EFFECTIVE REINFORCEMENT LEARNING FOR COLLABORATIVE MULTI-AGENT DOMAINS

QIANGFENG PETER LAU

Bachelor of Computing (Hons.)
Computer Science
National University of Singapore

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE

2012

A Blank Page

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis. This thesis has also not been submitted for any degree in any university previously.

Qiangfeng Peter Lau

12 September 2012

A Blank Page

Acknowledgements

To my dearest Chin Yee, thank you for the love, support, patience, and encouragement you have given me. To my parents and family, thank you for the concern, care, and nurture you have given to me since the beginning.

I appreciate and thank both Professor Wynne Hsu and Associate Professor Mong Li Lee for their patient guidance and advice throughout the years of my candidature.

I thank Professor Tien Yin Wong, the research and grading team at the Singapore Eye Research Institute for providing high quality data used in part of this thesis.

Special thanks to Assistant Professor Bryan Low and Dr. Colin Keng-Yan Tan for providing me with invaluable feedback that improved my work.

To my friends, thank you for the company, advice, and lively discussions. It would not have been the same without all of you.

I acknowledge and am thankful for the funding received from the A*STAR Exploit Flagship Grant ETPL/10-FS0001-NUS0. I have also benefited from the facilities at the School of Computing, National University of Singapore, without which much of the experiments in this thesis would have been difficult to complete.

Finally, I thank the research community whose work has enriched and inspired me to develop this thesis, and the anonymous reviewers whose insights have honed my contributions.

A Blank Page

Publications

Parts of this thesis have been published in:

1. Lau, Q. P., Lee, M. L., and Hsu, W. (2013). Distributed relational temporal difference learning. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. IFAAMAS

2. Lau, Q. P., Lee, M. L., and Hsu, W. (2012). Coordination guided reinforcement learning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pages 215–222. IFAAMAS

3. Lau, Q. P., Lee, M. L., and Hsu, W. (2011). Distributed coordination guidance in multi-agent reinforcement learning. In *Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 456–463. IEEE Computer Society

The other published works during my course of study related to the fields of retinal image analysis and data mining in order of relevance are:

1. Cheung, C. Y.-L., Tay, W. T., Mitchell, P., Wang, J. J., Hsu, W., Lee, M. L., Lau, Q. P., Zhu, A. L., Klein, R., Saw, S. M., and Wong, T. Y. (2011a). Quantitative and qualitative retinal microvascular characteristics and blood pressure. *Journal of Hypertension*, 29(7):1380–1391

2. Cheung, C. Y.-L., Zheng, Y., Hsu, W., Lee, M. L., Lau, Q. P., Mitchell, P., Wang, J. J., Klein, R., and Wong, T. Y. (2011b). Retinal vascular tortuosity, blood pressure, and cardiovascular risk factors. *Ophthalmology*, 118(5):812–818

3. Cheung, C. Y.-L., Hsu, W., Lee, M. L., Wang, J. J., Mitchell, P., Lau, Q. P., Hamzah, H., Ho, M., and Wong, T. Y. (2010). A new method to measure peripheral retinal vascular caliber over an extended area. *Microcirculation*, 17(7):1–9

4. Cheung, C. Y.-L., Thomas, G., Tay, W., Ikram, K., Hsu, W., Lee, M. L., Lau, Q. P., and Wong, T. Y. (2012). Retinal vascular fractal dimension and its relationship with cardiovascular and ocular risk factors. *American Journal of Ophthalmology*, In Press

5. Cosatto, V., Liew, G., Rochtchina, E., Wainwright, A., Zhang, Y. P., Hsu, W., Lee, M. L., Lau, Q. P., Hamzah, H., Mitchell, P., Wong, T. Y., and Wang, J. J. (2010). Retinal vascular fractal dimension measurement and its influence from imaging variation: Results of two segmentation methods. *Current Eye Research*, 35(9):850–856

6. Lau, Q. P., Hsu, W., Lee, M. L., Mao, Y., and Chen, L. (2007). Prediction of cerebral aneurysm rupture. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 350–357. IEEE Computer Society

7. Lau, Q. P., Hsu, W., and Lee, M. L. (2008). Deepdetect: An extensible system for detecting attribute outliers & duplicates in XML. In Chan, C.-Y., Chawla, S., Sadiq, S., Zhou, X., and Pudi, V., editors, *Data Quality and High-Dimensional Data Analysis: Proceedings of the DASFAA 2008 Workshops*, pages 6–20. World Scientific

8. Hsu, W., Lau, Q. P., and Lee, M. L. (2009). Detecting aggregate incongruities in XML. In Zhou, X., Yokota, H., Deng, K., and Liu, Q., editors, *Proceedings of the 14th International Conference on Database Systems for Advanced Applications (DASFAA)*, volume 5463 of *Lecture Notes in Computer Science*, pages 601–615. Springer

Contents

Summary

Online reinforcement learning (RL) in collaborative multi-agent domains is difficult in general. The number of possible actions that can be considered at each time step is exponential in the number of agents. This curse of dimensionality poses serious problems for online learning as exploration requirements are huge. Consequently, the learning system is left with lesser opportunities to exploit. Apart from the exploration challenge, the learning models for multiple agents can quickly become complex as the number of agents increase, and agents may have communication restrictions that vary dynamically with the state.

This thesis seeks to address the challenges highlighted above. Its main contribution is the introduction of a new kind of expert knowledge based on coordination between multiple agents. These are expressed as constraints that provide an avenue for guiding exploration towards states with better goal fulfilment. Such fragments of knowledge involving multiple agents are referred to as coordination constraints (CCs). CCs are declarative and are closely related to propositional features used in function approximation. Hence they may be (re)used for both purposes.

For a start, this work presents a centralized coordination guided reinforcement learning (CGRL) system that learns to employ CCs in different states. This is achieved through learning at two levels: the top level decides on CCs while the bottom level decides on actual primitive actions. Learning a solution in this augmented problem solves the original multi-agent problem. Coupled with relational learning, experiments show that CCs result in better policies and higher overall goal achievement than existing approaches.

Then, a distributed version of CGRL was developed for domains whereby communication between agents changes over time. This necessitates that learned parameters are distributed among agents. To do so, localized learning was designed for individual agents with coordination where possible. Thus, demonstrating that CCs are able to improve multi-agent learning in a distributed setting as well, albeit with some drawbacks in terms of model complexity.

Next, this thesis deals with issues of model complexity in the distributed case by introducing a distributed form of relational temporal difference learning. This is achieved by an agent localized form of relational features and a message passing scheme. The solution allows agents to generalize learning respectively over its interactions with other agents and among groups of agents whenever a communication link is available. The results show that the solution improves performance over non-relational distributed approaches while learning less parameters, and performs competitively with the centralized approach.

Subsequently, a novel preliminary application was developed for the medical imaging domain of retinal image analysis to illustrate the flexibility of multi-agent RL. The objective of retinal image analysis is to extract measurements from the vascular structure in the human retina. Interactively editing an extracted vascular structure from the retinal image to improve accuracy is cast as a collaborative multi-agent problem. Consequently, the methods described in this thesis may be applied. Experiments were conducted on a real world retinal image data set for evaluation and further discussion on how this application can be further improved.

Last, the thesis concludes and provides suggestions for future work for RL in collaborative multi-agent domains.

List of Figures

A Blank Page

List of Tables

List of Algorithms

A Blank Page

Glossary

A_i Action domain of agent i, also used to represent the agent itself. 14

$Perm(N, n)$ A function that returns the set of permutations of a subset of size n, i.e., the n-permutations, from the set $\{1, ..., N\}$. 84, 146

Q General action value function. 19, 65

Q^* Optimal action value function. 17

Q^π Action value function under policy π. 15

Q_i General agent decomposition of Q for agent i. 114

R_i Component i of decomposed reward function. 39

V General state value function. 19, 58

V^* Optimal state value function. 17, 64

V^π State value function under policy π. 15

$\Gamma(i)$ The set of neighbours in the CG for agent i in the current state. 36, 113, 155

α Step size parameter that controls the size of an update. 20

$\mathbf{a}_i$ A projection on the action tuple, $\mathbf{a}$, for action variables accessible by agent i. 113

$\mathbf{s}$ A joint state tuple from joint state space $\mathcal{S}$. 113

$\mathbf{s}_i$ A projection on the state tuple, $\mathbf{s}$, for state variables accessible by agent i. 113

ϵ-greedy Policy that takes a random action with ϵ probability and a greedy action with $1 - \epsilon$ probability. 20, 69, 73, 120

η Number unique agents from which the variables of a predicate comes from. 146

γ Discount factor in $[0, 1]$ used to control the impact of the future. 15

$\hat{Q}_i$ Single max-marginal of Q for agent i used in max-plus. 37

$\mathbf{a}$ Action (tuple) from joint action space $\mathcal{A}$. 14, 113

$\mathcal{A}$ Action space of an MDP, may be joint. 13

$\mathcal{A}_b$ Subset of the joint action space $\mathcal{A}$. 55

$\mathcal{P}$ Transitional probability model of an MDP. 13

$\mathcal{R}$ Reward model of an MDP, or a reward function. 13, 15, 39

$\mathcal{S}$ State space of an MDP. 13

$\mathcal{X}_N$ Set of all subsequences of the sequence $1, 2, ..., N$. 75

π Policy. 14, 15, 60, 114

π^* Optimal policy. 16, 64

ψ A two level system's sub-policy, i.e., $\psi(b, s)$. 57, 60, 117

ψ' A two level system's policy, i.e., $\psi'(\langle b, s \rangle)$. 57, 60, 117

$\vec{f}_{s,a}$ Feature vector for state s and action a. 22, 68

$\vec{w}$ Vector of weights for function approximation. 22, 68

f Feature or basis function for function approximation. 21, 68

q_i An additively decomposed component of Q that depends on agent i. 32

$q_{i,j}$ An additively decomposed component of Q that depends on pair of agents i, j. 32

w Weight for function approximation. 22, 68

BE Bucket elimination algorithm. 33, 74

CC Coordination constraints used to guide exploration. 8, 53, 116, 175, 188, 210

CG Coordination graph, may represent communication structure. 31, 74, 113, 122

COP Constraint optimization problem. 73, 121

DEC-MDP Decentralized Markov decision process. 112, 146

GLIE Greedy in the limit of infinite exploration. 20, 68, 70, 124

HRL Hierarchical reinforcement learning, often task-based. 6, 25

MDP Markov decision process. 6, 13

PF Propositional features based on predicates. 81, 123, 142

RF Relational features based on predicates. 84, 142, 146, 188

RL Reinforcement learning. 6, 15

RRL Relational reinforcement learning. 8, 46

RTS Real-time strategy (game). 2, 97

TD Temporal difference learning. 19, 68, 142, 188, 210

A Blank Page

Chapter 1

Introduction

An autonomous agent is one that is able to make sense of its environment and take logical actions to complete a task without human intervention. For example, a robotic vacuum cleaner moves to various locations in a room by itself in order to clean it. Assuming a simplified robotic vacuum cleaner that may only do one of: switching on or off the vacuum or move in a particular direction, the robot has to make sequential decisions, picking actions to perform at different points in time. Being able to make optimal decisions in appropriate situations to get the job done makes the robotic cleaner autonomous.

Put a team of these vacuum cleaners in the room and problems will appear. The cleaners may start to collide or re-clean parts of the room already cleaned by another cleaner resulting in less than optimal efficiency. One of the reasons for this is due to conflicting action selection, e.g. two robots each detect that no obstacle is on the left and right of them respectively, they collide when they try to move to the same spot simultaneously. Clearly, these robots will have to collaborate by coordinating their actions to achieve their aim of cleaning the room efficiently. In fact this is often necessary when multiple possibly conflicting actions have to be taken in parallel.

Many real world problems do not have known optimal solutions that work for all situations. This makes agents that are capable of learning to act and coordinate through experience particularly useful. However, with reference to our previous example, each

additional vacuum cleaner exponentially increases the space of actions to choose from. This has a direct effect on increasing the complexity of learning as well. Some real world problems that have large action spaces that can benefit from having efficient coordinated solutions include: computers that are part of a computing grid network deciding whether to reboot themselves when they begin to get faulty (Guestrin et al., 2001), animating crowds of autonomous virtual agents (e.g. humans, road traffic) in a life-like manner for realistic simulation or video rendering (Conde and Thalmann, 2006), urban traffic control (Kuyer et al., 2008), among others.

Over the years, there have been increasing interest for research in a myriad of multi-agent games. A popular genre is Real Time Strategy (RTS) games (Guestrin et al., 2003; Buro, 2004; Marthi et al., 2005; Wilson et al., 2007; Balla and Fern, 2009; Judah et al., 2010). These computer games normally include: multiple players, large numbers of player controllable units, tactical and strategic decisions, resource gathering, production, and exploration. The goal of each game may vary but is most often the annihilation of the enemy player's forces. Some popular commercial RTS games include: Warcraft™ and Starcraft™ series by Blizzard and the Age of Empires™ series by Microsoft. A typical instance of an RTS game requires the human player to issue strategic commands to anywhere between 10 to 100 units, while the computer aids the player in low-level tactical control. Coordination between the in-game units are often crucial to success.

Currently, the state of the art for Artificial Intelligence (AI) in commercial RTS games relies on hard-coded static scripts or finite state machines (Yue and de Byl, 2006). This results in AI for RTS games being of poor aid to the player, or mostly predictable and seldom a match for the human player. However, machine learning of fine-grained control for complex domains like RTS games is not straightforward. Beside their entertainment value, games like RTS also serve as a test bed for evaluating multi-agent machine learning ideas.

Figure 1.1: An example tactical RTS game of 10 versus 10 marines.

1.1 Efficient Multi-Agent Learning & Control

Consider a tactical RTS game in Figure 1.1 rendered by the Open RTS system (Buro, 2004). There are two teams, red and blue, of 10 marines each. The world has a square boundary, no obstacles and is fully observable with $240^2 = 57600$ locations. Each marine has a number of hit points representing its health that depletes to zero as the marine comes under attack. With 10 marines and considering positions alone, the upper bound on the state space is $\binom{57600}{10} > 10^{41}$. Assuming in the depicted state that each marine may take a step in one of the eight standard compass directions or simply idle, the joint action space for one team is 9^{10}. The large numbers in this example are due to the *curse of dimensionality* – as more agents are introduced, naive solutions for learning fine-grained control quickly become intractable.

The goal of this thesis is to further develop methods that *efficiently* learn control policies for collaborative multi-agent domains (e.g. Figure 1.1). Such a policy is a mapping from the state space to the joint action space. Here, 'efficiently' means that emphasis is placed on the overall learning rate rather than the actual computation of the mapping from state to action. In particular, the focus is on the online (incremental) learning problem where the collaborative agents seek to maximize the global goal while concurrently learning to do so. Incremental learning approaches are generally more

flexible in their application than batch learning. For example, if some generative model of the environment is available, agents may learn in simulation (offline), before being deployed to learn online in the real environment.

Achieving this goal brings about a variety of benefits. This includes better automation of real-world systems that involve multiple physical or virtual agents. Furthermore, better multi-agent AI can bring about a more interesting and challenging experience for millions of gamers world-wide.

1.2 Research Challenges

Highlighted next are a number of major research challenges that exists for machine learning in multi-agent domains. Overcoming these challenges will allow learning to generalize more effectively to various multi-agent domains with similar issues.

1.2.1 Exploration Versus Exploitation

To learn good coordinated policies agents need to explore, however to achieve the goal they need to exploit. Already an ongoing research problem in single agent domains, this trade-off is further aggravated in multi-agent domains due to the exponentially large joint action space to be explored. The large action space tilts the trade-off towards lengthy exploration as otherwise, learned information will be unreliable for effective exploitation. For online learning to be viable, exploration has to be better managed. Hence the number of agent interactions with the environment for learning a good coordinated policy has to be optimized.

1.2.2 Limited Communication & Distribution

Closely related are the issues of communication and distribution. Collaborative agents that work towards a global goal may communicate with each other to better solve the problem. However, it may not always be the case that all agents can communicate with every other agent or it may incur too much overhead to do so. Where the communi-

cation structure between agents are fixed, e.g. computers on a wired network, we may be concerned with fine-grained issues such as the quantity of messages passed between them. But, a bigger challenge is when communication is dynamic, for example: network links are disrupted, agents are physically mobile, or removed from the game in the case of RTS, resulting in isolated groups of agents. Here, distribution is important as critical components of the machine learning method must not entirely reside in any one agent.

1.2.3 Model Complexity & Encoding Knowledge

Machine learning often requires optimizing the parameter values of some model. With multiple agents, the parameters to be learned may also increase exponentially. This results in slow learning and poor generalization. Furthermore, models usually requires the user to encode some expert background knowledge of the problem. For multiple agents, this may become a tedious process for the user. Simplifying the representational requirements of such knowledge will have a direct impact of the practical applicability of the learning method. Furthermore, representations rich in semantics may offer generalization capabilities over the large joint state and action spaces. Such generalizations have the potential to improve learning efficiency.

1.2.4 Others

Highlighted above are some main challenges for multi-agent learning that this work intends to confront. By no means is the list exhaustive. Notably, joint policy execution, i.e., the mapping from the current state to an action for each agent, must be computed in a reasonable amount of time. Other challenges in multi-agent domains include: handling non-stationary environments that arise if agents learn independent solutions without communication, partial observability due to the presence of other agents or imperfect sensors, synchronization between agents, heterogeneous agent roles, team formation and discovery, credit assignment, and adversarial settings where agents have selfish goals.

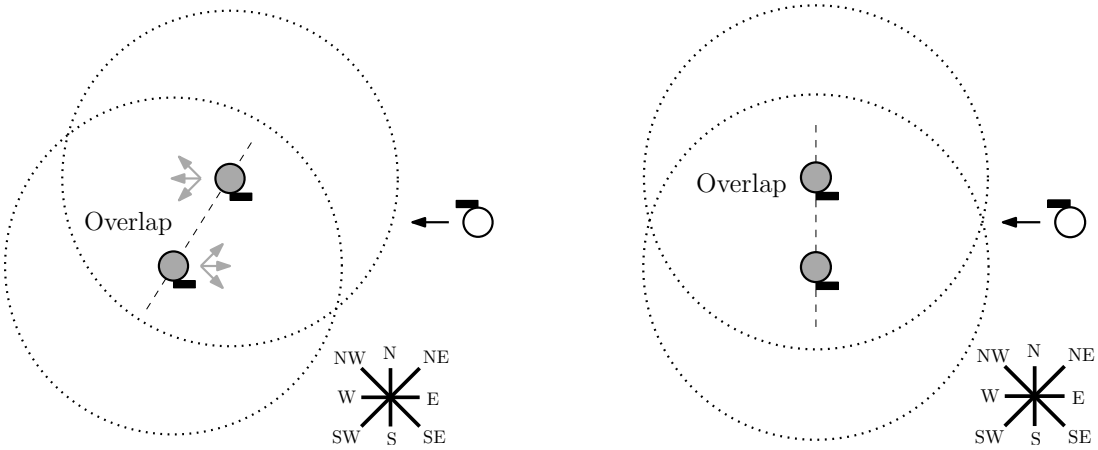
1.3 Existing Approaches & Gaps

Advances in the area of reinforcement learning (RL) have made applications to problems that have high dimensional joint action spaces increasingly practical. The basic RL framework is a machine learning method that learns to maximize a reward signal over a given discrete time horizon from interaction with the environment. It is particularly attractive for problems where a complete notion of optimality is unknown (Sutton and Barto, 1998). Furthermore, with a good environment simulator, it is possible to learn offline and continue to improve the learned results online in an incremental fashion. This allows an intelligent agent to continue to learn from and adapt to changes in the environment.

Expert knowledge is commonly employed in large-scale RL in a variety of ways. In particular, hierarchical RL (HRL) handles single agent Markov decision processes (MDPs) by recursively partitioning them into smaller problems using a task hierarchy (Sutton et al., 1999; Dietterich, 2000; Andre and Russell, 2002). The task hierarchy constrains the solution space (policies) of the learning problem so that only relevant actions for a task can be selected at each time step. Learning a good task selection policy will direct exploration towards the more promising parts of the MDP, mitigating some challenges in lengthy exploration.

As described, learning to make sequential decisions for multiple collaborating agents is a difficult problem in general. The HRL and other task-centric single agent methods have been adapted into a multi-agent setting. At each time step, agents are individually constrained to the actions allowed for the current task they are assigned to. Task assignment may be part of the learning process (Marthi et al., 2005; Ghavamzadeh et al., 2006) or derived separately (Proper and Tadepalli, 2009). Once each agent's task is selected, it will have a constrained (reduced) set of actions to consider. However, this framework cannot be easily extended to incorporate constraints based on coordination among multiple agents. Illustrated below, is an example of the useful effects in directing exploration that are derived from coordination-based knowledge.

Example 1.1 (RTS coordination knowledge)**.** *Consider the simplified view of states*



(a) Before alignment **(b)** After alignment

Figure 1.2: Example of coordination-based knowledge in a tactical RTS game. Figures show simplified states. Solid circles represent marines (2 for grey team and 1 for white team). Dotted circles indicate the range of their rifle (black bar). Black arrow indicates movement direction for white marine. Dashed line shows alignment of grey marines. Grey arrows in (a) are movement actions that may result in the state in (b).

from a tactical RTS game in Figure 1.2. Two marines from the grey team are shown with an oncoming enemy marine from the white team. The objective is to destroy the enemy. Each marine may move in 8 compass directions as shown in the figure or stay put and shoot at the enemies within range of its rifle. In Figure 1.2a, the joint action space of the grey team is $9^2 = 81$. *Careful examination reveals that much of this space is of less importance for exploration as they may not lead quickly to the goal of destroying the enemy. For example, the grey marines should not move in such a way that prevents both from shooting at the enemy at the same time. An ideal situation is illustrated in Figure 1.2b where the overlapped shooting range is aligned to the enemy's approach. Once the shooting begins, the white marine may only shoot at one grey marine while both grey marines can shoot at it. With this simple coordination strategy, the joint action space in Figure 1.2a is reduced by 81% to that of* $\{NW, W, SW, stay\} \times \{NE, E, SE, stay\} - \{\langle stay, stay \rangle\}$ *which has a size of 15.*

The effects of multi-agent coordination knowledge described in Example 1.1 do not fit well within procedural task definitions. This is because the user of task-based systems will have to encode increasingly complex joint procedural tasks as the number

of agents increases. Existing works usually delegate such coordination knowledge as basis features (Marthi et al., 2005) or as static rules (Proper and Tadepalli, 2009). This lack of an active involvement of coordination knowledge for exploration serves as a strong motivation to further investigate its utility in directing exploration for multi-agent problems.

Another line of works use static fixed heuristics to bias the policies of the original agents towards better exploration (Bianchi et al., 2007; Zhang et al., 2009, 2010). However, many of these and the task-based methods do not fully satisfy the distribution requirements we have described in Section 1.2.2. Namely, they are either centralized approaches (Marthi et al., 2005; Proper and Tadepalli, 2009) or they have learning components that rely on a fixed communication structure between the agents (Zhang et al., 2009, 2010).

In terms of handling model complexity (see Section 1.2.3), various existing works seek to generalize or approximate similar model parameters to reduce the number of parameters to be learned (Stone and Sutton, 2001; Silver et al., 2007; Sutton and Barto, 1998, chap. 8). Of particular interest are the works in relational reinforcement learning (RRL) that make use of declarative relationships between objects to generalize model parameters (Guestrin et al., 2003; Tadepalli et al., 2004; Asgharbeygi et al., 2006). However, study on how RRL may be used in a multi-agent setting is preliminary (Croonenborghs et al., 2005; Ponsen et al., 2010). Hence more work is needed to bring RRL ideas into the distributed case.

1.4 Overview of Contributions

The main contribution is to exploit coordination knowledge in multi-agent RL to improve the learning rate of useful policies by modelling coordination among agents as hard constraints. These hard constraints are referred to as coordination constraints (CCs), and are used to guide (limit) the joint action space for exploration. Unary constraints defined on single agents are a special case of CCs.

CCs dynamically depend on the state to guide exploration. Deciding which CCs to employ in different states is part of the learning process, enabling the RL system to learn to guide itself during exploration. The next example describes situations where this is desirable.

Example 1.2 (RTS dynamic coordination)**.** *In Example 1.1, the CC that indicates the grey marines movements should be constrained may not be suitable if one of the grey marines is badly wounded. It may make more sense for the healthy grey marine to engage the enemy first while the wounded marine supports from behind.*

The new learning methods must be able to handle most of the challenges identified in Section 1.2. In addition, it should integrate easily with existing works in multi-agent RL as far as possible. This thesis makes the following specific contributions.

1.4.1 Coordination Guided Reinforcement Learning

First, the problem is approached from a centralized perspective. This corresponds to domains where communication is free, e.g. an RTS game. A model-free two level RL system is presented where the top level learns to place CCs on the bottom level to guide exploration for the solution to the original problem (Lau et al., 2012). Equations for two level temporal difference learning are formulated and we describe how action selection under constraints can be computed. Next, we highlight the close relationship between basis features used in function approximation and CCs that allows the user to reuse definitions for both. The system can be incorporated with RRL and other existing works on coordination. Experiment results show that the inclusion of coordination knowledge in guiding exploration outperforms existing methods.

1.4.2 Distributed Coordination Guidance

The second contribution distributes the coordination guided RL system into the individual agent boundaries such that no critical component resides in any one agent (Lau et al., 2011). Each agent carries their own portion of learned parameters. This allows

the system to be applied in domains where agents' communication structure changes over time. In particular, agents are able to learn from local information by communicating with their current neighbours and observing their local reward. Experiment results show that agents are able to learn more effectively with CCs under the communication restrictions.

1.4.3 Distributed Relational Reinforcement Learning

The third contribution deals with issues of model complexity in the distributed setting. Existing work in centralized RRL provides the means to generalize learning over semantically similar situations. This results in models with less parameters to learn. However, there has been little work in this regard for the distributed setting. As the use of CCs introduces more learning parameters into the system, it will be prudent to mitigate the increase in parameters for the distributed case. We propose an internal and external relational generalization scheme for multiple agents (Lau et al., 2013). Agents provide locally learned parameters to their current neighbours that they may communicate with. By combining such information from each agent's neighbours with respect to relational semantics, experiences can be shared among agents. These ideas are incorporated with existing multi-agent distributed RL and the work in distributed coordination guidance. Experiment results show improvement in learning efficiency and competitiveness with centralized RL methods.

1.4.4 Application in Automating Retinal Image Analysis

The last contribution investigates a preliminary prototype application of the ideas in this thesis to the real-world domain of retinal image analysis. This is a novel application of RL to the field of retinal image analysis. Currently, the state of the art for practical large scale retinal image analysis involves a computer assisted feature extraction process (Cheung et al., 2010, 2011a,b, 2012). Much manual time and expertise are required to verify and edit the extracted vascular structure of retinal images before measurements of interests are recorded. Hence the motivation for a fully automated

solution. We formulate the correct editing of retinal vascular structure as a multi-agent MDP problem. Subsequently, we integrate RL solution methods with retinal image analysis system currently in active use. Last, we analyse the utility of our prototype application on real world data from population studies and discuss directions in which the solution can be improved as well as the challenges faced. This application demonstrates the wide applicability of multi-agent RL methods.

1.5 Organization

The rest of the thesis is organized as follows, Chapter 2 describes the basic details of RL that are important for our task. Next, Chapter 3 reviews the literature for coordinated multi-agent reinforcement learning. Then, we present the centralized coordination guided RL system and describe CCs in detail in Chapter 4. In Chapter 5, we develop a distributed version of coordination guided RL that is applicable in domains with dynamic communication between agents. Further in Chapter 6, we describe methods for relational generalization in the distributed case. Subsequently, we present the preliminary application in retinal image analysis in Chapter 7. Last, in Chapter 8, we conclude and discuss the possible directions for future work.

A Blank Page

Chapter 2

Preliminaries

In this chapter, we review the basic concepts that form the foundation of reinforcement learning such as: Markov decision processes, value functions, policies, and optimality. Then, we highlight the temporal difference learning algorithm, its variants, and a common function approximation method used for large value functions. Finally, we present in brief the concept of semi-Markov decision processes that is used in the following chapter on related work.

2.1 Markov Decision Processes

Reinforcement Learning is a learning framework for a class of discrete time decision problems that are called finite Markov decision processes (Puterman, 1994; Bertsekas and Tsitsiklis, 1996). More formally, a Markov decision process (MDP) is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$ where,

- $\mathcal{S}$ is a finite set of states,
- $\mathcal{A}$ is a finite set of actions,
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is a reward model such that $\mathcal{R}(s, a, s')$ is the expected value of reward, r, for taking action a in s and reaching state s', where r is the reward signal received from the environment,
- $\mathcal{P}$ is a transitional probability model such that $\mathcal{P}(s'|s, a)$ gives the probability that taking action a in state s will result in reaching state $s' \in \mathcal{S}$.

A typical episodic MDP starts from an initial state, s_0, and experiences a sequence of transitions,

$$s_0, a_0, r_0, s_1, a_1, r_1, ..., s_t, a_t, r_t..., s_\emptyset$$

where s_t, a_t, r_t is the state , action and reward at time t respectively, and $s_\emptyset$ is the terminal state. Such a sequence is termed an *episode* and arbitrary sequences can also be referred to as *trajectories*. Episodic MDPs are special cases of continuous (infinite horizon) MDPs, that do not end. Continuous MDPs may represent episodic MDPs simply by specifying that every terminal state $s_\emptyset$ transits to itself with certainty regardless of action taken.

A solution to the MDP is a policy, $\pi : \mathcal{S} \mapsto \mathcal{A}$. This is a function that maps the current state to an action to be taken in it. Alternatively, π may be a stochastic policy, in which case it will be defined as $\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. Then the stochastic policy will take action a in state s with probability $\pi(s, a)$. Being able to develop a policy to make decisions based on the current state reveals that some assumption of the problem is made. This assumption can be summarized as the *Markov Property*. In short, this means that the probability of reaching the next state and next reward depends only on the current state and the action taken in it, i.e. the current state is informative enough to include all necessary past information required within it.

A simple extension of the MDP to the multi-agent case is that of the *factored* MDP where for N agents the joint action space may be factored into multiple variables, i.e., $\mathcal{A} = A_1 \times A_2 \times ... \times A_N$. Then, a joint action in this space is given by $\mathbf{a} = \langle a_1, ..., a_N \rangle$, where $a_i \in A_i$. The notation, $\mathbf{a}$, is used when there is a need to emphasize the action is a tuple of values, otherwise we simply write a. Note that to facilitate discussion we have assumed a one to one mapping between agent and action variables. Agents that have more than one action variable may be presumed to be composed of other sub-agents.

2.2 Reinforcement Learning

In reinforcement learning (RL), we wish to learn a policy by interacting with the environment (Sutton and Barto, 1998) without given transition ($\mathcal{P}$) and expected reward ($\mathcal{R}$) models. At each time step, the environment sends the state and a reward signal to the agent from a user specified reward function[1] that encodes the goal of the problem. RL usually learns value functions and derive policies from them. One such function is the *state-value function* $V^\pi : \mathcal{S} \mapsto \mathbb{R}$. $V^\pi(s)$ gives the expected return[2] of being in state s if the policy π is being followed,

$$V^\pi(s) = E_\pi\{R_t | s_t = s\} \tag{2.1}$$

$$= E_\pi\{\sum_{t'=0}^{\infty} \gamma^{t'} r^{t+t'} | s_t = s\} \tag{2.2}$$

$$= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^\pi(s')] \tag{2.3}$$

Equation 2.1 states that the state value function based on policy π gives the expected total reward, R_t beginning from the current state s_t and subsequently experiencing new states by taking actions determined by π until termination. Equation 2.3 is a *Bellman equation* that shows how $V^\pi(s)$ is related to its successor states $V^\pi(s')$ under the stochastic policy π. Essentially, we sum over each possible action and each possible successor state, the reward experienced and the expected reward from there on after. The real variable $\gamma \in [0, 1]$ is a constant called the *discount factor* that controls proportionally how much future states matter to the current state.

Another function of interest that RL may seek to learn instead of V^π is the *action value function*, $Q^\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. $Q^\pi(s,a)$ returns the expected total reward of taking

[1] Where it is clear, we also write the reward function as $\mathcal{R}$.
[2] A return is a function on the sequence of rewards.

action a in state s and following policy π to termination, i.e.,

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} \tag{2.4}$$

$$= \sum_{s'} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^{\pi}(s')] \tag{2.5}$$

Equation 2.4 states that the expected total reward R_t depends on the current state s_t and action a_t taken in it. Equation 2.5 expresses $Q^{\pi}(s,a)$ in terms of the state-value functions of successor states, $V^{\pi}(s')$. By substituting Equation 2.5 in 2.3 we have,

$$V^{\pi}(s) = \sum_{a} \pi(s,a) Q^{\pi}(s,a) \tag{2.6}$$

$$= Q^{\pi}(s, \pi(s)) \tag{2.7}$$

where Equation 2.7 is a shortened notation for expectation over a. Then, the Bellman equation for Q^{π} relating the current state and the next is,

$$Q^{\pi}(s,a) = \sum_{s'} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma Q^{\pi}(s', \pi(s'))] \tag{2.8}$$

With these value functions, policies may be derived from them in a few ways. For example, the ϵ-greedy policy always chooses a greedy action,

$$\pi(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q^{\pi}(s,a) \tag{2.9}$$

with probability $1 - \epsilon$ and an arbitrary non-greedy action otherwise.

A basic approach to represent action value functions such as Q, would be the tabular approach. Each entry in the table for Q is keyed by the state and action, and has the corresponding data value, $Q(s,a)$. Although the tabular approach is feasible for small state and action spaces, it does not scale well to larger domains such as multi-agents ones. Other common methods to approximate functions will be mentioned in Section 2.4.

The optimal policy, π^* for an MDP, always seeks to maximize its expected rewards

in any given state. Such a policy necessarily has optimal state and action value functions, V^* and Q^*. These give higher or equal expected return when compared to any other value functions for arbitrary policy π, V^π and Q^π. Otherwise a new optimal state and action function can be constructed using the policy that gives a higher value, contradicting the fact that V^* and Q^* are already optimal. Therefore,

$$V^*(s) = \max_\pi V^\pi(s) \tag{2.10}$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \tag{2.11}$$

and the *Bellman optimality equations* for the optimal value functions are,

$$V^*(s) = \max_a Q^*(s,a) = \max_a \sum_{s'} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^*(s')] \tag{2.12}$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma \cdot \max_{a'} Q^*(s',a')] \tag{2.13}$$

Then, the deterministic optimal policy π^* based on V^* or Q^* is given as,

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^*(s')] \tag{2.14}$$

$$= \operatorname*{argmax}_a Q^*(s,a) \tag{2.15}$$

From Equation 2.15, one can observe that the optimal policy is a greedy policy.

2.2.1 Model-Free Versus Model-Based Learning

The benefit of learning action value functions over state value functions is obvious. Learning V^* is called a *model-based* approach as it requires learning the state transition model $\mathcal{P}$ and reward model $\mathcal{R}$ for action selection (see Equation 2.14). These models can be difficult to specify in general and especially so for multi-agent domains that require them to be decomposable for efficient action selection. However, by directly learning action value function Q^* in the *model-free* approach, we immediately have π^* without any special requirements on $\mathcal{P}$ and $\mathcal{R}$ (see Equation 2.15). Furthermore, when

a good model design is available, RL with Q, planning, and model building can be integrated rather independently using search based methods (Gelly and Silver, 2007; Silver et al., 2008; Sutton and Barto, 1998, chap. 9.2).

In general, to learn policies, model-free RL uses *policy iteration* that consists of two main steps. *Policy Evaluation* seeks to learn the value function (e.g. Q^π) for the current policy π. *Policy Improvement* seeks to improve the current given policy π. This is achieved by changing the action to take in the state by maximizing the learned value function (e.g. Equations 2.14 and 2.15) from the previous policy evaluation step. The two steps described are repeated in sequence iteratively until the policy and its value function converges to the optimal policy.

2.2.2 Direct Policy Search Versus Value Functions

Another approach to RL is to avoid learning the value functions and directly learn the stochastic policy, $\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. An example of this on a small two agent soccer domain through gradient descent on a differentiable policy is reported in Peshkin et al. (2000). They find that learning the action value function performs poorer than policy search when there is partial observability. A more comprehensive empirical comparison was done by Kalyanakrishnan and Stone (2009). They found that RL with value functions can learn significantly faster than policy search when accurate function approximation is available (see Section 2.4) and the Markov property holds. Conversely, when there is partial observability or the function approximation is poor, policy search may learn better policies than value function approaches in the long term. But, value functions perform better in the short term.

Interestingly, Kalyanakrishnan and Stone (2009) found that combining the two approaches, by first learning the value function then the policy directly, can yield the benefits of both approaches. This results in better performance than either method alone. Guestrin et al. (2002) also presented a method to fuse both approaches for multi-agent domains by first learning the value function. Since RL with value functions has shown to be an important first step in fusion with policy search methods, we focus on learning

value functions in this thesis.

The next few sections details an incremental approach to RL for action value functions called temporal difference learning, and other methods in its context that are relevant to applying model-free RL. For the rest of this chapter, whenever the policy is obvious we use V and Q to denote the state and action value function respectively.

2.3 Temporal Difference Learning

Temporal difference (TD) learning is a method to combine both policy evaluation and policy improvement into a single step for policy iteration (Sutton, 1988). In its simplest form, it allows an RL agent to update its value function after each action is taken. This makes the updates incremental and allows an agent to adapt to changes during the course of a single episode.

Assuming that tabular functions are used, the outline of a general TD algorithm for a single episode is shown in Algorithm 2.1. At each iteration, the next state and reward in the current time step is observed, then an update for action function Q is performed before the subsequent iteration. Note that there are two policies involved, the exploration policy π and the learning policy ψ. If $\pi = \psi$, i.e. the exploration policy is also the learning policy, then an *on-policy* update method is said to be used, otherwise the TD algorithm uses an *off-policy* update method. Both methods of updating have been proven to cause Q^{ψ} to converge to Q^{*} in the limit of episodes being experienced if some constrains on π and ψ are observed. Two popular methods, SARSA and Q-

Algorithm 2.1 General TD learning algorithm for one episode

Input: exploration policy π, learning policy ψ, initial tabular action value function Q
Output: updated function Q

1: Observe the initial state s_0
2: $s \leftarrow s_0$; $a \leftarrow \pi(s)$ # Compute action for s
3: **while** $\neg terminal(s)$ **do**
4: Take action a, observe the next state s' and reward r.
5: Compute an update for the value $Q(s, a)$ using s', r and policy ψ.
6: $s \leftarrow s'$; $a \leftarrow \pi(s)$
7: **end while**

learning, for on-policy and off-policy updates respectively are presented next to explain the details of the TD learning algorithm.

2.3.1 SARSA

SARSA (Rummery and Niranjan, 1994; Sutton, 1996) is a form of on-policy TD learning algorithm that performs updates using the tuple $\langle s_t, a_t, r_t, s_{t+1}, a_{t+1} \rangle$, hence SARSA. Being an on-policy update method, the exploration policy, $\pi = \psi$, the learning policy. Therefore, in Algorithm 2.1 Step 5, the following update is performed,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, \pi(s_{t+1})) - Q(s_t, a_t)] \tag{2.16}$$

where α is the *step size* parameter that controls the learning rate – a value of zero indicates no learning is taking place. In some applications, α is dependent on t and decays with time. If π is a Greedy in the Limit of Infinite Exploration (GLIE) policy and if α fulfils the conditions,

$$\sum_{t=1}^{\infty} \alpha_t = \infty \qquad \text{and} \qquad \sum_{t=1}^{\infty} \alpha_t^2 < \infty. \tag{2.17}$$

Then under SARSA, Q will converge to the optimal action function Q^* (Rummery and Niranjan, 1994). An example of a step size that satisfies the above conditions is $\alpha_t = 1/t$. A GLIE policy is one that will visit each action infinite number of times and converge to the greedy policy in the limit. An example of such a policy is the ϵ-greedy policy, where an exploratory action is chosen with ϵ probability and a maximal action with $1 - \epsilon$ probability. As the parameter ϵ decays towards zero over time the policy becomes increasingly greedy in the limit.

2.3.2 Q-learning

Q-learning (Watkins, 1989) is an off-policy update method that uses an exploration policy (π) to take actions in the environment but uses the greedy policy (ψ) for updates

to the action function. Essentially in Algorithm 2.1, for Step 5 the following update is performed,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \cdot \max_{a^* \in \mathcal{A}} Q(s_{t+1}, a^*) - Q(s_t, a_t)] \tag{2.18}$$

Similar to SARSA, if the conditions for α in Equation 2.17 is met, Q-learning will converge to the action value function Q^* for the optimal policy π^*. Although Q-learning may learn optimal policies, it may not perform as well in practice as SARSA in the online case as it does not take into account the randomness of the exploration policy. The two methods' policies will become equivalent if the exploration policy is GLIE.

2.4 Function Approximation

In MDPs with large state and action spaces, tabular representation for action value functions quickly become infeasible. To this end, function approximation is often used to estimate Q. In general, any incremental regression technique may be used to estimate Q. One popular method for function approximation is the use of Artificial Neural Networks (ANN). This approach has shown to be very effective in creating game players for Backgammon (Tesauro, 1994) and is widely used in other domains.

Another popular and simpler approach to function approximation that has also been studied widely is linear function approximation (Lagoudakis and Parr, 2001; Stone and Sutton, 2001; Guestrin et al., 2002; Marthi et al., 2005). In linear function approximation for action value functions, we approximate Q through a set of *basis functions*, $f_i \in F$. Each basis function $f_i : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is also called a feature and encodes some aspect of the state action space. For example, in the domain of a single marine RTS game, $health(s, a)$ may be the health level of the marine in state s regardless of the action and $collide(s, a)$ may be a binary feature that returns a value in $\{0, 1\}$ if the marine will collide with some object after taking action a.

The function Q may be approximated in linear form using N features by,

$$Q(s,a) = \sum_{i=1}^{N} w_i f_i(s,a) \tag{2.19}$$

where w_i is the corresponding weight of each feature to be learned. Gradient descent methods may then be used for TD updates to the weights. Let $\vec{w} = \langle w_1, ..., w_N \rangle$ be a vector of the weights and $\vec{f}_{s,a} = \langle f_1(s,a), ..., f_N(s,a) \rangle$ be a vector of the feature values for state s and action a. Note that in the multi-agent case, features may only depend on a subset number of action variables.

The general gradient descent update equation for learning is,

$$\vec{w} \leftarrow \vec{w} + \alpha[r + \gamma Q(s', \pi(s')) - Q(s,a)]\nabla\vec{Q}_{s,a} \tag{2.20}$$

where $\nabla\vec{Q}_{s,a}$ is a vector of update gradients (e.g. partial derivatives) of Q for s, a. The commonly used $\nabla\vec{Q}_{s,a}$ in TD updates for SARSA and Q-learning respectively are,

$$\vec{w} \leftarrow \vec{w} + \alpha[r + \gamma Q(s', \pi(s')) - Q(s,a)]\vec{f}_{s,a} \tag{2.21}$$

$$\vec{w} \leftarrow \vec{w} + \alpha[r + \gamma \cdot \max_{a^* \in \mathcal{A}} Q(s', a^*) - Q(s,a)]\vec{f}_{s,a} \tag{2.22}$$

These update gradients are commonly used because of their computational complexity that is typically linear in the number of weights. Update time can be further reduced the special case of binary features if implementations are optimized for $f_i(s,a) = 0$ and $\vec{f}_{s,a}$ is sparse. The SARSA update in Equation 2.21 converges within an error bound of the optimal approximation, where the error with the true function is minimal, with same condition at Equation 2.17 for α (Tsitsiklis and Roy, 1997). Additionally, with more stringent conditions, SARSA has been show to converge to the optimal approximation (Perkins and Precup, 2002; Melo et al., 2008). But in the case of off-policy methods such as Q-learning, learning updates with Equation 2.22 is not proven to converge to an optimal approximation to Q^* in general due to the gradient (Baird, 1995). A better gradient was presented in Lagoudakis and Parr (2001), but at the cost of quadratic time

complexity. More recently, a convergent off-policy and linear time complexity updating method was presented in the works of Sutton et al. (2009b,a) that uses two types of gradients.

2.5 Semi-Markov Decision Processes

A generalization of the MDP with actions that may take more than one time step to complete is the semi-Markov decision process (SMDP). This model is necessary for explaining some of the related work in Chapter 3. SMDP may be defined as modifying the reward model $\mathcal{R}$ and transitional probability model $\mathcal{P}$ to include provision for an action $a \in \mathcal{A}$ that may take N number of time steps to complete (Sutton et al., 1999; Dietterich, 2000). $\mathcal{R}$ is redefined as $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{Z}^+ \mapsto \mathbb{R}$, and $\mathcal{P}$ is now the joint probability that given state s and action a we reach state s' in N time steps, i.e. $\mathcal{P}(s', N|s, a)$. The Bellman equation, one step SARSA and one step Q-learning updates are then,

$$Q(s,a) = \sum_{s'} \mathcal{P}(s', N|s, a)[\mathcal{R}(s, a, s', N) + \gamma^N Q^\pi(s', \pi(s'))] \tag{2.23}$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r_N + \gamma^N Q(s', \pi(s')) - Q(s,a)] \tag{2.24}$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r_N + \gamma^N \max_{a^* \in \mathcal{A}} Q(s', a^*) - Q(s,a)] \tag{2.25}$$

Note that for Equations 2.24 and 2.25, r_N is the sum of the reward received from the environment over N time steps.

A Blank Page

Chapter 3

Literature Review

This chapter reviews selected approaches to handle the challenges of model-free reinforcement learning (RL) in multi-agent problems. We begin by reviewing hierarchical task based methods that were developed to improve the learning rate for single agent Markov decision processes (MDPs) by using procedural expert knowledge. Next, methods to handle the various important aspects of multi-agent RL such as joint action selection, value function representation, and learning updates are described. Then, hierarchical learning concepts for multi-agent problems are discussed. Finally, we briefly review reward shaping and relational reinforcement learning. The ideas introduced in the subsequent chapters will be discussed in relation to some of the works highlighted in this chapter.

3.1 Single Agent Task Based Learning

This section presents an overview of the methods used to improve the learning rate of single agent reinforcement learning. Hierarchical RL (HRL) approaches were originally conceived to deal with the problem of lengthy exploration trajectories through the MDP before rewards are encountered. In other words, the problem was one of the depth of exploration required. This is resolved by the introduction of tasks – procedural sub-problems that may in turn be decomposed further to sub-tasks. The effect is to decompose the original MDP into sub-MDPs (i.e., sub-problems) where more focused

solutions (sub-policies) may be discovered. Tasks correspond to their same real-world concept humans use and are encodings of expert knowledge.

Choosing of tasks is part of the RL process, hence tasks are higher level actions that are represented in a task hierarchy. As tasks may last multiple time steps, these methods use SMDPs (see Section 2.5) as the problem formulation. This allows HRL to propagate rewards multiple steps back in time where the high level task action is first selected. Hence the temporal abstraction. The specification of a task generally consists of a few components that give rise to a sub-MDP:

1. A set of starting states in which the task may be initialized. That is, the action to perform the task is available for selection.
2. A set of terminal states that ends the task, returning control to the parent task. This may be specified as a probability distribution over states that depends on the history of visited states within the task.
3. A sub-policy for the sub-MDP. The sub-policy limits the agent's actions to only those allowed for the task.
4. An optional pseudo-reward model that is used to improve task transfer among problems and encourage faster learning in the task. Introducing pseudo-rewards is a double-edged sword as they may pollute the original reward signal resulting in sub-optimal behaviour.

Next, we briefly review the three common HRL methods: Options (Sutton et al., 1999), MAXQ decomposition (Dietterich, 2000), and hierarchical abstract machines (Parr and Russell, 1997; Andre and Russell, 2002; Marthi et al., 2006).

3.1.1 Options

Sutton et al. (1999) details how RL may work with temporally extended action, i.e. a sequence of actions that follow their own given policy. Such temporally extended actions are called *options*. Options may come in two variants:

1. A Markov option is similar to an RL formulation.

2. A semi-Markov option has a policy that considers histories of the states and actions taken within the option and also for its a termination probability distribution. The latter allows options to self-terminate after a certain amount of time.

An MDP augmented with a set of options becomes an SMDP. The framework was presented as a two level hierarchy with the RL framework with its standard actions (called primitive actions), extended with options. It was shown that options can speed up learning and convergence to the optimal value functions under certain conditions.

An interesting point to note is that options may be interrupted during execution allowing more flexibility in their usage. Interruption also allows convergence to optimality for options that do not fit well in their entirety as a sub-task to a problem by prematurely truncating them after certain number of steps. Choosing to interrupt is done by comparing the state value with the action value, i.e., for current option o, if $Q(s, o) < V(s)$, then terminate o.

3.1.2 MAXQ Decomposition

Dietterich (2000) proposed another HRL method called MAXQ decomposition. Using this method, the value function for an RL task is hierarchically decomposed to additive sub-task components. The purpose of such decompositions of the Q function is to focus the observed rewards on the relevant sub-task (or sub-routine) in hope that the learning rate for the sub-task will be improved. Each sub-task component is a miniature MDP problem. Solving each of them will solve the original MDP. Sub-tasks are added to the action space $\mathcal{A}$ much like how options are added as actions in Sutton et al. (1999). Within a sub-task, primitive actions or calls to other sub-tasks may be performed. In essence, introducing extra decisions in the form of calling sub-tasks partitions the single action space in a hierarchical manner. Good design of these subtasks and their hierarchy has been shown to improve learning.

This hierarchical decomposition of the value function can be represented as a directed acyclic graph, called a MAXQ graph, to be executed as described in the following examples.

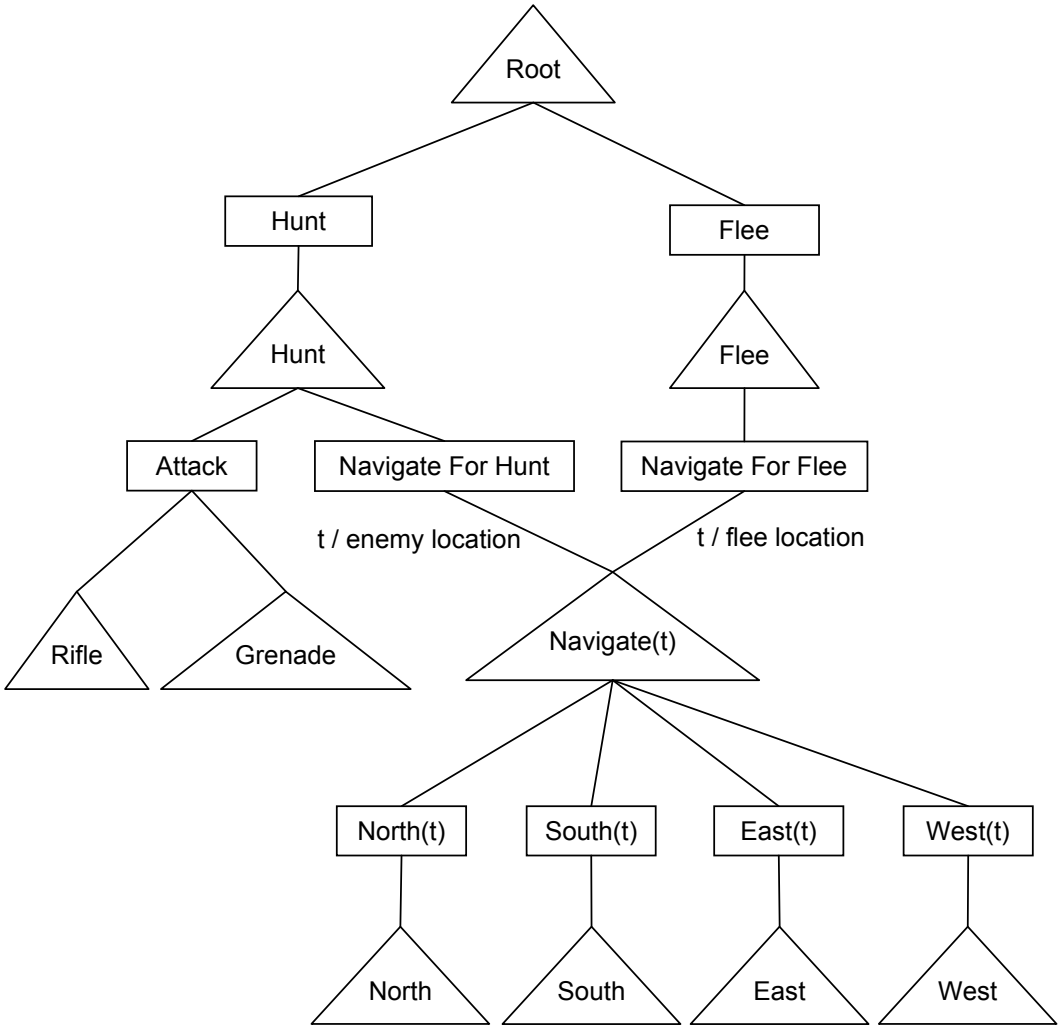


Figure 3.1: Example of a MAXQ decomposition graph, shown for a single marine RTS game. Actions are: 4 movement directions, and usage of 2 weapons. Rectangles are Q nodes and triangles are MAX nodes. Direction of edges are top-down.

Example 3.1 (MAXQ graph)**.** *Suppose that we have a simple RTS game whereby each team consists of one marine armed with a rifle and a grenade in a small square environment with no obstacles. The marine may move in one of four standard compass directions or use one of its weapons. The episode ends when the enemy marine is defeated (positive reward), the marine is defeated (negative reward) or fled from battle (zero reward). One possible MAXQ graph is given in Figure 3.1. The triangles are called MAX nodes that correspond to primitive actions (leaf nodes) or sub-tasks. The rectangles are called Q nodes and they correspond to the possible action or sub-task selections available to choose from in a particular sub-task. All edges are top down directed from the root node.*

Example 3.2 (MAXQ execution)**.** *An example execution of a MAXQ RL agent using this graph will first begin from* Root*. A decision has to be made between two sub-tasks: to Hunt the enemy, or to Flee the battle. Once a decision is made, the sub-task is executed. When some terminating condition is met, control is returned from a sub-task back to the parent task. For example, if sufficient damage has been taken, the* Hunt *sub-task will terminate and control returned to* Root *where the agent may decide to call the* Flee *sub-task. Sub-tasks may be parametrized as shown by the* Navigate *sub-task. Both instances of the* Navigate *sub-task will have to be learned.*

Dietterich (2000) further proposes that sub-goals in the form of local rewards be added by the domain expert to encourage sub-tasks to converge locally. The argument is that the sub-goal rewards will enable learnt sub-tasks to become more portable across different MDP problems for re-usability. However, this compromises optimality. In HRL, expert knowledge imposes constraints on the MDP hence learning may only converge to *hierarchical optimality* – optimality that is consistent with the constraints placed on the MDP. The sub-goal rewards for the sub-tasks in MAXQ compromise this further and MAXQ learning converges to the weaker form of *recursive optimality*.

3.1.3 Hierarchical Abstract Machines

Hierarchical abstract machines (HAMs) were first presented in Parr and Russell (1997) as a means to impose a hierarchy on the MDP using finite state machines augmented with certain special states. From a high level perspective, HAMs define a partially specified program due to certain *choice* states that are meant to be decided upon by the RL agent at runtime, thus completing the program. These choice states are similar to the high level actions that correspond to tasks. Tasks are represented as subroutines in the program that may contain choices for other subroutines or a limited set of primitive actions. The MDP is augmented with the hierarchy by adding the program call stack to the state and choice states to the actions. Hence the goal is to learn a program completion that consequently solves the original task.

Andre and Russell (2002) introduced a specialized language ALisp that is Lisp augmented with non-deterministic constructs for the purpose of specifying HAMs programmatically by the user. The paper further details a three part decomposition of the action value function Q as opposed to the two part decomposition in Dietterich (2000). Decomposing the Q function based on sub-routines serves the purpose of improving the learning rate of the sub-routine by propagating rewards further back in time. The benefit of the three-part decomposition is that sub-goals do not need to be specified for sub-routines. This is because the external component of the Q function decomposition models the reward that comes from after the return of a sub-routine (i.e. at the *stop* state). Hence an RL learning agent that uses HAMs will be able to converge on hierarchically optimal value functions that is better than recursive optimality in MAXQ.

However, the external component of the Q function decomposition depends on many states as it comes from the sequences of events after a sub-routine has returned to the parent routine. Thus, although no sub-goals are specified, RL with this system may lead to slow convergence to hierarchical optimality, i.e., a reduced learning rate. This concern was addressed to a certain extent in Marthi et al. (2006).

3.1.4 Discussion

Each of the three approaches to HRL is a form of temporal abstraction to the RL problem. Their aim is to solve the problem of slow learning due to the depth of exploration required. Of the HRL methods, both MAXQ and HAM specify criteria of how state space abstraction can be carried out for sub-tasks in Dietterich (2000) and Andre and Russell (2002) respectively. State space abstractions allow sub-tasks to generalize more over the state space, potentially speeding up the learning rate. Alternatively, they allow tasks to depend on a reduced number of states to mitigate the increase in space requirements of their tabular value functions due to the hierarchy.

Of the three HRL methods explored, MAXQ and HAMs have specific details on how the programmer may encode procedural knowledge for decision making. The three HRL methods started out as a way to represent and incorporate useful real-world knowl-

edge to improve RL. Subsequently, automated techniques of learning such knowledge have been studied for building options (Konidaris and Barto, 2007; Jong et al., 2008) and MAXQ task hierarchies (Mehta et al., 2008).

3.2 Coordination Graphs

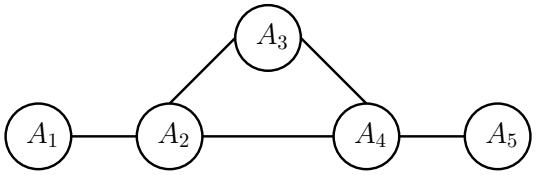


Figure 3.2: Example of a coordination graph

The concept of the *coordination graph* (CG) have often been used (Guestrin et al., 2001, 2002) to specify a structure among a group of agents. This structure can be interpreted in two ways. First, a CG specifies a coordination structure, i.e., it encodes if agents need to coordinate. Consider the CG given in Figure 3.2. There are five agents $A_1, ..., A_5$, each represented as a vertex. Between each pair of agents, an edge exists between them if they need to coordinate. CG as a coordination structure can be used to *additively decompose* value functions much like how Bayesian networks (Neapolitan, 2003) factor a joint probability distribution. For example, based on the Figure 3.2, we may specify Q as sub-components,

$$\begin{aligned} Q(s, \mathbf{a}) &= Q(s, a_1, ..., a_5) \\ &= q_a(s, a_1, a_2) + q_b(s, a_2, a_3, a_4) + q_c(s, a_4, a_5) \end{aligned}$$

Hence a tabular representation will have q_b as its largest component involving 3 agents that is usually smaller than a joint tabular representation of all 5 agents in Q.

The other interpretation of the CG is that it specifies a communication structure among agents. For example, in Figure 3.2, the agents may only communicate if an edge exists among them. This interpretation is useful in modelling the communication limitations among agents. Hence agents may only explicitly coordinate their joint actions if

they are able to communicate. When they are unable to communicate they choose their actions independently. It is possible for agents to still have implicit coordination if each agent models other agents internally. However, for simplicity, we will presume that communication limitations place a hard restriction on the coordination structure among agents, i.e., agents may only coordinate as far as communication allows.

In general, the structure of the CG may depend on the state. In this case, the dynamic CG changes accordingly to the current state value. Depending on interpretation, this can mean that agents coordination requirements are different in each state, or that communication links between agents are dynamic.

Finally, choosing a maximizing joint action, $\text{argmax}_{\mathbf{a}\in\mathcal{A}} Q(s, \mathbf{a})$, is essential for control and to learn an optimal policy for a given MDP. CGs provide a useful structure for maximization with the value function. The next two sections describe approaches for computing the maximizing joint action for centralized and distributed problems respectively.

3.2.1 Centralized Joint Action Selection

For centralized joint action selection using a CG, we presume that action value function can be *additively decomposed* into various components where each component depends on a subset number of agents. An edge exists between a pair of agents in the CG if there exists some component that depends on both of them. To illustrate, we assume Q may be additively decomposed into sub-components that depend on at most two agents out of the N agents in $\mathcal{A}$, i.e.,

$$Q(s, \mathbf{a}) = Q(s, a_1, ..., a_N) = \sum_{i=1}^{N} q_i(s, a_i) + \sum_{i,j\in[1,N] \mid i<j} q_{i,j}(s, a_i, a_j) \tag{3.1}$$

The components Q_i and $q_{i,j}$ may consist of weighted features, $w \cdot f$, used for linear function approximation (Section 2.4). For example, using linear function approximation in an RTS game, we may have some pairwise binary features such as, $collide_{1,2}$ and $attack_{1,2}$ that encode if marines 1 and 2 will collide and if marines 1 and 2 are attack-

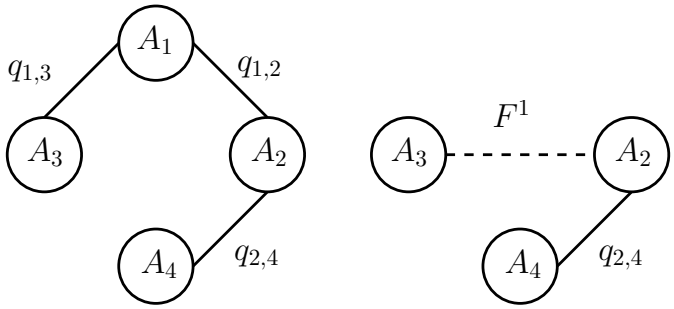
ing the same target respectively. Note that these features only depend on the state and action variables for marine 1 and 2. Assuming that there are no other pairwise features, then

$$q_{1,2}(s, a_1, a_2) = w_a \cdot collide_{1,2}(s, a_1, a_2) + w_b \cdot attack_{1,2}(s, a_1, a_2)$$

where w_a and w_b are the learned weights of their respective features. Note that in the case of the edges, if a particular $q_{i,j} = 0$ for the current state, the edge need not be represented in the CG. To find the joint action that maximizes Q, i.e.,

$$\pi(s) = \underset{\mathbf{a} \in \mathcal{A}}{\operatorname{argmax}}\, Q(s, \mathbf{a})$$

the *bucket elimination* (BE) algorithm (Dechter, 1996, 1999) can be applied using the CG. The BE algorithm is essentially the same as the variable elimination algorithm for inference in Bayesian Networks (Zhang and Poole, 1996). We describe how it is used via an example.



(a) Four agents with three components

(b) Elimination of a_1

Figure 3.3: Example of bucket elimination on a coordination graph

Example 3.3 (Bucket Elimination)**.** *Suppose we have four agents such that*

$$\mathbf{a} = \langle a_1, a_2, a_3, a_4 \rangle \in \mathcal{A} = A_1 \times A_2 \times A_3 \times A_4,$$

and a Q function that is a sum of three pairwise components with no single components as shown in Figure 3.3a. We wish to compute, by eliminating variables a_1, a_2, a_3, a_4 in

some order,

$$\max_{\mathbf{a}\in\mathcal{A}} Q(\mathbf{a}) = \max_{a_1,a_2,a_3,a_4} q_{1,2}(a_1,a_2) + q_{1,3}(a_1,a_3) + q_{2,4}(a_2,a_4) \tag{3.2}$$

$$= \max_{a_2,a_3,a_4} q_{2,4}(a_2,a_4) + \max_{a_1} q_{1,2}(a_1,a_2) + q_{1,3}(a_1,a_3)$$

$$= \max_{a_2,a_3,a_4} q_{2,4}(a_2,a_4) + F^1(a_2,a_3) \tag{3.3}$$

$$= \max_{a_3,a_4} F^2(a_3,a_4)$$

$$= \max_{a_4} F^3(a_4)$$

$$= F^4$$

where F^4 is a constant and functions,

$$F^1(a_2,a_3) = \max_{a_1} q_{1,2}(a_1,a_2) + q_{1,3}(a_1,a_3)$$

$$F^2(a_3,a_4) = \max_{a_2} q_{2,4}(a_2,a_4) + F^1(a_2,a_3)$$

$$F^3(a_4) = \max_{a_3} F^2(a_3,a_4)$$

Note that s has been left out for clarity. Equations 3.2 and 3.3 correspond to Figures 3.3a and 3.3b respectively. They show that action variable a_1 can be eliminated by computation of the function $F^1(a_2,a_3)$ for each possible pair (a_2,a_3). This computation is stored in the table, F^1, along with the maximal (argmax) value,

$$u^1(a_2,a_3) = \operatorname*{argmax}_{a_1} q_{1,2}(a_1,a_2) + q_{1,3}(a_1,a_3),$$

i.e., an entry in table F^1 contains $\langle a_2, a_3, F^1(a_2,a_3), u^1(a_2,a_3)\rangle$ such that (a_2,a_3) is an index. Repeating this process for the tables F^2, F^3, and F^4, we achieve a bottom up computation of $\max_{\mathbf{a}} Q(\mathbf{a})$. Then, we can set each a_i in $\mathbf{a}^* = \langle a_1^*, a_2^*, a_3^*, a_4^*\rangle$ by doing a reverse pass of the elimination order. For example, in the order,

$$a_4^* = \operatorname*{argmax}_{a_4\in A_4} F^3(a_4) = u^4$$

$$a_3^* = \underset{a_3 \in A_3}{\operatorname{argmax}} F^2(a_3, a_4^*) = u^3(a_4^*)$$

$$a_2^* = \underset{a_2 \in A_2}{\operatorname{argmax}} F^1(a_2, a_3^*) = u^2(a_3^*)$$

$$a_1^* = \underset{a_1 \in A_1}{\operatorname{argmax}} q_{1,2}(a_1, a_2) + q_{1,3}(a_1, a_3) = u^1(a_2^*, a_3^*)$$

where the $u^i(\cdot)$ values were previously computed and stored in the F^i tables.

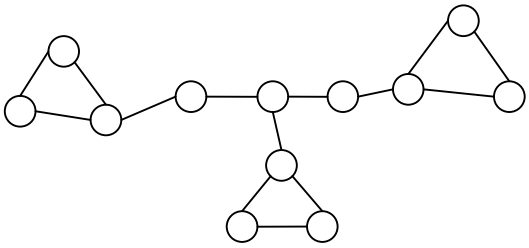


Figure 3.4: A CG with induced tree width of 3

Let w be the *induced tree width* of the CG, and n be the size of each action variable's domain. The time and space complexities of BE are exponential in w, i.e., $O(n^w)$ and $O(n^{w-1})$ respectively (Dechter, 1999). To obtain the induced tree width, we first triangulate the coordination graph by adding an edge between two nodes if they belong to a cycle of size greater than three. Then, the size of the largest clique in the graph is the induced tree width. Consequently, it is desirable to have coordination graphs with small induced tree widths. An example of a graph with induced tree width of 3 is shown in Figure 3.4, the maximum can be computed in $O(n^3)$ time and $O(n^{3-1})$ space.

There are a few approaches for further improving computational and space efficiency of BE. First, expert knowledge may be used to reduce the density of the CG. In Marthi et al. (2005) for RTS games, to ensure that the induced tree width remains small, pruning of CG edges was be carried out based on spatial knowledge. For example, a pair of agents, (i, j), that are spatially far apart will be considered as having zero value for their $q_{i,j}$ regardless of their actions taken. This effectively removes edges between the vertices in the CG.

Second, Kok and Vlassis (2004) presented the idea that the CG may be pruned using a predefined coordination structure in the form of value rules that define the CG based on the state. These value rules also make up a linear decomposition of the Q

function similar to linear function approximation. They demonstrated that their method of representing the joint state action space in a sparse manner using rules is effective for a predator-prey problem in which two predators need to coordinate to explicitly capture a prey.

Third, where good lower and upper bounds on the Q function is available, the space requirements of the maximized functions, F^i, can be further reduced in practice by combining BE with branch and bound techniques. This hybrid algorithm was presented in Larrosa and Dechter (2003).

Last, is to make use of approximate methods to estimate the maximizing value of Q based on the CG. These methods make use of message passing between agents and are described in the next section.

3.2.2 Distributed Joint Action Selection

In Kok and Vlassis (2006, Section 3.1), a message passing method called *max-plus* was introduced to approximate the maximum value and action of an action value function represented as a CG. As long as agents may communicate along the edges in the CG, max-plus is a distributed form of joint action selection. Furthermore, the passing of messages is iterative, hence action selection may be stopped at any iteration and the current results returned.

Max-plus requires the Q function to be decomposed into components involving up to at most a pair of agents as shown in Equation 3.1. Hence the CG directly corresponds to components of the function with q_i components at the vertices and $q_{i,j}$ components at the edges. From here on, we specify Q functions without the state s as it is fixed during action selection. Let $\Gamma(i)$ be the set of neighbours of i. At each iteration, each agent i sends a message, $\mu_{i,j}$, to neighbouring agents $j \in \Gamma(i)$ in the CG, defined as

$$\mu_{i,j}(a_j) = \max_{a_i \in A_i} \left[q_i(a_i) + q_{i,j}(a_i, a_j) + \sum_{k \in \Gamma(i) - \{j\}} \mu_{k,i}(a_i) \right] - \kappa_{i,j} \tag{3.4}$$

where the normalizing constant $\kappa_{i,j}$ is the average

$$\kappa_{i,j} = |A_j|^{-1} \cdot \sum_{a_j \in A_j} \mu_{i,j}(a_j) \tag{3.5}$$

that prevents the messages from growing too large in CGs with cycles.

After the desired number of iterations, each agent i computes its max-marginal Q function, $\hat{Q}_i$, using the single agent component q_i and its received messages $\mu_{j,i}$, where $j \in \Gamma(i)$,

$$\begin{aligned} \hat{Q}_i(a_i) &= q_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{j,i}(a_i) && (3.6) \\ &= \max_{\{\mathbf{a}' \in \mathcal{A} | a'_i = a_i\}} Q(\mathbf{a}). && (3.7) \end{aligned}$$

Equation 3.7 expresses that the max-marginal $\hat{Q}_i$ maximizes Q, for every value of $a_i \in A_i$, over the possible subset configurations of actions for the other agents. This corresponds to a bottom up computation of a spanning tree rooted at agent i. Hence for

$$\mathbf{a}^* = \langle a_i^*, ..., a_N^* \rangle = \operatorname*{argmax}_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{a}), \tag{3.8}$$

the maximal action a_i^* for each agent can be individually selected by

$$a_i^* = \operatorname*{argmax}_{a_i \in A_i} \hat{Q}_i(a_i). \tag{3.9}$$

Figure 3.5 illustrates the messages passed, as described above, for a similar CG to Figure 3.2 and the max-marginals computed at each agent. Note that Equation 3.9 only holds when the joint action $\mathbf{a}^*$ is unique. When $\mathbf{a}^*$ is not unique, which is common in many RL problems, computing an additional pairwise marginal $\hat{Q}_{i,j}$ is required. We will discuss this case further in Section 4.3.4 page 77.

The max-plus algorithm is analogous to the max-product algorithm used for maximum a posteriori (MAP) estimation in probabilistic models with the same pairwise component among variables. It is only exact and guaranteed to converge when the CG

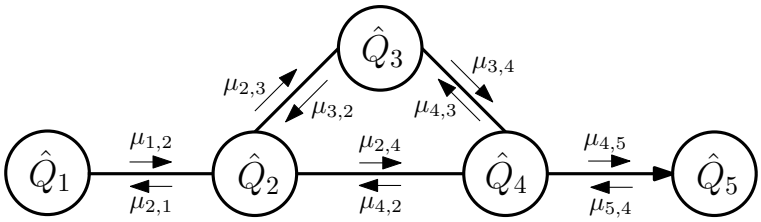


Figure 3.5: Passing $\mu_{i,j}$ messages between neighbours in max-plus and the max-marginals computed at each agent (vertex).

is a tree or, a graph with at most one cycle and one unique solution (Wainwright et al., 2004). Therefore, in the general case of CG with many cycles, it is an approximation and may diverge. To mitigate this, actions may be selected after every iteration and the best action stored. An exact message passing method for any CG with cycles is given in Petcu and Faltings (2005). However, like bucket elimination, it requires passing messages of size exponential in induced tree width to be exact.

While only allowing Q to consist of up to at most pairwise function components may be restrictive, it has the advantage of simplicity. The message passing scheme directly corresponds to the CG in terms of communication requirements (edges) and agent boundaries (vertices). If the function components in Q involve more than two agents, distributed action selection can be computed using factor graphs (Kschischang et al., 2001). However, unlike CGs, vertices and edges in factor graphs do not directly correspond to agents and their communication links. Hence more work is required to adapt factor graphs for situations where agents have dynamic communication.

3.3 Flat Coordinated Reinforcement Learning

In this section, we present methods for multi-agent reinforcement learning with the action value function. In the simplest centralized case, updating the Q function with the joint state and action spaces is the same as the update equations described in Chapter 2 while joint action selection is carried out using the methods for coordination graphs as described in the previous section. However, the Q function and MDP's reward signal can be further decomposed among agents so that various updates can be carried out locally in parallel. This aids distribution and is described next.

3.3.1 Agent Decomposition

One natural way of decomposing the RL problem among agents is to represent the Q function as a sum of individual agent functions. In addition, the rewards may also be decomposed into a sum of individual agent rewards, one for each agent. Suppose a factored MDP for N agents has a joint action space, $\mathcal{A} = A_1 \times A_2 \times ... \times A_N$ and a reward model that is additively decomposed for each agent, i.e.,

$$\mathcal{R}(s, a, s') = R_1(s, a, s') + ... + R_N(s, a, s') = \sum_{i=1}^{N} R_i(s, a, s') \tag{3.10}$$

With the decomposed reward function R_i, the reward signal from the environment is similarly decomposed as $r = \sum_{i=1}^{N} r_i$, where each agent i receives r_i at each time step. Then the Q function may also be additively decomposed to give,

$$Q(s, a) = Q_1(s, a) + ... + Q_N(s, a) = \sum_{i=1}^{N} Q_i(s, a) \tag{3.11}$$

where in general, the state and action available to each agent i may be further restricted. For example, we may define $Q_i(\mathbf{s}_i, \mathbf{a}_i)$ for agent i as only being able to access state and action variables of itself and its neighbouring agents instead of the full joint state and action space.

3.3.2 Independent Updates

The simplest form of updates is that of no coordination. This is referred to as the independent learners approach (Claus and Boutilier, 1998). Each agent i selects their action, a_i independently and performs TD updates based on,

$$Q_i(\mathbf{s}_i, a_i) \leftarrow Q_i(\mathbf{s}_i, a_i) + \alpha[r_i + \gamma \max_{a_i^* \in A_i} Q_i(\mathbf{s}_i', a_i^*) - Q_i(\mathbf{s}_i, a_i)] \tag{3.12}$$

where agents may be restricted to only certain state variables and we have used the greedy policy. An immediate benefit is that action selection is highly efficient as agents do not coordinate their actions. However, this form of local updating is sub-optimal

for domains where agents need to coordinate their actions to achieve a common goal (Russell and Zimdars, 2003), i.e., the sum of Q_i functions, as in Equation 3.11, is not a good estimate of Q.

3.3.3 Global Updates

The next updating method performs global TD updates and is referred to as coordinated RL in Guestrin et al. (2002). Assuming that each agent i may only access a subset of the state $\mathbf{s}_i$ and action $\mathbf{a}_i$ variables, the updates are,

$$Q_i(\mathbf{s}_i, \mathbf{a}_i) \leftarrow Q_i(\mathbf{s}_i, \mathbf{a}_i) + \alpha[r + \gamma \max_{\mathbf{a}^* \in \mathcal{A}} Q(\mathbf{s}', \mathbf{a}^*) - Q(\mathbf{s}, \mathbf{a})] \tag{3.13}$$

for the greedy policy. Note that the reward signal used is the global reward r and the TD update is based on the global Q. Action selection can be performed using BE or max-plus. The benefit of this method is that it does not require the reward signal to be decomposed. However, it requires the global values of $Q(\mathbf{s}, \mathbf{a})$ to be available. Although computing the global Q values may be achieved by message passing over a fixed network, they will not be always available in the case where the CG consists of disjoint sub-graphs due to dynamic communication.

3.3.4 Local Updates

In problems where the user may provide a precise agent decomposition of the reward signal, learning can be improved as each agent will receive a more accurate share of rewards. Local TD updating is an intermediary between independent and global updates. One local update equation for agent based decomposition (Kok and Vlassis, 2006, Section 5.1) is,

$$Q_i(\mathbf{s}_i, \mathbf{a}_i) \leftarrow Q_i(\mathbf{s}_i, \mathbf{a}_i) + \alpha[r_i + \gamma Q_i(\mathbf{s}'_i, \pi(\mathbf{s}'_i)) - Q_i(\mathbf{s}_i, \mathbf{a}_i)], \tag{3.14}$$

$$\text{where} \qquad \pi(\mathbf{s}) = \operatorname*{argmax}_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}, \mathbf{a}) \tag{3.15}$$

is a joint greedy policy, and we assume that agents may access subset state $\mathbf{s}_i$, and action $\mathbf{a}_i$ variables. Typically these are from the neighbouring agents in the CG. Although the updates are local, the joint action is selected globally using π via algorithms such as BE or max-plus. For tabular functions, this form of updating can converge to an optimal solution (Russell and Zimdars, 2003) even when each Q_i function is updated individually by agent i. This result is useful for the distributed setting where the global Q function is the sum of local Q_i functions, distributed among agents.

Kok and Vlassis (2006, Section 5.2) further details an *edge based* decomposition along the edges of the CG as compared to the agent (vertex) based decomposition shown in Equation 3.11, i.e., let E be the set of edges (i, j) in the CG,

$$Q(\mathbf{s}, \mathbf{a}) = \sum_{(i,j)\in E} Q_{i,j}(\mathbf{s}_i, \mathbf{s}_j, a_i, a_j) \tag{3.16}$$

where $\mathbf{s}_i, \mathbf{s}_j$ are the state values accessible by agent i and j respectively. They then propose two updating schemes for this edge decomposition. Edge decomposition is useful for tabular representation of Q functions as space requirements scale linearly in the number of neighbours instead of exponentially for Equation 3.11. However, it is unclear how the pairwise $Q_{i,j}$ functions are distributed among agents when the CG is not fixed due to dynamic communication. Furthermore, it is possible to mitigate space requirements of Equation 3.11 using function approximation.

Yet another form of local updating with communicated values from neighbouring agents is that of distributed value functions (Schneider et al., 1999; Ferreira and Khosla, 2000). The global Q function is a sum of N local functions update with,

$$Q_i(\mathbf{s}_i, a_i) \leftarrow Q_i(\mathbf{s}_i, a_i) + \alpha[r_i + \gamma \sum_{j=1}^{N} \beta(i, j) \max_{a_j^* \in A_j} Q_j(\mathbf{s}'_j, a_j^*) - Q_i(\mathbf{s}_i, a_i)] \tag{3.17}$$

where $\beta(i, j)$ is some weighting scheme that is zero if (i, j) does not exists in the current CG and non-zero for (i, i). Note that unlike Equation 3.14, each agent's actions, a_j, are selected independently instead of jointly, but updates involve Q_j values from other agents. Hence this form of learning tends to be less coordinated than RL with Equation

3.14.

3.3.5 Others

Apart from the learning update methods presented above. There are other forms of coordinated RL. Kok et al. (2005) focused on learning the structure (topology) of the coordination graph itself. This corresponds to learning a factoring of the Q function by analysing which agents (vertices) should have edges between them, indicating that they require coordination in some particular state. To do this, they store statistics of the MDP trajectory and perform a statistical test (t-test) to compare if there is a significant change in the Q function value when joint actions between two agents are allowed in that state. If so, a function component involving the two agents in that state replaces the individual function components involving each of the agents.

While many methods use CGs to represent additively decomposed value functions, other representations have also been explored. Sallans and Hinton (2004) proposed a centralized alternative to CGs by using a different model and a sampling method to compute the optimal action. Their method uses a model called the *product of experts* as a function approximator to Q. This model contains probabilistic semantics albeit without encoding causal links (conditional probabilities) like a Bayesian network. The variables of this model consist of the state space, action space and hidden variables. Inference is approximated through the use of Markov chain Monte Carlo (MCMC) sampling methods like Gibbs sampling. For learning they use a SARSA method with a Boltzmann exploration policy. To select a maximal action, any form of MCMC sampling may be used and Boltzmann exploration ensures that the optimal action will be selected with high probability over time. Their experiments show that RL is tractable for joint action spaces of size 2^{40}. However there is no measure of how much additional sampling is required if more agents are added to the problem.

The term 'coordinated reinforcement learning' has also been used to describe RL for solving other types of problem formulations. One such formulation is that of a *Markov game* whereby each agent explicitly models other agents. This is necessary because

in the absence of communication, independent learning agents may not converge to global optimality due to the non-stationary environment caused by other learning agents (Panait and Luke, 2005, Section 3.2). A recent example of learning in Markov games by predicting other agents' policies is given in Zhang and Lesser (2010). Markov games are usually adversarial. Other problem formulations include that of decision-theoretic planning where each agent take actions that span multiple time steps to solve multiple tasks (goals). An agent's actions may have dependencies on another agent's action in the past or future. Chen et al. (2005) presented an RL approach to solving such problems where action dependencies over time are provided as constraints. Their solution requires an offline phase to process these constraints.

3.4 Hierarchical Multi-Agent Learning

A variety of hierarchical approaches have been developed for multi-agent RL. Unlike in single agent HRL (see Section 3.1), here, a hierarchy refers to the learning architecture, where additional decisions have to be made that affect the original decisions (actions) made by the RL system. These additional, or higher level, decisions may have their semantics based on different concepts. Below, we explore the methods based on task based RL and agent organization. They are various methods that encode expert knowledge to improve learning in problems with large joint action spaces.

3.4.1 Task Based Approach

A number of works build on the procedural task encodings from single agent HRL by specifying a task hierarchy per agent. First, Marthi et al. (2005) presented a centralized method for multi-agent RL by extending the ALisp language (see Section 3.1.3) with multiple threads, one for each agent. At each time step, each agent may be executing some higher level task action or a low level primitive action. These are selected jointly using coordination graphs. The action value function is represented using linear function approximation. Experiments were carried out on RTS games. Although based on

HRL, the action value function is not decomposed over tasks hence this method does not make use of temporal abstraction. However, once tasks have been selected, agents may be individually restricted to a subset of actions. Hence exploration may be directed based on individual agents.

Second, Ghavamzadeh et al. (2006) presented another task based method called Cooperative HRL that is based on the MAXQ task decomposition (see Section 3.1.2). Similar to Marthi et al. (2005), each agent has its own task hierarchy. However, when choosing actions and updating value functions, agents only take into account other agents' actions at special *cooperative* sub-tasks that are defined by the user. This approach assumes all state and action values are observable to every agent, i.e., it is a centralized approach. To decentralize, additional *communicate* and *not-communicate* sub-tasks are added to each agent's MAXQ hierarchy. If an agent chooses to communicate, it may take additional actions to send and receive information from other agents, otherwise control and learning is individual.

Third, Proper and Tadepalli (2009) described a centralized two level system whereby the upper level assigns tasks to agents and the bottom level carries out the learning of the actual primitive action in the MDP. The MDP has additional states to indicate agents are assigned a task. However, unlike the previous two methods, the tasks are not higher level decisions learned by the system. Rather, agents are assigned tasks by finding the maximal task assignment that maximizes the current Q function. Exhaustive search of this nested maximization over tasks and primitive actions is prohibitive for all but the simplest problems. Hence stochastic local search techniques are employed. Primitive joint action selection uses max-plus while updating and value function representation is carried out using edge based decomposition (see Section 3.3.4) in Kok and Vlassis (2006, Section 5.2).

In the three described extensions of single agent task based learning to the multi-agent setting, only Ghavamzadeh et al. (2006) makes use of temporal abstractions in learning its value functions and is distributed. Both works of Marthi et al. (2005) and Ghavamzadeh et al. (2006) may constrain individual agent's primitive actions once tasks

are selected. In Proper and Tadepalli (2009), the main benefit from tasks is to reduce the number of state variables considered to only those relevant for the task each agent is assigned to, thus the space requirements of the tabular edge decomposed $Q_{i,j}$ functions are reduced.

3.4.2 Organization Based Approach

A different hierarchical concept was proposed in Zhang et al. (2009) that is inspired by real world human organizational structure. The original agents in the multi-agent problem are *worker* agents that learn their policies based on interacting with neighbouring agents. Then, additional *supervisor* agents are defined at a hierarchy above the worker agents to manage them. These supervisor agents may perform high level actions that sends suggestions to *adapt* (bias) the policies of a subset of worker agents and may in turn have higher level supervisors. A communication protocol between worker and supervisor agents was specified for a network with fixed structure. Supervisor agents generally observe state summaries over time to make their decisions. Although in theory supervisor agents may learn, hard-coded supervisor agents were used for evaluating the approach. Further, to determine the subset of worker agents that supervisors should be created to manage, Zhang et al. (2010) subsequently proposed a method for their organization based approach to self-organize. This was also based on a fixed network structure for agent communication.

3.5 Rewards for Learning

Another direction for improving the learning rate of RL is to modify the reward signal received from the environment. Ng et al. (1999) formalized how a shaping function may be used to additively modify the reward function of an MDP, $\mathcal{R}$, without compromising on optimality. Essentially the shaping function must be one that will give the same value to the same states. Hence they choose to use a potential difference function for shaping. This ensures that the net effect of entering a cycling of states from the shaping

function is always zero. The new reward function $\mathcal{R}'$ with shaping applied is given as,

$$\mathcal{R}'(s,a,s') = \mathcal{R}(s,a,s') + F(s,a,s') \tag{3.18}$$

$$= \mathcal{R}(s,a,s') + \gamma\Phi(s') - \Phi(s) \tag{3.19}$$

The new reward function $\mathcal{R}'$ is then used for any RL algorithm, by changing the reward signal to $r' = r + \gamma\Phi(s') - \Phi(s)$. Experiments show that shaping can lead to an increase in the learning rate albeit requiring intricate expert knowledge of the problem to craft the shaping function F.

In some situations it may be difficult for the user of RL to define good rewards. Where a good policy or sample trajectories is known, inverse reinforcement learning may uncover or be used to fine-tune the reward for small MDP problems (Ng and Russell, 2000). For multi-agent problems the more immediate issue is that of credit assignment – decomposing rewards among agents. This may be necessary for the distributed case or, desirable for more fine-grained updating of value functions where each agent updates based on rewards relevant to it. Marthi (2007) presented methods to automated agent decomposition of rewards and further learn shaping functions for them. These methods can reduce expert knowledge required by the user with regards to the reward signal.

3.6 Relational Reinforcement Learning

Multi-agent RL usually contains many variables in its MDP specification. For example, the state and action spaces are usually factored into variables pertaining to each agent. A relevant line of works that deals with representational and generalization issues in such problems is relational RL (RRL). An overview of RRL was given by Tadepalli et al. (2004).

In general, RRL brings about two key benefits. The first is that of representational simplicity. This is usually desirable in problems whereby specifying the entire joint state and action space is difficult due to their combinatorial nature. A common example

used to illustrate this is the blocks-world domain. In this domain, there are N labelled blocks and the actions are to stack or unstack blocks. The goal is to stack blocks in some order. Due to the possible permutations of stacked blocks, it is infeasible to apply tabular learning as both the states and actions grow with the permutations. Predicates, that are statements in first order logic (Russell et al., 1996, chap. 8), allows compact representation of the state and action spaces over block objects. Džeroski et al. (2001) introduced decision trees for regression using such predicates to represent the action value functions for RRL while Kersting and De Raedt (2004) presented convergence results for RL on a problem formulation that involves integrating MDP with logic programs. Croonenborghs et al. (2005) did a preliminary study on multiple agents in the blocks world. Ponsen et al. (2010) further described how relational learning may be used to learn a communication policy among multiple agents.

The second benefit is that of generalization along relational semantics. Instead of having one weight to be learned for features based on propositional statements, the features can be grouped together using a relational predicate and one weight learned for this new feature. Such representations of features for solving MDPs have been explored in Guestrin et al. (2003); Strens (2004); Asgharbeygi et al. (2006).

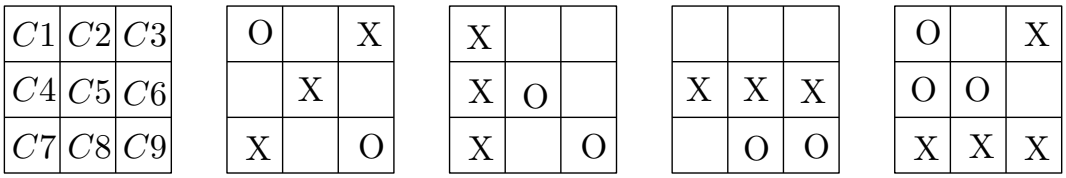


Figure 3.6: Relational generalization for tic-tac-toe game. Left-most shows the state variables. Tic-tac-toe states that follow in sequence satisfy: $lineX(C7, C5, C3)$, $lineX(C1, C4, C7)$, $lineX(C4, C5, C6)$, and $lineX(C7, C8, C9)$.

For example, in the tic-tac-toe game described in Asgharbeygi et al. (2006), suppose the state has 9 cells (relational objects), i.e., $C1, ...C9$ (see Figure 3.6). The predicate for a line of crosses may be specified as,

$$\begin{aligned} lineX(c_i, c_j, c_k) \quad := \quad & X = symbol(c_i) = symbol(c_j) = symbol(c_k) \\ & \wedge [Column(c_i, c_j, c_k) \vee Row(c_i, c_j, c_k) \\ & \vee Diagonal(c_i, c_j, c_k)] \end{aligned}$$

where c_i, c_j, c_k are variables of any 3 of the 9 cell objects, $symbol$ is a function that returns either 'X' or 'O' for a cell, and predicates $Column, Row, Diagonal$ indicate that the 3 cells form a valid line. A binding of objects to the variables in $lineX$ is called a grounding, that can be either true or false, e.g., $lineX(C1, C2, C3)$ will be true if $C1, C2, C3$ are marked with 'X' and $Row(C1, C2, C3)$ is true. In a propositional feature representation, each possible binding is a binary basis function with a corresponding weight to be learned. However, in relational learning, these weights can be combined to give the state value function,

$$V(s) = \sum_P w_P \cdot |I_P(s)| \tag{3.20}$$

where P is a predicate, e.g. $lineX$, $I_P(s)$ is the set of bindings that give true groundings for P, and w_P is the weight of relational feature $|I_P(s)|$, i.e., the number of true groundings.

In other words, each relational feature is the count of true groundings of predicate P in state s. For $lineX$, this will mean the same weight w_{lineX} will be updated regardless of the line being a column, row or diagonal. This is illustrated in Figure 3.6 where each of the example states shown will result in an update[1] for w_{lineX} because each state provides one grounding for $lineX$.

These ideas are closely related to the idea of sharing weights of feature templates. For example, Silver et al. (2007) illustrated in the game of Go, shape features (i.e. patterns of white and black stones) where the weights are generalized across stone colours and translations of these shapes around the game board. We intend to use relational features to further generalize over agents and their actions in multi-agent problems.

[1] We assume there is a dummy terminal state after these states.

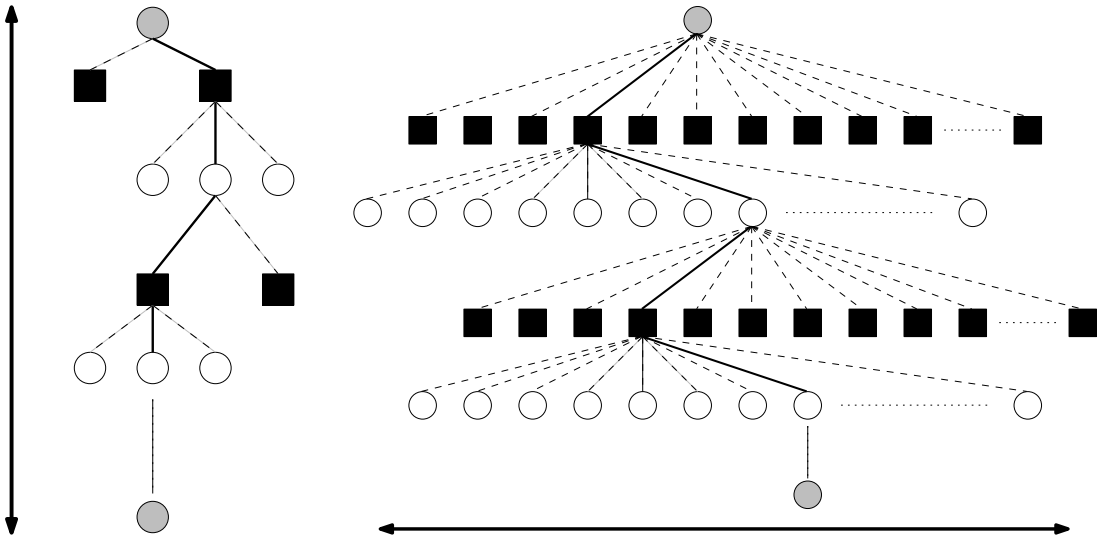
Chapter 4

Coordination Guided Reinforcement Learning

This chapter proposes to guide reinforcement learning (RL) with expert coordination knowledge for multi-agent problems managed by a central controller. The aim is to use expert coordination knowledge to restrict the joint action space and to direct exploration towards more promising states, thereby improving the learning rate. Such coordination knowledge is modelled as a set of constraints. We present a two level RL system that utilizes these constraints for online applications. This declarative approach towards specifying coordination in multi-agent learning allows knowledge sharing between constraints and features for function approximation. Results on different configurations of a soccer game and a tactical real-time strategy (RTS) game domains show that coordination constraints improves the online learning rate compared to using only unary constraints. The two level RL system also outperforms existing single-level approach that utilizes joint action selection via coordination graphs. Part of this work has been published in Lau et al. (2012).

4.1 Motivation

Learning to make sequential decisions for multiple collaborating agents is a difficult problem in general. This is especially so when agents require coordination to achieve



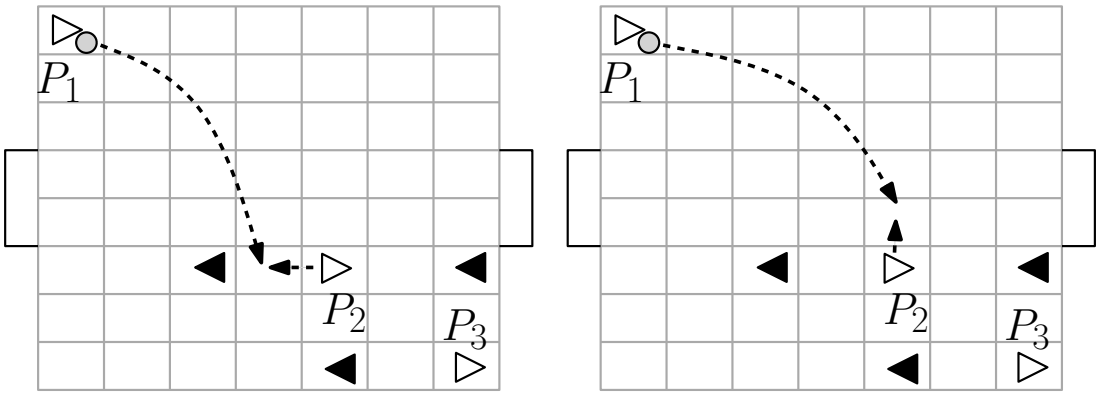
(a) Single agent **(b)** Five agents

Figure 4.1: The depth and width problems illustrated by single agent and multi-agent trajectories in the MDP respectively. Circles are states, grey circles are initial and terminal states. Solid squares are (joint) actions. Solid lines indicate the trajectory from top to bottom, dashed lines indicate possible actions or state transitions.

a common goal. The space of their action combinations is exponential in the number of agents, quickly rendering RL with naive exploration impractical. For online RL to be viable, we need to optimize the number of interactions with the environment. This motivates us to develop a solution to improve the learning rate of good policies in multi-agent RL (MARL).

Existing works in single agent task based RL makes use of the human concept of tasks as an abstraction to decompose complex problems into simpler ones to give a task hierarchy (Sutton et al., 1999; Dietterich, 2000; Andre and Russell, 2002) as described in Section 3.1 page 25. The learning systems learns to choose between tasks. When tasks are selected, the actions of an agent may be constrained to those relevant for the task. In addition, temporal abstraction allows rewards to be propagated further back in time, improving learning in problems where the depth of exploration required is lengthy.

For multi-agent problems, each agent has a set of actions whose Cartesian product forms the joint action space. This space is exponential in the number of agents and therefore, RL with naive exploration results in slow learning. This is because the large space results in a huge branching factor in the tree of transitions experienced by the



(a) Bad Pass **(b)** Good Pass

Figure 4.2: Example of coordination-based knowledge in simplified soccer. Figures show possible actions in a grid world state with white and black teams. Triangles are players that score in the direction they point to. The ball is the grey circle and dashed lines represent possible actions. (a) is a bad pass example while (b) is a good pass.

agents during exploration. Hence the more immediate problem is one of width instead of depth.

To illustrate, consider Figure 4.1 that shows conceptually the depth and width issues. In the single agent MDP shown in Figure 4.1a, the agent has two possible actions to choose from in every state and receives a reward in the terminal state at the bottom. Task based RL may propagate the rewards to earlier states faster by a decomposition of tasks into subtasks and updating the value function back when a (sub)task is initiated. On the other hand, if there are five of these agents as shown in Figure 4.1b, there are 2^5 joint actions to choose from, resulting in a large branching factor. To address this issue of width, multi-agent extensions to task based methods may use tasks to restrict each individual agent's available actions (Marthi et al., 2005; Ghavamzadeh et al., 2006; Proper and Tadepalli, 2009). However, as we noted earlier in Example 1.1 page 6, there exists coordination knowledge that can further constrain joint actions of agents but are difficult to incorporate in task based methods. We describe another example from the soccer game domain below.

Example 4.1 (Soccer coordination knowledge)**.** *Consider Figure 4.2 which depicts a state in a simplified soccer game and player P_1 has the ball. Let N, S, E, W, be the four compass directions. P_1's action set is $A_1 = \{S, E, pass2, pass3, shoot\}$ where*

$pass2$ and $pass3$ denote passing the ball to players P_2 and P_3 respectively, and $shoot$ denotes the action to kick the ball into the goal. Players P_2 and P_3 have the action set $A_2 = \{N, S, E, W\}$ and $A_3 = \{N, W\}$ respectively. We denote a joint action as $\langle a_1, a_2, a_3 \rangle \in A_1 \times A_2 \times A_3$. The size of this joint action space is $5 \times 4 \times 2 = 40$. Notice that many actions from this space are unlikely to lead to a winning state. For example in Figure 4.2a, P_1 certainly should not pass the ball to P_2 if P_2 is moving adjacent to an opponent as the ball can easily be intercepted. With this simple coordination strategy, the set of disallowed joint actions is $\{pass2\} \times \{S, E, W\} \times A_3$. Similarly, P_1 should not pass the ball to P_3 and the set of disallowed joint actions is $\{pass3\} \times A_2 \times A_3$. Immediately, the size of the joint action space is reduced by 35%.

The example of a bad pass in soccer described above does not fit well within individual agent task definitions. In the multi-agent extensions to task based RL, using this coordination knowledge to reduce agents' actions will not be possible within a single agent task. Alternatively, we might define joint-tasks on agents, i.e., instead of a task hierarchy for each agent we may have a task hierarchy defined on a pair of agents. For example, in the case of soccer, we may have a task involving players P_1 and P_2 such that bad pass in an offensive sub-task that can constrain the joint action space, $A_1 \times A_2$. However, we will not be able to do the same for pairs of players (P_1, P_3) or (P_2, P_3) as the task hierarchies will overlap. Furthermore, defining a joint-task hierarchy for all three players requires the user to encode procedural knowledge for all three. As the joint actions to consider increase exponentially with the number of agents such knowledge may become infeasible for the user to specify in detail. Hence learning may become limited to coarse-grained high level strategic decisions for the entire team of agents instead of fine-grained low level actions.

These considerations motivate us to develop learning method that allows fine-grained control of agents while incorporating coordination knowledge such as bad pass into the learning process to better direct exploration.

4.2 Aims & Approach

The expert coordination knowledge described in Example 4.1 and Example 1.1 page 6 are referred to as *coordination constraints* (CCs). CCs are hard constraints involving the state and action variables of multiple agents. We consider unary constraints like those used in task based methods on individual agents as special cases of CCs.

The main aim of this chapter is to handle the research challenges of managing exploration for multi-agent problems while being mindful of model complexity and knowledge representation (see Section 1.2 page 4). For a start, we consider a centralized approach to the problem and ignore the challenge of distribution.

We propose a two level RL system where the top level learns to select the appropriate CCs to constrain the bottom level's learning of joint actions. This is because not all CCs are useful in every state. Deciding which CCs to employ in different states is part of the learning process, enabling the RL system to effectively guide itself. An example, of the benefits of dynamically employing CCs for the soccer game follows.

Example 4.2 (Soccer dynamic coordination)**.** *In Example 4.1, the CC that P_1 should not pass the ball to P_2 is appropriate since there is an opponent next to P_2. However, this CC may not be suitable and should not selected if P_2 is standing directly in front of the goal as it may be better to pass to P_2 so that P_2 can try to score as illustrated in Figure 4.3.*

The proposed CCs are useful in addition to existing methods of modelling coordina-

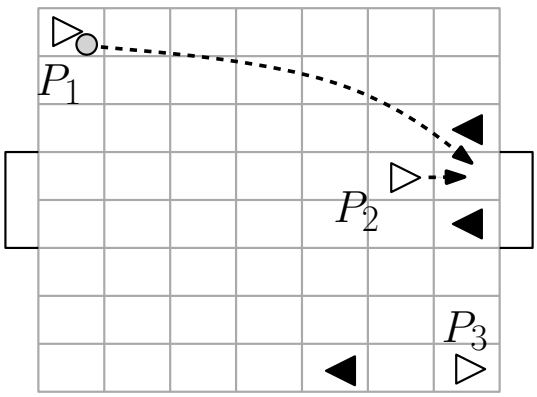


Figure 4.3: State in soccer where a "bad pass" should be allowed.

tion with a communication structure such as a coordination graph (CG) for joint action selection (Section 3.2 page 31). Furthermore, such a two level RL system is different from RL for constrained MDPs (Geibel and Wysotzki, 2005; Wu and Durfee, 2005; Geibel, 2006) as the two level system dynamically learns to use different constraints to improve learning, instead of using static constraints to prevent failure or entering of risky error states. Moreover, there is no additional cost signal to be minimized.

Using CCs for online RL has two challenges. First, the system must be able to efficiently learn to activate the various CCs from its interaction with the environment. However, different combinations of activated CCs lead to large number of bottom level value functions to be learned. We address this by formulating learning equations that exploit similarities among the bottom level components of our system. Second, choosing CCs at the top level introduces an exponential top level joint action space to explore. We overcome this by identifying those CCs which can never be violated in a given state. Once identified, these CCs are deactivated, reducing the top level action space for exploration.

Our model-free approach frees the user from designing models for large MDPs with many variables. A major benefit of this system is that existing predicate definitions of features for function approximation can be reused to specify CCs. This sharing of predicate components between CCs and features aids the user in encoding knowledge for the top level of the system.

Similar to the origins of task based RL discussed in Section 3.1.4 page 30, this chapter proposes and verifies a new way to incorporate expert coordination knowledge to improve multi-agent RL. Subsequent future work may use it as a basis to further reduce reliance on the expert. We describe the two level learning system next.

4.3 Two Level Learning System

The problem formulation we use is that of a factored MDP. This is essentially the same as the basic MDP formulation described in Section 2.1 page 13. Except that with N

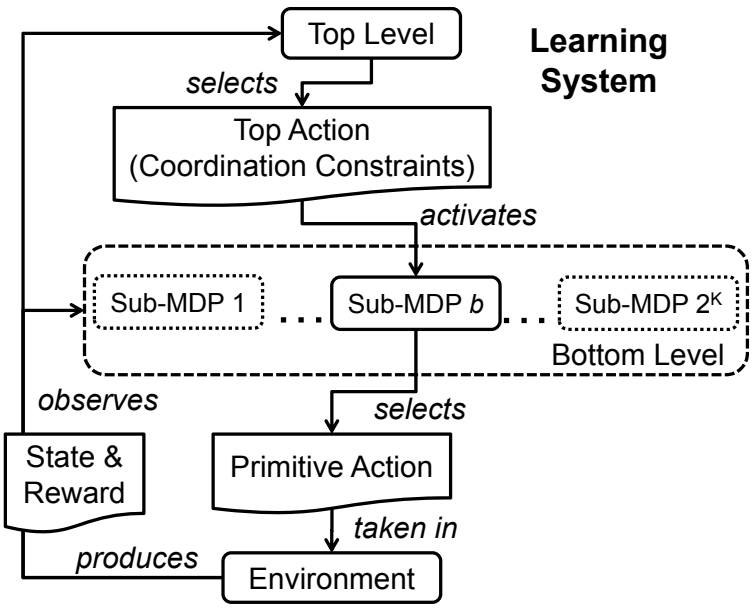


Figure 4.4: Centralized two level learning system

agents, the joint action space is factored as $\mathcal{A} = A_1 \times ... \times A_N$, where A_i is the action variable that corresponds to the i-th agent. $\mathcal{S}$ may also be factored into multiple variables in a similar way. The set of available actions at state s is written as $\mathcal{A}(s)$. We will make use of the original MDP's reward signal. Hence the user of the system does not need to decompose the reward function among agents.

Now, we present the system from a centralized viewpoint. Figure 4.4 depicts the proposed two level learning system. The top level determines which CCs are relevant to the current state. Its action space $\mathcal{A}_0$ consists of the *activation* or *deactivation* of each CC. With K constraints, the size of $\mathcal{A}_0$ is 2^K. However, in practice, the number of activated CCs is typically small. The activated CC restricts the bottom level to a sub-MDP, $b \in [1, 2^K]$, as shown in Figure 4.4. The joint action space of sub-MDP b, denoted by $\mathcal{A}_b$, is a subset of the original joint action space $\mathcal{A}$. This allows the bottom level to learn the primitive joint actions quickly. After the bottom level has taken an action in the environment, execution returns to the top level. Hence the system constantly oscillates between the top and bottom levels while operating in the environment.

The learning system's interaction with the environment is summarized in Algorithm 4.1. The system interacts externally with the environment through Line 6. The selection of top level action and the activations of the sub-MDP through CCs are performed

Algorithm 4.1 Centralized two level learning overview

```
s ← s_0                                              # observe initial state s_0
while ¬terminal(s) do
    Select top level action b ∈ A_0(s)               # at top level
    Activate CCs in b                                # activate sub-MDP b
    Select primitive action a within A_b(s)          # at bottom level
    Take action a, observe reward r and next state s'
    Perform learning updates using r, s'
    s ← s'
end while
```

internally by the RL system.

4.3.1 Augmented Markov Decision Process

To model the system's two level learning, we augment the original factored MDP's state space with an index $b \in [0, 2^K]$ that keeps track of the position within the hierarchy. The top level corresponds to $b = 0$, and $b \in [1, 2^K]$ refers to one of the sub-MDPs. Then, given a factored MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$, the *augmented* MDP with the two level hierarchy is $\langle \mathcal{S}', \mathcal{A}', \mathcal{R}', \mathcal{P}' \rangle$ where,

- $\mathcal{S}' = [0, 2^K] \times \mathcal{S}$, the state includes the index to the top level or some bottom level sub-MDP whose primitive actions in s are constrained. $\langle b, s \rangle \in \mathcal{S}'$ is an augmented state.
- $\mathcal{A}' = \mathcal{A}_0 \cup \mathcal{A}$, the joint actions space now also includes top level (de)activation actions. Let an action in $\mathcal{A}'$ be $\tilde{a}$.
- $\mathcal{R}'$, the reward model is derived from $\mathcal{R}$ such that the reward is zero for steps that transit from top level to bottom level i.e.,

$$\mathcal{R}'(\langle b, s \rangle, \tilde{a}, \langle b', s' \rangle) = \begin{cases} \mathcal{R}(s, \tilde{a}, s') & \text{if } b \in [1, 2^K] \wedge b' = 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

  Note that $\tilde{a} \in \mathcal{A}$ is a primitive action when $b \in [1, 2^K]$.
- $\mathcal{P}'$ is the transition probability model such that the top to bottom level transitions are deterministic, otherwise transitions in the original state follow that of

the original MDP's $\mathcal{P}$ i.e.,

$$\mathcal{P}'(\langle b', s' \rangle | \langle b, s \rangle, \tilde{a}) = \begin{cases} \mathcal{P}(s'|s, \tilde{a}) & \text{if } b \in [1, 2^K] \wedge b' = 0 \text{ where } \tilde{a} \in \mathcal{A} \\ 1 & \text{if } b = 0 \wedge b' = \tilde{a} \text{ where } \tilde{a} \in \mathcal{A}_0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Given the above definition, the sequence of transitions in the augmented MDP with respect to the original is,

$$\underbrace{\langle 0, s_0 \rangle, b_0, r', \langle b_0, s_0 \rangle}_{s_0}, a_0, r_0, \underbrace{\langle 0, s_1 \rangle, ...,}_{s_1,...,} \underbrace{\langle 0, s_t \rangle, b_t, r', \langle b_t, s_t \rangle}_{s_t}, a_t, r_t, ..., \underbrace{\langle 0, s_\emptyset \rangle}_{s_\emptyset} \tag{4.3}$$

where $r' = 0$ for all transitions, t reflects the time steps in the original MDP, and r_t is the original MDP's reward signal and $\langle 0, s_\emptyset \rangle$ is the terminal state. The under braces show the steps in the augmented MDP from the top level to the bottom that take place within the same state as the original MDP. Figure 4.5 depicts the two level learning system's interactions with the environment from two perspectives: (a) the augmented MDP as a new environment, and (b) the original environment. In the following section we define the policies and value functions of the augmented MDP for learning.

4.3.2 Policies & Value Functions

Here, we define policies and value functions for the augmented MDP and simplify them for the learning equations in the following section. We also show that learning in the augmented MDP solves the original MDP problem.

Quality of Policies

The solution to the augmented MDP is the two level policy $\psi' : \mathcal{S}' \mapsto \mathcal{A}'$. This can be expressed as a set of sub-policies, $\{\psi(0, s), \psi(1, s), ...\psi(2^K, s)\}$, written as two main

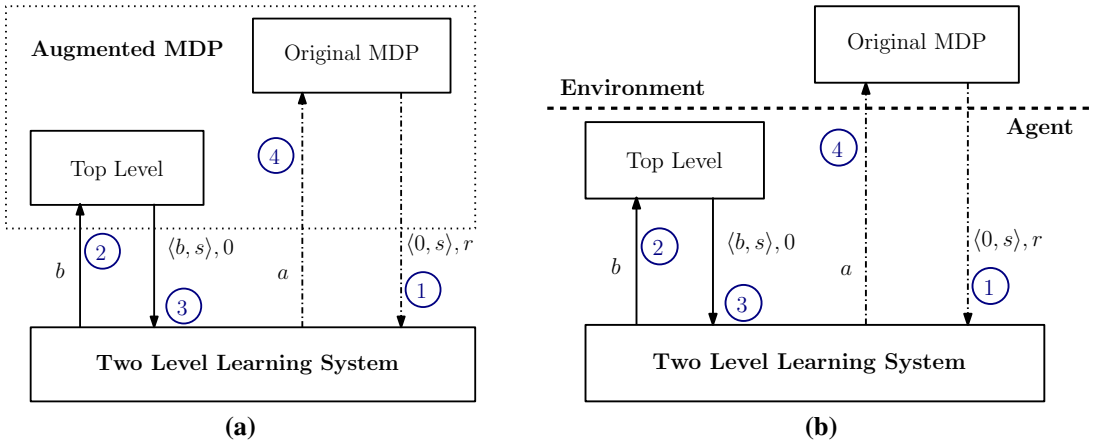


Figure 4.5: Interaction between two level learning system and environment. Numbers in circles shows the sequence of interactions. Solid arrows are deterministic interactions, dashed-dotted arrows are stochastic. (a) Boundary of augmented MDP (environment) and the two level learning system (agent) from the agent perspective. (b) Boundary of the original environment defined by the original MDP and the agent (system) acting in it. Note that the learning system always receives zero reward from the top level of the system.

parts, one for the top and another for the bottom level, i.e,

$$\psi'(\langle b, s\rangle) = \begin{cases} \psi(0,s) \in \mathcal{A}_0(s) & \text{if } b = 0 \\ \psi(b,s) \in \mathcal{A}_b(s) & \text{otherwise, i.e., } b \in [1, 2^K] \end{cases} \tag{4.4}$$

We assume $\forall b \in A_0, A_b \neq \emptyset$, i.e., there is always at least one action that can be selected after activating the CCs indicated by b. We leave it to the user to design CCs such that the problem is not over-constrained in any state. Now that we have defined the augmented MDP for two level learning, we proceed to define its value functions.

We define the state value function of the augmented MDP with ψ' as,

$$V^{\psi'}(\langle b, s\rangle) = \sum_{\langle b', s'\rangle \in \mathcal{S}'} \mathcal{P}'(\langle b', s'\rangle | \langle b, s\rangle, \psi(b,s))[\mathcal{R}'(\langle b, s\rangle, \psi(b,s), \langle b', s'\rangle) + \gamma' V^{\psi'}(\langle b', s'\rangle)] \tag{4.5}$$

where we use the simplified notation for expectation over the policy (see Equation 2.7 page 16), $\gamma' = 1$ if the transition is from top level to bottom and $\gamma' = \gamma$, the original MDP's discount factor, if transiting between original states. The γ' indicates that we

remove discounting for the top level case when $b = 0$ and only discount at each time step corresponding to that of the original MDP 's state transitions. Because $\mathcal{P}'$ is deterministic when transiting between top and bottom levels and $\mathcal{R}'$ is zero during those transitions, we have for the top level where $b = 0$,

$$V^{\psi'}(\langle 0, s\rangle) = \sum_{\langle b', s'\rangle \in \mathcal{S}'} \mathcal{P}'(\langle b', s'\rangle | \langle 0, s\rangle, \psi(0, s))[\mathcal{R}'(\langle 0, s\rangle, \psi(0, s), \langle b', s'\rangle) + \gamma' V^{\psi'}(\langle b', s'\rangle)] \tag{4.6}$$

$$= \mathcal{P}'(\langle \psi(0, s), s\rangle | \langle 0, s\rangle, \psi(0, s))[\mathcal{R}'(\langle 0, s\rangle, \psi(0, s), \langle \psi(0, s), s\rangle) + \gamma' V^{\psi'}(\langle \psi(0, s), s\rangle)] \tag{4.7}$$

$$= 1 \cdot [0 + 1 \cdot V^{\psi'}(\langle \psi(0, s), s\rangle)] \tag{4.8}$$

$$= V^{\psi'}(\langle \psi(0, s), s\rangle). \tag{4.9}$$

and, for the bottom level where $b > 0$, based on Equations 4.1 and 4.2, we have,

$$V^{\psi'}(\langle b, s\rangle) = \sum_{\langle 0, s'\rangle \in \mathcal{S}'} \mathcal{P}'(\langle 0, s'\rangle | \langle b, s\rangle, \psi(b, s))[\mathcal{R}'(\langle b, s\rangle, \psi(b, s), \langle 0, s'\rangle) + \gamma' V^{\psi'}(\langle 0, s'\rangle)] \tag{4.10}$$

$$= \sum_{s' \in \mathcal{S}} \mathcal{P}(s' | s, \psi(b, s))[\mathcal{R}(\langle b, s\rangle, \psi(b, s), \langle 0, s'\rangle) + \gamma V^{\psi'}(\langle 0, s'\rangle)]. \tag{4.11}$$

Then, we simplify the augmented state value function in Equation 4.5 using Equations 4.9 and 4.11 as,

$$V^{\psi'}(\langle b, s\rangle) = \begin{cases} V^{\psi'}(\langle \psi(0, s), s\rangle) & \text{if } b = 0 \\ \sum_{s'} \mathcal{P}(s' | s, \psi(b, s))[\mathcal{R}(s, \psi(b, s), s') + \gamma V^{\psi'}(\langle 0, s'\rangle)] & \text{otherwise} \end{cases} \tag{4.12}$$

Note that the original MDP's discount factor, transition and reward model are explicitly stated.

Having defined the state value function, we first explain stochastic policies before

presenting the theoretical results relating policies to the original MDP. For stochastic policies, actions are selected for each state with some probability. Deterministic policies are a special case of stochastic policies where all probability mass (unity) is assigned to only one value. A stochastic policy for an MDP is written as π: $\mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, i.e., $\pi(s, a) = P(a|s)$ the probability of choosing action a given current state s. The Bellman equation for the state value function (see Equation 2.3 page 15) can be rewritten to take expectation over the stochastic policy, i.e.,

$$V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\} \tag{4.13}$$

$$= \sum_{a} \pi(s, a) \sum_{s'} \mathcal{P}(s'|s, a)[\mathcal{R}(s, a, s') + \gamma V^{\pi}(s')] \tag{4.14}$$

In the case of a two level policy ψ' written in parts in Equation 4.4, the stochastic versions are,

$$\psi : \{0\} \times \mathcal{S} \times \mathcal{A}_0 \mapsto [0, 1], \quad \psi(0, s, b) = P(b|s) \tag{4.15}$$

$$\psi : [1, K] \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1], \quad \psi(b, s, a) = P(a|s, b) \tag{4.16}$$

Note that in Equation 4.16 actions can be constrained by b in certain states by having the $P(a|s, b) = 0$ entries in the probability table (see Equation 4.67 for an example). We now present the first lemma.

Lemma 4.1. *Given any two level policy ψ' for the augmented MDP. Let a control policy for the original MDP be, $\pi(s, a) := \sum_b \psi(0, s, b)\psi(b, s, a)$, where $\psi(0, \cdot)$ and $\psi(b, \cdot)$ are top and bottom policies of ψ' respectively. Then, for all original MDP states, s, $V^{\pi}(s) = V^{\psi'}(\langle 0, s\rangle)$.*

Proof. Suppose we are given the augmented MDP stochastic policy ψ'. We need to show that the evaluated state values $V^{\psi'}(\langle 0, s\rangle)$ in the augmented MDP will be equal to $V^{\pi}(s)$ when the two level system executing ψ' is evaluated as a *black-box* control policy, π, in the original MDP.

We begin by showing that the state value function, $V^{\psi'}$, is the expected return from

a sequence of rewards exclusively generated by the original MDP's reward model, i.e.,

$$V^{\psi'}(\langle 0, s\rangle) = E_{\psi'}\{r' + r_t + r' + \gamma r_{t+1} + ... + r' + \gamma^{t'} r_{t+t'} + ... \mid s_t = s\} \tag{4.17}$$

$$= E_{\psi'}\{r_t + \gamma r_{t+1} + ... + \gamma^{t'} r_{t+t'} + ... \mid s_t = s\} \quad , \text{ as } r' = 0 \tag{4.18}$$

$$= E_{\psi'}\{\sum_{t'=0}^{\infty} \gamma^{t'} r_{t+t'} \mid s_t = s\} \tag{4.19}$$

$$= E_{\psi'}\{R_t \mid s_t = s\} \tag{4.20}$$

where R_t in Equation 4.20 only consists of the rewards from the original MDP as illustrated in Equation 4.3. This result is straightforward as the augmented MDP does not modify the original reward signal.

Next we show that the probabilities involved in Equation 4.20 that govern original state transitions from s to s' are the same as that given by the transition probability in the original MDP. We rewrite Equation 4.5 after it has been simplified by Equation 4.12 for the case $b = 0$ for stochastic policies as,

$$V^{\psi'}(\langle 0, s\rangle) = \sum_b \psi(0, s, b) V(\langle b, s\rangle) \tag{4.21}$$

$$= \sum_b \psi(0, s, b) \sum_a \psi(b, s, a) \sum_{s'} \mathcal{P}(s'|s, a)[\mathcal{R}(s, a, s') + \gamma V^{\psi'}(\langle 0, s'\rangle)] \tag{4.22}$$

$$= \sum_a \overbrace{\sum_b \psi(0, s, b)\psi(b, s, a)}^{\substack{\text{Control policy } \pi \text{ in} \\ \text{original MDP using } \psi'.}} \sum_{s'} \mathcal{P}(s'|s, a)[\mathcal{R}(s, a, s') + \gamma E_{\psi'}\{R_t \mid s_t = s'\}] \tag{4.23}$$

as $\pi(s, a) := \sum_b \psi(0, s, b)\psi(b, s, a)$, by relabelling,

$$= \sum_a \pi(s, a) \sum_{s'} \mathcal{P}(s'|s, a)[\mathcal{R}(s, a, s') + \gamma E_{\pi}\{R_t \mid s_t = s'\}] \tag{4.24}$$

$$= \sum_a \pi(s, a) \sum_{s'} \mathcal{P}(s'|s, a)[\mathcal{R}(s, a, s') + \gamma V^{\pi}(s')] \tag{4.25}$$

$$= V^{\pi}(s). \tag{4.26}$$

Note that at Equation 4.23, the term $\sum_b \psi(0, s, b)\psi(b, s, a)$ is the control policy derived by the two level system executing ψ', which we relabel as π. This indicates that the probability of the two level system control policy eventually choosing a in s is the expectation over the choice of top action, i.e., CC activations. No other probabilities are present since transiting from the top level to the bottom level is deterministic by definition.

The probabilities used for the expectation are governed by the same probabilities in the original MDP and the augmented MDP does not introduce any new terminal states. These imply that in the episodic MDP case, a two level policy that terminates in the augmented MDP must terminate in the original MDP. □

The consequence of Lemma 4.1 is that evaluating the control policy π derived from the two level system executing ψ' in the original MDP results in the same state values as that of ψ' being evaluated in the augmented MDP. This implies that the quality of the policy for an augmented MDP remains the same when the policy is applied to the original MDP using the two level system.

Next, we show that the two level policy does not limit the proposed system's ability to represent equivalent policies for the original MDP. This indicates that the two level policy for the augmented MDP has enough flexibility to represent any original MDP solution regardless of the CCs designed for the system.

Lemma 4.2. *Given any policy π for the original MDP, and given any augmented MDP based on the original MDP, there exists a two level policy, ψ' for the augmented MDP, such that $V^{\psi'}(\langle 0, s\rangle) = V^{\pi}(s)$.*

Proof. As the transition probabilities and reward model of augmented MDPs are fixed with respect to the original MDP by definition, the only difference between augmented MDPs is the set of CCs provided by the user that constitutes the top level action space. Let $b_\emptyset$ be the top level action where all CCs are deactivated. The top action $b_\emptyset$ exists in any augmented MDP by definition. The simplest representation of a two level policy ψ' given π is one where the top part of ψ' deterministically deactivates all CCs in all states,

i.e.,

$$\psi(0, s, b) = \begin{cases} 1 & \text{if } b = b_\emptyset \\ 0 & \text{otherwise} \end{cases} \tag{4.27}$$

Next, we define the bottom policy for the sub-MDP with respect to $b_\emptyset$, that has the same primitive joint action space as the original MDP, to be π, i.e.,

$$\psi(b_\emptyset, s, a) = \pi(s, a). \tag{4.28}$$

Since the top policy is deterministic, the probability values of $\psi(b, s, a)$ when $b \neq b_\emptyset$ do not matter. That is, executing the two level system with ψ' in the original MDP gives the control policy,

$$\sum_b \psi(0, s, b)\psi(b, s, a) = \psi(0, s, b_\emptyset)\psi(b_\emptyset, s, a) \tag{4.29}$$

$$= 1 \cdot \pi(s, a) \tag{4.30}$$

$$= \pi(s, a) \tag{4.31}$$

Next, the state value function of π' can be expanded as follows,

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} \mathcal{P}(s'|s, a)[\mathcal{R}(s, a, s') + \gamma V^\pi(s')] \tag{4.32}$$

$$= \sum_a \pi(s, a) \sum_{s'} \mathcal{P}(s'|s, a)[\mathcal{R}(s, a, s') + \gamma E_\pi\{R_t \mid s_t = s'\}] \tag{4.33}$$

$$= \sum_a \psi(0, s, b_\emptyset)\psi(b_\emptyset, s, a) \sum_{s'} \mathcal{P}(s'|s, a)[\mathcal{R}(s, a, s') + \gamma E_{\psi'}\{R_t \mid s_t = s'\}] \tag{4.34}$$

$$= \overbrace{\psi(0, s, b_\emptyset) \sum_a \psi(b_\emptyset, s, a)}^{\text{Top to bottom transition is deterministic with zero reward.}} \sum_{s'} \mathcal{P}(s'|s, a)[\mathcal{R}(s, a, s') + \gamma E_{\psi'}\{R_t \mid s_t = s'\}] \tag{4.35}$$

$$= \psi(0, s, b_\emptyset) V^{\psi'}(\langle b_\emptyset, s \rangle) \tag{4.36}$$

$$= \sum_b \psi(0, s, b) V^{\psi'}(\langle b, s \rangle) \qquad \text{, note that } \psi(0, s, b) = 0 \text{ for } b \neq b_\emptyset. \tag{4.37}$$

$$= V^{\psi'}(\langle 0, s \rangle) \tag{4.38}$$

Note that at Equation 4.35 we can insert and rewrite the equation using the augmented MDP's transition probabilities and reward for augmented states transiting from top level to bottom level like in Equation 4.5. However we omit them as those transitions are completely deterministic and their reward is always zero as shown in Equation 4.9. Hence we have proven that at least one two level policy exists that can represent π for any augmented MDP. □

Lemma 4.2 implies that given π in the original MDP, we can find, or create from π, a two level policy ψ' that is of equivalent quality when evaluated in the augmented MDP. We have proven Lemma 4.2 by using the extreme case where all CCs are deactivated. In practice, it is usually the case that more than one such two level policy exists. The intuition behind this is observed when representing $\pi(s, a)$ using the expectation, $\sum_b \psi(0, s, b)\psi(b, s, a)$. This results in an increase in parameterization of π that does not reduce the flexibility in representing the probabilities. Hence there will not be a case whereby some primitive actions are always removed from the primitive joint action space. However, note that the same cannot be said if static constraints are added to the augmented MDP that are not placed on the original MDP. Next, we present the theorem on optimality based on Lemmas 4.1 and 4.2.

Theorem 4.3. *Given an optimal policy ψ'^* for the augmented MDP, the derived control policy π^* for the original MDP is optimal for the original MDP.*

Proof. We need to prove that ψ'^* being optimal implies that we can execute the two level system with ψ'^* to give the control policy π^* that is optimal in the original MDP. From Lemma 4.1, we have $V^{\pi^*}(\cdot) = V^{\psi'^*}(\langle 0, \cdot \rangle)$. Next, we need to show that $V^{\pi^*} = V^*$, i.e., π^* is the optimal solution to the original MDP. We prove this by contradiction. Suppose that there exists π^{**} that performs better than π^*, i.e., $V^{\pi^{**}} > V^{\pi^*}$. Then by Lemma 4.2, there exists ψ'^{**} such that $V^{\psi'^{**}}(\langle 0, \cdot \rangle) = V^{\pi^{**}}(\cdot)$ and therefore $V^{\psi'^{**}} > V^{\psi'^*}$. However

this contradicts the optimality of ψ'^*. Hence π^* must be optimal. □

From Theorem 4.3, we can learn optimal solutions in the augmented MDP to find optimal solutions in the original MDP. The additional top level and CCs do not prevent the learning system from discovering optimal primitive action sequences as in general the system can learn to deactivate CCs that lead to poor returns. This allows the system flexibility in figuring out good policies. In practice, we hope that a rational user designs useful CCs. Hence CCs should bias the bottom level's availability of primitive actions towards faster discovery of promising actions. Next, we describe the action value functions required for model-free learning.

Simplifying Action Value Functions

From here on, we will be dealing with the two level policy, ψ'. Therefore, to reduce clutter we drop the superscript ψ' when writing value functions that are obviously related to ψ'. The action value function Q for the augmented MDP is

$$Q(\langle b, s\rangle, \tilde{a}) = \sum_{\langle b', s'\rangle \in \mathcal{S}'} \mathcal{P}'(\langle b', s'\rangle | \langle b, s\rangle, \tilde{a})[\mathcal{R}'(\langle b, s\rangle, \tilde{a}, \langle b', s'\rangle) + V(\langle b', s'\rangle)] \quad (4.39)$$

where we have omitted discounting. Like V, we will only discount when the primitive state s transits to s'. We simplify Q by writing it in two parts, first $\forall b \in [1, 2^K]$,

$$\begin{aligned}
Q(\langle 0, s\rangle, b) &= \sum_{\langle b', s'\rangle \in \mathcal{S}'} \mathcal{P}'(\langle b', s'\rangle | \langle 0, s\rangle, b)[\mathcal{R}'(\langle 0, s\rangle, b, \langle b', s'\rangle) + V(\langle b', s'\rangle)] && (4.40)\\
&= \mathcal{P}'(\langle b, s\rangle | \langle 0, s\rangle, b)[\mathcal{R}'(\langle 0, s\rangle, b, \langle b, s\rangle) + V(\langle b, s\rangle)] && (4.41)\\
&= 1 \cdot [0 + V(\langle b, s\rangle)] && (4.42)\\
&= V(\langle b, s\rangle) \qquad \text{, substituting Equation 4.12,} && (4.43)\\
&= \sum_{s' \in \mathcal{S}} \mathcal{P}_b(s' | s, \psi(b, s))[\mathcal{R}(s, \psi(b, s), s') + \gamma V(\langle 0, s'\rangle)] && (4.44)\\
&= \sum_{s' \in \mathcal{S}} \mathcal{P}_b(s' | s, \psi(b, s))[\mathcal{R}(s, \psi(b, s), s') + \gamma Q(\langle 0, s'\rangle, \psi(0, s'))], && (4.45)
\end{aligned}$$

note that at Equation 4.43, $V(\langle b, s\rangle) = Q(\langle b, s\rangle, \psi(b, s))$. We could have stopped at Equation 4.43, however we have further expanded it to Equation 4.45, as we wish to derive update equations for it based on the reward signal from the original MDP. The variable b in Equation 4.45 represents a top level action in $\mathcal{A}_0$ and also a sub-MDP with a constrained action space. We subscript $\mathcal{P}$ with b to indicate primitive actions that are disallowed based on b. Equation 4.45 expresses the expected reward of taking a top level action b and following $\psi(b, \cdot)$ for one step before returning to the top and following $\psi(0, \cdot)$ thereafter.

Then, $\forall b \in [1, 2^K]$, where $a \in \mathcal{A}_b(s)$,

$$Q(\langle b, s\rangle, a) = \sum_{\langle b', s'\rangle \in \mathcal{S}'} \mathcal{P}'(\langle b', s'\rangle | \langle b, s\rangle, a)[\mathcal{R}'(\langle b, s\rangle, a, \langle b', s'\rangle) + V(\langle b', s'\rangle)] \quad (4.46)$$

$$= \sum_{s' \in \mathcal{S}} \mathcal{P}'(\langle 0, s'\rangle | \langle b, s\rangle, a)[\mathcal{R}'(\langle b, s\rangle, a, \langle 0, s'\rangle) + V(\langle 0, s'\rangle)] \quad (4.47)$$

$$= \sum_{s' \in \mathcal{S}} \mathcal{P}_b(s'|s, a)[\mathcal{R}(s, a, s') + \gamma V(\langle 0, s'\rangle)] \quad (4.48)$$

$$= \sum_{s' \in \mathcal{S}} \mathcal{P}_b(s'|s, a)[\mathcal{R}(s, a, s') + \gamma Q(\langle 0, s'\rangle, \psi(0, s'))]. \quad (4.49)$$

Equation 4.49 expresses the expected reward at the bottom level that returns to the top level immediately after one step and following $\psi(0, \cdot)$ thereafter.

Notice that for the function, $Q(\langle b, s\rangle, a)$, its domain is the cross product of a large space of size 2^K for b and the constrained joint action space $\mathcal{A}_b$. We can represent this using one $Q(\langle \cdot, s\rangle, a)$ function for each of the 2^K values of b. This corresponds to having one function per sub-MDP, where updating these functions requires exponential space and time. However we observe that, although $\mathcal{P}_i$ and $\mathcal{P}_j$ for two sub-MDPs have different domains as their actions are from $\mathcal{A}_i$ and $\mathcal{A}_j$ respectively, their probabilities are contained in the original MDP's $\mathcal{P}$. This is because the transition probability is a conditional probability where values are normalized over the state space but not the action space. As a result, the values $\mathcal{P}_b(s'|s, a)$ where $a \notin \mathcal{A}_b(s)$ are undefined. Therefore, the probability tables of $\mathcal{P}_i$ and $\mathcal{P}_j$ are sub-tables of $\mathcal{P}$ that do not contain the values of a not in $\mathcal{A}_i$ and $\mathcal{A}_j$ respectively.

Consequently, given 2^K sub-MDPs, $\forall s \in \mathcal{S}, i, j \in [1, 2^K], a \in \mathcal{A}_i(s) \cap \mathcal{A}_j(s)$, we have

$$Q(\langle i, s\rangle, a) = Q(\langle j, s\rangle, a) \tag{4.50}$$

that indicates the various sub-MDP functions are the same for their intersected domains. This leads to a single bottom level function definition: $\forall b \in [1, 2^K], a \in \mathcal{A}_b(s)$,

$$U(s, a) = Q(\langle b, s\rangle, a) \tag{4.51}$$

that is independent of b. Incidentally, U is similar to the action value function for the original MDP but constrained to the joint actions in $\mathcal{A}_b(s)$ by the two level policy ψ'.

Now, we redefine the top action value function as,

$$\begin{aligned} W(s, b) &= Q(\langle 0, s\rangle, b) && (4.52)\\ &= V(\langle b, s\rangle) && (4.53)\\ &= Q(\langle b, s\rangle, \psi(b, s)) && (4.54)\\ &= U(s, \psi(b, s)). && (4.55) \end{aligned}$$

Equation 4.55 indicates that the top level value function is an expectation of the bottom function U with respect to the primitive action space. Finally, the Bellman equations for W and U in terms of themselves are,

$$W(s, b) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, \psi(b, s))[\mathcal{R}(s, \psi(b, s), s') + \gamma W(s', \psi(0, s'))], \tag{4.56}$$

$$U(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a)[\mathcal{R}(s, a, s') + \gamma W(s', \psi(0, s'))]. \tag{4.57}$$

With the above equations we can select a maximizing primitive action in s by separately computing,

$$\psi(0, s) = \underset{b \in \mathcal{A}_0(s)}{\operatorname{argmax}} W(s, b), \tag{4.58}$$

followed by

$$\psi(b, s) = \underset{a \in \mathcal{A}_b(s)}{\operatorname{argmax}} U(s, a). \tag{4.59}$$

Using the definitions of W and U, we develop the update equations for RL in the next section.

4.3.3 Update Equations

We use linear function approximation to learn the value functions W and U as described in Section 2.4 page 21. This employs a linear combination of m number of features, f_p, with weights, w_p, to be learned. In other words, given a function $F(s, a)$, we want to find $\vec{w} = \langle w_1, ..., w_m \rangle$ such that,

$$F(s, a) = \vec{w} \cdot \vec{f}_{s,a}, \tag{4.60}$$

is a dot product where $\vec{f}_{s,a} = \langle f_1(s, a), f_2(s, a), \cdots, f_m(s, a) \rangle$.

To obtain $\vec{w}$ for each optimal value function, we perform on-policy temporal difference (TD) updates using: the two level policy $\psi' = \{\psi(b, \cdot) \mid b \in [0, 2^K]\}$ where each $\psi(b, \cdot)$ is a GLIE policy (Dietterich, 2000), and online samples of the form

$$\overbrace{\langle \langle 0, s \rangle, b, \underbrace{\langle b, s \rangle, a, r, \langle 0, s' \rangle, b'}_{\text{Used to update U}} \rangle}^{\text{Used to update W}} \tag{4.61}$$

where $b' = \psi(0, s')$. The first two entries in the sample denote that the top level is in the augmented state $\langle 0, s \rangle$ and it chooses the action $\psi(0, s) = b \in \mathcal{A}_0(s)$. The next two entries in the sample indicate the state $\langle b, s \rangle$ of the bottom level and the primitive action $\psi(b, s) = a \in \mathcal{A}_b(s)$ taken by it. The final two entries indicate that both levels observe reward r and go to next state s'. Note that when the bottom level policy chooses an exploratory action, i.e., $\psi(b, s)$, it does so by choosing a random action within the constrained joint action space $\mathcal{A}_b(s)$ as specified by the top level action b.

Let W and U be approximated as

$$W(s,b) \approx \vec{w_W} \cdot \vec{f_{W}}_{s,b}, \tag{4.62}$$

$$U(s,a) \approx \vec{w_U} \cdot \vec{f_{U}}_{s,a}, \tag{4.63}$$

and α be the step size parameter that decreases over time. Then, the weights $\vec{w_W}$ and $\vec{w_U}$ are updated as follows:

$$\vec{w_W} \leftarrow \vec{w_W} + \alpha[r + \gamma W(s',b') - W(s,b)]\vec{f_{W}}_{s,b} \tag{4.64}$$

$$\vec{w_U} \leftarrow \vec{w_U} + \alpha[r + \gamma W(s',b') - U(s,a)]\vec{f_{U}}_{s,a}. \tag{4.65}$$

The values involved in these updates are illustrated by the braces in Equation 4.61. The update Equations 4.64 and 4.65 correspond to the Bellman Equations 4.56 and 4.57 respectively. For both update equations, we are updating between two consecutive states in the original environment. Therefore, for Equation 4.56 we look ahead an extra (pseudo) step in the augmented MDP to update the estimate of W with the original reward signal r.

To explain how the updating equations guide exploration, we first rewrite the top and bottom sub-policies of ψ' as probabilities for ϵ-greedy policies. The probability of selecting a top level action b is

$$\psi(0,s,b) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}_0|} & \text{if } b = \text{argmax}_{b' \in \mathcal{A}_0} W(s,b') \\ \frac{\epsilon}{|\mathcal{A}_0|} & \text{otherwise} \end{cases}, \tag{4.66}$$

and similarly for the primitive action a given b,

$$\psi(b,s,a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}_b(s)|} & \text{if } a = a^* \\ \frac{\epsilon}{|\mathcal{A}_b(s)|} & \text{if } a \in \mathcal{A}_b(s) - \{a^*\} \\ 0 & \text{otherwise, i.e., } a \notin \mathcal{A}_b(s) \end{cases} \tag{4.67}$$

where $a^* = \text{argmax}_{a' \in \mathcal{A}_b(s)} U(s, a')$. In general, the ϵ parameters used for the top and bottom sub-policies need not be the same. Recall that the probability, $\pi(s, a) \in [0, 1]$, of selecting a primitive action a using the two level policy in s is the expectation,

$$\pi(s, a) = \sum_{b \in A_0} \psi(0, s, b)\psi(b, s, a). \tag{4.68}$$

The probabilities in Equations 4.66 and 4.67 may be specified using other forms of GLIE policies such as those used for Boltzmann exploration. But, note that for the bottom policy in Equation 4.67 the probability of selecting actions outside of the current sub-MDP b (i.e. CC restrictions) is always zero. Next, we illustrate the guiding effect of CCs through two examples on the simple MDP shown in Figure 4.6.

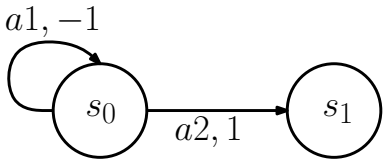


Figure 4.6: Guiding learning in a simple deterministic MDP. There are two states, initial state s_0 and terminal state s_1. Taking action $a1$ in s_0 transits to itself with a -1 reward while $a2$ transits to s_1 with 1 reward and the episode ends.

Example 4.3 (Useful CC)**.** *For the MDP in Figure 4.6, suppose we have the top level action space $\mathcal{A}_0 = \{b0, b1\}$ for one CC where $b0$ refers to deactivating the CC while $b1$ activates it. Let the CC be one that removes action $a1$ in s_0, i.e., resulting in $\mathcal{A}_{b1}(s_0) = \{a2\}$. Suppose $\alpha = 1$, $\epsilon = 0.1$, W and U are tabular functions initialized to zero. The ϵ-greedy policy chooses with equal probability when there is no maximum. For the first time step using the two level RL system,*

$$\begin{aligned}
\pi(s_0, a1) &= \sum_{b \in A_0} \psi(0, s_0, b)\psi(b, s_0, a1) \\
&= \psi(0, s_0, b0)\psi(b0, s_0, a1) + \psi(0, s_0, b1)\psi(b1, s_0, a1) \\
&= \underbrace{0.5(0.5)}_{\textit{deactivated}} + \underbrace{0.5(0)}_{\textit{activated}} \\
&= 0.25,
\end{aligned}$$

$$\begin{aligned}\pi(s_0, a2) &= \sum_{b \in A_0} \psi(0, s_0, b)\psi(b, s_0, a2) \\ &= \underbrace{0.5(0.5)}_{\textit{deactivated}} + \underbrace{0.5(1)}_{\textit{activated}} \\ &= 0.75.\end{aligned}$$

Suppose that the two level policy has chosen $b0$ *and* $a1$*. Then* $W(s_0, b0) = -1$ *and* $U(s_0, a1) = -1$*. The probabilities for the next step are,*

$$\begin{aligned}\pi(s_0, a1) &= \frac{0.1}{2}(\frac{0.1}{2}) + (0.9 + \frac{0.1}{2})(0) \\ &= 0.0025, \\ \pi(s_0, a2) &= \frac{0.1}{2}(0.9 + \frac{0.1}{2}) + (0.9 + \frac{0.1}{2})(1) \\ &= 0.9975.\end{aligned}$$

We tabulate these probabilities from the two level system with that of plain RL in Table 4.1 for various actions selected in the first step. Plain RL is assumed to use a ϵ*-greedy policy with* $\epsilon = 0.1$ *and having chosen the same primitive action as two level learning after the first step with on-policy TD updates. Notice that when the CC is useful, the probability of picking a good primitive action is always higher than in the plain RL case. This example demonstrates that the two level RL system will increase probability of choosing good actions when the given CC is useful for the problem.*

Time	Actions	Probabilities	Plain RL	Two Level RL	$\psi(0, s_0, b1)$
First Step	N.A.	$\pi(s_0, a1)$	0.50	0.2500	0.50
		$\pi(s_0, a2)$	0.50	0.7500	
Next Step	$b0, a1$	$\pi(s_0, a1)$	0.05	0.0025	0.95
		$\pi(s_0, a2)$	0.95	0.9975	
	$b0, a2$	$\pi(s_0, a1)$	0.05	0.0475	0.05
		$\pi(s_0, a2)$	0.95	0.9525	
	$b1, a2$	$\pi(s_0, a1)$	0.05	0.0025	0.95
		$\pi(s_0, a2)$	0.95	0.9975	

Table 4.1: Example probabilities of actions in the simple MDP in Figure 4.6 with a useful CC for a single update assuming different actions chosen for the first step. Plain RL and Two Level RL columns show the corresponding probabilities for the state action pair in the Probabilities column.

Example 4.4 (Useless CC). *In this example we use the same description as in Example 4.3 for the simple MDP in Figure 4.6, except that the CC is now one that removes action $a2$ in s_0 instead, i.e., resulting in $\mathcal{A}_{b1}(s_0) = \{a1\}$. Assuming that all other parameters are the same, suppose the two level RL system has chosen $b1$ and $a1$ for the first time step. Then, $W(s_0, b1) = -1$, $U(s_0, a1) = -1$ and the probabilities for the two level RL system for the next step are,*

$$\begin{aligned}\pi(s_0, a1) &= (0.9 + \frac{0.1}{2})(\frac{0.1}{2}) + (\frac{0.1}{2})(1) \\ &= 0.0975, \\ \pi(s_0, a2) &= (0.9 + \frac{0.1}{2})(0.9 + \frac{0.1}{2}) + (\frac{0.1}{2})(0) \\ &= 0.9025.\end{aligned}$$

Time	Actions	Probabilities	Plain RL	Two Level RL	$\psi(0, s_0, b1)$
First Step	N.A.	$\pi(s_0, a1)$ $\pi(s_0, a2)$	0.50 0.50	0.7500 0.2500	0.50
Next Step	$b0, a1$	$\pi(s_0, a1)$ $\pi(s_0, a2)$	0.05 0.95	0.9525 0.0475	0.95
	$b0, a2$	$\pi(s_0, a1)$ $\pi(s_0, a2)$	0.05 0.95	0.0975 0.9025	0.05
	$b1, a1$	$\pi(s_0, a1)$ $\pi(s_0, a2)$	0.05 0.95	0.0975 0.9025	0.05

Table 4.2: Example probabilities of actions in the simple MDP in Figure 4.6 with a useless CC for a single update assuming different actions chosen for the first step. Plain RL and Two Level RL columns show the corresponding probabilities for the state action pair in the Probabilities column.

We tabulate the probabilities for the next step given the three possible top and bottom action sequences selected by two level learning in Table 4.2. For the cases where the first steps actions are $b0, a2$ and $b1, a1$ observe that the two level RL system will still result in a higher probability for $\pi(s_0, a2)$, albeit less probability than plain RL. However, notice that there will be a low tendency to activate the useless CC as the probability $\psi(0, s_0, b1) = 0.05$ is low for those cases. For the case of $b0, a1$, we see that two level RL actually tends towards the primitive action $a1$. This can be explained by observing that $\psi(0, s_0, b1) = 0.95$, i.e., having not received a good reward, the system

will prefer to consider the CC before realizing it is useless.

The above two examples illustrate that two level learning with useful CCs can guide exploration towards states with higher rewards by increasing the probability of selecting certain primitive actions. When the CCs are not useful, the top level of the system will tend to deactivate them and learn without the CCs. As CCs are specified by the user, like task based methods (Sutton et al., 1999; Dietterich, 2000; Andre and Russell, 2002), we presume the user is rational and will design useful CCs for the learning system. For ϵ-greedy policies, as ϵ goes to zero, similar to plain RL, the two level policy becomes the greedy policy. Example 4.4 presented an extreme case for a useless CC. In practice, we will want the system to learn to deactivate CCs when they are not useful in certain states (see Example 4.2). We will evaluate empirically, the overall impact of two level learning with CCs later in Section 4.4. In the following section we describe action selection in the proposed system.

4.3.4 Action Selection Under Constraints

Applying the bottom component of the two level policy $\psi(b, \cdot)$ often requires selecting a primitive action within $\mathcal{A}_b(s)$ that maximizes the value functions. For example, the ϵ-greedy bottom level policy is to select a maximal action

$$\psi(b, s) = \operatorname*{argmax}_{a \in \mathcal{A}_b(s)} U(s, a)$$

with $1-\epsilon$ probability, or a random action within $\mathcal{A}_b(s)$ with ϵ probability. In our system, $\mathcal{A}_b(s)$ is subjected to the constraints activated by the top level action b. This implies that the problem of finding a maximal action $\operatorname{argmax}_{a \in \mathcal{A}_b(s)} U(s, a)$ can be modelled as a constraint optimization problem (COP) over the original primitive action space, $\mathcal{A}$, as follows:

$$\begin{aligned} & \operatorname*{argmax}_{a \in \mathcal{A}} U(s, a), \\ \text{subject to:} \quad & c_{b,1}(s, a), ..., c_{b,p}(s, a) \quad c_{0,1}(s, a), ..., c_{0,q}(s, a) \end{aligned} \tag{4.69}$$

where the constraints $c_{b,j}(s,a)$ are activated by the top level action b, termed dynamic CCs, and the constraints $c_{0,l}(s,a)$ are always activated regardless of b, termed static CCs.

Let each constraint be a function on a subset of variables in $\mathcal{S}$ and $\mathcal{A}$ that returns $-\infty$ if violated, or 0 otherwise. The objective function to maximize for the COP is

$$g(s,a) = U(s,a) + C_0(s,a) + C_b(s,a) \tag{4.70}$$

where C_0 and C_b are the sum of their respective constraints $c_{0,l}$ and $c_{b,j}$. Note that to switch to selecting random actions within $\mathcal{A}_b(s)$, we can simply replace U with a random function. Likewise, the selection of action for the top level, $\operatorname{argmax}_{b \in \mathcal{A}_0(s)} W(s,b)$, can be similarly modelled.

Depending on the characteristics of the COPs, we employ different strategies to solve them. Next, we review two main methods used to solve action selection based on coordination graphs (see Section 3.2 page 31), an exact and an approximate method.

Exact Solver

The exact solver is based on bucket elimination (BE) (Dechter, 1999) as described in Section 3.2.1 page 32, with additional handling of hard constraints using methods from constraint programming. The main BE algorithm is given in Algorithm 4.2. There are N agents and each agent i corresponds to an action variable, a_i, with domain A_i. We assume that the objective function g to maximize may be additively decomposed into a coordination graph (CG). Hence the objective function is a sum of function components that may depend some of the N action variables, i.e.,

$$g(s,a) = \sum_{n=1}^{N} \sum_{i_1,\ldots,i_n \in \mathcal{X}_N} f_{i_1,\ldots,i_n}(s, a_{i_1}, \ldots, a_{i_n}) \tag{4.71}$$

where $\mathcal{X}_N$ is the set of all subsequences of the sequence $1, 2, ..., N$. For the case where g consists of at most binary (pairwise) components, we have the simplified case of

$$g(s, a) = \sum_{i=1}^{N} f_i(s, a_i) + \sum_{i,j \in [1,N] | i<j} f_{i,j}(s, a_i, a_j). \tag{4.72}$$

Let the function $scope(f)$ return the set of action variables involved in a function component f. Algorithm 4.2 depends on two other functions:

PICKVARIABLE that chooses an action variable from a given set of variables based on some heuristic. We have use the *most constrained variable* heuristic to always pick the variable involved in the highest number of constraints first. And,

ELIMINATE is depicted in Algorithm 4.3. It eliminates a variable a_i by summing it out from the coordination graph to give a new *maximized* function component for which a_i is maximized for every neighbour of a_i in the CG. An example of this process is given in Example 3.3 page 33.

As the state variable is constant while action selection is carried out, we do not mention it in the algorithms.

Algorithm 4.2 Recursive bucket elimination algorithm

Input: $\mathcal{A}$ – set of action variables, $\mathcal{F}$ – set of function components
Global: $a[1..N]$ – array of action value assignments initialized to null values
Return: maximized value for g
BUCKETELIMINATION($\mathcal{A}, \mathcal{F}$)

1: **if** $\mathcal{A} = \emptyset$ **then**
2: **return** $f()$, where $\{f\} = \mathcal{F}$ # f is a constant function
3: **end if**
4: $a_i \leftarrow$ PICKVARIABLE($\mathcal{A}$)
5: $F_i = \{f \in \mathcal{F} \mid a_i \in scope(f)\}$ # F_i is all functions that involve a_i
6: Initialize f_i to a table of $-\infty$ values with $scope(f_i) = [\bigcup_{f \in F_i} scope(f)] - \{a_i\}$
7: Let $n = |scope(f_i)|$, $A[1..n]$ be the domains of the action variables in $scope(f_i)$ and $a'[1..n]$ be auxiliary action values.
8: # f_i is passed by reference
9: ELIMINATE($scope(f_i) - \{a_i\}, a_i, F_i, A[], a'[], f_i$)
10: $\mathcal{F} \leftarrow \mathcal{F} \cup \{f_i\} - F_i$
11: $r \leftarrow$ BUCKETELIMINATION($\mathcal{A}, \mathcal{F}$)
12: # look up the maximum value for $a[i]$ using values in $scope(f_i)$
13: $a[i] \leftarrow lookup(f_i, a[1], ..., a[N])$
14: **return** r

Algorithm 4.3 Eliminate function with extensions for hard constraints

Input:

- $\mathcal{A}$ – remaining variables to maximize,
- a_i – the variable to eliminate,
- F – functions involving a_i,
- $A[1..n]$ – array of action variable domains in $scope(f)$,
- $a[1..n]$ – array of action values for variables in $scope(f)$,
- f – current function to store, passed by reference

ELIMINATE($\mathcal{A}, a_i, F, A[], a[], f$)

1: **if** $\mathcal{A} = \emptyset$ **then**
2: $f(a[1], ..., a[n]) \leftarrow \max_{a^* \in A[i]} \sum_{g \in F} g(a^*, a[1], ..., a[n])$ # maximize for a_i
3: $lookup(f, a[1], ..., a[n]) \leftarrow a^*$
4: **else**
5: Let $\mathcal{C}$ be the set of constraints over any variables in $scope(f)$
6: $\forall a_j \in scope(f)$, remove domain values for $A[j]$ using $\mathcal{C}$ with constraint propagation
7: $a_j \leftarrow$ PICKVARIABLE($\mathcal{A}$)
8: Let $\mathcal{C}_j$ be set of constraints that involve a_j
9: **for** $a \in A[j]$ **do**
10: $a[j] \leftarrow a$
11: $A[j] \leftarrow \{a\}$ # set the domain of a_j to a singleton
12: **if** $\neg \exists c \in \mathcal{C}_j$ s.t. $c(a_i, a[1], ..., a[n]) = -\infty$ **then**
13: ELIMINATE($\mathcal{A} - \{a_j\}, a_i, F, A[], a[], f$)
14: **end if**
15: **end for**
16: **end if**

The ELIMINATE function is extended with constraint propagation methods to better make use of hard constraints, i.e., the CCs that are activated in the current state. Before entering the ELIMINATE function from BUCKETELIMINATION, we first initialize the new maximized component, f_i to $-\infty$ (see Algorithm 4.2 Line 6), for every possible value. Then, this tabular function is passed by reference to ELIMINATE.

The for loop at Algorithm 4.3 Line 9, and the recursive call to ELIMINATE within, enumerates every joint action for the variables in the scope of f. This enumeration gives rise to a worst case time complexity that is exponential in $|scope(f)|$. Storing f gives rise to the worst case space complexity that is exponential in $|scope(f)| - 1$. Note that $|scope(f)|$ corresponds to the induced tree width of the CG.

Constraint propagation is used in Algorithm 4.3 Line 6 to remove values from the action variables' domains (Apt, 2003, Chapter 7). This includes methods such as, node consistency that directly reduces individual action domains based on violations in unary

constraints, and arc consistency that removes values if there is no value in other variables' domains that will together satisfy a higher arity constraint. Furthermore, domains are set to singleton values when enumerating actions at Line 11. This may in turn remove more actions in a subsequent recursive call. We have further employed forward checking (Apt, 2003, Section 8.4.1) before the recursive call at Line 12. This is similar to the preliminary *filtered join* method presented in Sánchez et al. (2004) for BE. These methods may reduce the time complexity in practice. Furthermore, if f is sparse, the space complexity can be reduced since all unvisited entries in the table have the pre-initialized value of $-\infty$.

Where the CG is dense, i.e., has many edges, BE is often not feasible in practice. In this case we turn to the approximate solver described next.

Approximate Iterative Solver

If the problem consists of features and constraints that can be additively decomposed into component functions involving up to two action variables (e.g. Equation 4.72), we can utilize the max-plus algorithm (Kok and Vlassis, 2006) for an approximate solution in dense CGs. Max-plus is a message passing method where agents iteratively exchange messages to compute marginal (objective) functions, one for each, such that each agent may maximize their action variable individually when the maximization is unique. However, as previously mentioned in Section 3.2.2 page 36 and Kok and Vlassis (2006), agents should not select their actions individually when the maximization is not unique. We review max-plus in this case and present a solution derived from Wainwright et al. (2004) for a similar max-product algorithm.

The max-plus algorithm is presented in Algorithm 4.4. At each message passing iteration, each agent i sends the messages,

$$\mu_{i,j}(a_j) = \max_{a_i \in A[i]} \left[f_i(a_i) + f_{i,j}(a_i, a_j) + \sum_{k \in \Gamma(i) - \{j\}} \mu_{k,i}(a_i) \right] - \kappa_{i,j} \tag{4.73}$$

at Line 15 where $\kappa_{i,j}$ is a normalizing constant as stated in Equation 3.5 page 37. Once

Algorithm 4.4 Max-plus algorithm for non-unique maximals

Input: ϵ – threshold for fixed point, `max_iter` – maximum number of iterations
Global: g – function to maximize, f_i, $f_{i,j}$ – component functions of g
Return: Maximal joint action
MAXPLUS(ϵ, `max_iter`)

Let $A[1..N]$ be the domains of the action variables
$\forall c \in \mathcal{C}$, remove domain values from $A[\cdot]$ with constraint propagation
Initialize arrays $a^*[1..N]$ and $a[1..N]$ to random actions
$g^* \leftarrow g(a^*)$ # current best action value
Initialize $\forall i, j \in [1, N]$ messages $\mu_{i,j}(\cdot) \leftarrow 0$
$\delta \leftarrow 0$; $\delta' \leftarrow \infty$; $m \leftarrow 0$ # loop variables
while $|\delta - \delta'| > \epsilon$ **and** $m <$ `max_iter` **do**
  $\delta \leftarrow \delta'$; $\delta' \leftarrow 0$
  # pass messages
  for $i \in [1, N]$ **do** # for each agent
    for $j \in \Gamma(i)$ **do** # for each neighbour in set of neighbours $\Gamma(i)$
      for $a_j \in A[j]$ **do** # for each action value
        $v \leftarrow \max_{a_i \in A[i]} \left[f_i(a_i) + f_{i,j}(a_i, a_j) + \sum_{k \in \Gamma(i) - \{j\}} \mu_{k,i}(a_i) \right] - \kappa_{i,j}$
        $\delta' \leftarrow max(\delta', |\mu_{i,j}(a_j) - v|)$ # find the max. change in message value
        $\mu_{i,j}(a_j) \leftarrow v$ # send message step
      end for
    end for
  end for
  # compute joint action
  for $i' \in [1, N]$ **do** # given a depth-first visit order in the CG
    $i \leftarrow$ ORDERING(i')
    if $root(i)$ **then**
      $a[i] \leftarrow \text{argmax}_{a'_i \in A[i]} \hat{g}_i(a'_i)$ # root agent
    else
      $p \leftarrow$ PARENT(i) # parent in depth-first visit order
      $a[i] \leftarrow \text{argmax}_{a'_i \in A[i]} \hat{g}_{p,i}(a[p], a'_i)$ # with constraints $\mathcal{C}$
    end if
  end for
  if $g(a) > g^*$ **then** # keep if action better than previous
    $g^* \leftarrow g(a)$; $a^* \leftarrow a$
  end if
  $m \leftarrow m + 1$
end while
return a^*

the messages have been sent, agents compute the max-marginals $\hat{g}$. For the unique maximum case,

$$\hat{g}_i(a_i) = f_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{j,i}(a_i). \tag{4.74}$$

However, since the maximization need not be unique, another pairwise max-marginal is required, i.e.,

$$\hat{g}_{i,j}(a_i, a_j) = \max_{\{\langle a_1,\ldots,a_N\rangle \in \mathcal{A} | a'_i = a_i \wedge a'_j = a_j\}} g(a_1, ..., a_N) \tag{4.75}$$

$$= f_i(a_i) + f_j(a_j) + f_{i,j}(a_i, a_j) + \sum_{j' \in \Gamma(i) - \{j\}} \mu_{j',i}(a_i) + \sum_{i' \in \Gamma(j) - \{i\}} \mu_{i',j}(a_j) \tag{4.76}$$

$$= f_{i,j}(a_i, a_j) + \hat{g}_i(a_i) - \mu_{j,i}(a_i) + \hat{g}_j(a_j) - \mu_{i,j}(a_j). \tag{4.77}$$

Using Equation 4.74 and 4.77 the maximal action is selected via a pre-determined depth-first visit order of a spanning tree on the CG given by ORDERING at Line 21. Starting from the root agent, we maximize $\hat{g}_i$ to get the action $a[i]$. Then, with this action we proceed down the ordering of agents. At each agent, its parent in the depth-first visit order is guaranteed to have been visited. Hence we select at Line 26,

$$\operatorname*{argmax}_{a_i \in A[i]} \hat{g}_{p,i}(a[p], a_i) \tag{4.78}$$

where p is the parent agent of i and $a[p]$ has already been selected. This scheme is optimal if the CG is a tree. However, in a general graph with cycles, this form of action selection is also approximate.

The hard constraints are used in Algorithm 4.4 in a few ways. First, the selection of actions and passing of messages for each agent i are done on the domain $A[i]$ where constraint propagation with the constraints in $\mathcal{C}$ has been applied. Next, at Line 26 we enforce constraints by forward checking. This is done implicitly in Equation 4.78 where we not only maximize over $A[i]$ but we disregard values of a_i such that there exists some violated constraint $c_{p,i}(a[p], a_i) = -\infty$. As this constraint will be part of the $f_{p,i}$ component, $f_{p,i}(a[p], a_i) = -\infty$ when c is violated. We illustrate the action selection with an example.

Example 4.5 (Max-plus selecting actions)**.** *Consider the CG in Figure 4.7. The component between agents 1 and 4 is zero. The CG has two cycles hence max-plus is inexact.*

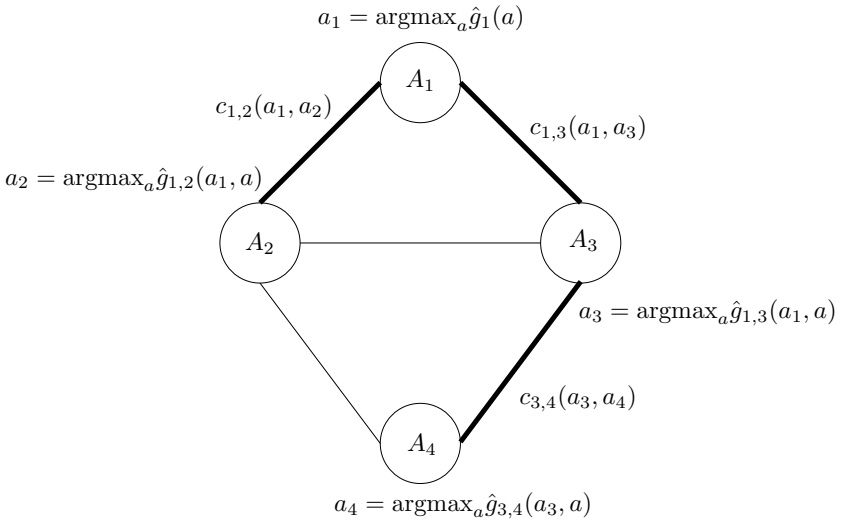


Figure 4.7: Max-plus action selection in a CG of 4 agents. The thick edges indicate a spanning tree whereby the depth-first visit ordering for agents is 1, 2, 3, 4.

The thick edges indicate a spanning tree that gives rise to the depth-first visit ordering. The max-marginals for computing actions in the depth-first agent order 1, 2, 3, 4 is shown. Constraints are enforceable along the edges of this spanning tree during action selection.

At each iteration of the max-plus algorithm, the number of messages passed is linear in the number of edges of the CG. Action selection requires each agent to maximize over their own domain that is typically smaller than the complexity of the CG. Hence it has worse case complexity that is linear. As the max-marginals are not guaranteed to converge and the action selection scheme is also approximate, we store and return the best action found throughout the iterations. If the function components of the objective function g have scope greater than two, more general solvers over factor graphs may be used (Kschischang et al., 2001) while exact computation of the max-marginals can be achieved at the cost of exponential in induced tree width message sizes (Petcu and Faltings, 2005).

4.3.5 Features & Constraints

In this section, we show how existing predicate definitions of features (basis functions) defined for function approximation can be reused to specify the CCs. We further de-

scribe how the top and bottom levels' features may share predicate components in their design and highlight a type of features that may be useful for certain multi-agent problems. These methods address the challenges of model complexity for the action value functions and of encoding knowledge.

Predicates are a natural way to encode expert knowledge as features for RL. They have been used in in first order logic (Russell et al., 1996, chap. 8) to generalize statements over objects. With predicates, we can derive the corresponding list of propositional features (PFs) by binding the variables of predicates to specific objects. We illustrate an example of deriving features from a predicate below.

Example 4.6 (Propositional features from a bad pass predicate)**.** *In the soccer domain, the expert knowledge of a bad pass can be written as the predicate:*

$$BadPass(s, a_x, a_y) := HasBall(P_x) \wedge IsPass(P_y, a_x) \wedge MoveNextToOpp(s, a_y),$$

where P_x is the player of the action variable a_x that is part of the state, $HasBall(P_x)$ is true if player P_x has the ball, $IsPass(P_y, a_x)$ is true if the action of player P_x, a_x, is to pass the ball to P_y, and $MoveNextToOpp(s, a_y)$ is true if the action of player P_y is to move next to an opponent.

Propositional features may be derived from BadPass by binding specific players (agents) action variables to it. For example, for P_1, P_2 we have the PF,

$$f_{1,2}^{BadPass}(s, a) = BadPass(s, a_1, a_2)$$

and for P_1, P_3,

$$f_{1,3}^{BadPass}(s, a) = BadPass(s, a_1, a_3).$$

The value of a PF is in $\{0, 1\}$.

Throughout this thesis we use alphabet subscripts to indicate unbound variables of predicates and numbers for bound instance of objects, e.g., specific agents' actions.

Relationship Between Constraints, Features, and Learning

Propositional features based on predicates are commonly employed in existing RL systems to approximate the value function (Marthi et al., 2005; Kok et al., 2005; Asgharbeygi et al., 2006). We also utilize PFs for the bottom level function U. An immediate advantage is that the PFs can be reused for specifying CCs. For each PBF f, we can formulate it into a constraint:

$$c(s, a) = -\infty \cdot f(s, a) \tag{4.79}$$

If $f(s, a) = 1$, the constraint $c(s, a)$ returns $-\infty$ signifying that the condition has been violated.

Reusing PFs in U as CCs has an added advantage during bottom level action selection. Instead of specifying individual constraints $c_{i,j}$ to sum for C_i in the objective function in Equation 4.70, we can simply set the corresponding PFs' weights of the activated CCs to $-\infty$. In so doing, the set of actions that are disallowed by the CCs will never be chosen by exact solvers due to its $-\infty$ weight. Then, the objective function g given the activated CCs, b, becomes,

$$g(s, a) = \underbrace{\sum_j w_j \cdot f_j(s, a)}_{\text{Normal PFs \& deactivated CCs}} + \underbrace{\sum_{k \in b} -\infty \cdot f_k(s, a)}_{\text{CCs activated in } b} \tag{4.80}$$

After selecting actions, we restore the original weights of these PFs during the updates (Equations 4.64 and 4.65). This is because in practice, incomplete COP solvers like max-plus may still select actions that violate certain activated CCs. When this happens, we can learn the weights for the violated constraint PFs that are useful for updating other weights. Hence PFs can be used both as constraints for guiding exploration and for function approximation.

Note that the predicates for features that are used as CCs assert what should not be. But the canonical form for COPs, e.g., Equation 4.69, constraints state what should be. We have written it this way as it is generally easier for a domain expert to criticize taking

particular actions in states instead of specifying an immediate solution. Converting to canonical form is simply taking the negation of the bound predicate (proposition) as described next.

Example 4.7 (Bad pass predicate as a CC)**.** *Consider the $BadPass$ predicate from Example 4.6 for players P_1 and P_2. When activated, its component in the COP's objective function is $-\infty \cdot BadPass(s, a_1, a_2)$. However its canonical form that the COP is subjected to is the negation $\neg BadPass(s, a_1, a_2)$, i.e., player P_1 should not make a bad pass to P_2.*

Top Level Features

The top level value function W is also a linear approximation of some features. Here, we show how the bottom level PFs can also be reused for the top level features. We observe that the activation of a constraint is often dependent on the state the system is in. Hence, we encode such state-dependent activation knowledge as the top-level features in the following manner: Let $Activated(c)$ be true if constraint c is activated. For each c corresponding to some PFs in the bottom level, we conjunct $Activated(c)$ or its negation with selected state predicates of agents involved in c. We give an example of a simple strategy that allows us to design top level features easily by reusing bottom level features' predicates below.

Example 4.8 (Top Level Features for $BadPass$ CCs)**.** *In the soccer game, we would like to deactivate the $BadPass_{1,2}$ constraint if the receiving player P_2 is near the enemy goal, i.e., $NearGoal(P_2)$ is true. This is because it may turn out to be better to take a chance at scoring. Hence, we define a predicate*

$$NearGoal(P_y) \wedge \neg Activated(BadPass_{x,y})$$

for each pair of players to capture this condition.

In practice, for our experiments, we programmatically construct top level PFs using the $Activated$ predicate. For each constraint c, we conjunct $Activated(c)$ or its negation

with sets of state predicates of agents involved in c that may be relevant. This reduces the work on the user's part as compared to manual enumeration.

4.3.6 Relational Features

For applications where the agents are homogeneous or their quantity changes over time, a new class of features called relational features (RFs) can be utilized for relational learning (see Section 3.6 page 46). Here, we modify the relations used in Guestrin et al. (2003); Asgharbeygi et al. (2006) for function approximation of the state value function to that of the action value function.

Let ρ be a predicate and n be the number of agents (action variables) that is involved in the predicate. For N agents, an RF based on ρ is then an aggregation of bound PFs,

$$\varrho_\rho(s,a) = \sum_{i_1,...,i_n \in Perm(N,n)} \tau_\rho \cdot \rho(s, a_{i_1}, ..., a_{i_n}) \tag{4.81}$$

where τ_ρ is a scaling factor and $Perm(N,n)$ is a function that returns all permutations of a subset of size n from the set $\{1,...,N\}$, i.e., it is the set of n-permutations where the number of such permutations are $|Perm(N,n)| = {}_N\mathrm{P}_n = \frac{N!}{(N-n)!}$. In the basic case, the scaling factor may be the constant 1. Then ϱ_ρ is the count of true valid bindings for predicate ρ. This is similar to the features used for relational temporal difference learning in Asgharbeygi et al. (2006). In our case we use the total number of possible bindings as the scaling factor, i.e., for single agent predicates it is the inverse count of agents $\tau_\rho = N^{-1}$, for pairwise agent predicates it is N^{-2}. For certain domains like RTS games, the number of agents N changes, hence τ scales relational features dynamically with the state. Note that Equation 4.81 also indicates that RFs are additively decomposable back into their PFs.

A single weight is learned for an RF. Hence the action value function U may be represented linearly as

$$U(s,a) = \sum_i w_i f_i(s,a) + \sum_\rho w_\rho \varrho_\rho(s,a) \tag{4.82}$$

where f_i are PFs or other kinds of features, and ϱ_ρ are RFs. W may be similarly specified. As RFs generalizes over the number of action variables, they may greatly reduce the number of weights required for W as elaborated next.

Example 4.9 (Top Level Generalization)**.** *Consider the top level predicate for features,*

$$\rho(s, BadPass_{x,y}) := NearGoal(P_y) \wedge \neg Activated(BadPass_{x,y}),$$

described in Example 4.8 where $BadPass_{x,y}$ is the top level $BadPass$ CC for any pair of players P_x, P_y derived from the top level action. For N agents, there are $N(N-1)$ top level action variables for $BadPass_{x,y} \in \{0, 1\}$ such that the top level action,

$$b = \langle ..., \underbrace{BadPass_{1,2}, BadPass_{1,3}, ..., BadPass_{x,y}, ..., BadPass_{N,N-1}}_{N(N-1)\ BadPass\ CCs}, ... \rangle$$

Using RFs defined by equation Equation 4.81 we learn a single weight instead of $N(N-1)$ weights if PFs were used, i.e., the RF is,

$$\varrho_\rho(s, b) = \sum_{x,y \in [1,N]} \rho(s, BadPass_{x,y}).$$

Where suitable for the domain, relational features can improve the learning rate in two ways. They can be used as additional features to the PFs to enrich the feature representation, or as a replacement for PFs to simplify the feature representation. In both cases they provide generalization for large state action spaces commonly found in multi-agent domains. We have presented RFs as one method to simplify representation for the top level value function. For the next section we will discuss how the top level efficiency issues can be addressed.

4.3.7 Top Level Efficiency Issues

In many domains, we observe that the top level action space, $\mathcal{A}_0$, may be heavily constrained based on the current state, s. This yields a smaller $\mathcal{A}_0(s)$ to explore, and

consequently, faster learning. In fact, this reduction to $\mathcal{A}_0(s)$ can be directly derived from the predicates used to create the CCs. If it can be inferred that a CC cannot be violated in the current state s, then the CC need not be activated. This can be done in $O(K)$ time as we only need to inspect the predicates of each of the K CCs.

Example 4.10 (Reducing $BadPass$ CCs to choose from)**.** *Consider the $BadPass_{2,3}$ CC. Since only player P_1 has the ball (see Figure 4.2), $HasBall(P_2)$ is false and CC for $BadPass_{2,3}$ can be deactivated. We can also deactivate $BadPass_{x,y}$ CCs for other pairs of players where P_x does not have the ball, thus reducing the quadratic number of $BadPass_{x,y}$ CCs to a linear number of $BadPass_{1,y}$ CCs.*

Another observation is that agents who are very far apart do not need to coordinate. We can define a $Nearby_{x,y}$ predicate that is true if two agents are within a given distance and conjunct it with those predicates involving them. Hence, multi-agent CCs of agents that are not currently nearby cannot be violated. This simple strategy has proven effective in reducing $\mathcal{A}_0$ for directing top level exploration in practice.

The top level action variables of the system may consist of a number of CCs that is quadratic in number of agents. While the above two observations can directly deactivate many of the CCs, there may be situations whereby using max-plus is still inefficient. In such situations, if the activating CCs are presumed to be independent, we may use PFs for W that only involve the state and one action variable corresponding to one CC. Consequently, top level actions can be selected independently in $O(K)$ time while bottom level actions are selected jointly. This turns out to be sufficient for good performance for our experiments as shown in Section 4.4.

4.3.8 Learning Algorithm

The detailed learning algorithm of the two level RL system which we call *coordination guided reinforcement learning* (CGRL), is given in Algorithm 4.5. The algorithm is presented by an agent (system) driven point of view that interacts with the original environment (see Figure 4.5b page 58). The algorithm is an on-policy algorithm as we

Algorithm 4.5 Coordination guided reinforcement learning

1: Observe initial state s
2: $b \leftarrow \psi(0, s)$ # select top level action from ϵ-greedy policy
3: $a \leftarrow \psi(b, s)$ # select bottom level action from ϵ-greedy policy
4: **while** $\neg terminal(s)$ **do**
5: Take action a, observe reward r and state s'
6: $b' \leftarrow \psi(0, s')$ # select CCs to activate
7: $\vec{v} \leftarrow \vec{w_U}$ # backup all weights in w_U
8: Activate CCs with b' # activated CCs' weights in w_U are set to $-\infty$
9: $a' \leftarrow \psi(b', s')$ # select primitive actions
10: $target \leftarrow r + \gamma W(s', b')$ # target to update towards
11: $\vec{w_W} \leftarrow \vec{w_W} + \alpha[target - W(s, b)]\vec{f}_{W\,s,b}$ # update W
12: $\vec{w_U} \leftarrow \vec{v}$ # restore weights for updating
13: $\vec{w_U} \leftarrow \vec{w_U} + \alpha[target - U(s, a)]\vec{f}_{U\,s,a}$ # update U
14: $s \leftarrow s'$; $b \leftarrow b'$; $a \leftarrow a'$
15: **end while**

update according to the actions taken in the environment, and it is a form of temporal difference learning.

Each time step in the environment corresponds to one iteration of the while loop. At Line 7, the weights for activated CCs in the top level action b' are backed up before being set to $-\infty$ in the next line for primitive action selection. This is straightforward for normal features and PFs. For the case of RFs, U is first additively decomposed before the weight w_c for f_c, that is an activated CC c, is set to $-\infty$. i.e.,

$$\begin{aligned} U(s,a) &= w_1 \cdot f_1(s,a) + w_2 \cdot f_2(s,a) + ... + w_c \cdot f_c(s,a) + ... + w_m \cdot f_m(s,a) \\ &= w_1 \cdot f_1(s,a) + w_2 \cdot f_2(s,a) + ... + (-\infty) \cdot f_c(s,a) + ... + w_m \cdot f_m(s,a). \end{aligned}$$

Finally, when a terminal state s' is observed, note that the value of $W(s', b') = 0$ at Line 10.

4.4 Experiments

We carried out experiments to evaluate the proposed approach on two domains: simplified soccer and tactical RTS (Buro, 2004). The environments are fully observable and episodic. All learning is online as no experience is saved and replayed. A video of the

#	Domain	RL Agents	Action Selection	Value Functions	Features
1	Soccer	2	Exact	Exact	N.A. (Tabular)
2	Soccer	4	Exact	Approximate	Propositional
3	Tactical RTS	10	Approximate	Approximate	Relational

Table 4.3: Overview of centralized CGRL experiment settings.

sample runs of our policies can be viewed at: http://youtu.be/aloAOTBEUZ4[1].

To visualize our results we use two main types of plots. The first is the plot of *average reward* for each block of M episodes, e.g., see Figure 4.8a where $M = 10,000$. A block of episodes is a grouping of sequential episodes experienced by the RL players. This plot shows the quality of the policy as episodes are experienced. The second is the plot of *cumulative average reward* per episode, e.g., see Figure 4.8b. This is the average reward over all previously experienced episodes, i.e., the i-th data point is the average reward of i episodes. This plot shows the overall goal achievement in the online setting as we wish to maximize the goal even while learning from the outset. An important point to note is that all plots start from the point *one* and not at zero. Depending on the scale of each individual plot, this may refer to a different number of initial episodes.

There are a total of four types of experiments. The first three types of experiments are designed to investigate the performance of the two level learning system on increasingly complex domains as shown in Table 4.3. Each experiment progressively includes other methods that make RL in complex multi-agent domains practical. In the subsequent sections we introduce the reinforcement learning players used and describe the two domains as they appear in the experiments. The last experiment presents empirical runtime results of the various RL methods on a large multi-agent problem.

4.4.1 Reinforcement Learning Players

We compare four types of RL players including methods from previous works.

1. *Independent player* – each agent selects their own action and learn their strategy independently (Claus and Boutilier, 1998) with their own value functions.

[1]In the video, CGRL-Solo is called Solo and CGRL-Full is called Coordinated. Alternative link: http://www.comp.nus.edu.sg/~plau/cgrl.wmv

2. *Coordinated player* – RL using single level on-policy learning with coordination graphs defined by features for joint action selection (Guestrin et al., 2002). Global updates are used for learning the weights (see Section 3.3.3 page 40). The coordinated player is a suitable baseline as it represents a multi-agent RL system without any coordination guidance.
3. *CGRL-Solo player* – a representative of having individual task hierarchies for each agent. In general, task based methods may allow other kinds of expert knowledge such as procedural knowledge and pseudo rewards to be encoded in addition to constraining the primitive action space for exploration. To compare the same type of knowledge for guiding exploration, we use CGRL with only single agent unary constraints. Hence tasks correspond to high level actions that activate some combination of unary constraints.
4. *CGRL-Full player* – uses our full two level learning system with coordination constraints defined on multiple agents.

The coordinated and CGRL players use the same features for their bottom level value function. Hence they have the same CGs defined by the features. The independent player has only features that involve single agents to represent their individual action value functions.

Comparing different types of expert knowledge used in various RL systems is inherently difficult. For example, it will be difficult to objectively compare hierarchical RL (see Section 3.4 page 43) with CGRL as the former uses procedural task decomposition while the latter uses declarative predicates as constraints. If CGRL performs better than HRL, it may boil down to a poor task decomposition or a weakness in the RL method. Yet there is no straightforward way to figure out which is the case without being able to compare the quality of task decomposition with the predicate CCs. Therefore, we have chosen the coordinated player and CGRL-Solo as state-of-the-art baselines that make use of the exact same predicates for features and CCs. Furthermore, they make use of the same techniques for joint action selection.

For the experiments, discounting (γ) is used in conjunction with the reward scheme to encourage learning outcomes, i.e., it is a part of problem modelling. The step size (α) and exploration (ϵ) parameters are set by empirical observation on a small number of episodes. The α parameters are set based on learning stability considerations while the ϵ parameters are set to ensure reasonable amount of exploration.

4.4.2 The Simplified Soccer Game Domain

In a game of simplified soccer, the objective is to score the first goal in the shortest time. The soccer field is a *grid world*. Soccer players can stay, move in 4 directions, or the player with the ball may pass or shoot with a probabilistic chance of success weighted by distance. The ball changes ownership to the opposing team if the player with the ball collides with any player or if a pass fails. A failure to score a goal results in the ball going to the nearest player to the goal. In each time step, submitted player actions are randomly shuffled and executed. Rewards are 1 for winning, and -1 for losing. The game is initialized by randomly placing soccer players in the half of the field corresponding to their home goal. The learners are pitted against three scripted strategy opponents:

1. *Random* players take actions at random;
2. *Defensive* players stay around the home quarter of the field and move to intercept the ball if it enters, the player with the ball does a solo counter-attack;
3. *Aggressive* players always go for the ball, once attained, the player with the ball heads for the goal while each of the other players stays near (marks) their respective enemy players.

The detailed description of the predicates used for this domain are given in Section A.1 page 230.

4.4.3 Experiment 1: Only Exact Methods

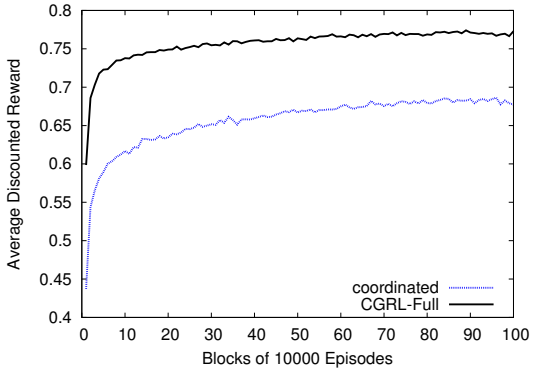
For the first type of experiments, we examine our proposed two level RL system without any approximation, using exact tabular functions and action selection via enumeration. This is to investigate if the extra top level is indeed useful for learning performance. A tabular function is equivalent to a linear function approximation where each feature is a boolean variable corresponding to an entry in the table. We only compare tabular coordinated and CGRL-Full players.

The soccer field is 6×4 units and the RL players have two soccer players versus one soccer player for the scripted opponent. RL players are evaluated against all three scripted strategies. RL players used discounting ($\gamma = 0.99$), ϵ-greedy policies with constant $\epsilon = 0.1$, and constant step size ($\alpha = 10^{-3}$). The CGRL-Full player used only pairwise CCs and no unary CCs for this experiment. Its top level has 3 dynamic CCs giving a top joint action space of size 2^3. There are only 3 dynamic CCs based on the 3 pairwise CC predicates described in Section A.1.3 page 234 as there is only one pair of agents. With tabular functions the number of parameters (table entries) to learn are more than 10,000 for each RL player. Hence we run the experiments for many episodes.

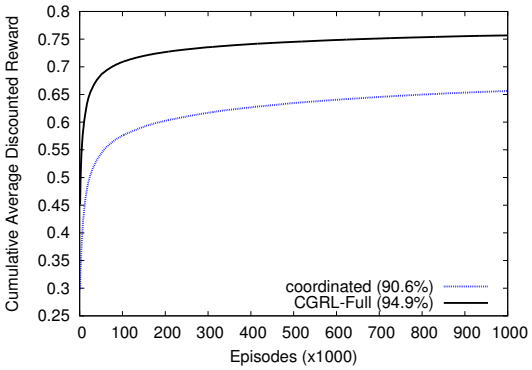
Figures 4.8, 4.9 and 4.10 show the results for the three scripted opponents respectively. One million episodes are used against random, and 10 million episodes against both defensive and aggressive opponents. Each curve is averaged over 10 runs. From the results we see that RL players have poorer goal achievement against harder opponents. The CGRL-Full player performs consistently better than the coordinated player. This verifies that the top level of our system is stable and improves learning performance when there are no other factors such as feature quality or approximate action selection.

4.4.4 Experiment 2: Function Approximation

For the second type of experiments, we evaluate the RL system using propositional features (PFs) for linear value function approximation and exact action selection through bucket elimination given in Algorithm 4.2. Experiments take place in a 12×8 soccer

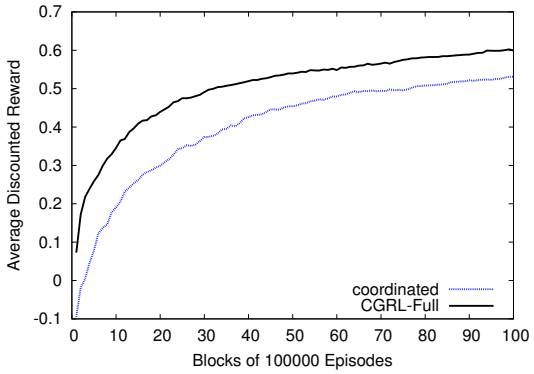


(a) Average Discounted Reward

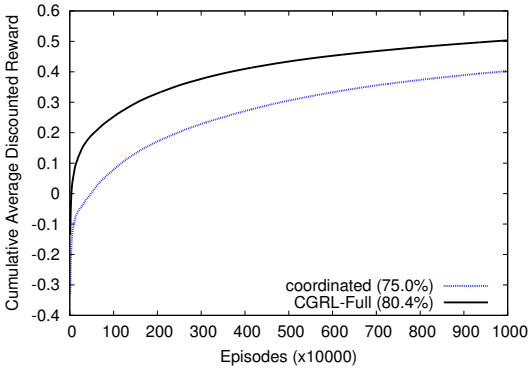


(b) Cumulative Average Discounted Reward (final win rate shown in brackets)

Figure 4.8: Centralized Exp. 1: Soccer results for random opponent. One million episodes averaged over 10 runs for each RL player.

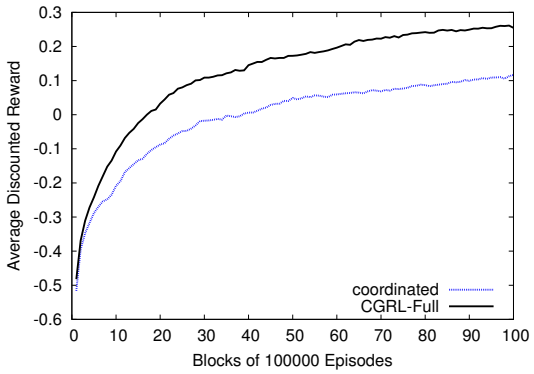


(a) Average Discounted Reward

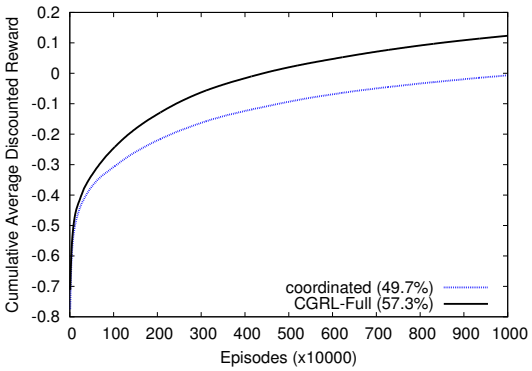


(b) Cumulative Average Discounted Reward (final win rate shown in brackets)

Figure 4.9: Centralized Exp. 1: Soccer results for defensive opponent. 10 million episodes averaged over 10 runs for each RL player.



(a) Average Discounted Reward



(b) Cumulative Average Discounted Reward (final win rate shown in brackets)

Figure 4.10: Centralized Exp. 1: Soccer results for aggressive opponent. 10 million episodes averaged over 10 runs for each RL player.

RL player	exploration (ϵ)			step size (α)		
	initial	final	decay	initial	final	decay
Independent & CGRL	1.0	0.01	0.998	0.1	0.01	0.998
Coordinated	1.0	0.01	0.998	0.2	0.10	0.998

Table 4.4: Centralized Exp. 2: Table of parameters for soccer experiments.

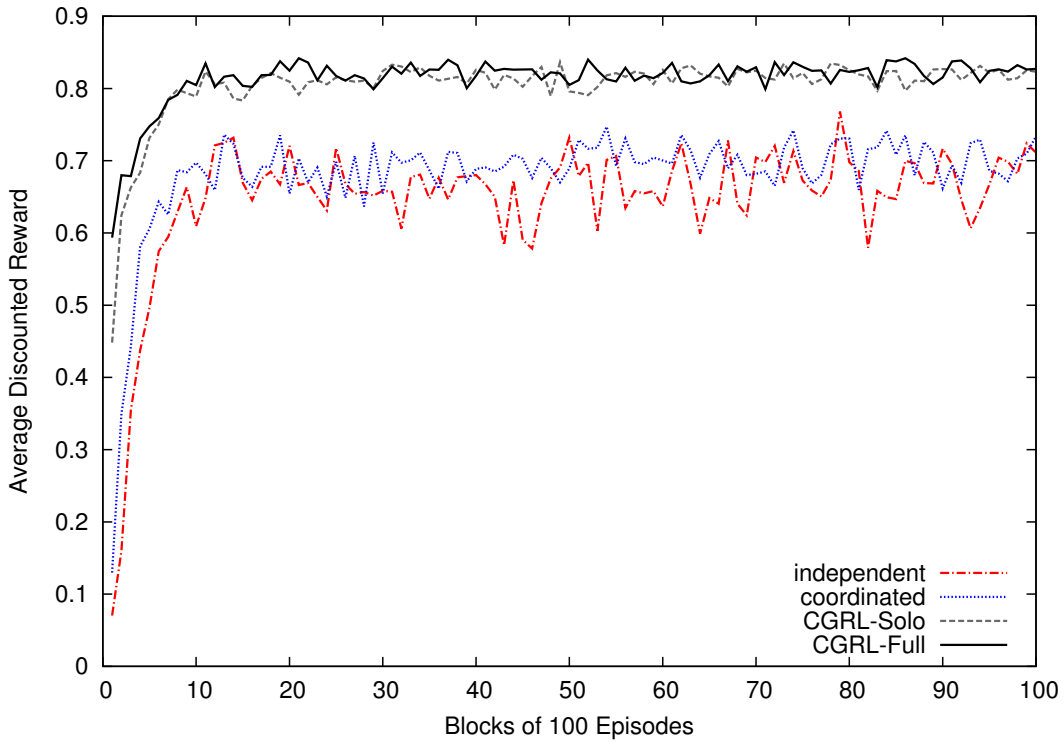
RL player	Weights	Total CCs	Static CCs	Dynamic CCs
Independent	944	8	8	0
Coordinated	1049	8	8	0
CGRL-Solo	1497	36	8	28
CGRL-Full	1821	66	20	46

Table 4.5: Centralized Exp. 2: Quantity of feature weights and CCs for soccer experiments with 4 agents.

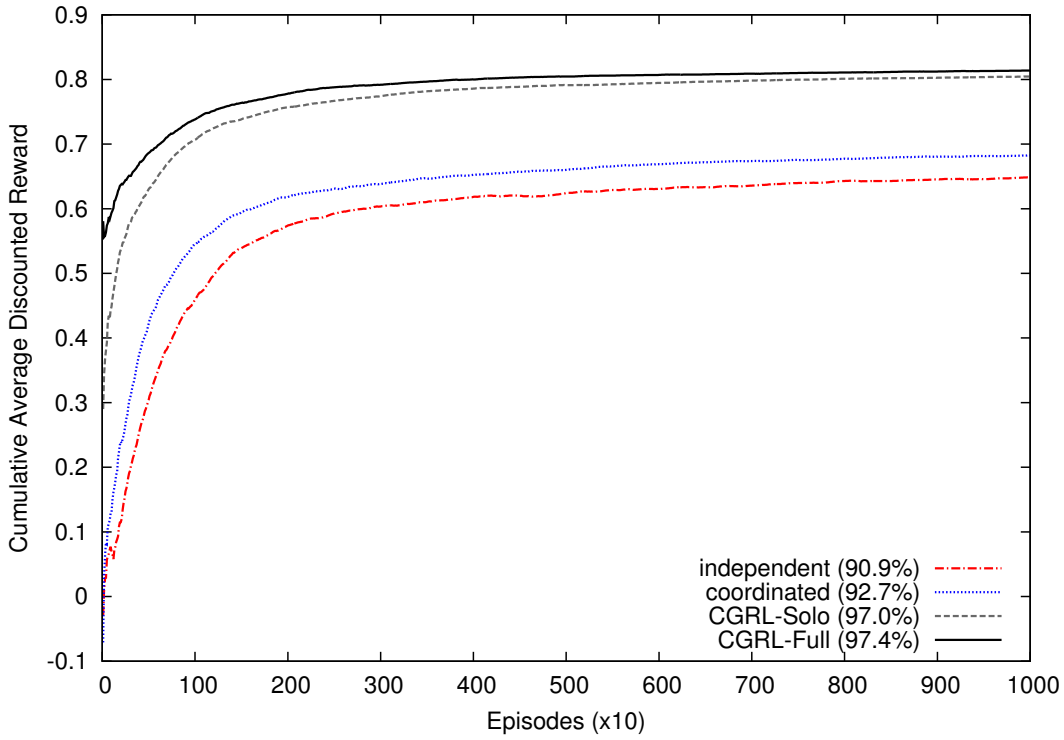
field. Learners have 4 soccer players versus 6 players for each of the scripted opponents. The size of the state space is at least 10^{13} and the size of the action space is 8^4. Agents coordinate if they are within a Manhattan distance of 10. For the size of the grid world, the CG formed by the agents is fully connected in most states. The learners used discounting with $\gamma = 0.99$, and ϵ-greedy policies with decaying step size (α) and exploration (ϵ) parameters as stated in Table 4.4.

Propositional features were used for soccer as good policies may require agents to have specific roles. The total number of feature weights to be learned and CCs used are given in Table 4.5. The static CCs for independent, coordinated and CGRL-Solo players are the simple and usually used single agent constraints such as preventing boundary collision and passing (or shooting) when a soccer player does not have the ball. For multi-agent CCs, we used 12 static collision CCs and 18 dynamic CCs including: $BadPass$, not jointly intercepting as opponent with the ball, and jointly blocking opponents' movements.

The results for soccer versus the three opponents are given in Figures 4.11, 4.12, and 4.13. The RL players are evaluated online for 10,000 episodes and each curve is averaged over 10 runs. The overall win rate is shown in brackets. In all three opponent setups, we see that the CGRL methods outperforms existing works that use coordinated and independent RL. This indicates that CGRL is a useful learning method over existing approaches. Furthermore, coordinated RL out performs independent RL. This shows

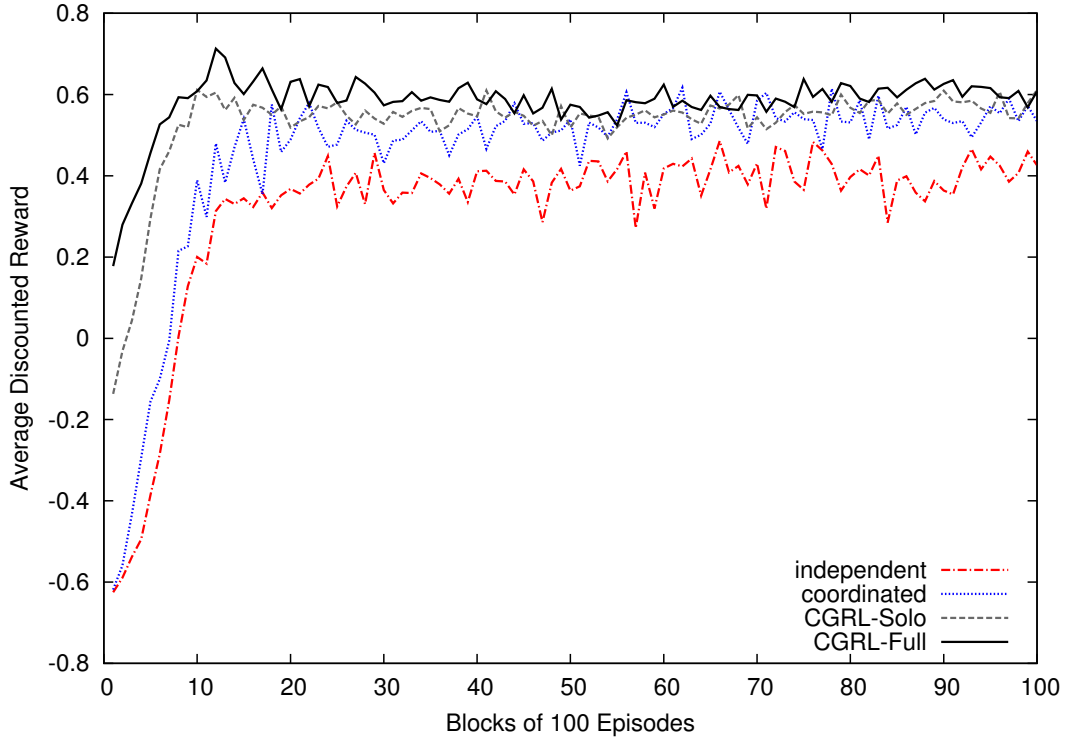


(a) Average Discounted Reward

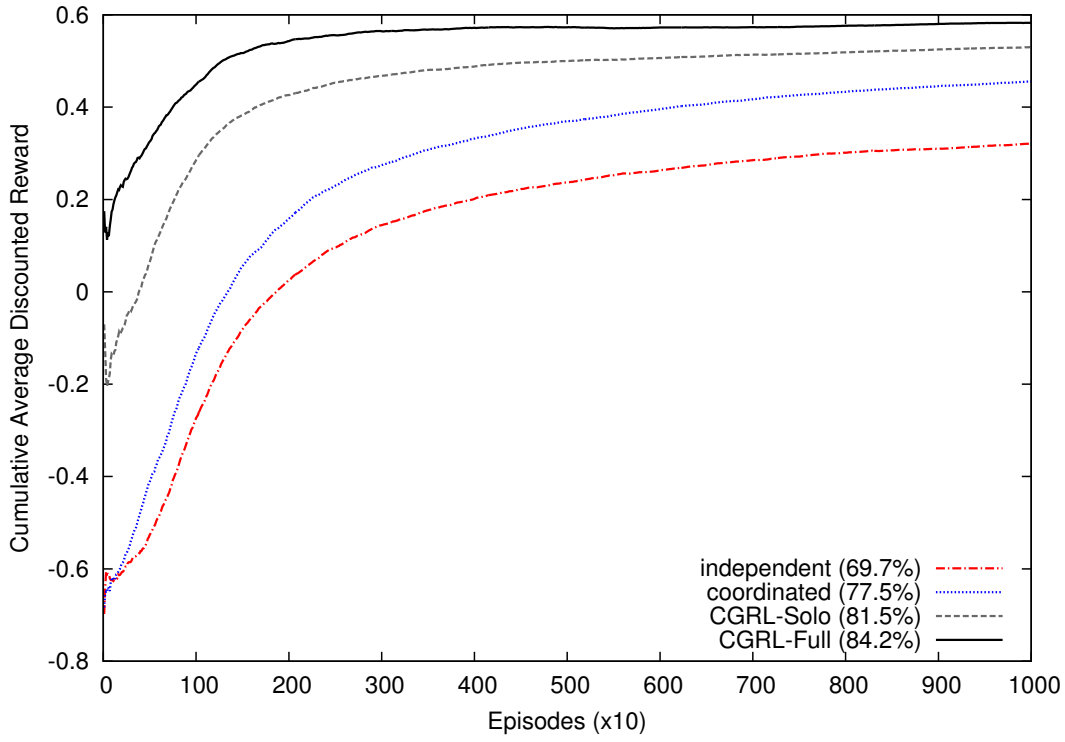


(b) Cumulative Average Discounted Reward (final win rate shown in brackets)

Figure 4.11: Centralized Exp. 2: Soccer results for random opponent. 10,000 episodes averaged over 10 runs for each RL player.

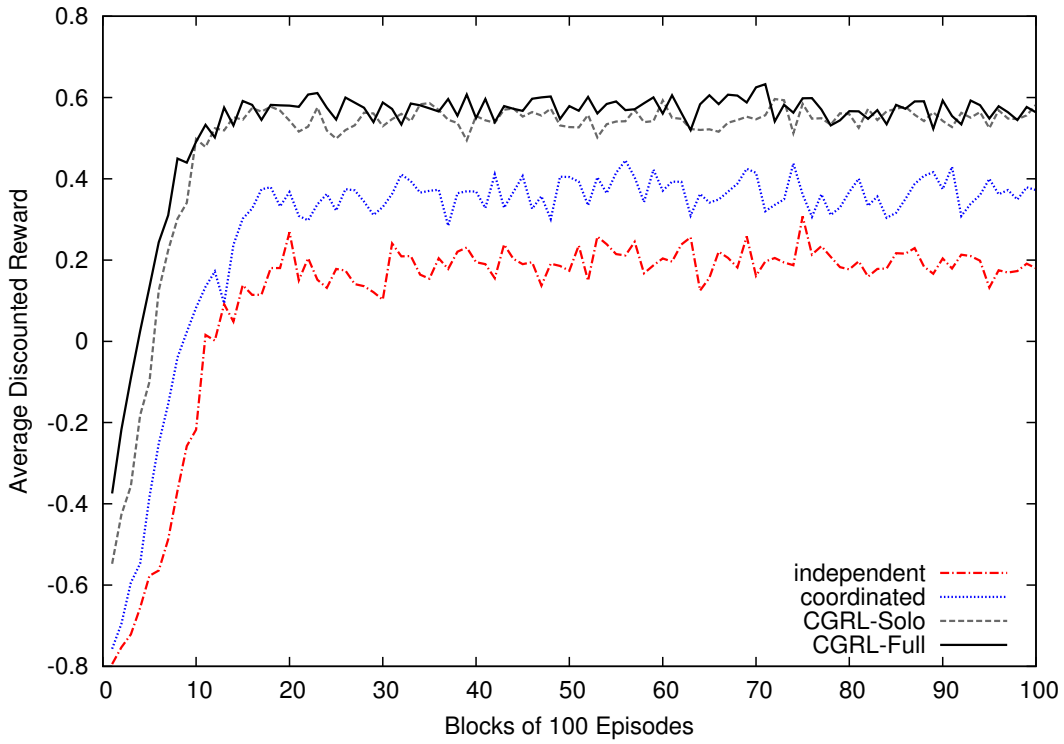


(a) Average Discounted Reward

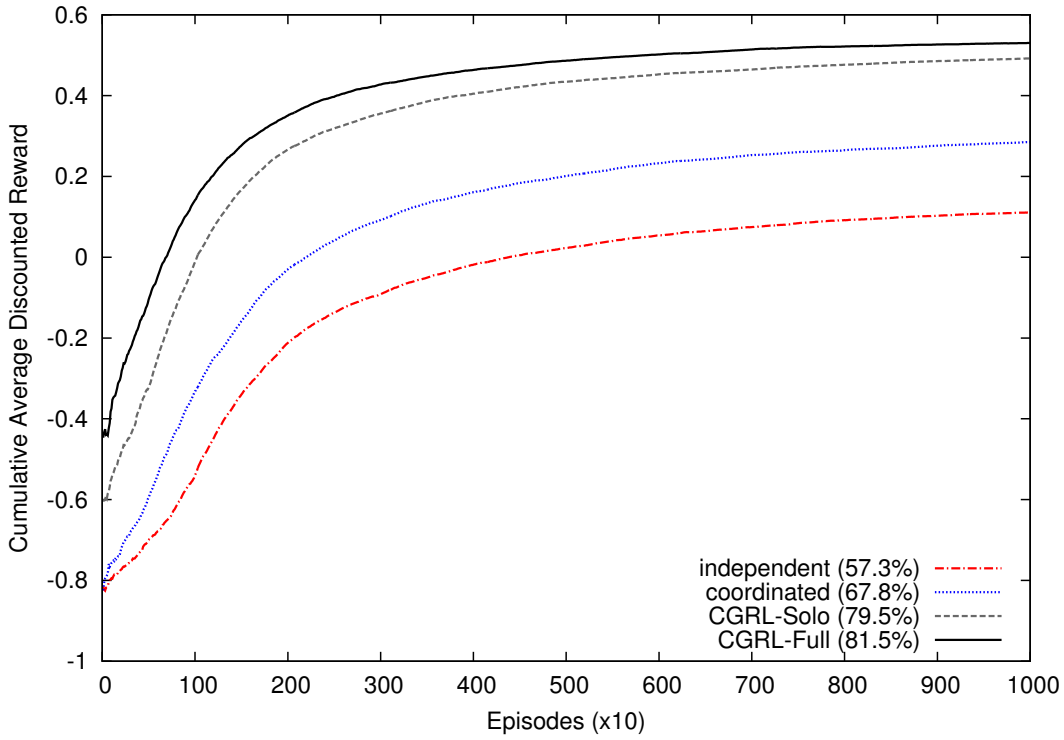


(b) Cumulative Average Discounted Reward (final win rate shown in brackets)

Figure 4.12: Centralized Exp. 2: Soccer results for defensive opponent. 10,000 episodes averaged over 10 runs for each RL player.



(a) Average Discounted Reward



(b) Cumulative Average Discounted Reward (final win rate shown in brackets)

Figure 4.13: Centralized Exp. 2: Soccer results for aggressive opponent. 10,000 episodes averaged over 10 runs for each RL player.

that the soccer game requires coordination to achieve good results and more so when the opponent has a coherent strategy such as being defensive or aggressive instead of having no strategy (random).

We see that CGRL consistently obtains better policies in Figures 4.11a, 4.12a, and 4.13a over the existing methods. The difference in quality of policy between CGRL-Solo and CGRL-Full against the random opponent is minimal. However, it is more distinguished against the defensive opponent, followed by the aggressive opponent. In terms of overall goal achievement while learning online, we see that CGRL-Full performs slightly better than CGRL-Solo in Figure 4.11b, and subsequently, much better in Figure 4.12b and Figure 4.12a. Further, the CGRL-Full player performs better much earlier compared to the other RL players. This advantage is the result of better early exploration in contrast to the other RL players. This indicates that CCs defined over multiple agents are effective in the online setting.

4.4.5 The Tactical Real-Time Strategy Domain

We have used the Open RTS[2] game system to evaluate our methods. The goal in tactical RTS is to eliminate the enemy team of marines quickly in a 240×240 *point based* map. Each marine occupies a point on the map with a fixed radius and a number of hit points. When its hit points reaches zero, it is destroyed. A marine's action domain consists of the 8 compass directions, an attack action for each possible enemy, and idle. The size of the action space is at least 10^{10} while the huge state space consists of all the possible marines' positions and hit points. An example of the game state is shown in Figure 1.1 page 3. Rewards are -0.1 per time step and 10^3 for eliminating the opposing team.

We pit the RL players against two scripted opponents:

1. *Aggressive* marines head for the nearest enemy and shoot enemies in range,
2. *Unpredictable* marines move in random directions and shoot enemies in range. The unpredictable opponent may be strong if its marines move in the same direction towards the enemy, or weak if they scatter.

[2] http://skatgame.net/mburo/orts/

Opponents' marines are able to shoot and move at the same time giving them an advantage. RL players must quickly learn to shoot and exploit teamwork as marines die easily. This makes it difficult to explore winning episodes. We use four setups in our experiments:

1. 10 RL marines versus 10 aggressive marines,
2. 10 RL marines versus 13 unpredictable marines,
3. 10 RL marines versus 13 aggressive marines,
4. 10 RL marines versus 5 unpredictable super marines.

The super marines have twice the firepower and hit points, hence coordination for the RL players is very crucial for success.

For tactical RTS, we include a new RL player, *CGRL-Static*, which utilized only the static collision CCs in addition to the capabilities of the solo player. This is to compare if the static collision CCs used provide most of the benefit of the CGRL-Full player.

4.4.6 Experiment 3: Relational Features & All Approximations

The third type of experiment integrates methods to deal with large multi-agent problems where the number of agents change over time. We incorporate the use of relational features (RFs) for generalization of learning and approximate primitive action selection using the max-plus algorithm presented in Algorithm 4.4 for 10 iterations.

For each setup, the RL players use no discounting ($\gamma = 1$) with the same decaying parameters. Parameters for RL for the various setups are given in Table 4.6. Setup 2 was given more exploration due to the unpredictable nature of the opponent that increases with the number of its marines.

RFs were used for RTS as the number of agents varies over time, except for the independent player that learns separate policies using PFs. The number of features weights and CCs are given in Table 4.7. Note that we use lesser features than in soccer to encode knowledge. Furthermore, relational features greatly reduce the number of weights to learn. Static CCs for independent, coordinated and CGRL-Solo players are those that

Setup	exploration (ϵ)			step size (α)		
	initial	final	decay	initial	final	decay
1, 3, & 4	1.0	0.01	0.998	0.01	10^{-4}	0.998
2	1.0	0.10	0.998	0.01	10^{-6}	0.998

Table 4.6: Centralized Exp. 3: Table of parameters for RTS experiments.

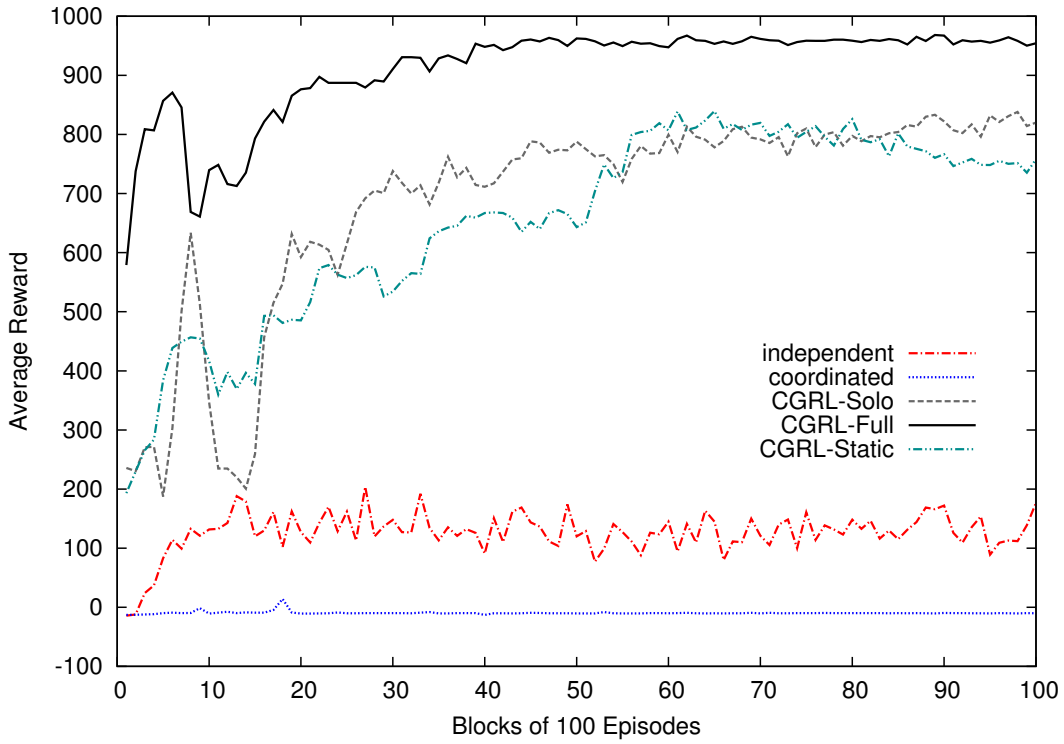
RL player	Weights	Total CCs	Static CCs	Dynamic CCs
Independent	350	30	30	0
Coordinated	53	30	30	0
CGRL-Solo	70	90	30	60
CGRL-Static	70	135	75	60
CGRL-Full	98	225	75	150

Table 4.7: Centralized Exp. 3: Quantity of feature weights and CCs for RTS experiments with 10 agents.

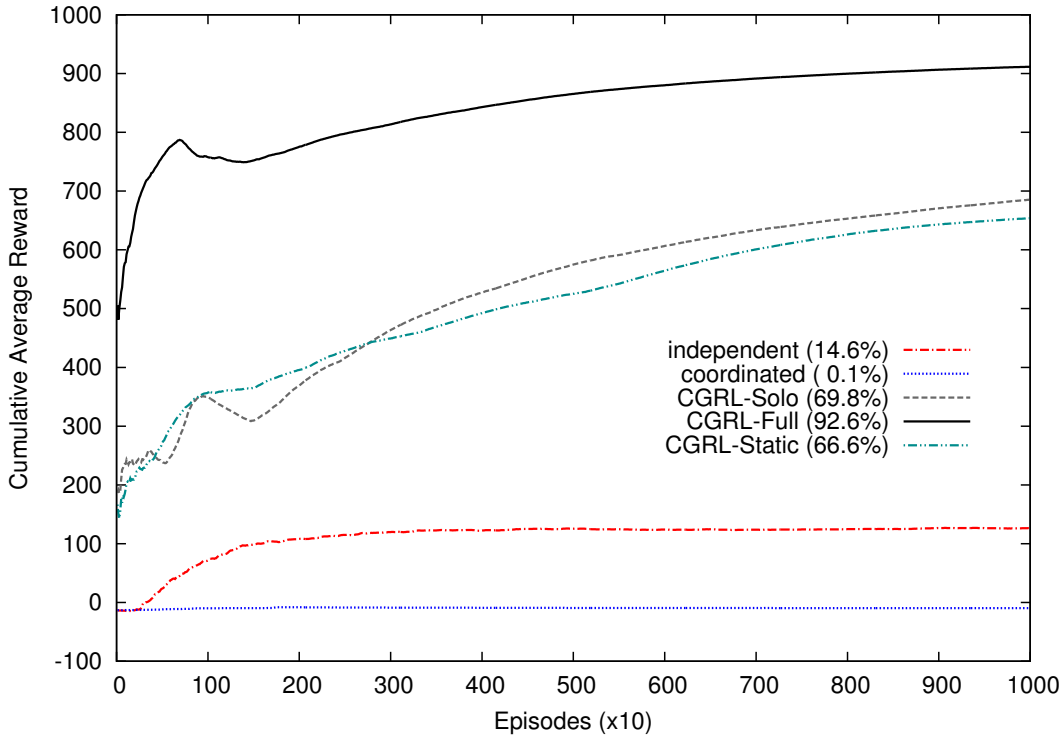
are simple constraints used on single agents, e.g. disallowing attacking enemies that are out of range, or a move action that collides with a boundary. For multi-agent CCs, we used 45 static collision CCs, and 90 dynamic CCs for troop formation to maximize overlapping firepower on enemies and to protect weaker team members. Further details of predicates used to create the features are given in Section A.2 page 235. Agents need to coordinate if they are within 30 points of each other, otherwise their binary features are set to zero.

Figures 4.14 to 4.17 show the results for the four RTS setups respectively. Total percentage wins are shown in brackets. We observe that all the RL players' policies converged over time. As seen in soccer, all CGRL players outperformed existing approaches of independent and coordinated RL. The CGRL-Full player was able to learn quickly in all the four setups with better policies and high overall goal achievement. The coordinated player experienced few winning episodes and ended up trying to lose as fast as possible to reduce the total negative reward obtained from each time step. The independent player managed to learn some strategy in Figure 4.14 and wins more episodes than the coordinated player in the other setups. However, its reward is less than the coordinated player in those setups.

The results for the CGRL-Static and CGRL-Solo players are mostly comparable. This is because the CGRL-Static marines tend to spread out more often and are de-

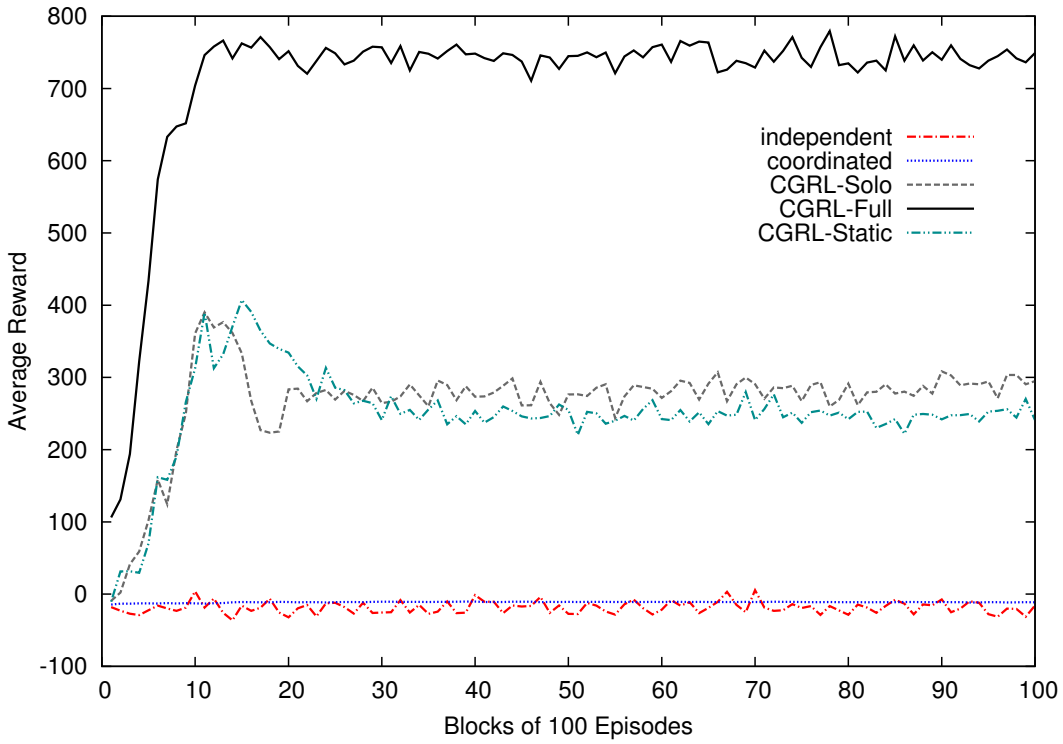


(a) Average Reward

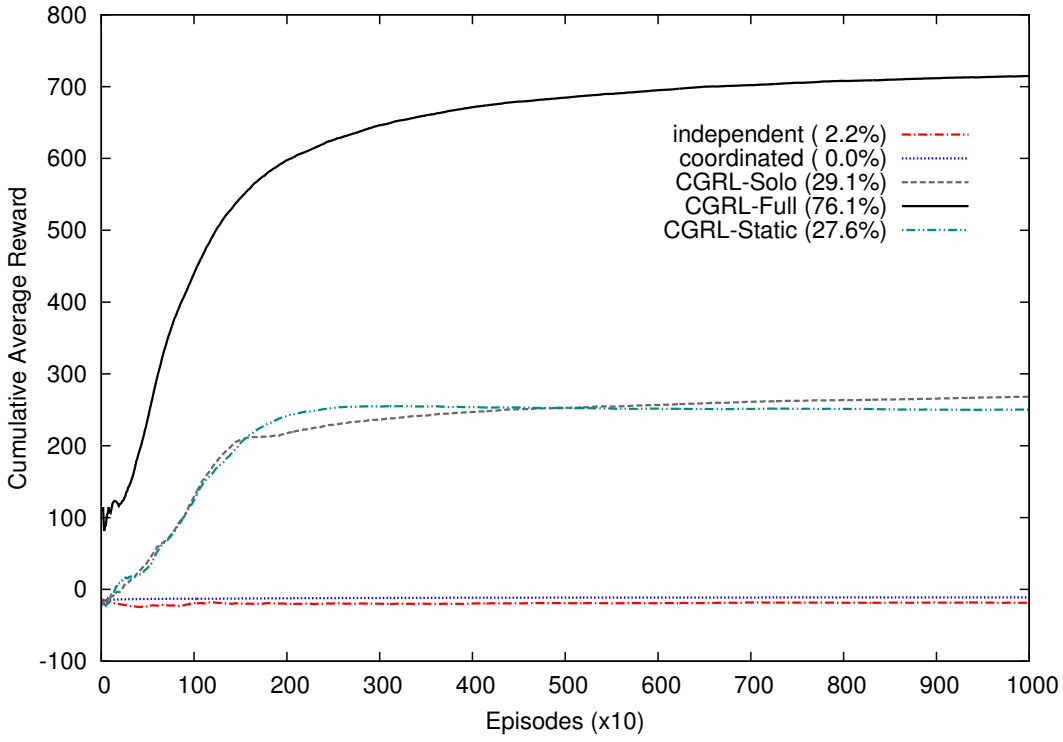


(b) Cumulative Average Reward (final win rate shown in brackets)

Figure 4.14: Centralized Exp. 3: RTS results for 10 versus 10 aggressive marines. 10,000 episodes averaged over 10 runs for each RL player.

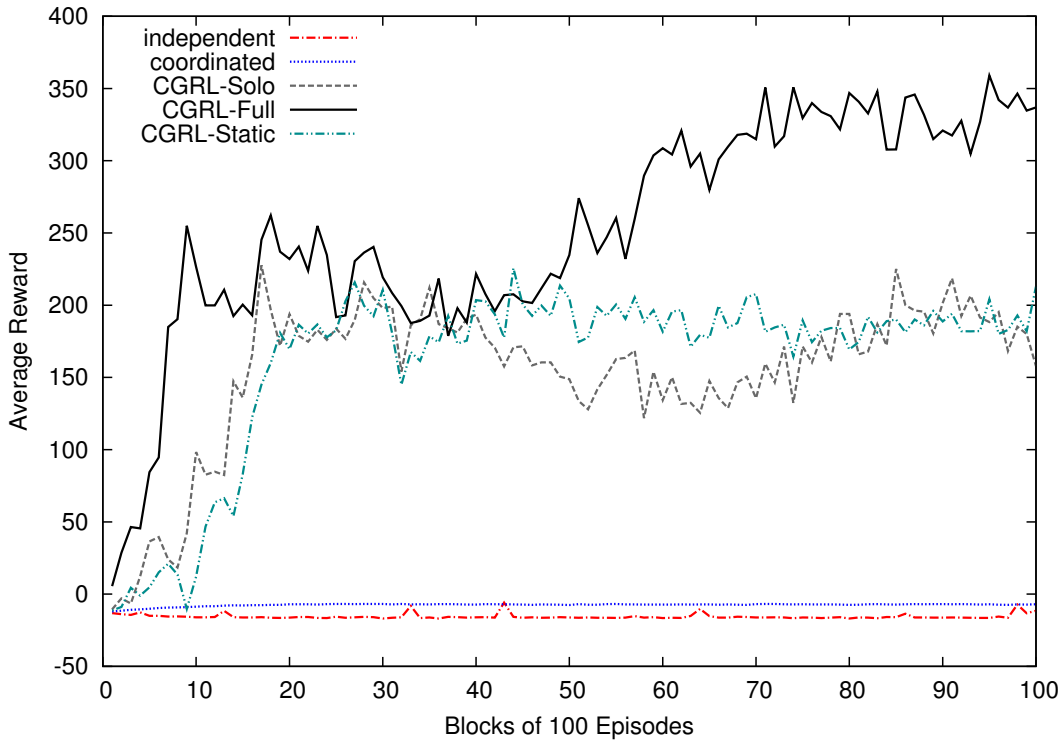


(a) Average Reward

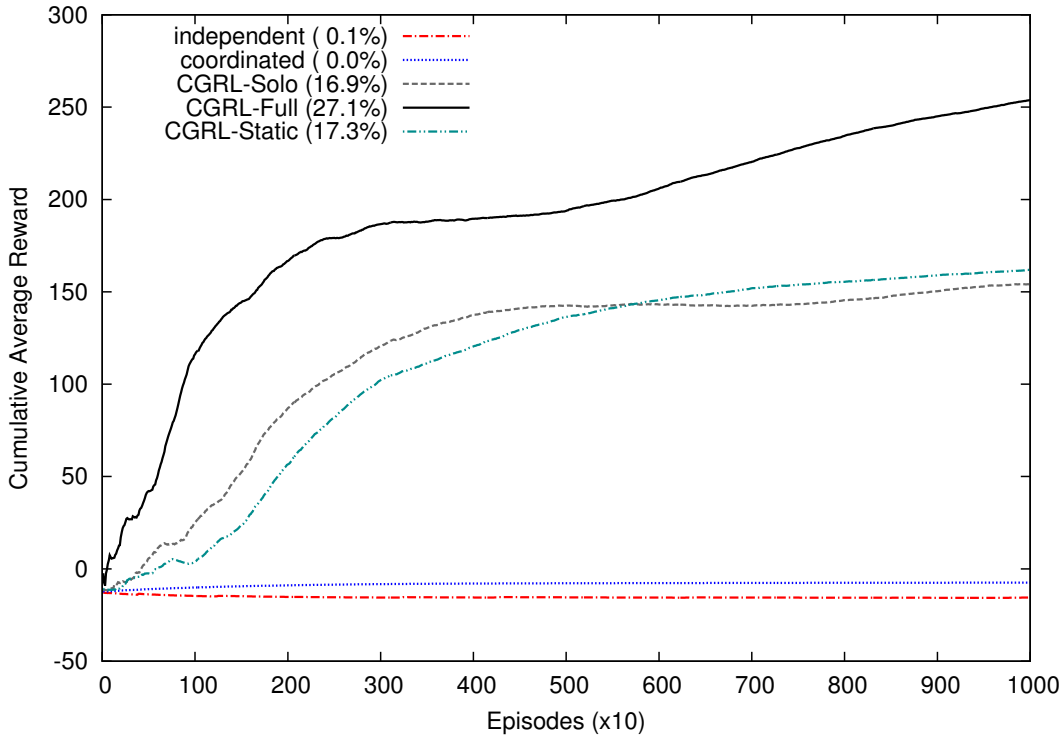


(b) Cumulative Average Reward (final win rate shown in brackets)

Figure 4.15: Centralized Exp. 3: RTS results for 10 versus 13 unpredictable marines. 10,000 episodes averaged over 10 runs for each RL player.

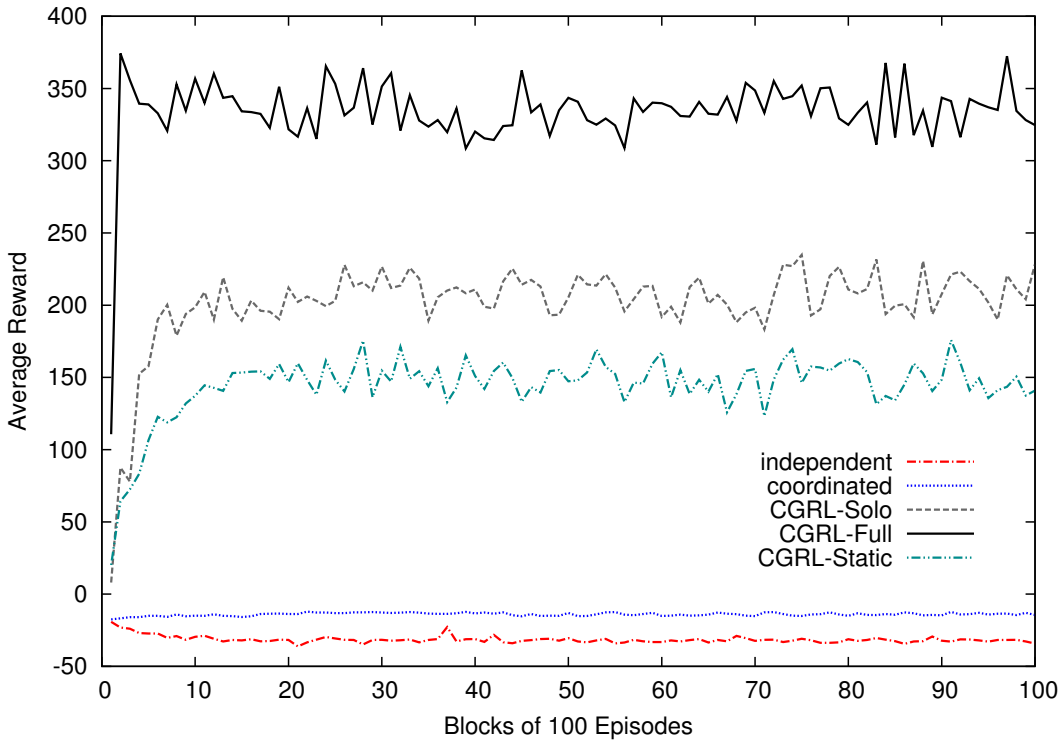


(a) Average Reward

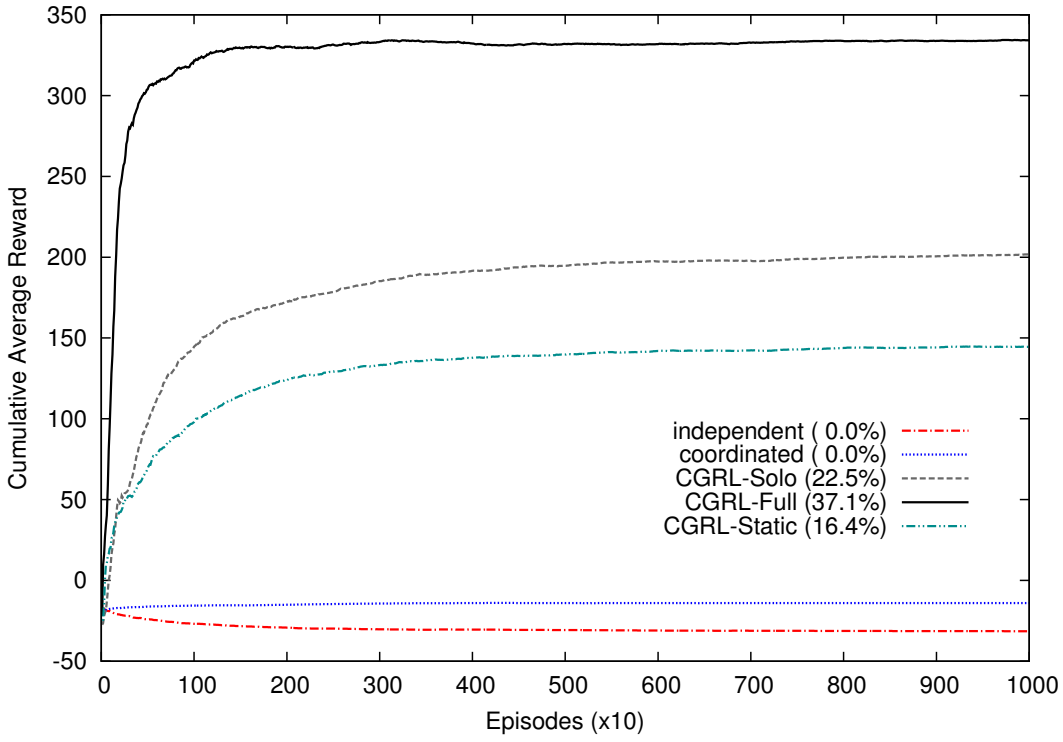


(b) Cumulative Average Reward (final win rate shown in brackets)

Figure 4.16: Centralized Exp. 3: RTS results for 10 versus 13 aggressive marines. 10,000 episodes averaged over 10 runs for each RL player.



(a) Average Reward



(b) Cumulative Average Reward (final win rate shown in brackets)

Figure 4.17: Centralized Exp. 3: RTS results for 10 versus 5 unpredictable super marines. 10,000 episodes averaged over 10 runs for each RL player.

stroyed easily when isolated. This occurs while ϵ is high and it has yet to learn a good formation. After sufficient exploration, the CGRL-Static player is mostly competitive with the CGRL-Solo player, although it learned a poorer policy in Setup 4.

On the other hand, the performance gains by the CGRL-Full player with dynamic CCs is large compared to the other players. Hence CCs are obviously crucial for tactical RTS. It is clear that most learning benefits came from the dynamic CCs. The CGRL-Full player has more coordination than the CGRL-Solo player. This is also confirmed in our video which shows the CGRL-Full player overcoming the enemy force simply by having better coordination among its marines.

4.4.7 Actual Runtime Results

The last experiment investigates runtime performance of the various RL players on a large multi-agent problem, namely the tactical RTS game. This experiment is conducted on computers with two quad-core Intel® Xeon® E5440 2.83Ghz CPUs with 12MB of L2 cache, 16GB of main memory, and running Linux. Each RL player is trained using a single process and a single thread. The CPU time is measured on the time spent within the RL players for action selection and learning (updating). Each RL player experiences a total of 10,000 episodes without prior learning. The total CPU time for these episodes are divided by the total number of time steps to give the mean CPU time per step.

Figure 4.18 shows the runtime results for the experiment of 10 versus 10 aggressive marines previously depicted in Figure 4.14. The results presented are the average of 10 runs for each RL player. The learning updates for linear function approximation runs in linear time in the number of features. Therefore, runtime is mostly dominated by joint action selection using the max-plus algorithm. This is verified by observing the low runtime of the independent RL player that selects actions independently for each agent.

Notice that the CGRL players are more time efficient than the coordinated RL player even though they have the additional top level for guiding exploration. There are two likely reasons. First, the action selection for CC activations is equivalent to independent action selection as there are no top level features that involve more than two top level

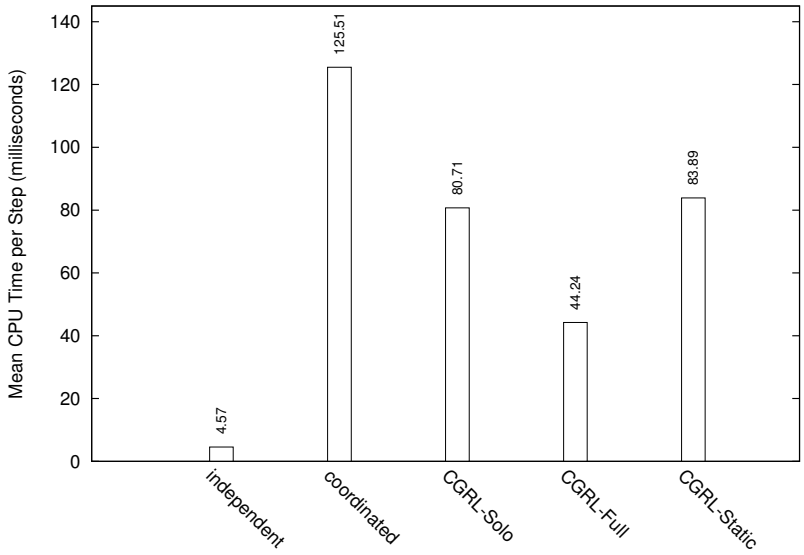


Figure 4.18: Runtime results of centralized RL players for the experiment in Figure 4.14. The mean CPU time taken per step for 10,000 episodes averaged over 10 runs.

action variables. Thus the top level does not incur a large runtime overhead during action selection. Second, the activated and static CCs allows pruning of the action space during bottom level joint action selection. Pruning is achieved via constraint propagation and forward checking described in Section 4.3.4. The CGRL-Full player's result further verifies the impact of CCs on pruning as it has additional dynamic pairwise CCs compared to CGRL-Solo and CGRL-Static. Hence CGRL is able to improve runtime performance in addition to learning performance.

4.5 Discussion

The results in the previous section indicate that allowing the RL system to guide its exploration via dynamic multi-agent CCs is effective for improving the learning rate in MDPs with large joint action spaces. The three experiments progressively incorporated approximations to deal with increasingly complex domains. Results in Section 4.4.3 demonstrate that the system is stable without any additional factors from features or approximate action selection coming into play. Throughout the experiments, the results show that CGRL retains its advantages when integrated with various methods that are commonly used to handle other essential aspects of RL.

In detail, CGRL with multi-agent CCs outperforms task based RL with coordination graphs in terms of beneficial effects from constraining the joint action space during exploration. The benefits are amplified for larger domains such as RTS where it is difficult to explore winning episodes. This is because a few marines lost early in the episode can result in a huge disadvantage such that subsequent good actions taken still result in a loss. Yet CGRL with multi-agent CCs is able to effectively explore winning episodes. This is even when some activated CCs are unenforceable as approximate action selection (max-plus) is used. The coordinated RL players based on coordination graphs had computational benefits during the selection of exploitative actions. CGRL improves on this by guiding the selection of exploratory actions in addition to the existing benefits of coordinated RL. Furthermore, the increase in number of feature weights to learn in the RTS domain scales reasonably with the use of RFs. Hence the advantage of CGRL compared with other approaches.

For all the experiments, rewards are provided only for the ultimate goal, that is to defeat the opponent in the shortest time possible. No fine-grained rewards were decomposed among agents, nor were any rewards given as sub-goals within episodes. For example, negative rewards if collision or a bad pass is made in soccer, and positive rewards if an enemy marine was destroyed in RTS. This is unlike existing work evaluated on similar domains (Guestrin et al., 2003; Marthi et al., 2005). However, CGRL was able to learn better policies than other methods without fine-grained rewards. This is an advantage as less expert knowledge for specifying rewards is required.

Compared to coordinated RL, CGRL requires the user to pick predicates to be used as CCs. This additional requirement on the user's part may appear to be a practical limitation on domains where it is difficult for the user to identify good CCs. Example 4.4 illustrated that useless CCs do not prevent good policies from being discovered but they may result in slower learning. Hence, automatically determining which predicates to use as CCs will be useful for improving practical application and is an important direction for future work.

Previous works in task based RL for multiple agents require users to define tasks,

terminating conditions, reward decomposition among tasks and agents, and new features for every level in the task hierarchy to represent them (Marthi et al., 2005; Ghavamzadeh et al., 2006; Proper and Tadepalli, 2009). They learn high level actions to constraint the exploration of primitive MDP actions based on single agent tasks and it is not straightforward to incorporate coordination among agents.

In contrast, the proposed two level RL system employs declarative CCs and allows existing predicates for features to be reused as CCs to guide itself. Encoding coordination knowledge for CCs involves defining predicates with similar semantics as those used to encode features for function approximation. Hence the user may encode knowledge without having to switch between procedural semantics and declarative semantics.

The work in Proper and Tadepalli (2009) presented a two level method where the top level assigns tasks and the bottom level learns with the task restrictions. CGRL differs as the top level explores CC activations that are defined on multiple agents, and learns two value functions that eliminates the need for a costly nested maximization when selecting CCs to activate.

The proposed CCs are distinct from methods that learn coordination structure (Kok et al., 2005) within the value functions themselves. In CGRL, CCs are used to direct exploration by specifying subsets of the joint action space to be pruned. This is dynamically learned by the top level of CGRL.

In the organization based approach of Zhang et al. (2009), fixed heuristic supervisor agents biased base agents' policies with a coarse-grained approach. In contrast, CGRL employs fine-grained RL at the top level. For example, instead of grouping bad pass CCs into one large bad pass CC that involves all soccer players, fine-grained CC over pairs of players were used so that the learning system may activate bad pass for different pairs based on the current state of those players – the exact states are learned by the top level in CGRL.

Another branch of works deal with the more restrictive multi-agent problems known as Markov games where the focus is on handling the non-stationary environment due to independent learning and the setting is usually adversarial (see Section 3.3.5 page

42). In Bianchi et al. (2007), heuristics can be provided to influence learning when the policy selects a maximal action. CCs in CGRL differ as their heuristics do not affect exploratory actions and are used in an adversarial setting with a much smaller action space.

4.6 Conclusion

This chapter investigated the use of expert coordination knowledge to improve RL for multi-agent MDPs. Expert coordination knowledge are expressed as CCs. The system's top level learns to select CCs to guide the bottom level's exploration towards better experience. The results demonstrate that the proposed two level CGRL system leads to better policies in different domains compared to existing approaches such as coordination graphs. Furthermore, RL with CCs starts off with better performance and is advantageous for online applications as overall higher goal achievement is attained. These results show that CGRL has addressed the challenge of exploring large multi-agent MDPs to a certain extent.

An important benefit of CGRL is that the additional expert knowledge required by users of the system over existing well-established coordination graph and function approximation methods is minimal. Existing predicate definitions for features can be reused to define CCs. Furthermore, similar semantics are used to defined new CCs enabling them to enrich the current set of features. The use of predicates allows further relational generalization among features. These benefits addresses the challenges of model complexity and encoding knowledge for the centralized setting.

For multi-agent domains where communication is free among agents or when the communication links among agents is fixed, CGRL can effectively address the challenges stated above. However, critical parts of the system, e.g. the top level and the learned parameters, require these communication assumptions to hold. The next chapter presents a distributed approach to CGRL that can be applied in domains where communication links between agents change dynamically.

Chapter 5

Distributed Coordination Guidance

This chapter presents a distributed version of coordination guided reinforcement learning (CGRL). The focus is on the scenario where agents can communicate with their neighbours but this communication structure and the number of agents may change over time. As before, coordination knowledge is expressed as coordination constraints (CCs) to reduce the joint action space for exploration. However, the top level, that learns to decide dynamically which CCs are useful, is decentralized among the agents. This makes it suitable for the communication restrictions. Agents communicate during action selection while learning is performed within agents using locally updated value functions. Experiment results on the soccer and tactical RTS game domains show that distributed CGRL performs better than other existing approaches for dynamic communication. Part of this work has been published in Lau et al. (2011).

5.1 Motivation

Learning in collaborative multi-agent domains may require agents to communicate to achieve the global goal. There exists a multitude of existing works for learning under different communication assumptions (Panait and Luke, 2005, Section 4). In the simplest case, there is no cost to communication and the communication structure among agents can be modelled as a connected coordination graph (CG). Agents are allowed to communicate directly through the edges of CG and as the CG is connected, all agents

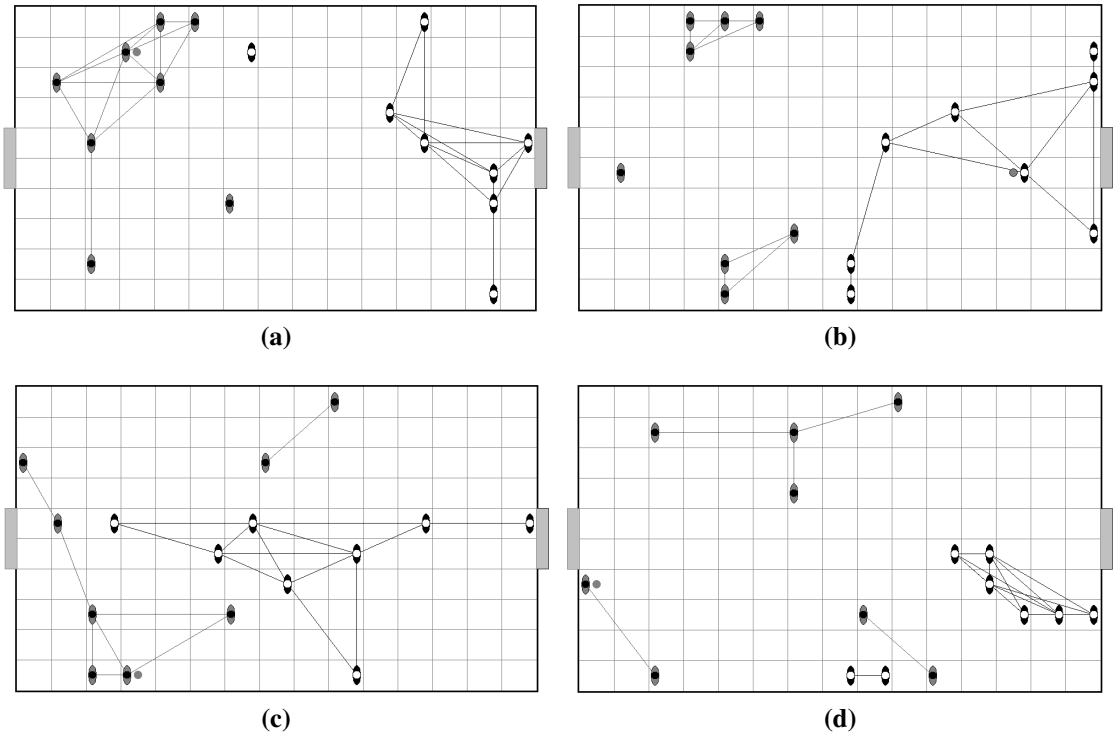


Figure 5.1: Example of dynamic communication structure in soccer (for grey and black teams) where communication restricted to agents within a Manhattan distance of 5. Lines between players indicate the communication links within the soccer team.

will be able to eventually route messages between any pair of agents. This is the multi-effector case where centralized approaches can be applied directly by modelling the problem as a factored MDP as we have done in the previous chapter. Note that although the learning problem may be centralized, distributed computation may be used. The other extreme case of communication is when there is no direct communication allowed between agents. Communication itself is an action that has to be handled as part of the learning problem.

This chapter deals with problem domains where direct communication among agents are allowed and the scenario is fully cooperative, but the communication structure changes dynamically with the state. An example for the soccer domain is given in Figure 5.1 where the lines between agents indicate the links in their communication structure. In this case, agents may communicate with different neighbouring agents at different times, forming different subgroups, i.e., disjoint CGs. Such problems can occur when agents are mobile entities that have a certain communication range or are part

of a network of computers where there may be intermittent disruption to communication links between them. Naturally, parameters to be learned have to be distributed among them. Hence critical learning components of the system should not be centralized in any one agent.

Many existing works focus on flat distributed learning in domains with varying assumptions on communication where agents learn local policies or local value functions (Claus and Boutilier, 1998; Schneider et al., 1999; Lauer and Riedmiller, 2000; Peshkin et al., 2000; Russell and Zimdars, 2003). However, few works have incorporated expert knowledge to guide learning in domains with changing communication structure. One such work previously discussed is the task based method presented in Ghavamzadeh et al. (2006) where communicating subtasks are new actions for agents to consider. Other task based approaches are centralized methods (Marthi et al., 2005; Proper and Tadepalli, 2009). Further, the organization based approaches are evaluated on fixed network problem domains (Zhang et al., 2009, 2010) where the extra agents used to supervise other agents can be pre-located somewhere within the network. These considerations constitute our motivation to devise new method to incorporate coordination knowledge in a distributed RL system to improve the learning rate for problems with dynamic communication.

5.2 Aims & Approach

The main aim of this chapter is develop a distributed learning system that may leverage on expert coordination knowledge to guide exploration during learning. To this end, we present a distributed CGRL system applicable to domains whereby agents have a dynamic communication structure. At different time steps, agents may observe the state variables and coordinate actions with the current neighbouring agents. Therefore, we focus on resolving the challenge of improving the overall learning rate of the system under limited communication (see Section 1.2.2 page 4).

Distributing coordination guidance entails three main challenges. First, as the com-

munication structure between agents is fluid, the top level of the system that learns to employ coordination knowledge in different states must not critically reside in any agent. Second, the parameters to learn must be distributed. Agents should be able to make use of any coordination they have learned when they can communicate with others, otherwise they should be able to function effectively on their own. Third, learning must not detract from the original goal. The original global reward signal of the system should be maximized.

The first and third challenges are addressed by formulating decomposed value functions to be learned. The learning updates for these value functions are local to each agent that involve a decomposed reward signal from the original signal. The second challenge is handled by local update equations for updating the local parameters of each agent's distributed value function. Then, message passing methods from distributed constraint optimization literature can be used to select joint actions that are limited by the current communication restrictions.

The distributed two level system is decentralized in the sense that learned parameters are split among the agents. Hence the loss of one agent (e.g. in an RTS game) or the separation of agents into subgroups do not render the system ineffective. The communication structure is modelled as a coordination graph among agents so that they may coordinate if they can communicate (see Section 3.2 page 31). As such the proposed approach is different from methods that learn when such coordination is (un)necessary (Kok et al., 2005). We next review the learning problem formulation for decentralization with communication before describing distributed coordination guidance.

5.3 Decentralized Markov Decision Process

The decentralized Markov decision process (DEC-MDP) formulation (Bernstein et al., 2002) is employed for the multi-agent RL system as follows. A DEC-MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ such that given N number of agents:

- $\mathcal{S} = S_0 \times S_1 \times ... \times S_N$ is the state space that is factored into state variable S_0 that

is observable by all agents and $S_1, ..., S_N$ variables that are local for each agent. A state is the tuple $\mathbf{s} = \langle s_0, s_1, ..., s_N \rangle \in \mathcal{S}$.

- $\mathcal{A} = A_1 \times ... \times A_N$ is the joint action space factored into N action variables, one for each agent. A joint action is the tuple $\mathbf{a} = \langle a_1, ..., a_N \rangle \in \mathcal{A}$.

- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the global transitional probability model, i.e., $\mathcal{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ is the probability of transiting to state $\mathbf{s}'$ after the joint action is taken in state $\mathbf{s}$.

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the global reward model that gives the reward received for taking action $\mathbf{a}$ in state $\mathbf{s}$ and reaching state $\mathbf{s}'$. $\mathcal{R}$ is decomposable into a sum of local rewards that we will present later in Equation 5.4.

Under different state values in $\mathcal{S}$, agents have a different communication structure. That is, $\mathbf{s} \in \mathcal{S}$ defines a coordination graph (CG) such that a vertex exists for each agent and an edge exists between agents that can communicate in $\mathbf{s}$. Each agent, i, may identify its set of neighbours, $\Gamma(i)$, in the CG using state values $s_0 \in S_0$ and $s_i \in S_i$ and send messages to them. Hence agents may coordinate and access the local state and action variables of their neighbours. Based on this communication restriction, we define the state tuple $\mathbf{s}_i$ and action tuple $\mathbf{a}_i$ that are accessible by agent i. The tuples $\mathbf{s}_i$ and $\mathbf{a}_i$ are projections of $\mathbf{s} \in \mathcal{S}$ and $\mathbf{a} \in \mathcal{A}$ respectively on agents in $\Gamma(i) \cup \{i\}$. This is illustrated for the action variables in Figure 5.2 for four agents with a given CG. In general, the CG may consists of disjoint sub-graphs and agents within the same sub-graph may synchronize their actions by sending messages.

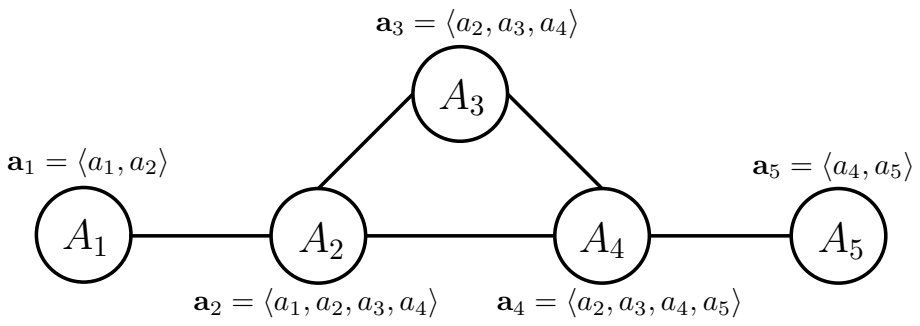


Figure 5.2: Example of the primitive action tuples accessible by each agent for a coordination graph that specifies the current communication links between agents.

A solution to the DEC-MDP is a global policy $\pi : \mathcal{S} \mapsto \mathcal{A}$. This policy can be expressed in terms of a global action value function that is the expected discounted sum of rewards, r, when taking $\mathbf{a}$ in $\mathbf{s}$ at time t and discount rate γ, i.e., $Q^{\pi}(\mathbf{s}, \mathbf{a}) = E_{\pi}\{\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} | \mathbf{s}, \mathbf{a}\}$. This is written as the global Bellman equation,

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) = \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}(\mathbf{s}' | \mathbf{s}, \mathbf{a})[\mathcal{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma Q^{\pi}(\mathbf{s}', \pi(\mathbf{s}'))] \tag{5.1}$$

Then, we may express the maximizing policy as $\pi(\mathbf{s}) = \text{argmax}_{\mathbf{a} \in \mathcal{A}(\mathbf{s})} Q^{\pi}(\mathbf{s}, \mathbf{a})$, where $\mathcal{A}(\mathbf{s}) \subseteq \mathcal{A}$ indicates that certain actions may be disallowed in the current state. To learn an optimal policy π^* we need only to learn the optimal action value function Q^* directly, without actually learning the model $\mathcal{P}$ (Section 2.2.1 page 17). This is useful in the case of DEC-MDP as it frees the user from defining a decomposable transition model for $\mathcal{P}$ which can be difficult for multi-agent problems.

For distribution, the global policy π needs to be expressed as local parts, one for each agent. This can be done by decomposing Q^{π} into a sum of local components, Q_i^{π}, i.e.,

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) = \sum_{i=1}^{N} Q_i^{\pi}(\mathbf{s}_i, \mathbf{a}_i) \tag{5.2}$$

where $\mathbf{s}_i, \mathbf{a}_i$ contain the state and action values of agent i and its current neighbours. Because the CG is not static, in general the domain of Q_i^{π} may include all the possible projections of $\mathbf{s}$ and $\mathbf{a}$. For example, if agents are close enough such that the CG is fully connected, $\mathbf{a}_i = \mathbf{a}$, and if they are far apart such that none may communicate, then $\mathbf{a}_i = a_i$.

The global policy can be learned by updating local value functions Q_i^{π} using the local interactions with the environment (Russell and Zimdars, 2003; Kok and Vlassis, 2004),

$$Q_i^{\pi}(\mathbf{s}_i, \mathbf{a}_i) \leftarrow Q_i^{\pi}(\mathbf{s}_i, \mathbf{a}_i) + \alpha[r_i + \gamma Q_i^{\pi}(\mathbf{s}_i', \mathbf{a}_i') - Q_i^{\pi}(\mathbf{s}_i, \mathbf{a}_i)] \tag{5.3}$$

where $\mathbf{a}'_i$ is a projection on the global joint action $\mathbf{a}'$ chosen from the global policy π, α is the step size parameter, and $r_i = R_i(\mathbf{s}_i, \mathbf{a}_i)$ is the reward observed by agent i such that the global reward is written as,

$$\mathcal{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \sum_{i=1}^{N} R_i(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i). \tag{5.4}$$

Note that in our problem formulation, the CG is not fixed. This indicates that the state and action variables in $\mathbf{s}'_i, \mathbf{a}'_i$ may not correspond to those in the previous time step, $\mathbf{s}_i, \mathbf{a}_i$. We will explain how this is handled as we describe our system next.

5.4 Distributed two level System

A conceptual overview of the proposed distributed CGRL (DistCGRL) system architecture is shown in Figure 5.3 for four hypothetical agents, $A_1, ..., A_4$, that are able to

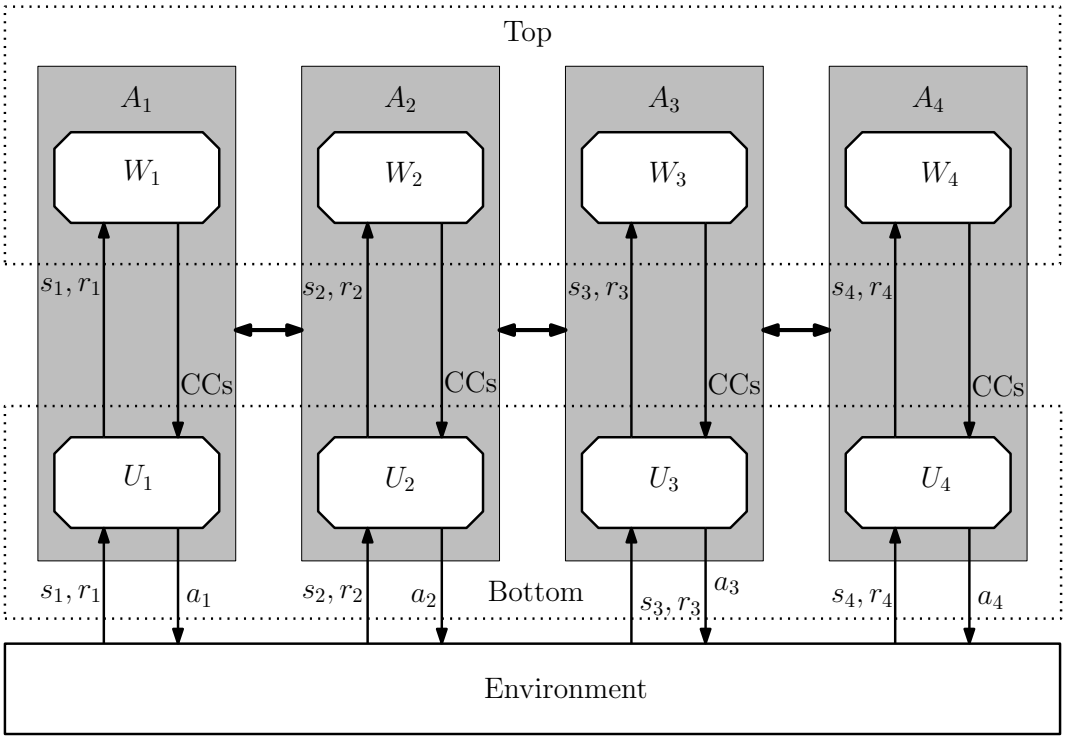


Figure 5.3: Conceptual architecture of the DistCGRL system with four agents. Grey boxes are the agent boundaries. Within each agent, two white boxes indicate its learning components. Those at the top are grouped as the top level and similarly for the bottom. For each agent, top level components observe the same state and reward as the bottom level.

communicate with their adjacent agents (double headed arrows). Each white box represents a learning component, grouped into global top and bottom level components (dotted boxes). At each time step, each agent's top level component decides on the coordination constraints (CCs) to *activate* for the bottom level component. These CCs may be communicated between the agents involved in them. Then, the bottom level component takes into account the CCs in selecting the actual joint action $\langle a_1, a_2, a_3, a_4 \rangle$ to be taken in the environment. Both levels' components observe the next local state and local reward and update themselves internally.

The concept of CCs and their close relationship with predicates used to define propositional features for function approximation was previously discussed in Section 4.3.5 page 80. We proceed to describe their use in DEC-MDPs for the rest of this section.

5.4.1 Augmented DEC-MDP

As with the centralized CGRL in the previous chapter (see Section 4.3.1 page 56), the DEC-MDP is augmented with the choices of CCs as the top level actions. Let there be K number of CCs. Then the top level joint action space is $\mathcal{A}_0$ such that an action $\mathbf{b} \in \mathcal{A}_0$ is the tuple $\langle b_1, b_2, ..., b_K \rangle$. Each variable in the tuple has the domain $\{0, 1\}$ that represents deactivation or activation of the CCs. Then, the augmented DEC-MDP is the tuple $\langle \mathcal{S}', \mathcal{A}', \mathcal{P}', \mathcal{R}' \rangle$ such that,

- $\mathcal{S}' = \mathcal{A}_\Theta \times \mathcal{S}$ is the state space, where $\mathcal{A}_\Theta = \mathcal{A}_0 \cup \{\Theta\}$ and Θ is a dummy constant value that indicates that the system is in the top level. A state is written as $\langle \mathbf{b}, \mathbf{s} \rangle \in \mathcal{S}'$.
- $\mathcal{A}' = \mathcal{A}_\Theta \cup \mathcal{A}$ is the action space that includes both top and bottom level actions.
- $\mathcal{R}' : \mathcal{S}' \times \mathcal{A}' \times \mathcal{S}' \mapsto \mathbb{R}$ is the reward model that like in Equation 4.1 page 56, it is zero for all top to bottom state transitions and otherwise, based on the original DEC-MDP's $\mathcal{R}$ where each agent i receives r_i from the environment at each time step.
- $\mathcal{P}' : \mathcal{S}' \times \mathcal{A}' \mapsto \mathcal{S}'$ is the transition probability model. Similar to the augmented

MDP (see Equation 4.2 page 57), it is deterministic when transiting from top to bottom level otherwise it is the same as the original DEC-MDP's $\mathcal{P}$.

The solution to the augmented DEC-MDP is the two level global policy written in two parts: a top policy and bottom policy, i.e.,

$$\psi'(\langle \mathbf{b}, \mathbf{s} \rangle) = \begin{cases} \psi(\Theta, \mathbf{s}) & \text{if } \mathbf{b} = \Theta \\ \psi(\mathbf{b}, \mathbf{s}) & \text{otherwise, i.e., } \mathbf{b} \in \mathcal{A}_0 \end{cases} \tag{5.5}$$

where $\psi(\Theta, \mathbf{s}) \in \mathcal{A}_0$ and $\psi(\mathbf{b}, \mathbf{s}) \in \mathcal{A}_{\mathbf{b}}$. The bottom level policy $\psi(\mathbf{b}, \cdot)$ only selects actions in the primitive joint action space constrained by $\mathbf{b}$ in state $\mathbf{s}$. As was in the centralized case, the control policy used by the agents acting in the environment is given by,

$$\pi(\mathbf{s}) = \psi(\psi(\Theta, \mathbf{s}), \mathbf{s}) \tag{5.6}$$

which indicates: first choosing a top level action and then, choosing a bottom level primitive action. The global top and bottom policies correspond to the dotted boxes in Figure 5.3. However, in the DEC-MDP case, the global policy consists of local parts that are distributed among agents. We describe the value functions for the policies next.

5.4.2 Value Functions Within Agents

The solution to the DEC-MDP is represented by a set of local policies that together constitute the global policy. To decompose the global policies, we use the approach of decomposing their value functions among agents as previously described in Equation 5.2, but for both levels of the overall multi-agent learning system.

Let $\mathbf{b}_i$ be the top level action tuple for agent i that is a projection on $\mathbf{b}$. Unlike the primitive joint actions where one domain A_i exists for each agent i, the individual top level actions b_k from $\mathbf{b} = \langle b_1, b_2, ..., b_K \rangle$ may be related to multiple agents. Hence $\mathbf{b}_i$ is the action tuple $\langle b_{i,1}, ..., b_{i,n} \rangle$ such that the n number of CCs corresponding to

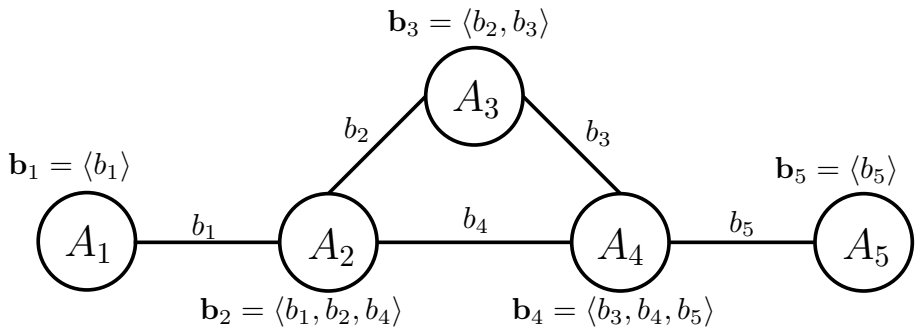


Figure 5.4: Example of top level action tuples accessible by each agent. There are 5 pairwise CCs corresponding to $b_1, ..., b_5$. Each agent may access CCs that they are involved in and that involves a current neighbour in the CG.

$b_{i,1}, ..., b_{i,n}$ involve only agent i and its current neighbours in $\Gamma(i)$. We illustrate this with an example.

Example 5.1 (Distributing top level actions)**.** *Consider Figure 5.4 with the same four agents and communication links as Figure 5.2. Suppose there are only CCs involving pairs of agents. These CCs correspond to the edges in the CG shown in the graph. The top level action tuples* $\mathbf{b}_1, ..., \mathbf{b}_5$ *that each agent may access are shown at each agent vertex of the graph. There may be other pairwise CCs, for example, one for agents* 1 *and* 3*. However, because these two agents are not neighbours, these CCs are not part of the accessible top level actions in the current state.*

The global action value functions for the top and bottom policies are given by the Bellman equations,

$$W(\mathbf{s}, \mathbf{b}) = \sum_{\mathbf{s}'} \mathcal{P}(\mathbf{s}'|\mathbf{s}, \psi(\mathbf{b}, \mathbf{s}))[\mathcal{R}(\mathbf{s}, \psi(\mathbf{b}, \mathbf{s}), \mathbf{s}') + \gamma W(\mathbf{s}', \psi(\Theta, \mathbf{s}'))] \tag{5.7}$$

$$U(\mathbf{s}, \mathbf{a}) = \sum_{\mathbf{s}'} \mathcal{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a})[\mathcal{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma W(\mathbf{s}', \psi(\Theta, \mathbf{s}'))] \tag{5.8}$$

that correspond to Equations 4.56 and 4.57 on page 67 respectively[1]. The functions are in turn additively decomposed into distributed functions, one for each of the N agents,

$$W(\mathbf{s}, \mathbf{b}) = \sum_{i=1}^{N} W_i(\mathbf{s}_i, \mathbf{b}_i) \tag{5.9}$$

[1]We have omitted the two level policy ψ from the superscripts of these functions.

$$U(\mathbf{s}, \mathbf{a}) = \sum_{i=1}^{N} U_i(\mathbf{s}_i, \mathbf{a}_i). \tag{5.10}$$

Based on these agent decompositions, each agent i carries its share of the learned W_i and U_i functions within themselves as shown in Figure 5.3 where the grey boxes indicate the boundaries of individual agents. Next we discuss distributed control.

5.4.3 Distributed Control

Localized Policies

The top and bottom levels' policies are represented by local parts such that each group of agents that belong in the same connected CG will participate in selecting actions for that group. In the extreme case where all agents can communicate in the current state, this is the same as the centralized case. A policy that selects the maximum action will maximize over all variables in the action value function. For the more general case where the CG is disjoint, only sub-groups of agents will be maximized as shown in the example below.

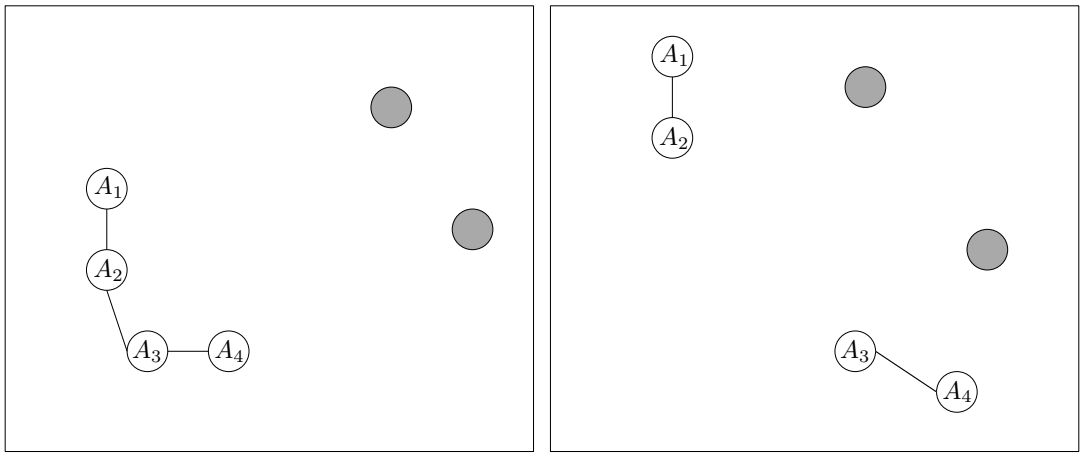
Example 5.2 (Localized policies)**.** *Figure 5.5 depicts, in simplified terms, states in an RTS game with the white team consisting of four marines and the grey team of two. In Figure 5.5a the marines are fully connected hence the entire sum of local functions, W and U, are used to select joint actions for agents 1 to 4. However, in the state shown in Figure 5.5b, the marines have moved apart into two groups. Assuming we are selecting the maximal action and CCs between pairs of agents, for the top level action,*

$$\psi(\Theta, \mathbf{b}) = \operatorname*{argmax}_{\mathbf{b} \in \mathcal{A}_0} W(\mathbf{s}, \mathbf{b}) = \operatorname*{argmax}_{\mathbf{b} \in \mathcal{A}_0} \sum_{i=1}^{4} W_i(\mathbf{s}_i, \mathbf{b}_i) = \langle b^*_{1,2}, b^*_{3,4} \rangle$$

where we can separately maximize in two parts,

$$b^*_{1,2} = \operatorname*{argmax}_{b_{1,2}} W_1(\mathbf{s}_1, b_{1,2}) + W_2(\mathbf{s}_2, b_{1,2}),$$

$$b^*_{3,4} = \operatorname*{argmax}_{b_{3,4}} W_3(\mathbf{s}_3, b_{3,4}) + W_4(\mathbf{s}_4, b_{3,4}),$$



(a) Connected **(b)** Disjoint

Figure 5.5: Example of policies as local parts for a simplified tactical RTS game. Communication links shown as lines between marines. The four marines (agents) in the white team in the state depicted in (a) is fully connected. In (b) the marines are in two disjoint sub-graphs.

that are the maximal top level actions for (de)activating the CCs between the pairs of agents. Similarly for the maximal primitive action we have,

$$\psi(\mathbf{b}^*, \mathbf{s}) = \underset{\mathbf{a}\in\mathcal{A}_{\mathbf{b}^*}}{\operatorname{argmax}} U(\mathbf{s}, \mathbf{a}) = \underset{\mathbf{a}\in\mathcal{A}_{\mathbf{b}^*}}{\operatorname{argmax}} \sum_{i=1}^{4} U_i(\mathbf{s}_i, \mathbf{a}_i) = \langle a_1^*, a_2^*, a_3^*, a_4^* \rangle$$

where $\mathbf{b}^* = \langle b_{1,2}^*, b_{3,4}^* \rangle$*, such that we separately maximize,*

$$\langle a_1^*, a_2^* \rangle = \underset{\langle a_1, a_2\rangle \in \mathcal{A}_{b_{1,2}^*}^{1,2}}{\operatorname{argmax}} U_1(\mathbf{s}_1, a_1, a_2) + U_2(\mathbf{s}_2, a_1, a_2),$$

$$\langle a_3^*, a_4^* \rangle = \underset{\langle a_3, a_4\rangle \in \mathcal{A}_{b_{3,4}^*}^{3,4}}{\operatorname{argmax}} U_3(\mathbf{s}_3, a_3, a_4) + U_4(\mathbf{s}_4, a_3, a_4),$$

where $\mathcal{A}_{b_{i,j}^*}^{i,j} = A_i \times A_j$ *subjected to the (de)activation of the CC,* $b_{i,j}^*$*.*

Example 5.2 demonstrates how the local action value functions within agents for the top and bottom levels may be used to localize the policies for control. In general, agents within the same connected CG may synchronize their local policies. For example, if ϵ-greedy policies are used, the agents in the same group may choose to select a random exploratory joint action or exploit the current value functions. Note that the decomposed functions W_i and U_i may in turn be further decomposed into sub-components using

features for function approximation.

Computing Joint Actions

We now turn to how the selection of joint actions are computed. Suppose the top level actions have already been selected, i.e., the CCs have been activated. Computing the maximal bottom level action is to maximize the objective function[2] for a COP,

$$\psi(\mathbf{b}, \mathbf{s}) = \underset{\mathbf{a} \in \mathcal{A}_{\mathbf{b}}(\mathbf{s})}{\operatorname{argmax}} U(\mathbf{s}, \mathbf{a}) = \underset{\mathbf{a} \in \mathcal{A}}{\operatorname{argmax}} g_{\mathbf{b}}(\mathbf{s}, \mathbf{a}) \tag{5.11}$$

where $\mathcal{A}_{\mathbf{b}}(\mathbf{s})$ is the joint action space constrained by $\mathbf{b}$ in $\mathbf{s}$, and the objective function, $g_{\mathbf{b}}$, that enforces these constraints is,

$$g_{\mathbf{b}}(\mathbf{s}, \mathbf{a}) = U(\mathbf{s}, \mathbf{a}) + C_0(\mathbf{s}, \mathbf{a}) + C_{\mathbf{b}}(\mathbf{s}, \mathbf{a}) \tag{5.12}$$

$$= \sum_{i=0}^{N} [U_i(\mathbf{s}_i, \mathbf{a}_i) + C_{0_i}(\mathbf{s}_i, \mathbf{a}_i) + C_{\mathbf{b}_i}(\mathbf{s}_i, \mathbf{a}_i)] \tag{5.13}$$

$$= \sum_{i=0}^{N} g_{\mathbf{b}_i}(\mathbf{s}_i, \mathbf{a}_i), \tag{5.14}$$

where C_{0_i} are the sum of static CCs involving $\mathbf{s}_i, \mathbf{a}_i$ from an agent's neighbours, and $C_{\mathbf{b}_i}$ is the sum of dynamic CCs activated by the top level action tuple $\mathbf{b}_i$ accessible by each agent in the state $\mathbf{s}$. The CCs return $-\infty$ if violated or zero otherwise. As described in Example 5.2, the local $g_{\mathbf{b}_i}$ functions for each agent i are summed for agents that are connected in the CG for finding the maximal action.

COPs using the the local functions $g_{\mathbf{b}_i}$ can be solved by distributed COP methods. This typically involves message passing among neighbours. If each C_{0_i}, $C_{\mathbf{b}_i}$, and U_i can be additively decomposed into components involving up to two agents in their scope, the max-plus algorithm described in Section 4.3.4 page 77 can be used directly on each disjoint CG with minor adaptations (Kok and Vlassis, 2006). Otherwise, for components involving more agents, the max-sum method can be used (Stranders et al., 2010). These methods are approximate for CG with cycles, hence some CCs may not

[2] See also the centralized case in Equation 4.70 page 74.

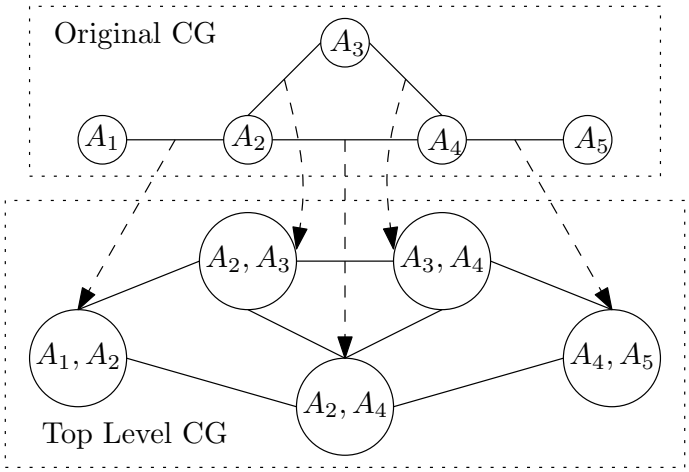


Figure 5.6: Example of the top level coordination graph derived from the original.

be enforceable. Nevertheless they have been show to be effective in practice.

Next, we deal with the question of who makes the decisions for CCs that involve multiple agents. Consider Figure 5.6 where each agent is represented as a vertex in the original CG from Figure 5.4. Suppose that each edge corresponds to a pairwise CC. Then, each vertex in the top level CG corresponds to the top level action variable for one CC. The edges in the top level CG indicate that there exists some pairwise component of the top level function W that involves two CCs. Hence, a similar agent decomposable objective function as Equation 5.12 can be used to compute the top level action, $\psi(\Theta, \mathbf{s})$.

For each vertex of the top level CG, we use a scheme where one predetermined agent in the vertex computes the activation of the CC while the other agents relay their local values of W_i that is relevant to the CC as well as messages from their neighbouring agents. With this scheme, the described distributed COP solutions can be used to compute $\psi(\Theta, \mathbf{s})$ using local W_i values. Note that the restriction that only neighbouring agents may communicate still holds. If no component in W exists that is defined on multiple top level action variables, the top level CG has no edges and the activation of each CC can be selected independently by each group of involved agents. This turns out to be effective as shown by the experiment results in Section 5.5.

Finally, CCs that are defined on agents currently out of communicable distance always remain deactivated. This is because agents will not be able to ensure that the

CCs will be enforced between them. Hence the action variables that represent CCs in the local top level action $\mathbf{b}_i$ for agent i will only be present only if the agents that each CC refers to are current neighbours of the agent. This was shown in Figure 5.4.

5.4.4 Local Function Representation & Updating

Representing the local value functions W_i and U_i for each agent i with tabular functions in the general case results in poor scalability. This is because when agents are close enough, the tables require storage space that is of size exponential in number of agents. In practice, we represent each agent's value functions using linear function approximation. Features that involve multiple agents may be scaled according to the number of involved agents. The next example shows how propositional features (PFs) from predicates may be divided among agents.

Example 5.3 (Local value functions)**.** *Suppose we have three agents, 1, 2, and 3. Three CCs are defined for pairs of agents, i.e., $b_{1,2}, b_{1,3}, b_{2,3}$. Given the predicate ρ that may be bound to two agents i, j giving propositions to represent the value functions, the top level functions W_i may be represented as,*

$$W_1(\mathbf{s}_1, \mathbf{b}_1) = w_{1,2}\overbrace{(0.5)\rho(s_1, s_2, b_{1,2})}^{\textit{local feature}} + w_{1,3}\overbrace{(0.5)\rho(s_1, s_3, b_{1,3})}^{\textit{local feature}}$$

$$W_2(\mathbf{s}_2, \mathbf{b}_2) = w_{2,1}(0.5)\rho(s_1, s_2, b_{1,2}) + w_{2,3}(0.5)\rho(s_2, s_3, b_{2,3})$$

$$W_3(\mathbf{s}_3, \mathbf{b}_3) = w_{3,1}(0.5)\rho(s_1, s_3, b_{1,3}) + w_{3,2}(0.5)\rho(s_2, s_3, b_{2,3}).$$

where $w_{i,j}$ are weights to be learned and 0.5 normalizes the pairwise propositions as each pairwise proposition will appear twice in the global top value function. Hence, when the above are summed, we obtain

$$\begin{aligned} &W_1(\mathbf{s}_1, \mathbf{b}_1) + W_2(\mathbf{s}_2, \mathbf{b}_2) + W_3(\mathbf{s}_3, \mathbf{b}_3) \\ &= \frac{w_{1,2} + w_{2,1}}{2}\rho(s_1, s_2, b_{1,2}) + \frac{w_{1,3} + w_{3,1}}{2}\rho(s_1, s_3, b_{1,3}) + \frac{w_{2,3} + w_{3,2}}{2}\rho(s_2, s_3, b_{2,3}) \\ &= W(\mathbf{s}, \mathbf{b}) \end{aligned}$$

that is the joint top level value function. The predicate $\rho(s_i, s_j, b_{i,j})$ *is zero if the agents* i *and* j *are too far apart to communicate in* $\mathbf{s}$.

The above example showed how PFs can be divided among individual agent's top level functions. The bottom level value functions U_i can also be approximated in the same way. A similar form of function approximation using value rules for the flat learning of the action value function was also presented in Kok and Vlassis (2004). If only features that involve up to two agents are used, the number of parameters (weights) that each agent needs to learn and store scales linearly with the number of agents in the problem. Hence agents do not need to update tables of exponential size.

Now, we update each local value function with the local temporal difference error using the local reward. Let the local value functions be linearly approximated as $W_i(\mathbf{s}_i, \mathbf{b}_i) = \vec{w}_{W_i} \cdot \vec{f}_{W_i,\mathbf{s}_i,\mathbf{b}_i}$ and $U_i(\mathbf{s}_i, \mathbf{a}_i) = \vec{w}_{U_i} \cdot \vec{f}_{U_i,\mathbf{s}_i,\mathbf{a}_i}$ where $\vec{w}_{W_i}$ and $\vec{w}_{U_i}$ are local weights and $\vec{f}_{F,\mathbf{s}_i,\mathbf{b}_i} = \langle f_1(\mathbf{s}_i, \mathbf{b}_i), ..., f_n(\mathbf{s}_i, \mathbf{b}_i)\rangle$ are feature values for an approximated function, F. The local update equations for both levels are,

$$\vec{w}_{W_i} \leftarrow \vec{w}_{W_i} + \alpha[r_i + \gamma W_i(\mathbf{s}'_i, \mathbf{b}'_i) - W_i(\mathbf{s}_i, \mathbf{b}_i)]\vec{f}_{W_i,\mathbf{s}_i,\mathbf{b}_i} \tag{5.15}$$

$$\vec{w}_{U_i} \leftarrow \vec{w}_{U_i} + \alpha[r_i + \gamma W_i(\mathbf{s}'_i, \mathbf{b}'_i) - U_i(\mathbf{s}_i, \mathbf{a}_i)]\vec{f}_{U_i,\mathbf{s}_i,\mathbf{a}_i} \tag{5.16}$$

where $\mathbf{b}'_i$ is the top level joint action of agent i and its neighbours projected from the global top level action, $\mathbf{b}' = \psi(\Theta, \mathbf{s}')$. We use ϵ-greedy policies for ψ where an exploratory action is chosen with ϵ probability and a maximal action with $1-\epsilon$ probability. These policies are GLIE when ϵ decreases over time. Note that we have not used the primitive action $\mathbf{a}'$ selected for $\mathbf{s}'$ by the policy as each W_i already encodes the current local estimate of the expected return.

The variables where the values in $\mathbf{s}_i$, $\mathbf{a}_i$, and $\mathbf{b}_i$ come from may be different in two consecutive states as the agents' neighbours may have changed. To handle this problem of dynamic communication changes, we only require each feature f_j to return *zero* when there exists variables in the $scope(f_j)$[3] that do not have corresponding values present in

[3]The scope of a function is the set of variables from its domain

$\mathbf{s}_i$, $\mathbf{a}_i$, or $\mathbf{b}_i$. Hence, the weights in Equations 5.15 and 5.16 that involve more than one agent will only be updated for the duration in which the agents remain as neighbours.

5.4.5 Agent's Algorithm

The DistCGRL algorithm executed by each agent is given in Algorithm 5.1. At each time step, each agent observes their state s_i and communicates it to their neighbouring agents during action selection in Lines 3, 11, 16, and 24. In the process, the

Algorithm 5.1 DistCGRL algorithm for one agent.

Remarks: Values $\mathbf{b}_i, \mathbf{a}_i, \mathbf{s}_i$ are from state and action variables of the current neighbours. Value a_i is the primitive action for agent i. Local state $\mathbf{s}_i$ is terminal if episode has ended or agent i is removed from the episode.

1: Observe initial state s_i
2: **if** $appointed(i)$ **then** # agent appointed to choose some top level action
3: $\mathbf{b}_i \leftarrow \psi(\Theta, \mathbf{s})$ # select top action using distributed COP with W_i
4: Send $\mathbf{b}_i$
5: **else**
6: Relay messages, receive $\mathbf{b}_i$
7: **end if**
8: Receive $\mathbf{s}_i$ as a consequence of the previous step.
9: $\vec{w}'_{U_i} \leftarrow \vec{w}_{U_i}$ # backup weights
10: Set weights in $\vec{w}_{U_i}$ to $-\infty$ if activated in $\mathbf{b}'_i$.
11: $\mathbf{a}_i \leftarrow \psi(\mathbf{b}, \mathbf{s})$ # select primitive action using distributed COP with U_i
12: $\vec{w}_{U_i} \leftarrow \vec{w}'_{U_i}$ # restore weights
13: **while** $\neg terminal(\mathbf{s}_i)$ **do**
14: Take action a_i, observe reward r_i and state s'_i
15: **if** $appointed(i)$ **then** # agent appointed to choose some top level action
16: $\mathbf{b}'_i \leftarrow \psi(\Theta, \mathbf{s}')$ # select top action
17: Send $\mathbf{b}_i$
18: **else**
19: Relay messages, receive $\mathbf{b}_i$
20: **end if**
21: Receive $\mathbf{s}'_i$ as a consequence of the previous step.
22: $\vec{w}'_{U_i} \leftarrow \vec{w}_{U_i}$ # backup weights
23: Set weights in $\vec{w}_{U_i}$ to $-\infty$ if activated in $\mathbf{b}'_i$.
24: $\mathbf{a}'_i \leftarrow \psi(\mathbf{b}', \mathbf{s}')$ # select primitive action
25: $target \leftarrow r_i + \gamma W_i(\mathbf{s}'_i, \mathbf{b}'_i)$ # local agent updates
26: $\vec{w}_{W_i} \leftarrow \vec{w}_{W_i} + \alpha[target - W_i(\mathbf{s}_i, \mathbf{b}_i)]\vec{f}_{W_i,\mathbf{s}_i,\mathbf{b}_i}$
27: $\vec{w}_{U_i} \leftarrow \vec{w}'_{U_i}$ # restore weights
28: $\vec{w}_{U_i} \leftarrow \vec{w}_{U_i} + \alpha[target - U_i(\mathbf{s}_i, \mathbf{a}_i)]\vec{f}_{U_i,\mathbf{s}_i,\mathbf{a}_i}$
29: $\mathbf{s}_i \leftarrow \mathbf{s}'_i$; $\mathbf{b}_i \leftarrow \mathbf{b}'_i$; $\mathbf{a}_i \leftarrow \mathbf{a}'_i$
30: **end while**

agents receive the joint actions $\mathbf{b}_i$ and $\mathbf{a}_i$. For the top level action selection in Lines 3 and 16, agents appointed to select CCs on behalf of other neighbouring agents selects the CCs while other agents relay messages from other appointed agents and also their top level component functions that involve the CC. The target to update towards (Line 25), computed at each time step, uses only the local reward experienced by the agent. Furthermore, no more communication is necessary during updating at Lines 25–28 as the joint state and actions of each agent and its neighbours have been received during action selection.

Like in centralized CGRL (see Section 4.3.5 page 80), the CCs may be represented by PFs (i.e., bound predicates). Their effects on the distributed COP's objective function (Equation 5.12) can be achieved by setting the weights of activated CCs to $-\infty$ (at Line 22) and restoring their weights during updating (at Line 27). This setting of weights does not require communication since the agents know which CCs are activated during action selection.

5.5 Experiments with Dynamic Communication

Two domains of simplified soccer and tactical RTS are used to evaluate our proposed approach. Communication is limited to a certain threshold distance between agents. We plot average global reward for blocks of episodes to visualize the quality of the global policy achieved by the local value functions. We also plot the cumulative average global reward to visualize overall goal achievement in the online setting. An important point to note is that all plots start from the point *one* and not at zero. Depending on the scale of each individual plot, this may refer to a different number of initial episodes.

5.5.1 Agent Setup

We compare four types of RL agents that use linear function approximation for their value functions.

1. *Independent learners* – each agent learns its own value function without any communication and select their actions individually (Claus and Boutilier, 1998).
2. *Coordinated learners* – each agent has a decomposed value function and performs on-policy updates using Equation 5.3 with a similar gradient as Equation 5.16. This is akin to the methods presented in Russell and Zimdars (2003); Kok and Vlassis (2004).
3. *DistCGRL-Solo learners* – each agent uses two level learning to select CCs to guide exploration. This is a representative of the distributed task based methods to compare the effect of guiding exploration using only individual agent constraints.
4. *DistCGRL-Full learners* – each agent uses the full DistCGRL method with all CCs.

For both soccer and RTS domains, the independent learner uses features defined on single agents. Coordinated learners use single and pairwise features that are the same as those used in the bottom level value functions of DistCGRL. No relational features are used as learning parameters are distributed among agents. Hence we only use PFs in conjunction with other normal features. Each agent uses only features described in Appendix A page 229 that it is involved in to represent its local value function. Distributed joint action selection is performed using the max-plus algorithm limited to 10 iterations. RL parameters are set based on the same considerations outlined in Section 4.4.1 page 88.

5.5.2 Results For Soccer

The soccer domain used the same game dynamics, scripted defensive, and aggressive opponents as described in Section 4.4.2 page 90. However, we use a larger variant of the game previously shown in Figure 5.1. The soccer field is a grid world of 15×10 units. Each team has 8 players. Players may communicate within a Manhattan distance of 5 as shown by the lines between players in the various states in Figure 5.1. This is different from the centralized soccer experiments where the entire team is part of the same CG in most states.

RL player	Weights	Total CCs	Static CCs	Dynamic CCs
Independent	1888	16	16	0
Coordinated	3120	16	16	0
DistCGRL-Solo	4016	72	16	56
DistCGRL-Full	7040	212	72	140

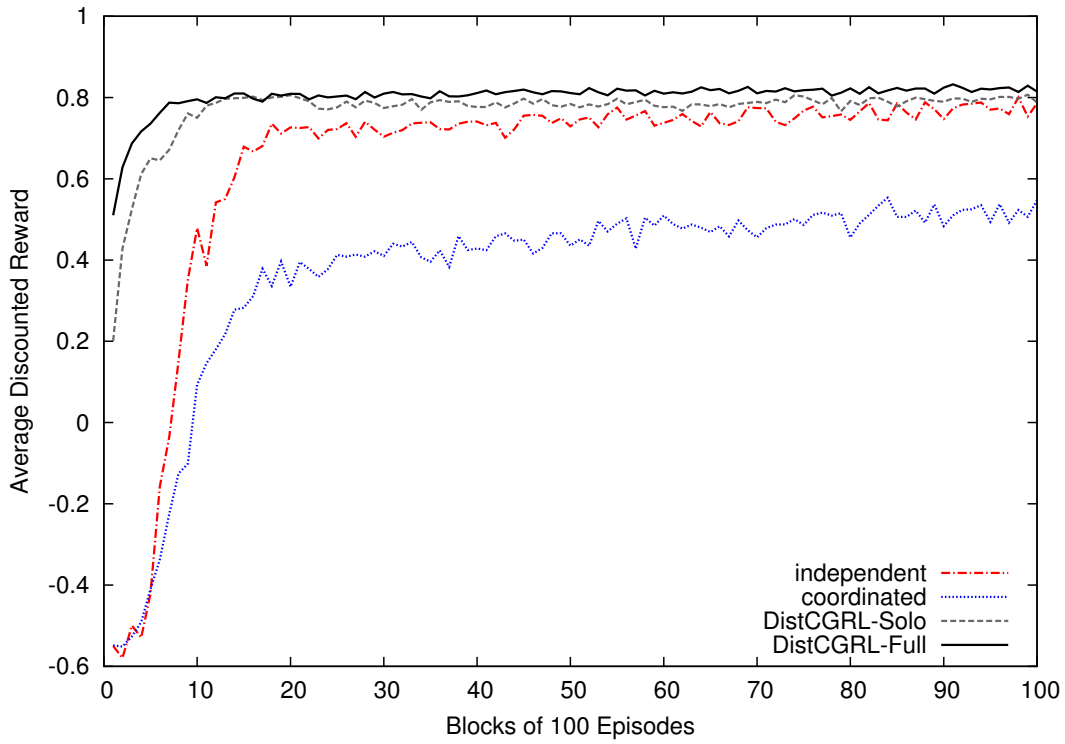
Table 5.1: Quantity of feature weights and CCs for distributed soccer experiments with 8 agents.

The global reward signal gives 1 for a win and -1 for a loss. This is decomposed into individual agent reward signals such that the agent that scores receives 1 while agents nearest to the home goal receive -1 equally divided among them for a loss. Hence during a loss, if only one agent is the nearest to the goal, it receives the full -1 reward while the other agents receive zero. All learners used: discounting of $\gamma = 0.99$, $\epsilon = \{1.0, 0.01, 0.998\}$, and $\alpha = \{0.01, 10^{-4}, 0.998\}$; where the first value indicates the initial parameter setting, the second indicates the final and the third indicates the decay rate per episode.

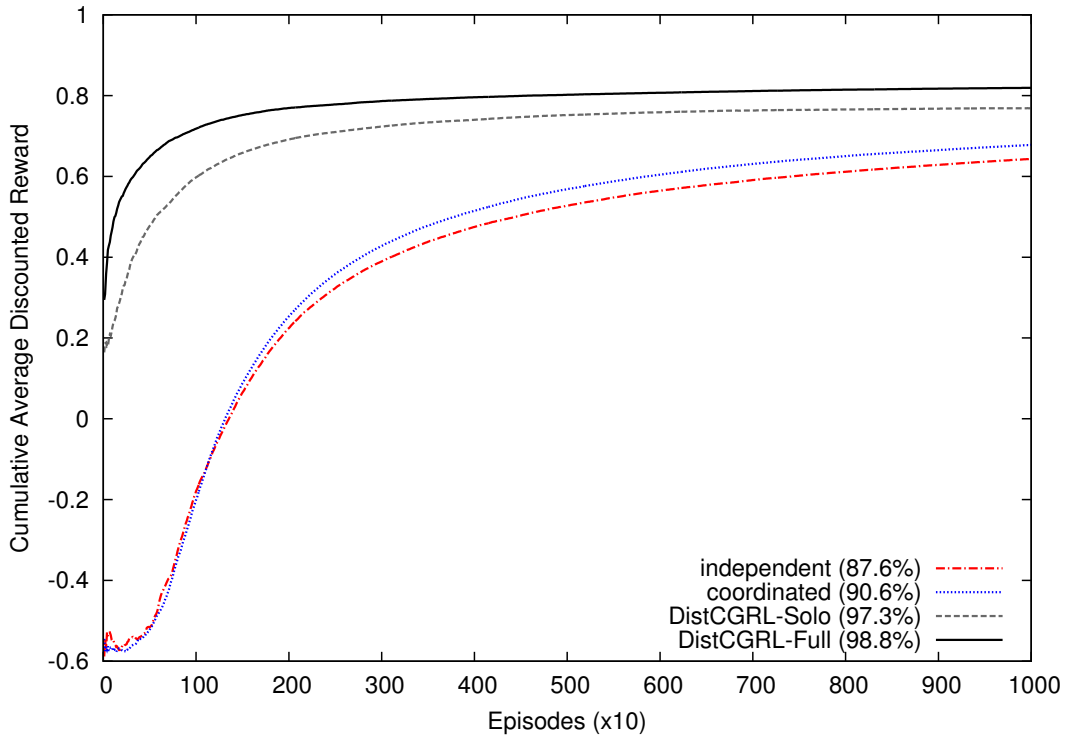
The quantities of feature weights and CCs are given in Table 5.1. The types of features and CCs are the same as the centralized experiment (see Section A.1 page 230) except that they are now scaled to 8 agents and some features are duplicated across agents due to distribution.

The results against the defensive opponent are given in Figure 5.7. In terms of quality of the global joint policy in Figure 5.8a, we see that the DistCGRL-Full learners reach near perfect policies (final win rate of 98.9%) a few hundred episodes earlier than the DistCGRL-Solo learners. For overall performance in Figure 5.8b, we see that DistCGRL-Full achieves the best global goal followed by DistCGRL-Solo. Surprisingly, the coordinated learners do poorer than the independent learners. This may indicate that overcoming the defensive opponent does not require very high coordination among the soccer players. Overall the DistCGRL learners perform better than coordinated and independent learners.

The results against the aggressive opponent are given in Figure 5.8. From Figure 5.8a, we see that the DistCGRL-Solo and DistCGRL-Full learners have learned better policies than the coordinated and independent learners throughout the 10,000 episodes

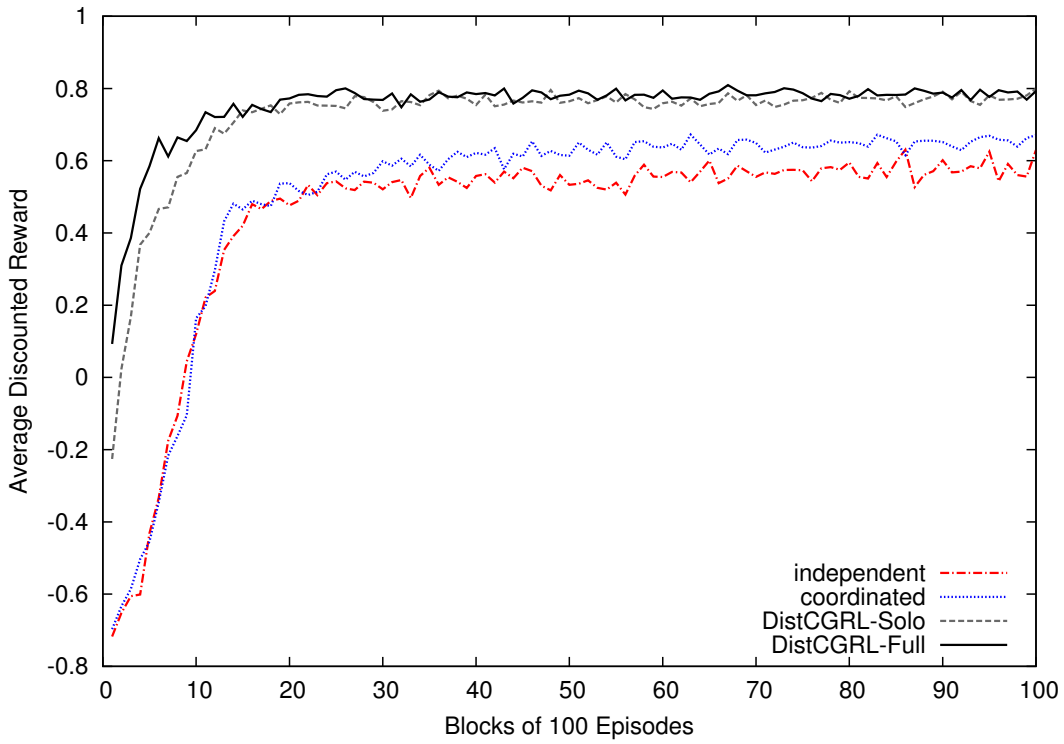


(a) Average Discounted Reward

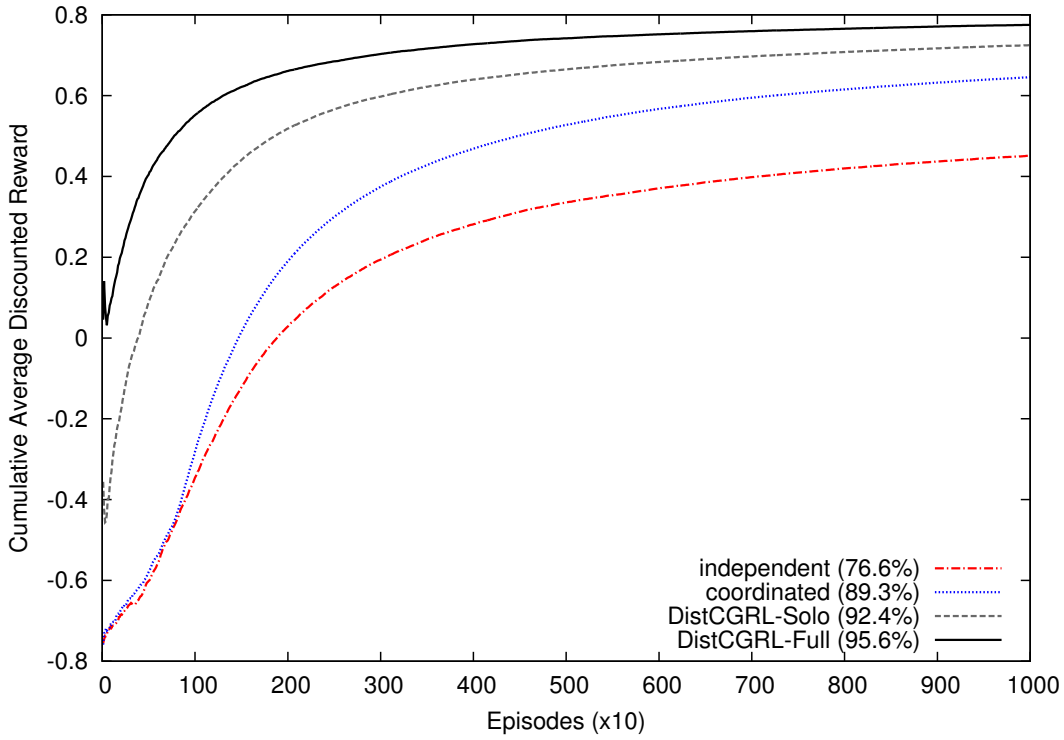


(b) Cumulative Average Discounted Reward (final win rate shown in brackets)

Figure 5.7: Distributed soccer results for defensive opponent. 10,000 episodes averaged over 10 runs for each type of learners.



(a) Average Discounted Reward



(b) Cumulative Average Discounted Reward (final win rate shown in brackets)

Figure 5.8: Distributed soccer results for aggressive opponent. 10,000 episodes averaged over 10 runs for each type of learners.

with the DistCGRL-Full learners having higher average reward than the DistCGRL-Solo learners. In Figure 5.8b, we see that the coordinated learners outperform the independent learners and the DistCGRL learners are much better than the flat learners. This indicates that coordination guidance can be useful in the distributed setting.

5.5.3 Results For Tactical Real-Time Strategy

The tactical RTS game we use for this section is similar to the one in Section 4.4.5 page 97. We pit the RL learners against unpredictable enemy marines and aggressive enemy marines. For each of the two setups, both teams have 10 marines. Communication is limited to an Euclidean distance of 30 points within the 240×240 point map.

The global rewards of -0.1 per time step and 1000 per win is decomposed among marines as follows. At each time step, marines receive the global reward, equally divided among the number of surviving marines. Marines that have been destroyed do not receive any reward and no longer participate in the rest of the episode.

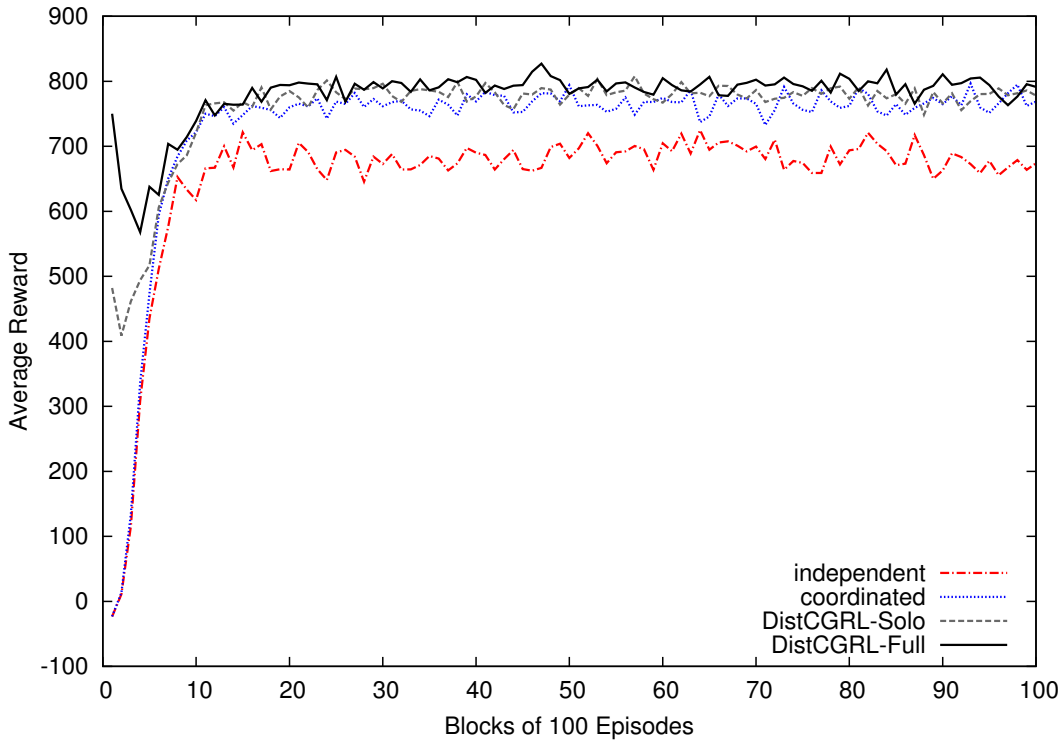
Both RTS setups used no discounting ($\gamma = 1$). The learning step size and exploration parameters are given in Table 5.2 while the quantities of feature weights and CCs are given in Table 5.3. Unlike the centralized CGRL, no relational features were used as the learning parameters are completely distributed among the marines. Hence the number of weights to learn is much larger in the distributed case. The features descriptions are given in Section A.2 page 235.

Setup	exploration (ϵ)			step size (α)		
	initial	final	decay	initial	final	decay
Unpredictable	1.0	0.01	0.998	0.01	10^{-4}	0.998
Aggressive	0.5	0.01	0.998	0.01	0.01	N.A.

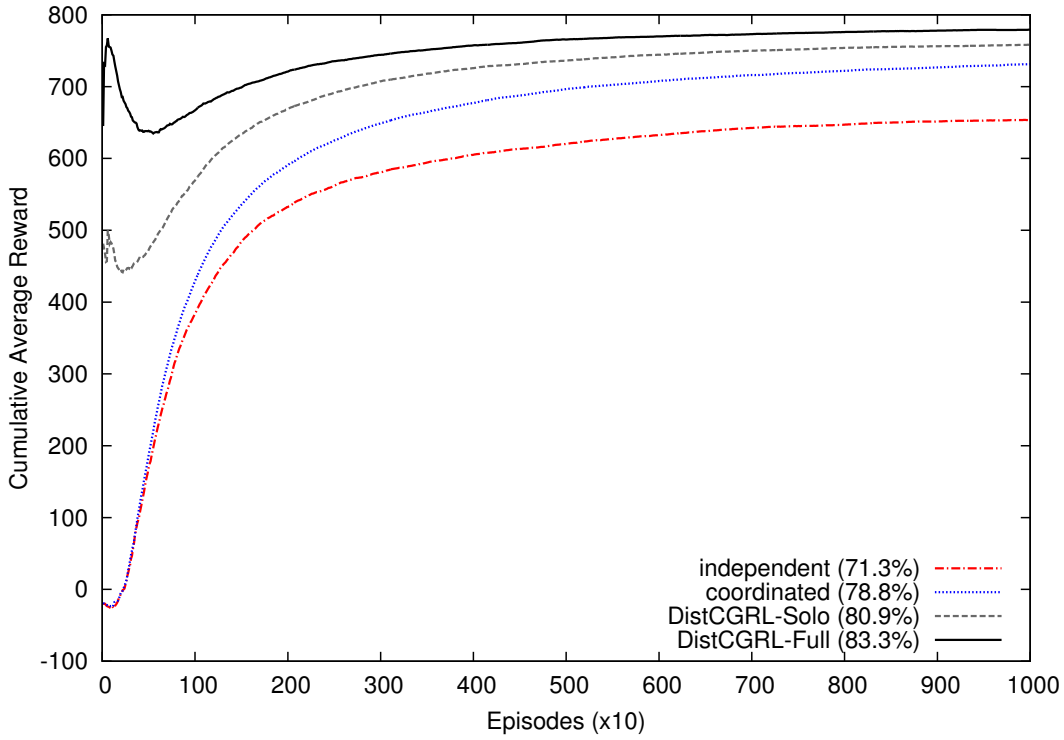
Table 5.2: Table of parameters for distributed RTS experiments.

Learners	Weights	Total CCs	Static CCs	Dynamic CCs
Independent	350	30	30	0
Coordinated	1970	30	30	0
DistCGRL-Solo	2140	90	30	60
DistCGRL-Full	4660	225	75	150

Table 5.3: Quantity of feature weights and CCs for distributed RTS experiments with 10 agents.

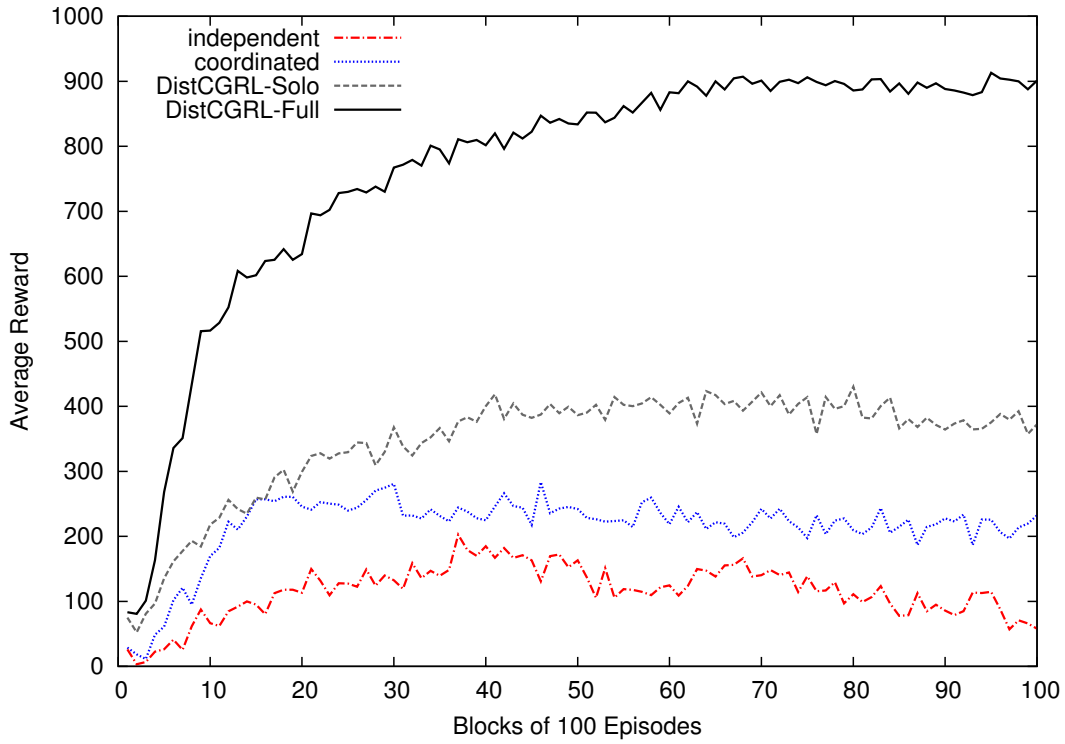


(a) Average Reward

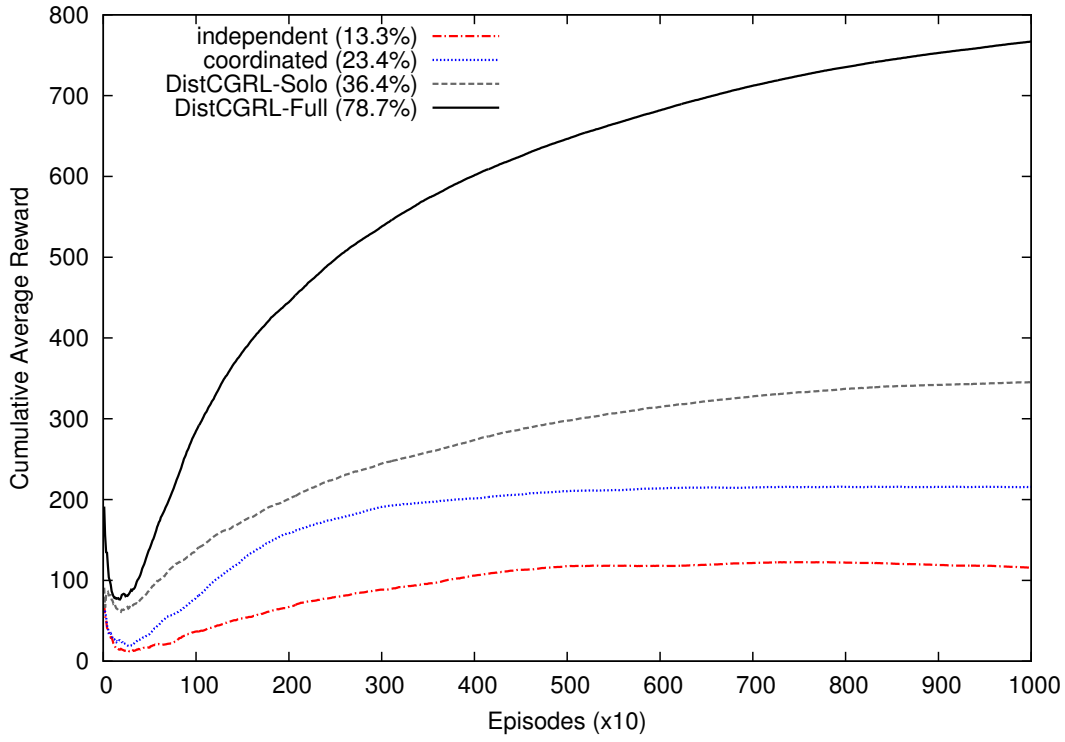


(b) Cumulative Average Reward (final win rate shown in brackets)

Figure 5.9: Distributed RTS results for 10 versus 10 unpredictable marines. 10,000 episodes averaged over 10 runs for each type of learners.



(a) Average Reward



(b) Cumulative Average Reward (final win rate shown in brackets)

Figure 5.10: Distributed RTS results for 10 versus 10 aggressive marines. 10,000 episodes averaged over 10 runs for each type of learners.

The results for 10 versus 10 unpredictable marines are shown in Figure 5.9. In terms of quality of global policies in Figure 5.9a, the DistCGRL learners have better policies than the coordinated and independent learners. The DistCGRL-Full learner has the best policy followed by DistCGRL-Solo and coordinated learners respectively. From Figure 5.9b the DistCGRL learners attained higher overall goal achievement with the same trend as the quality of policy shown by Figure 5.9a .

The results for the more difficult setup of 10 versus 10 aggressive marines are shown in Figure 5.10. A similar trend for the quality of policy is observed for the DistCGRL learners over the rest. The plots in Figure 5.10b show that for online learning, the DistCGRL-Full approach has a clear advantage over DistCGRL-Solo that uses only individual agent constraints.

5.5.4 Comparison with Centralized Approach

For the RTS game in the previous section, we notice that the gains from DistCGRL over flat approaches are not as significant as the gains in the centralized CGRL (see Section 4.4.6 page 98) that employs relational features (RFs). To illustrate we plot together the distributed and centralized results for the 10 versus 10 aggressive marines in Figure 5.11 for CGRL (CentCGRL) and the coordinated learners. An extra centralized CGRL learner (CentCGRL-PF) that uses only propositional features (PFs) instead of RFs is included. The parameter settings for the distributed and centralized learners are displayed in Table 5.4 while the amount of feature weights are compared in Table 5.5 where both CentCoordinated and CentCGRL uses RFs. The quantity and types of CCs used are the same for distributed and centralized cases (see coordinated and DistCGRL-Full in Table 5.3).

Learners	exploration (ϵ)			step size (α)		
	initial	final	decay	initial	final	decay
Centralized	1.0	0.01	0.998	0.01	10^{-4}	0.998
Distributed	0.5	0.01	0.998	0.01	0.01	N.A.

Table 5.4: Table of parameters for centralized and distributed RL for 10 versus 10 aggressive marines.

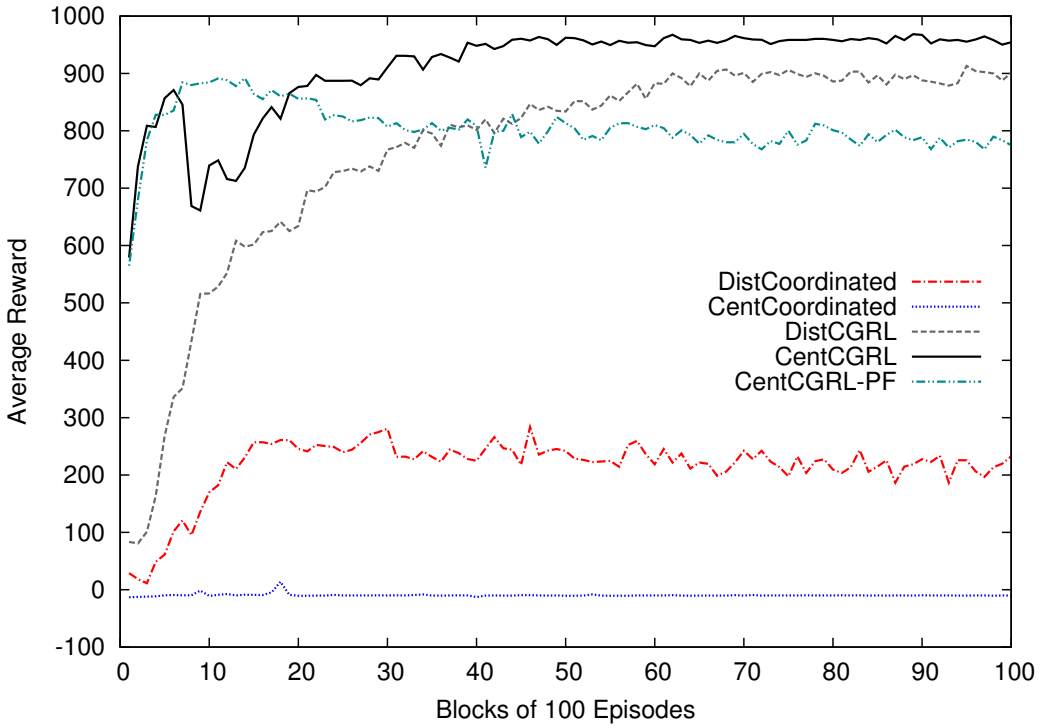
Learners	Weights	Ratio to Centralized with RFs
CentCoordinated	53	1.00
DistCoordinated	1970	37.17
CentCGRL	98	1.00
CentCGRL-PF	2500	25.51
DistCGRL	4660	47.55

Table 5.5: Comparing quantity of feature weights between centralized and distributed RTS experiments with 10 agents.

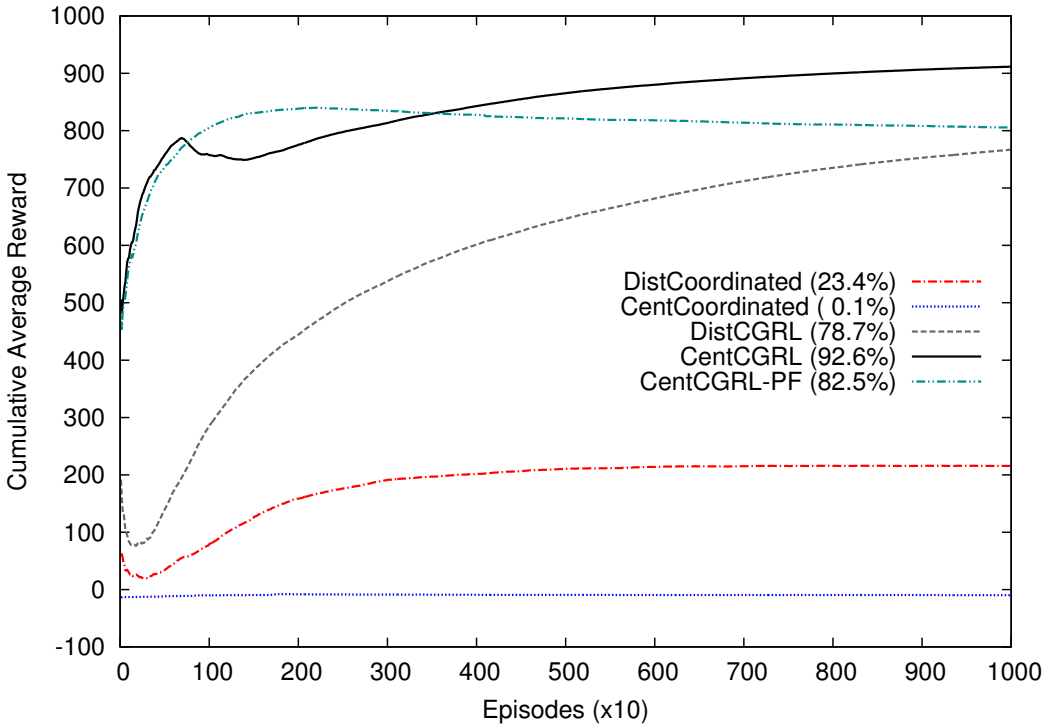
There are two main insights provided by Figure 5.11. First, notice that for the coordinated learners, the centralized version (CentCoordinated) performs poorer than the distributed version (DistCoordinated). This may be due to a few factors. For one, the RFs used in CentCoordinated may have over generalized too quickly, coupled with coarse global rewards that are not agent decomposed, CentCoordinated converged too quickly to losing as fast as possible. Having fine-grained agent decomposed reward and distinct PFs may have helped to better DistCoordinated's performance. Next, the result could be due to the step size and exploration parameter settings. The DistCoordinated learners exploits earlier than CentCoordinated allowing it to experience more winning episodes.

For the second insight, we see the reverse of the first when comparing centralized and distributed CGRL learners. The centralized CGRL (CentCGRL) outperforms DistCGRL in both quality of the solution (Figure 5.11a) and overall goal achievement (Figure 5.11b) although it only learned 98 weights compared to 4660. The result of CentCGRL-PF shows that centralized CGRL with PFs performs poorer than centralized CGRL with RFs. This shows that for RTS, the RFs are very useful for generalizing the top level of the system that selects CCs.

Furthermore, DistCGRL eventually learns a better policy than CentCGRL-PF as seen in Figure 5.11a. This can be attributed to the fined-grained agent decomposed rewards that provide a more precise reward signal to individual agents resulting in better policies as compared to coarse-grained global rewards. To illustrate, consider the scenario where a third of the marines approaches the enemy without opening fire and is destroyed, while the rest overcomes the enemy team with formation and fire-power.



(a) Average Reward



(b) Cumulative Average Reward (final win rate shown in brackets)

Figure 5.11: Comparing distributed and centralized learning in RTS for 10 versus 10 aggressive marines. 10,000 episodes averaged over 10 runs for each type of learners.

With agent decomposed rewards, only the surviving marines will receive a large positive reward while the third of the marines that did nothing useful would have only received the -1 negative reward at every time step they were alive. In contrast, for the centralized learners, the entire team would have received a large positive reward regardless of individual performance. Hence it will be harder to improve individual agents' behaviours from the global reward signal.

5.5.5 Actual Runtime Results

Next, we analyse the runtime performance of distributed CGRL with centralized CGRL and the other coordinated RL learners on the 10 versus 10 aggressive marines set up. The experiments are conducted on the same platform previously described in Section 4.4.7 page 104. The agents in the distributed RL learners are implemented within a single process with a single thread, i.e., the experiments do not exploit parallelism. The mean CPU time per time step for the first 10,000 episodes are measured for each agent and averaged over 10 runs. CPU time is only measured on computation within the agents. This includes action selection and updating.

Figure 5.12 shows the runtime results for the various RL learners. *Actual* refers to

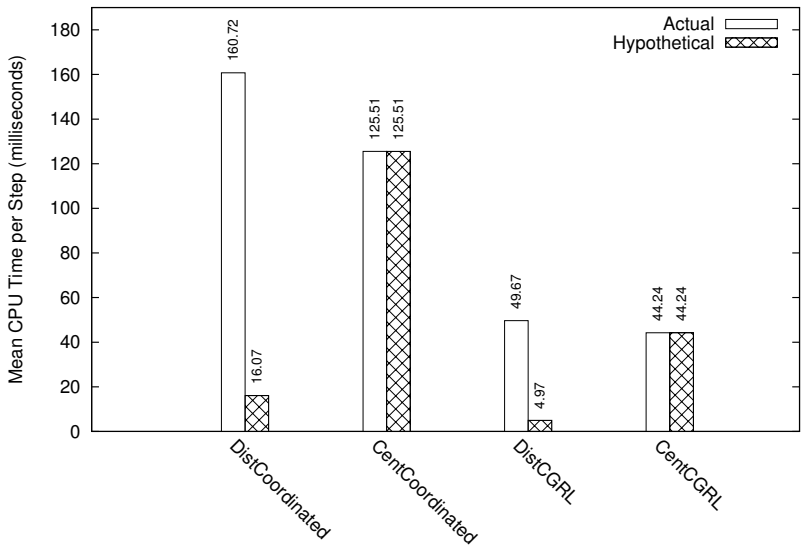


Figure 5.12: Runtime comparison of centralized and distributed RL learners for the experiment in Figure 5.11. The mean CPU time taken per step for 10,000 episodes averaged over 10 runs.

the runtime measured within the agents. *Hypothetical* refers to actual divided by 10 marines and is the hypothetical lower bound on the runtime if a parallel implementation is able to split the CPU time equally among the marines. This is a loose lower bound as it is usual for some marines to be destroyed in each episode. The results show that the increase in number of feature weights in the distributed case due to the propositional features do not heavily impact the runtime. This is expected as runtime is mostly dominated by joint action selection and is also largely affected by the activated CCs (see Section 4.4.7 page 104). Furthermore, the results show that there is room for runtime performance gains for the distributed RL learners if parallel computation is employed.

5.6 Discussion

The results in the previous section demonstrate that coordination knowledge, in the form of CCs, is effective at improving learning performance in the distributed multi-agent setting. Furthermore, the concept of employing CCs to guide knowledge is useful in a variety of distributed scenarios. However, the benefit achieved by distributed CGRL is less than centralized CGRL for certain domains such as RTS. This may be due to a lack of generalization through relational features resulting in the number of weights to be learned in the distributed case being much greater than the centralized case.

In terms of constraining the primitive action space to guide exploration, employing multi-agent CCs has advantages over using only individual agent constraints in distributed task based methods (Ghavamzadeh et al., 2006). This reinforces the notion that coordination knowledge can do more than being features for function approximation.

The use of CCs has been investigated under the assumptions of a dynamic communication structure. In terms of learning to use expert knowledge to guide exploration, few works have been evaluated under these communication restrictions. In Zhang et al. (2009), a framework was introduced where supervisor agents bias the policies of worker agents that select the primitive actions. However, the learning interactions of the supervisors were not defined. DistCGRL can be viewed from the perspective that each agent

carries local 'supervision' knowledge through CCs as the top level is distributed into the agent level. This is more applicable for domains where communication structure among agents change as we have presented.

5.7 Conclusion

This chapter introduced DistCGRL for domains where agents have a dynamic communication structure. The benefits of CCs in guiding exploration shown in the previous chapter are further reinforced by this chapter's results for the distributed case. In terms of the main research challenges, DistCGRL addresses the challenges of limited and changing communication. This is handled by decentralizing CC selection and distributing learned value functions for the top level of the CGRL system into the boundaries of individual agents. DistCGRL is shown to be effective for online learning, outperforming existing learning approaches in terms of overall goal achievement.

DistCGRL mainly dealt with issues in dynamic communication changes. Although DistCGRL does not directly address fine-grained communication costs, note that communication only takes place during joint action selection. These selection methods are message passing distributed COP solvers such as max-plus that are well studied in the literature (see Section 3.2.2 page 36). To handle fine-grained communication costs, the number of messages passed between agents can be limited as required. Hence DistCGRL may be adapted to better manage communication costs.

The migration from the centralized CGRL system to the distributed one is not without any drawbacks. Because of the requirement of dynamic communication, weights for learning are distributed among agents. This results in a lack of generalization through the use of relational features and an increase in model complexity. For domains such as the tactical RTS game where relational features can yield high performance, this drawback constitutes a step back in addressing the challenge of model complexity as shown in Section 5.5.4. Hence, the next chapter proposes techniques to mitigate this problem by allowing agents to exchange some learned knowledge between weights of features

that share relational semantics.

Chapter 6

Distributed Relational Temporal Difference Learning

Relational representations have great potential for rapidly generalizing learned knowledge in large Markov decision processes such as multi-agent problems. This chapter introduces relational temporal difference learning for the distributed case where the communication links among agents are dynamic. No critical components of the proposed method reside in any one agent. Relational generalization among agents' learning is achieved through the use of partially bound, local agent based, relational features and a message passing scheme. These methods are combined with existing multi-agent coordinated reinforcement learning (RL) and distributed coordination guided RL (CGRL). Experiments were conducted on soccer and real-time strategy game domains with dynamic communication. Results show that the proposed methods improve goal achievement in online learning with a large decrease in number of parameters to learn. Hence, model complexity can be reduced while simultaneously improving the performance of distributed RL. Part of this work has been published in Lau et al. (2013).

6.1 Motivation

Relational representations (see Section 3.6 page 46) have the potential to significantly reduce model complexity for multi-agent learning where huge joint state and action

spaces are common (Džeroski et al., 2001; Guestrin et al., 2003; Tadepalli et al., 2004; Croonenborghs et al., 2007). In particular, relational temporal difference (TD) learning (Strens, 2004; Asgharbeygi et al., 2006) generalizes propositional features (PFs) that are defined from predicates by combining them to give a single relational feature (RF) with one weight to be learned (see Section 3.6 page 46). This approach is flexible in that it allows RFs from predicates to be combined with other kinds of features for function approximation.

Previously, Chapter 4 described how RFs may be used to generalize TD learning of the action value functions in Section 4.3.6 page 84. Results in Section 4.4.6 page 98 and Section 5.5.4 page 134 have shown that RFs have good potential for improving learning performance in the RTS domain when coupled with centralized CGRL. To emphasize the generalization capability of RFs that allow a reduction in the number of learning parameters, we revisit the $NotAligned$ predicate previously described in Example 1.1 page 6.

Capital letters are used to denote first order predicate variables, i.e., variables that are not fixed to any particular object (agent). Small letters denote that the variables in the predicate's arguments are *bound* to particular objects to yield a propositional statement. For example, let $Pred1(A_i, A_j)$ be a predicate on any two agents. After *binding* $Pred1$ to agents a_x and a_y, we have the proposition $Pred1(a_x, a_y)$ which is either true or false depending on the actual values of a_x and a_y. Now we proceed with the example.

Example 6.1 (Not aligned predicate for RTS)**.** *Consider the coordination knowledge in Example 1.1 page 6 that describes bad alignment of marines' positions in a tactical RTS game. Bad alignment occurs when marines take movement actions that result in less than maximal firepower directed at oncoming enemies. This knowledge can be captured by a pairwise predicate on any two marines,* x *and* y*,*

$$NotAligned(S_x, S_y, A_x, A_y) := SameNearestEnemy(S_x, S_y) \\ \wedge\ DistanceDiff(S_x, A_x, S_y, A_y) > 0 \qquad (6.1)$$

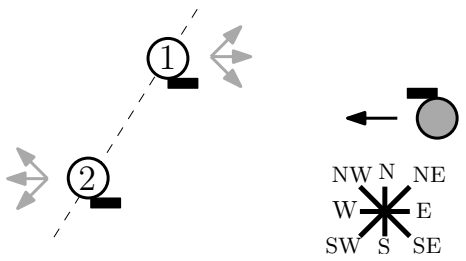


Figure 6.1: Example of actions (grey arrows) that will lead to white marines 1 and 2 being further unaligned to the enemy (dark grey) marine.

where $SameNearestEnemy$ is true if the two marines' nearest enemy is the same. Let this nearest enemy be E. Then, the function $DistanceDiff$ returns the magnitude of the difference in distance between pairs of marines (x, E), and (y, E), after both marines have each taken actions a_x and a_y in joint state s. Figure 6.1 depicts a state that will lead to $NotAligned$ being true for white marines 1 and 2 if they take the actions indicated by the grey arrows.

If only PFs are used, predicates involving pairs of agents such as $NotAligned$ will be bound to specific agents. This gives rise to a number of features that is quadratic in the number of agents in a centralized setting. For example, for $NotAligned$ and N marines, we will have $N(N-1)/2$ number of features as $NotAligned$ is symmetric. However, with RFs, we can define a single feature that takes into account all possible binding of agents to the predicates as shown in the example below.

Example 6.2 (Not aligned relational feature for RTS)**.** *Suppose we have N marines. Then, we define the RF for not aligned as*

$$\varrho_{NotAligned}(\mathbf{s}, \mathbf{a}) = \sum_{x,y \in [1,N] \mid x<y} \tau \cdot NotAligned(\mathbf{s}_x, \mathbf{s}_y, a_x, a_y)$$

where τ is a scaling factor that may be a constant, e.g., $N(N-1)/2$ to normalize the range of $\varrho_{NotAligned}$ to $[0, 1]$. For centralized coordinated learning using linear function approximation, the weight of this RF is then updated as,

$$w_{NotAligned} \leftarrow w_{NotAligned} + \alpha[r + Q^{\pi}(\mathbf{s}', \mathbf{a}') - Q^{\pi}(\mathbf{s}, \mathbf{a})]\varrho_{NotAligned}(\mathbf{s}, \mathbf{a})$$

using the observation $\langle \mathbf{s}, \mathbf{a}, r, \mathbf{s}', \mathbf{a}' \rangle$ *when following the policy* π *in the environment.*

Hence centralized updates involving knowledge of $NotAligned$ will be performed on the same weight $w_{NotAligned}$, effectively generalizing learning and reducing the quadratic in N number of learning parameters to only one parameter.

In spite of the many advantages of RFs, using them in reinforcement learning has thus far been mostly limited to centralized[1] RL where a designated controller is tasked to compute and store the learned parameters. Unfortunately, in a highly dynamic decentralized scenario, it is not possible to find such a designated controller as the communication links among agents change over time. For example, in the RTS game, marines lose communication when destroyed, similarly in some robotics application, malfunction may render agents unresponsive. In the distributed case, PFs cannot be aggregated into RFs as weights like $w_{NotAligned}$ will have to reside in some agent that may not always be accessible by other agents. Thus it is important that the learning system must be decentralized and that the critical components do not reside in any one agent.

Existing works in multi-agent relational RL are few, and in the distributed case, even rarer. The works in Guestrin et al. (2003); Croonenborghs et al. (2005) describe multi-agent relational learning with centralized controllers. While the work in Ponsen et al. (2010) investigated solving separate tasks for each agent that may seek other expert agents for advice where relational learning was used to identify such experts.

The above reasons constitutes the motivation to develop a relational TD learning method for distributed agents with dynamic communication. Consequently, the new learning method will be combined with distributed RL and the distributed CGRL approach presented in the previous chapter to further improve online learning.

6.2 Aims & Approach

The aim in this chapter is to develop a relational TD learning method for distributed agents that have dynamic communication links. The main challenge is to handle the

[1] In this chapter, *centralized* corresponds to the same problem but without communication restrictions.

communication restrictions while maintaining most of the learning benefits of relational generalization in the centralized case.

To approach the problem, RFs are first devised from predicates that are *local* to each agent. Unlike global RFs, these local RFs allow an agent based decomposition of a global value function that uses global RFs. This scheme enables agents to generalize their interactions with other agents within themselves. Additionally, the global value function may be distributed such that no agent is critical to the learning system.

For the next step, message passing between neighbouring agents is used to transfer current knowledge (learned weights) between agents' features that share relational semantics. These allows learning from interactions within a group of agents to be transferred to other groups whenever agents in separate groups come into contact with each other.

These proposed methods are evaluated on domains with dynamic communication such as the soccer domain and the tactical RTS game. Agents in these domain have a limited communication range. Furthermore in RTS, agents may be removed from the current episodes once they are destroyed and may only rejoin at the start of the next episode. The proposed methods for relational TD are coupled with distributed coordinated learning, and distributed CGRL to better handle the three research challenges laid out in Section 1.2 page 4, namely: exploration, model complexity, and distribution. To the best of our knowledge, the proposed methods are the first that decentralizes relation features among agents such that no agents are critical in the system.

The evaluation of distributed relational TD learning in this chapter focuses primarily on its generalization capability to reduce model complexity of the value functions while simultaneously improving learning efficiency. Less emphasis is placed on the reasoning benefits and capabilities of relational representations. Hence the proposed approach is more related to the works of Guestrin et al. (2003); Strens (2004); Asgharbeygi et al. (2006). These allow relational features to be combined with other types of features for function approximation as opposed to the family of methods for relational RL using decision trees (Džeroski et al., 2001; Croonenborghs et al., 2007; Ponsen et al., 2010).

6.3 Distributed Relational Generalizations

This section describes how relational generalization can be employed for distributed RL via an approach internal to each agent and an external approach that requires passing messages. The similar DEC-MDP problem formulation as described in Section 5.3 page 112 is employed for the distributed setting. Before presenting internal and external generalization, we review the details of TD learning with relational features assuming full communication.

6.3.1 Centralized Relational Temporal Difference Learning

We briefly review the current form of relational TD learning that was employed for centralized coordinated learning. Relational generalization can be achieved through the use of RFs that are generalizations over the possible bindings of objects to the arguments of first-order logical predicates that yield groundings when true. TD learning using RFs for the action value function is an adaptation of the work in Asgharbeygi et al. (2006) that was designed for the state value function.

Let ϱ_ρ represent an RF based on the predicate ρ with agent arity $\eta(p)$. The agent arity is the number unique agents from which the variables of a predicate come from, as opposed to the usual notion of the arity (or scope) of a function that is the number of arguments. For example, the predicate $EgPredicate(s_x, a_x, s_y, a_y)$ has an arity of 4 but an agent arity of $\eta(EgPredicate) = 2$. Recall that the function $Perm(N, n)$ returns the set of n-permutations from the set of all N agents. For a given N and n, the number of such permutations is $|Perm(N, n)| = {}_N\mathrm{P}_n = \frac{N!}{(N-n)!}$. For a given predicate ρ, the construction of an RF for N agents shown in Equation 4.81 page 84 for the centralized case is

$$\varrho_\rho(\mathbf{s}, \mathbf{a}) = \tau_\rho \cdot \sum_{i=1}^{{}_N\mathrm{P}_n} \rho(s_{i_1}, ..., s_{i_n}, a_{i_1}, ..., a_{i_n}) \tag{6.2}$$

where $n = \eta(\rho)$, τ_ρ is a scaling factor for predicate ρ, and $s_{i_1}, ..., s_{i_n}$, and $a_{i_1}, ..., a_{i_n}$ are projections on $\mathbf{s}$ and $\mathbf{a}$ respectively, based on some unique i-th n-permutation of the

agents. Equation 6.2 explicitly states that predicates may consist of the state variables and action variables accessible by some number of agents. When the $\tau_\rho = 1$, Equation 6.2 describes the RF as the count of number of true propositions for every binding of variables to the predicate ρ. This is similar to the count of true grounded literals for the state-only relational features described in Asgharbeygi et al. (2006).

Let F be the set of all predicates. Then, the global relational action value function, as a linear function approximation, is given by,

$$Q(\mathbf{s}, \mathbf{a}) = \sum_{\rho \in F} w_\rho \varrho_\rho(\mathbf{s}, \mathbf{a}) \tag{6.3}$$

and the weights can then be updated using

$$w_\rho \leftarrow w_\rho + \alpha[r + \gamma Q(\mathbf{s}', \mathbf{a}') - Q(\mathbf{s}, \mathbf{a})]\varrho_\rho(\mathbf{s}, \mathbf{a}) \tag{6.4}$$

for a centralized system where r is the global reward.

Next, we describe how partial relational generalization is achieved for the individual agent's local action value functions for the distributed case.

6.3.2 Internal Generalization

In the DEC-MDP, agents may only access their own and their current neighbours' state and action variables. As communication is dynamic, each agent should carry a part of the learning system within them. Hence each agent x carries the local Q_x function that are approximated using PFs. Agents may internally generalize the weights between related PFs involving themselves and other agents. This is achieved by the partial binding of predicates to give RFs local to each agent. The result enables the sharing of learned parameters from an agent x's interactions with another agent y in a particular state with x's interaction with other agents $z \neq y$. This is illustrated in Figure 6.2 where the grey agent generalizes its interactions shown as lines among the various other agents.

We elaborate on the purpose internal generalization using the $NotAligned$ predicate from before. From here on, without loss of generality, we take the presence of the action

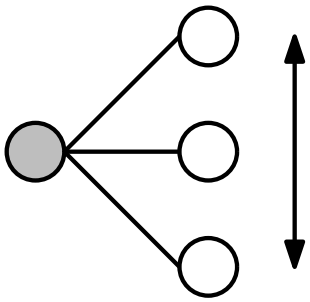


Figure 6.2: Internal relational generalization. Circles are agents and black lines are interactions. The line with arrowheads indicate the grey agent is internally generalizing interactions over the various other (white) agents.

variable a_x to also indicate the presence of the accessible state variables $\mathbf{s}_x$ and omit writing the state variables.

Example 6.3 (Not aligned internal RF)**.** *Consider the pairwise $NotAligned$ predicate defined in Equation 6.1. Suppose we have an RTS game with 4 marines and each agent has PFs created from predicates that involve themselves. Then, marine 1 will have the pairwise PFs,*

$$\{NotAligned(a_1, a_2), NotAligned(a_1, a_3), NotAligned(a_1, a_4)\}$$

while marine 2 has,

$$\{NotAligned(a_1, a_2), NotAligned(a_2, a_3), NotAligned(a_2, a_4)\}$$

and so on for the other marines. A partially bound predicate may be defined based on $NotAligned$ by fixing the first agent variable such that $x = 1$, e.g., $NotAligned(a_1, a_y)$ where a_y is any other agent. Then, the internal $NotAligned$ RF for marine x is

$$\varrho_{NotAligned,x}(\mathbf{a}_x) = \tau \cdot \sum_{y \in [1,4]-\{x\}} NotAligned(a_x, a_y). \tag{6.5}$$

where τ is a scaling factor, and $NotAligned$ is assumed to be symmetrical, i.e., $NotAligned(a_x, a_y) = NotAligned(a_y, a_x)$.

The example indicates that agent 1 may generalize its knowledge of $NotAligned$

from interaction with agent 2 to the other agents 3 and 4 as these interactions contribute towards updating the same feature weight. Furthermore, for N agents, we see that generalizing internally for pairwise predicates reduces the quadratic in N number of feature weights to a linear in N number of weights to be learned.

In general, for each predicate ρ, the partially bound, agent based local RF for agent x is defined as

$$\varrho_{\rho,x}(\mathbf{a}_x) = \frac{\tau_\rho}{n} \sum_{j=1}^{n} \sum_{i=1}^{N-1\mathrm{P}_{n-1}} \rho(a_{i_1}, ..., a_{i_{j-1}}, a_x, a_{i_{j+1}}, ..., a_{i_{n-1}}) \tag{6.6}$$

where there are N agents, $n = \eta(\rho)$, '$a_{i_1}, ..., a_{i_{j-1}}, a_x, a_{i_{j+1}}, ..., a_{i_{n-1}}$' is an n-permutation of the values in the tuple $\mathbf{a}_x$, and τ_ρ is a scaling factor which we will assume to be 1 from here on. The double summations indicate that the action variable a_x is inserted at the various positions j of the predicate ρ with respect to each $(n-1)$-permutation drawn from the other $N-1$ agents. That is, while the other $n-1$ agents' action variables may be drawn from the set of all other $N-1$ agents, agent x's participation in the RF is fixed. As before in Section 5.4.4 page 123, the predicate is false for agent action values not found in $\mathbf{a}_x$, i.e., $\rho(\cdot)$ returns false (zero) for agents that are out of communication range.

With Equation 6.6 and given the set of predicates F, local action value functions can be specified for each agent x as the linear function approximation,

$$Q_x(\mathbf{a}_x) = \sum_{\rho \in F} w_{\rho,x} \varrho_{\rho,x}(\mathbf{a}_x) \tag{6.7}$$

$$= \sum_{n=1}^{N} \sum_{\rho \in F_n} w_{\rho,x} \varrho_{\rho,x}(\mathbf{a}_x) \tag{6.8}$$

where $F_n \subseteq F$ is the set of predicates $\{\rho \in F \mid \eta(\rho) = n\}$ that is a partition of F such that $F = \bigcup_{n=1}^{N} F_n$. Equation 6.8 essentially groups the sums based on the agent arity of the RFs, assuming that the maximum agent arity is N.

Now we are ready to present the theoretical result that shows the agent based decomposition in Equation 6.7 is an additive decomposition of the global relational action

value function in Equation 6.3.

Theorem 6.1. *Given the same set of predicates F, the global relational value function $Q(\mathbf{s}, \mathbf{a}) = \sum_{x=1}^{N} Q_x(\mathbf{s}_x, \mathbf{a}_x)$ the sum of local relational value functions, if the local weight for each agent x's local RF $w_{\rho,x} = w_\rho$ the global weight for the global RF based on the same predicate ρ.*

Proof. For this proof we expand on the sum of N local agent action value functions while omitting the state variables as before. Assume that the scaling factor for local value functions and the global value function to be $\tau = 1$. It should be similar to prove the same theorem with conditions on other values of τ. Starting from the sum of local Q_x functions,

$$\sum_{x=1}^{N} Q_x(\mathbf{a}_x) = \sum_{x=1}^{N} \sum_{n=1}^{N} \sum_{\rho \in F_n} w_{\rho,x} \varrho_{\rho,x}(\mathbf{a}_x) \tag{6.9}$$

$$= \sum_{x=1}^{N} \sum_{n=1}^{N} \sum_{\rho \in F_n} \frac{w_{\rho,x}}{n} \sum_{j=1}^{n} \sum_{i=1}^{{}^{N-1}\mathrm{P}_{n-1}} \rho(a_{i_1}, ..., a_{i_{j-1}}, a_x, a_{i_{j+1}}, ..., a_{i_{n-1}}) \tag{6.10}$$

$$= \sum_{n=1}^{N} \sum_{\rho \in F_n} \sum_{x=1}^{N} \frac{w_{\rho,x}}{n} \sum_{j=1}^{n} \sum_{i=1}^{{}^{N-1}\mathrm{P}_{n-1}} \rho(a_{i_1}, ..., a_{i_{j-1}}, a_x, a_{i_{j+1}}, ..., a_{i_{n-1}}) \tag{6.11}$$

, as $w_{\rho,x} = w_\rho$, factor w_ρ,

$$= \sum_{n=1}^{N} \sum_{\rho \in F_n} w_\rho \sum_{x=1}^{N} \sum_{j=1}^{n} \sum_{i=1}^{{}^{N-1}\mathrm{P}_{n-1}} \frac{\rho(a_{i_1}, ..., a_{i_{j-1}}, a_x, a_{i_{j+1}}, ..., a_{i_{n-1}})}{n} \tag{6.12}$$

, by rearranging the sums,

$$= \sum_{n=1}^{N} \sum_{\rho \in F_n} w_\rho \sum_{i=1}^{{}^{N}\mathrm{P}_{n}} \sum_{j=1}^{n} \frac{\rho(a_{i_1}, ..., a_{i_n})}{n} \tag{6.13}$$

, as j is a redundant index, the last sum is a multiplication by n,

$$= \sum_{n=1}^{N} \sum_{\rho \in F_n} w_\rho \sum_{i=1}^{{}^{N}\mathrm{P}_{n}} \rho(a_{i_1}, ..., a_{i_n}) \tag{6.14}$$

$$= \sum_{n=1}^{N} \sum_{\rho \in F_n} w_\rho \varrho_\rho(a_{i_1}, ..., a_{i_n}) \tag{6.15}$$

$$= \sum_{\rho \in F} w_\rho \varrho_\rho(\mathbf{a}) = Q(\mathbf{a}) \tag{6.16}$$

Next we demonstrate that Equation 6.13 can be achieved by rearranging the sums in Equation 6.12. Notice that there are duplicate terms in each agent's local function, for example, $NotAligned(a_1, a_2)$ appears in agent 1 and agent 2 in Example 6.3. A predicate term of n agent arity with a particular permutation of its variables will appear n times due to the terms generated by the sum $\sum_{x=1}^{N} \sum_{j=1}^{n} \sum_{i=1}^{N-1 \mathrm{P}_{n-1}} (...)$ in Equation 6.12. This is because, the same n-permutation of the variables in the predicate ρ will be generated once by each of the n agents. This equivalence in number of terms can also be observed from the following equations relating the total number local terms in Equation 6.12 to the number of terms in Equation 6.13,

$$N \cdot n \cdot {}_{N-1}\mathrm{P}_{n-1} = n \cdot N \cdot \frac{(N-1)!}{[(N-1)-(n-1)]!} \tag{6.17}$$

$$= n \cdot (n-1)! \cdot \frac{N!}{(n-1)![(N-1)-(n-1)]!} \tag{6.18}$$

$$= n! \cdot \frac{N!}{(n-1)!(N-n-1+1)!} \tag{6.19}$$

$$= n! \cdot \frac{N!}{(n-1)!(N-n)!} \tag{6.20}$$

$$= n! \cdot \frac{N!}{n!(N-n)!} \cdot n \tag{6.21}$$

$$= {}_{N}\mathrm{P}_{n} \cdot n \tag{6.22}$$

In Equation 6.13, $\sum_{i=1}^{N \mathrm{P}_n} \sum_{j=1}^{n} (...)$ indicates that each n-permutation is summed n times. From here we eventually arrive at $Q(\mathbf{s}, \mathbf{a}) = \sum_{x=1}^{N} Q_x(\mathbf{s}_x, \mathbf{a}_x)$. □

Theorem 6.1 shows that the local relational agent based decomposition is able to collectively represent the same function values as the ideal global case. This indicates that representational power is maintained when using the local value functions. Consequently, if the true global value function is well approximated by a linear combination of global RFs, we expect the same error bound to apply when using the local value functions with local RFs. Hence, a successful scheme has been devised to make use of

RFs while ensuring that each agent keeps a portion of the value function. This decomposition solves the distributed requirements.

With the local value functions given in Equation 6.7, the local agent based on-policy TD update equations in Section 3.3.4 page 40 for TD learning are used to build a completely distributed RL system, i.e., for each agent x, its weights for each RF are updated via,

$$w'_{\rho,x} \leftarrow w_{\rho,x} + \alpha[r_x + \gamma Q_x(\mathbf{s}'_x, \mathbf{a}'_x) - Q_x(\mathbf{s}_x, \mathbf{a}_x)]\varrho_{\rho,x}(\mathbf{s}_x, \mathbf{a}_x) \tag{6.23}$$

As the global Q function is a sum of local Q_x functions, such a global function is itself a larger linear function approximation. Hence, we expect using Equation 6.23 to provide the same optimal convergence guarantees under the same conditions as stated in Perkins and Precup (2002); Melo et al. (2008). For distributed CGRL in Chapter 5 page 109, we can represent both top and bottom level local agent value functions with the described RFs in Equation 6.6.

Using the predicate representation, more complex predicates can be constructed from logical operators on simpler predicates. The local RFs used in internal generalization are based on first order predicates with the constraint that one of the variables is fixed to the particular agent that the local RF belongs to. Therefore, to compute valid bindings, a customized implementation or an existing inference engine for single agents can be used with this constraint for each agent as was done in Džeroski et al. (2001); Kersting and De Raedt (2004). As in standard distributed RL, joint action selection for local Q_x functions approximated by local RFs can be solved using the same message passing methods as that for distributed RL (see Section 3.2.2 page 36). This is because using Equation 6.7, the local Q_x functions remain additively decomposable into components based on the variables in the scope of the features.

However, local TD updates in Equation 6.23 provides a weaker form of relational generalization in contrast to the global update in Equation 6.4. The key difference is that the global update assumes no communication restrictions. Pan agents' RFs are involved in the TD update through the global function Q in Equation 6.4 whereas in the

local case, only each agent x's weights and RFs are involved in Q_x. Consequently, the next section explores a solution to further generalize learning with respect to relations.

6.3.3 External Generalization

Internal generalization allows an agent to generalize learning between its individual interactions over various agents. This introduces a form of shared experience and reduces the number of learning parameters in the system as a whole. However, there is still room for further generalization as interactions between other agents in the system do not play a large role in the updating of local agent's weights.

To illustrate the difference with centralized relational TD learning, consider the case where there are disjointed groups of agents for a certain time as shown in Figure 5.5 page 120. Agents may communicate with each other within their group but not outside of it. The true bindings to the same predicate in one group will not factor into the TD updates of agents in another group due to the communication restrictions. In contrast, these true bindings will have an impact in the global centralized case as they are involved in the global RF used in the update in Equation 6.3. Hence, local updates only allow an agent to generalize learning over its interactions with other agents within its communication group, but not over interactions between agents from other groups.

More specifically, consider the centralized $NotAligned$ RF in Equation 6.1. It allows all pairwise interactions between any two agents to share experience. That is, whenever $NotAligned(a_1, a_2), NotAligned(a_2, a_3)$, or $NotAligned(a_3, a_4)$ are true, they participate in updating the same weight. In contrast, Equation 6.5 only allows agent 1's individual experience to be generalized within itself. That is, whenever $NotAligned(a_1, a_2)$ or $NotAligned(a_1, a_4)$ are true they update the same weight in Agent 1, but $NotAligned(a_2, a_3)$ and $NotAligned(a_3, a_4)$ do not play a part in the update by the definition of local RFs in Equation 6.6. Short of neighbouring agents' influences during coordinated action selection, there is no other mechanism to pass on learned knowledge to other agents, as local updating shown in Equation 6.23 does not involve the value functions (learned parameters) of other agents.

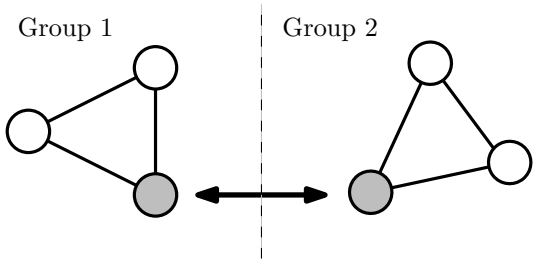


Figure 6.3: External relational generalization. Circles are agents and black lines are interactions. There are two disjoint groups of three agents. Within each group, interactions among agents are generalized. The line with arrowheads indicates that the grey agents should share what each has learned in its group when they finally meet.

The above observations motivate us to develop a form of sharing of experience whenever agents are in contact with each other. This is illustrated in Figure 6.3, where initially, only agents in their respective disjoint groups share experience. Subsequently, when the two grey agents are within communication range, they should be able to share what they have learned from their respective groups with each other. Theorem 6.1 further provides a clue to this goal through the condition that $w_{\rho,x} = w_\rho$. This suggests that to obtain comparable results to the centralized case, agents should seek to learn the same weights for each RF $\varrho_{\rho,x}$ that is based on the predicate ρ. We make use of this intuition to design a experience sharing scheme for external generalization.

We devise a message passing method to share the learned weights of the local RFs based on the relations that they represent. Under our communication restrictions, agents may send messages to their current neighbouring agents. This is used to pass messages for joint action selection with neighbours. In the process, agents are aware of their neighbours' local actions and states. Likewise, to achieve further relational generalization, additional messages may be passed. The intent is to average the weights of features related by their predicates so that all $w_{\rho,x} = w_\rho$ the true value of each weight in centralized relational TD learning. We first present a synchronized message passing scheme. Then, it is simplified as a piecewise asynchronous update scheme.

In a synchronized message passing system, messages are passed for a number of iterations within each DEC-MDP time step. At each iteration, agent x computes and sends the relational messages $\kappa_{x,y}(\rho)$ for each predicate, ρ, to its neighbour, agent $y \in$

$\Gamma(x)$,

$$\kappa_{x,y}(\rho) = \left[w_{\rho,x} + \sum_{z \in \Gamma(x) - \{y\}} \kappa_{z,x}(\rho) \right] + c_{x,y} \tag{6.24}$$

where $c_{x,y}$ is a normalizing constant to ensure messages do not go to infinity in a graph with cycles (Wainwright et al., 2004). The agent x sends the message $\kappa_{x,y}(\rho)$ to agent y that is an aggregation of the messages it received from its other neighbours, i.e., $\Gamma(x) - \{y\}$, from the previous iteration. In this message passing scheme, a fixed point exists in a CG without cycles.

Once the desired iteration is achieved or the κ messages have converged, each agent x updates the weight $w_{\rho,x}$ for the each local RF $\varrho_{\rho,x}$ based on predicate ρ with

$$w_{\rho,x} \leftarrow \beta_x w_{\rho,x} + \sum_{y \in \Gamma(x)} \beta_y \kappa_{y,x}(\rho), \tag{6.25}$$

where β_x are mixing parameters local to each agent x, that are used to normalize the messages or bias the messages from certain agents if desired. In the general case of a graph with cycles, the messages may not converge but the intermediary result by computing Equation 6.25 at every iteration is usually useful in practice (Kok and Vlassis, 2006).

After applying Equation 6.25 with each $\beta_x = 1/N$, if the CG is connected without cycles, each local RF's weight will be the global average of all agents' weights for that RF. This is observable by considering each agent's node in the CG as the root of a tree. For the weight of each RF based on the same predicate ρ, the messages from Equation 6.24 propagate weights from the leaves to the root of the tree. At each iteration, a node sends its parent its weight and the sum of its children's messages from the previous iteration, i.e., it sends a sum of the weights within the sub-tree rooted at the node. A message from a leaf node containing its weight value will require H steps before arriving at the root as part of the message of one of the root's child where H is the height of the tree. Therefore after H iterations, Equation 6.25 will divide the sum of

all weights for the RF based on ρ by N, yielding the global average. As agents send messages to all their neighbours for each weight, the same convergence to the global average applies for trees rooted at every agent for every RF. Hence every agent will have the same averaged weights.

CGs are formed dynamically in a DEC-MDP. This indicates that they may be disjoint in different states and contain cycles. Hence, rarely do all agents form a connected CG. Furthermore, messages have to be sent for each RF's weight which may pose a problem when the number of RFs is large. To address these issues, an asynchronous message and update scheme is adapted as follows. In each time step, each agent x immediately sends the weight $w_{\rho,x}$ of the local RF $\varrho_{\rho,x}$ as a message to its neighbours. Upon receiving any message $w_{\rho,y}$ from a neighbour $y \in \Gamma(x)$, the agent immediately updates the weight of $\varrho_{\rho,x}$ using the convex combination,

$$w_{\rho,x} \leftarrow (1-\beta)w_{\rho,x} + \beta w_{\rho,y} \tag{6.26}$$

$$= w_{\rho,x} + \beta[w_{\rho,y} - w_{\rho,x}] \tag{6.27}$$

One advantage of this scheme is that agents may send any subset of its weights as messages to its neighbours as permitted by its communication channel. Furthermore, every agent uses the same mixing parameter β.

The update given in Equation 6.27 echoes that of the TD updates using a step size parameter α described in Equation 6.23. The intent of both equations is to compute a form of averaging over streaming data. In the case of TD learning, the data comes in the form of one step transition and reward, where the goal is to estimate the expected return. In the case of external generalization, the data comes in the form of neighbouring agents' weights and the goal is to estimate the mean of these weights.

In our experiments, we use Equation 6.26 together with the general relational TD learning algorithm for distributed RL and distributed CGRL for each agent given in Algorithm 6.1. Note that agents will have known the states and actions of their neighbours after exchanging messages with them when selecting actions at Lines 2 and 4.

Algorithm 6.1 General distributed relational TD learning algorithm for one agent

1: Observe local state s_x.
2: Select and perform action a_x based on a policy with neighbours.
3: Observe local state s'_x and reward r_x.
4: Select and perform action a'_x based on a policy with neighbours.
5: Perform local updates for RF weights using $\langle \mathbf{s}_x, \mathbf{a}_x, r_x, \mathbf{s}'_x, \mathbf{a}'_x \rangle$.
6: Exchange external generalization messages with neighbours and update weights.
7: Set $s_x \leftarrow s'_x$ and $a_x \leftarrow a'_x$.
8: If s_x is not terminal go to Line 3.

6.4 Experiments

Two sets of experiments are conducted to evaluate the proposed solution. The first set of experiments investigates the use of RFs on distributed RL in two domains: simplified soccer and tactical RTS. The predicates used are shown in Section A.1 page 230 and Section A.2 page 235 for soccer and RTS respectively. These domains were also used in the previous chapters and other works. The second set of experiments demonstrate the potential in fusing distributed relational TD learning with distributed CGRL in the tactical RTS domain. The experiments compared a number of RL methods from previous literature and earlier chapters, namely:

1. *Independent* refers to independent learners see Agent 1 in Section 5.5 page 126,
2. *Coordinated* refers to distributed RL, see Agent 2 in Section 5.5 page 126,
3. *CentCGRL* refers to centralized CGRL introduced in Agent 4, Section 4.4.1 page 88,
4. *DistCGRL* refers to distributed CGRL, see Agent 4 in Section 5.5 page 126,
5. *Coordinated*($\beta = x$) refers to distributed relational TD learning,
6. *DistCGRL*($\beta = x$) refers to relational DistCGRL.

For relational TD, both coordinated and DistCGRL learners are investigated with various values for the external generalization parameter, $\beta \geq 0$. $\beta = 0$ indicates that only internal generalization is used. All agents use ϵ-greedy policies. To evaluate online learning, we plot agent performance over the number of episodes. An important point to note is that all plots start from the point *one* and not at zero. Depending on the scale of each individual plot, this may refer to a different number of initial episodes.

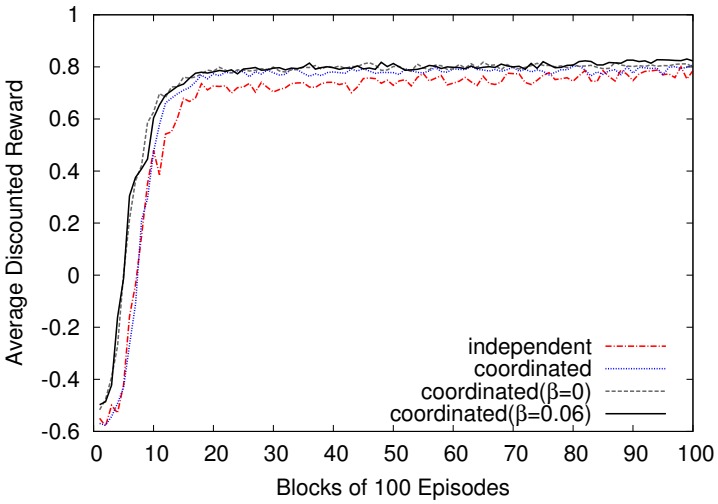
6.4.1 Results for Soccer

The distributed simplified soccer domain was previously described in Section 5.5.2 page 127. A 15×10 grid is used with two teams of 8 players each, grey and black. The objective is to score the first goal. Players may move in 4 directions, and pass or shoot when in possession of the ball. Communication is allowed only at a Manhattan distance of 5. The lines in Figure 5.1 page 110 describe the current CG based on different example soccer states. The RL players are pitted against a scripted opponent that defends their goal and counter-attacks once the ball is intercepted (see the Defensive player in Section 4.4.2 page 90). The total number of weights to learn for function approximation for each type of RL player is given in Table 6.1. The RL players use $\gamma = 0.99$ with decaying step size, $\alpha = \{10^{-2}, 10^{-4}, 0.998\}$, and exploration, $\epsilon = \{1, 10^{-2}, 0.998\}$. Parameters are written as $param = \{\text{initial, final, decay rate}\}$. Only the player that scores a goal receives a reward of 1 while a reward of -1 is evenly divided among soccer players nearest to their home goal when the opponent scores.

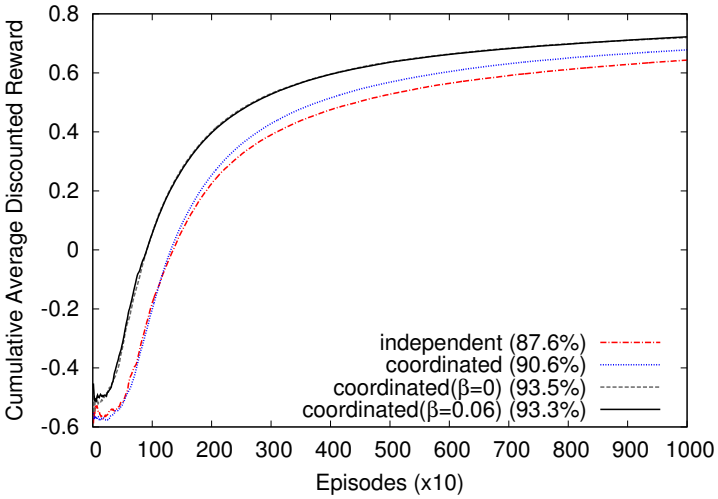
Figure 6.4 shows the results for online learning. Each plot is an average of 10 learning runs. In terms of overall performance in Figure 6.4b, we observe that internal relational generalization, coordinated($\beta = 0$), is beneficial to soccer over the coordinated player, while external generalization, coordinated($\beta = 0.06$), does not yield any benefit as its curve overlaps with $\beta = 0$. In Figure 6.4c we plot the change in reward, i.e., the reward for the relational players subtract the coordinated player for each episode. Both forms of relational generalization give rise to high benefit at the start. But, the benefit is lost over time for most settings of $\beta > 0$. This is expected as generalization allows fast initial learning, yet it is role specialization that gives a soccer team an edge over time. Nevertheless better performance is achieved while only using 66.15% of the quantity of

RL Method	# Weights	% of Coordinated
independent	1888	60.51
coordinated	3120	100.00
coordinated($\beta \geq 0$)	2064	66.15

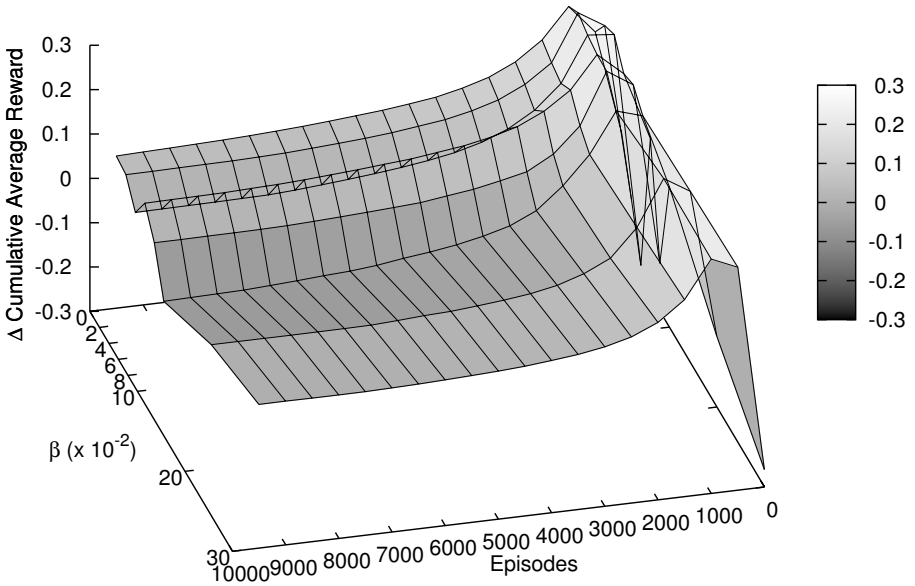
Table 6.1: Weights to learn for soccer. The last column expresses the number of weights as a percentage of the number of weights for the non-relational coordinated learner.



(a) Average discounted reward for the best learners



(b) Cumulative average discounted reward for the best learners, final win rate in brackets



(c) Change in reward over coordinated learner for various β

Figure 6.4: Soccer experiment results. 10,000 episodes with each plot averaged over 10 runs.

learning parameters of the coordinated player (see Table 6.1).

6.4.2 Results for Tactical Real-Time Strategy

For the tactical RTS domain, we use a 240×240 point based world as shown in Figure 1.1 page 3 and described in Section 5.5.3 page 131. There are 10 RL marines, each may move in 8 directions and shoot at any enemy within range, thus the joint action space is massive. Marines may only communicate within a range of 30 points. The RL marines are pitted against scripted aggressive marines that move towards their nearest enemy and start shooting (see Section 4.4.5 page 97). Rewards are only given to marines that are alive. The rewards are -1 per time step and 1000 for winning that are equally divided among the surviving marines.

For RTS, learners used constant $\alpha = 10^{-4}$ and $\epsilon = \langle 0.5, 10^{-2}, 0.998 \rangle$ while the CentCGRL learner is the same as described in Section 4.4.6 page 98. No discounting is used, i.e., $\gamma = 1$. The number of weights and RL parameters are shown in Table 6.2. Note that the use of internal generalization reduces the weights for the coordinated learner to 26.9% of the original. This is because most predicates are multi-agent, involving pairs of agents.

Figure 6.5 shows the results for the coordinated learners versus 10 enemy marines. From Figures 6.5a and 6.5b, it is clear that relational generalization is highly desirable in the tactical RTS domain where formation among marines is of paramount importance.

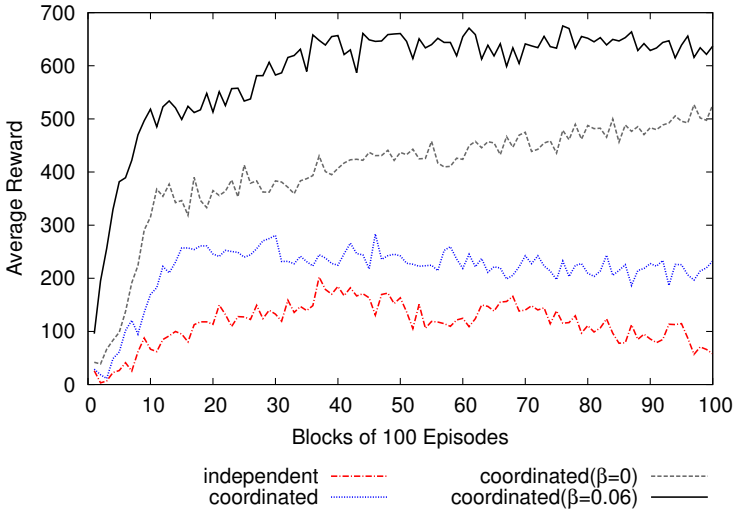
Type of RL	RL Method	# Weights Bottom	Top	Total	% of Non-Relational Coordinated RL
flat	independent	350	0	350	17.766
	coordinated	1970	0	1970	100.000
	coordinated($\beta \geq 0$)	530	0	530	26.904
two level	CentCGRL	53	45	98	2.103
	DistCGRL	1970	2690	4660	100.000
	DistCGRL($\beta \geq 0$)	530	450	980	21.030

Table 6.2: Weights to learn for RTS. Bottom indicates the weights for flat RL, e.g, the Q function, or the U function in CGRL while Top indicates the weights for top level function W in CGRL. The last column indicates the total number of weights expressed as a percentage of the total weights for the non-relational coordinated learners for each type of RL, flat and two level respectively.

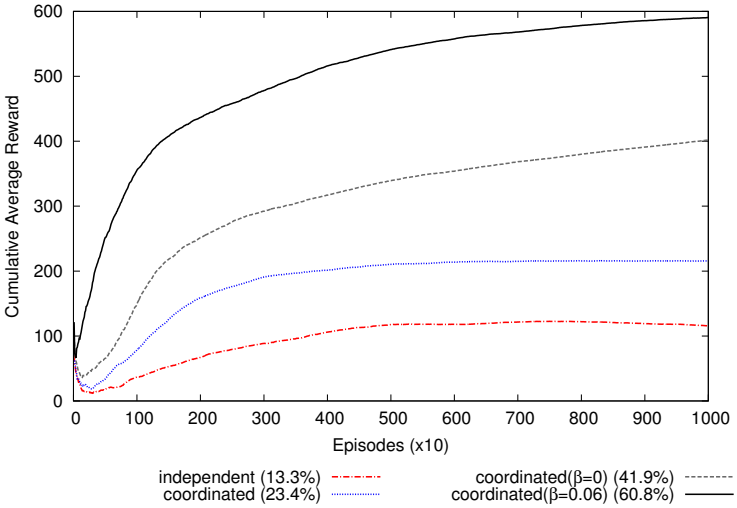
We see improvement from internal generalization in the coordinated($\beta = 0$) learner and further improvement when external generalization messages are exchanged when $\beta > 0$. Figure 6.5c shows the change in reward over the coordinated learner without relational learning. We observe that distributed relational TD learning is beneficial for most values of β and requires learning only a quarter of the number of weights for the non-relational case.

For the next set of experiments on distributed relational TD learning with DistCGRL (i.e., two level RL), we use the same tactical RTS setup of 10 RL marines versus 10 scripted marines. Figure 6.6 shows the results. We also include the result for the centralized version of CGRL, *CentCGRL* that uses RFs (see Figures 6.6a and 6.6b). We observe that DistCGRL without RFs is superior to coordinated($\beta = 0.06$), but its performance is nowhere near that of CentCGRL. However, once relational TD is introduced, DistCGRL outperforms CentCGRL and almost achieves a perfect win rate of 98.7% when $\beta = 0.06$, achieved with only 21% of the number weights in DistCGRL without RFs. The largest reduction comes from the top level function (see Table 6.2). The improved performance over the CentCGRL may be due to agent-decomposed rewards in contrast to the global reward scheme in CentCGRL. Agent-decomposed rewards provide fine-grained information for DistCGRL resulting in more precise updating. This effect was similarly observed and discussed previously in Section 5.5.4 page 134. Figure 6.6c plots the change in reward for the relational DistCGRL learners over the plain DistCGRL learner. Generally, external generalization further improves learning for most values of β.

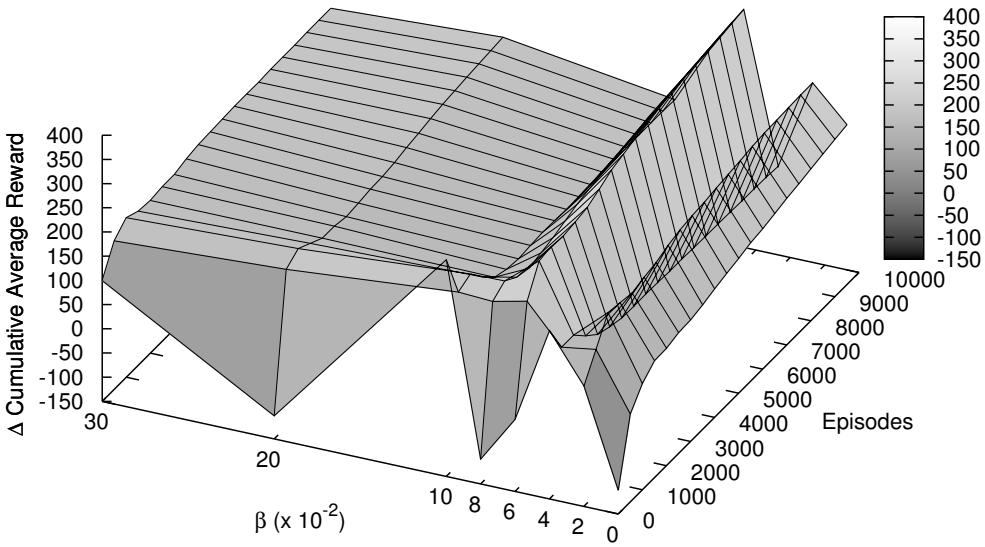
To further demonstrate the benefits of our relational TD concepts, we pit the RL marines against a harder scenario with 13 enemy marines. The results are shown in Figure 6.7. In this scenario, independent, coordinated, and DistCGRL methods without RFs are unable to learn to overcome their opponent within 10,000 episodes as shown in Figures 6.7a and 6.7b. However, relational generalization eventually enables DistCGRL to learn better policies than CentCGRL. This illustrates that rapid generalization is crucial for a fast-paced domain like tactical RTS whereby marines are destroyed easily.



(a) Average reward of best learners

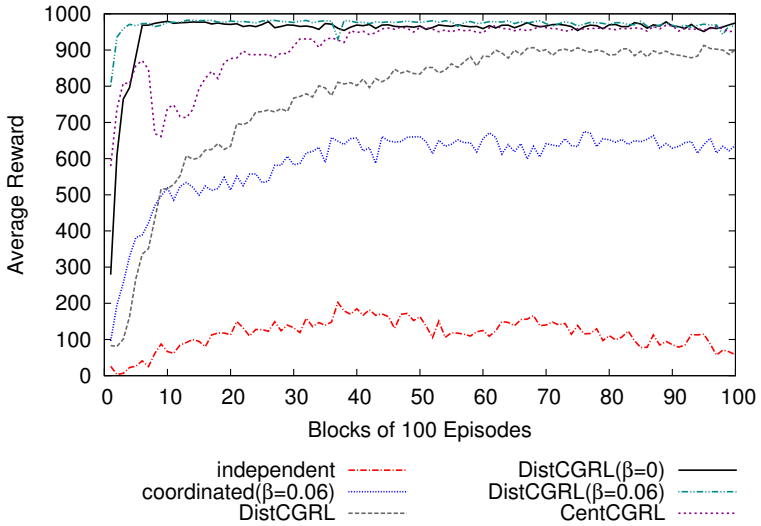


(b) Cumulative average reward of best learners, final win rate in brackets

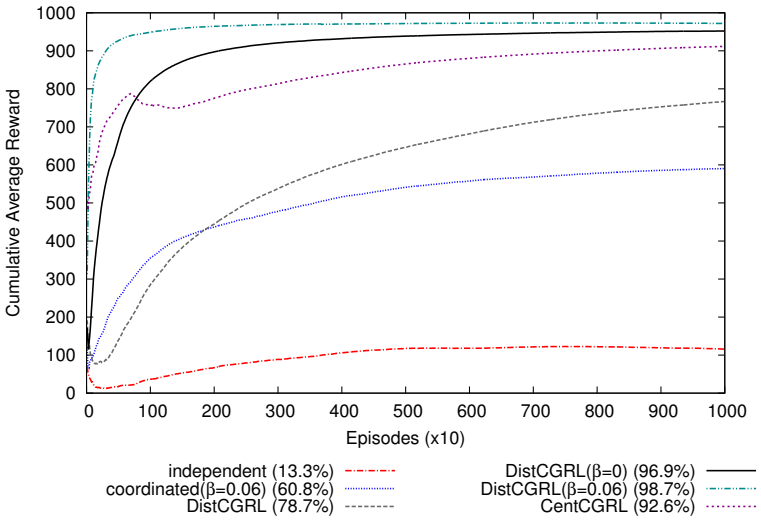


(c) Change in reward over coordinated learner for various β

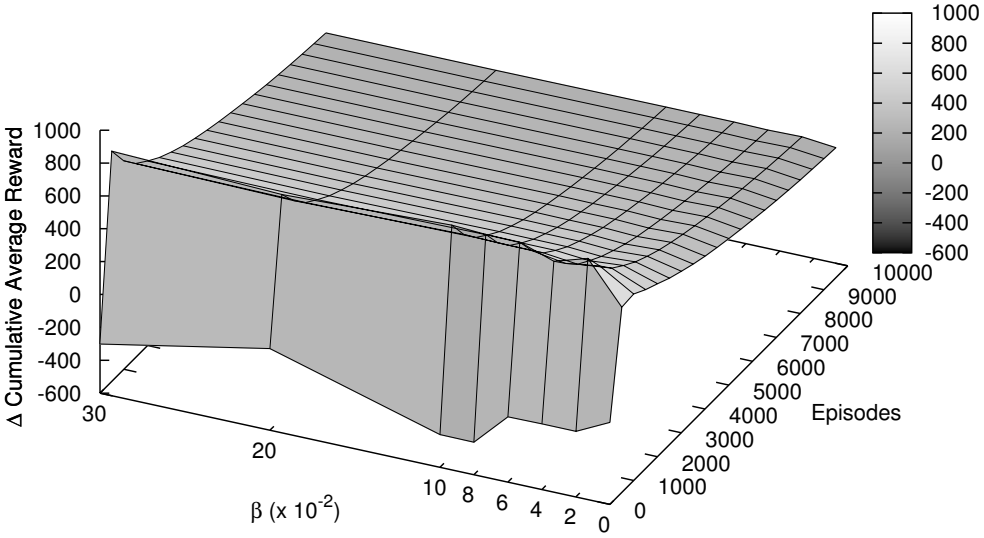
Figure 6.5: RTS experiment against 10 enemy marines using coordinated learners. 10,000 episodes with each plot averaged over 10 runs.



(a) Average reward of best learners

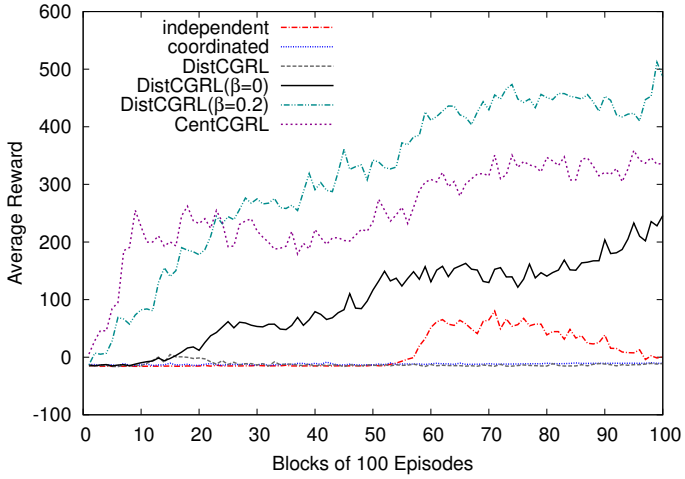


(b) Cumulative average reward of best learners, final win rate in brackets

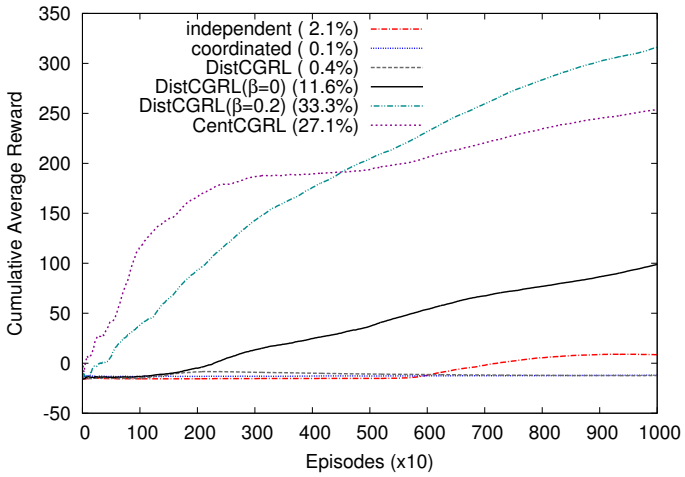


(c) Change in reward over DistCGRL learner for various β

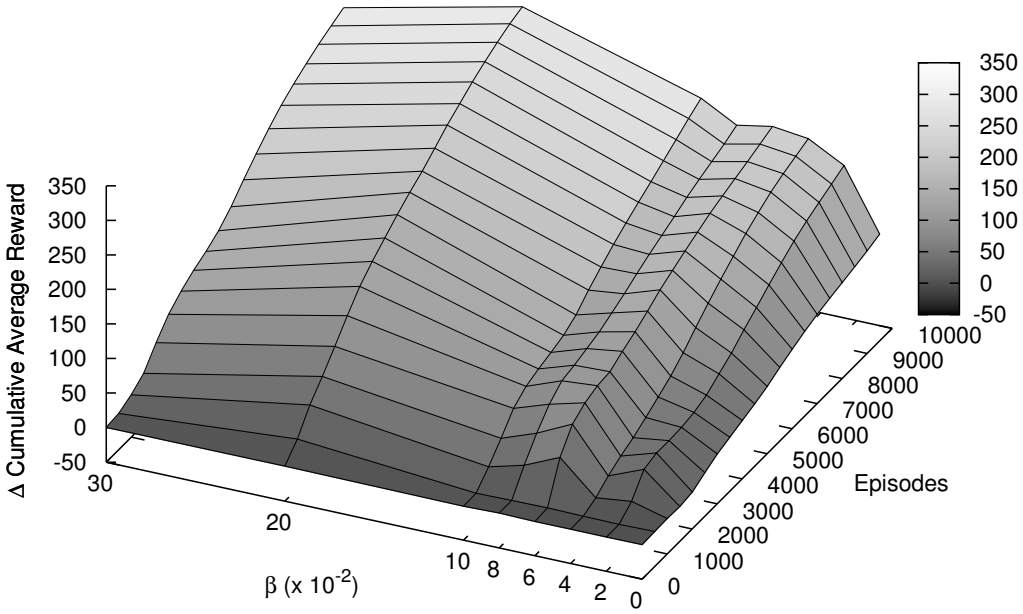
Figure 6.6: RTS experiment against 10 enemy marines using DistCGRL learners. 10,000 episodes with each plot averaged over 10 runs.



(a) Average reward of best learners



(b) Cumulative average reward of best learners, final win rate in brackets



(c) Change in reward over DistCGRL learner for various β

Figure 6.7: RTS experiment against 13 enemy marines using DistCGRL learners. 10K episodes with each plot averaged over 10 runs.

Consequently, this results in too few samples of winning episodes even with sophisticated exploration in DistCGRL without RFs. In Figure 6.7c, we see that external generalization improves learning throughout the episodes with the best improvement at $\beta = 0.2$.

6.4.3 Actual Runtime Results

We now turn to the actual runtime performance of the various RL learners. The experiments are conducted on the same platform described in Section 4.4.7 page 104. All RL learners were implemented as a single process and single threaded program.

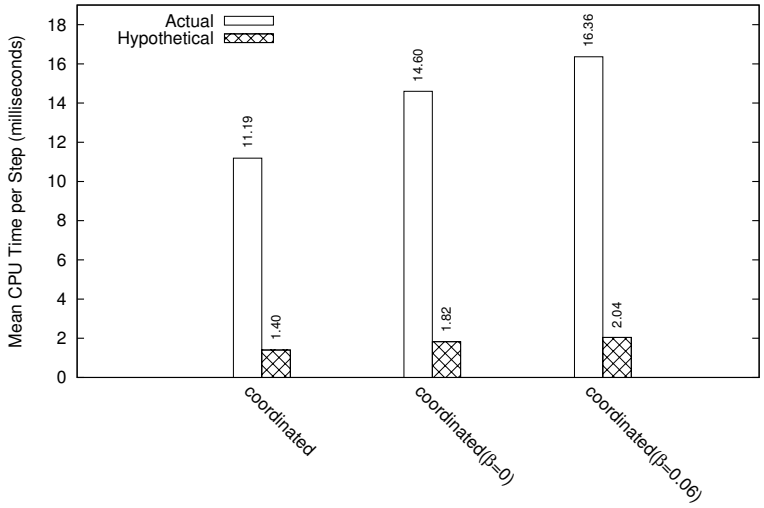


Figure 6.8: Runtime comparison of relational RL learners for the soccer game. The mean CPU time taken per step for 10,000 episodes averaged over 10 runs.

First, the runtime performance is measured on the soccer experiment previously presented in Section 6.4.1, Figure 6.4. The results are shown in Figure 6.8. Similar to Section 5.5.5 page 137, the *actual* runtime is the mean CPU time measured per time step while the *hypothetical* runtime is the actual runtime divided by the number of agents. The latter represents the loose lower bound for parallel implementations of the distributed RL learners. The results show that internal and external generalization each adds a small runtime overhead compared to coordinated RL when the number of agents are held constant.

Next, runtime performance is measured on the 10 versus 10 aggressive marines set up for RTS. Figure 6.9 shows the runtime performance of the various RL learners

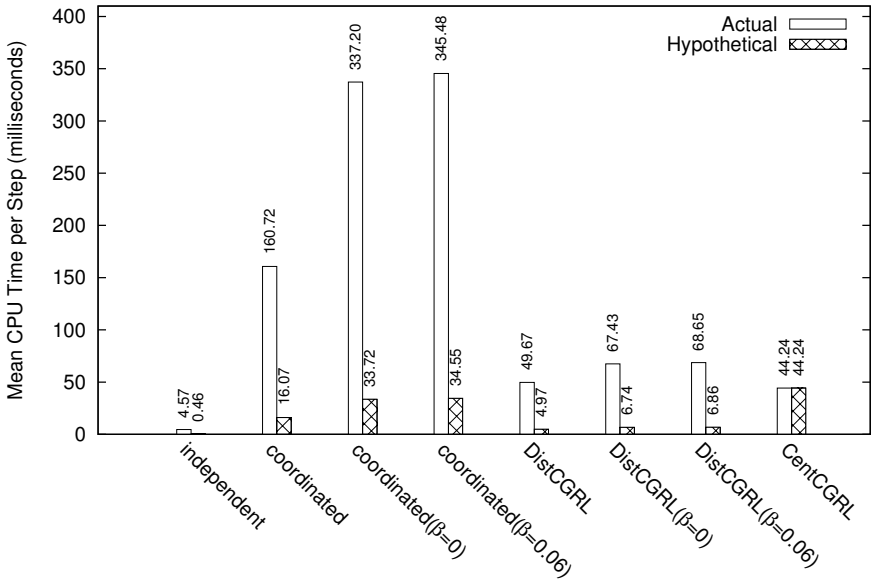


Figure 6.9: Runtime comparison of relational RL learners for 10 versus 10 aggressive marines. The mean CPU time taken per step for 10,000 episodes averaged over 10 runs.

depicted in Figures 6.5 and 6.6. Overall the two level learners (DistCGRL) are more time efficient than the flat learners (coordinated). This is due to the pruning effect of the CCs (see Section 4.4.7 page 104).

The runtime performance of the coordinated RL learners in Figure 6.9 shows that the runtime increases when internal generalization is applied in coordinated($\beta = 0$) and further increases when external generalization is used in coordinated($\beta = 0.06$). Unlike the soccer game's results in Figure 6.8, this increase in runtime is mostly due to the improvements in policy learned as shown in Figure 6.5. Better policies result in more marines surviving and consequently, slower joint action selection. In contrast, the poorer policy of the coordinated learner results in more losing episodes where all marines are destroyed. Hence better policies result in a higher runtime cost for the coordinated RL learners in the RTS domain.

For the DistCGRL learners in Figure 6.9, the differences in runtime performance with relational generalization are less pronounced. We observe a small performance overhead when comparing DistCGRL($\beta = 0$) and DistCGRL($\beta = 0.06$) respectively with non-relational DistCGRL. The relational DistCGRL($\beta \geq 0$) learners have correspondingly performed better as shown in Figure 6.6. Therefore in RTS, distributed

relational generalization is able to improve learning with a corresponding impact on runtime performance partly due to policy improvements.

6.5 Discussion

The experiment results indicate that distributed relational generalization is feasible in practice on various domains, and general enough to be applied to different RL methods. We have used a coarse-grained approach towards the specification of local RFs for internal generalization by applying it to all predicates. Results are expected to improve further if the user carefully defines a subset of predicates to apply internal generalization, especially for domains that require specialized roles, e.g., soccer. Furthermore, the results in Figure 6.4 suggest that for domains like soccer, generalization should be performed early but decreased over time to allow agents to develop specialized value functions.

For domains whereby specialization is less important, e.g., in RTS, relational generalization allows experience to be shared rapidly. The result is better goal achievement while having less parameters (weights) to learn and store. In some cases the performance improvement is dramatic as we have seen in Figures 6.6 and 6.7 where relational DistCGRL uses only 21% of the number of weights of a propositional DistCGRL approach.

Guestrin et al. (2003) observed that the homogeneity or heterogeneity of the domain may affect the learning efficiency gains from relational generalization in solving MDPs. In light of this, distributed relational TD learning may be preferred for a centralized application when it is feasible to provide agent decomposed rewards from the MDP. From this perspective, the β parameter in external generalization gives the user some control over the amount of relational generalization for each agent's learning, i.e., agents may retain some form of individuality. Results in RTS have shown that it is possible to outperform even centralized approaches with RFs with certain values of β. Last, most settings of β demonstrate improvement over learning without RFs, hence the user need

not fine-tune the value of β if the current goal achievement is acceptable.

Relational generalization for RL arose with the need to compactly represent state and action spaces in combinatorial domains such as the blocks (un)stacking domain (Džeroski et al., 2001; Kersting and De Raedt, 2004). Since then, many relational RL methods have focused primarily on state predicates and model-based RL. Asgharbeygi et al. (2006) allowed RFs to be used in conjunction with other kinds of features, and attempts have been made to increasingly address the distributed multi-agent setting (Croonenborghs et al., 2005). Closely related to relational generalization is the concept of feature templates whereby a feature may be generalized by introducing some parameter like how the relative layout of some pieces in certain board games like Go may exhibit similar importance for goal achievement when the layout is rotated (Silver et al., 2007). One of the first uses of relational generalization for a factored joint action space was in planning in RTS games (Guestrin et al., 2003) that generalized over classes of agents. In this work, we return to the family of earlier methods that make use of model-free learning, and predicates involving both the state and actions of agents, albeit with the new definition of local RFs for function representation.

Our proposed approach to distributed relational TD learning is different from other multi-agent RL works that seek to incorporate parts of the local value functions of other agents within each agent. The primary difference is the relational semantics that determine the form of learning experience to be shared among agents. In Schneider et al. (1999), agents' updates incorporate tabular Q-values from other agents' local value functions based on the current state and action values. However, in our approach agents may share learned weights for RFs that are not limited to the current state and action.

Previous works have shown that averaging tabular action value functions between agents can improve learning performance when some agents are experts (Ahmadabadi and Asadpour, 2002; Araabi et al., 2007). However, this is prohibitive for large action value functions and heterogeneous agents that do not share the same state-action pairs. Conversely, our approach allows sharing of weights among agents with different

individual state and action domains, so long as they share relational semantics that is expressible by some common predicates. Ponsen et al. (2010) investigated learning a multi-agent communication policy for domains where communication between agents is costly but expert agents exist. In contrast, our approach begins with no known expert.

We have used the general linear function approximation approach to model value functions. This is more akin to the work in Asgharbeygi et al. (2006) as opposed to relational regression tree family of methods used in Džeroski et al. (2001); Kersting and De Raedt (2004); Croonenborghs et al. (2005, 2007). The benefits of our approach include the smoothness of the function, the fast update time that is linear to the number of weights, and the ability to include non-predicate based features. Furthermore, in the event that the user specifies poor predicates, the effect is similar to having useless features in linear function approximation that do not sufficiently distinguish (i.e., correlate with) state action pairs.

The relational predicates used in this work are provided by the user. No distinction was made between the base level literals, that encode basic relational representation for states and actions, from higher level predicates such as $NotAligned$. Both are regarded as similar background knowledge since we do not assume a readily encoded domain is available, e.g., from the General Game Playing framework (Asgharbeygi et al., 2006). Works in centralized single agent relational TD have discussed possible solutions for automated construction of higher level predicates from lower level predicates (Kersting and De Raedt, 2004; Asgharbeygi et al., 2006). Adapting these ideas to the communication requirements of the distributed case will reduce dependence on the human expert.

6.6 Conclusion

To the best of our knowledge, this is the first work to bring the generalization capabilities of relational TD learning to the distributed case with dynamic communication. This was achieved using two parts, internal and external generalization. Internal generalization creates local RFs from partially bound predicates, enabling individual agents to

generalize over their interactions with various agents and greatly reduce the number of parameters to learn in the system as a whole. External generalization involves message passing of learned parameters that share relational semantics, allowing different groups of agents to share their experience with others. Experiment results on two domains show that the proposed methods leads to better online learning through rapid generalization of experience. Exceptional results are achieved when the domain involves homogeneous agents like the tactical RTS domain. Distributed relational TD was combined with distributed RL and CGRL to illustrate the applicability of this approach on various value function based RL methods. Furthermore, the competitive results with the centralized case shows that the challenge of distribution can be resolved while retaining much of the benefits of relational generalization.

Chapter 7

Application in Automating Retinal Image Analysis

In this chapter, we investigate a preliminary application of multi-agent reinforcement learning (RL) to the analysis of images of the human retina (retinal images). We describe the motivation for computer assisted grading of the retinal images as a means to extract features for further analysis in hope of finding disease indicators. Then, we present our attempt at further automating this process by introducing a post processing step whereby a system modifies the detected vascular structure. This post processing step interacts with the existing system in a feedback loop and is trained using RL on the problem domain formulated as a collaborative multi-agent Markov decision process. Results are evaluated on a real world data set from a population based study. Analysis shows that our system is able to improve images where the vascular structure is poorly identified at the start. We further identify the shortcomings of this approach and suggests how the system can be improved.

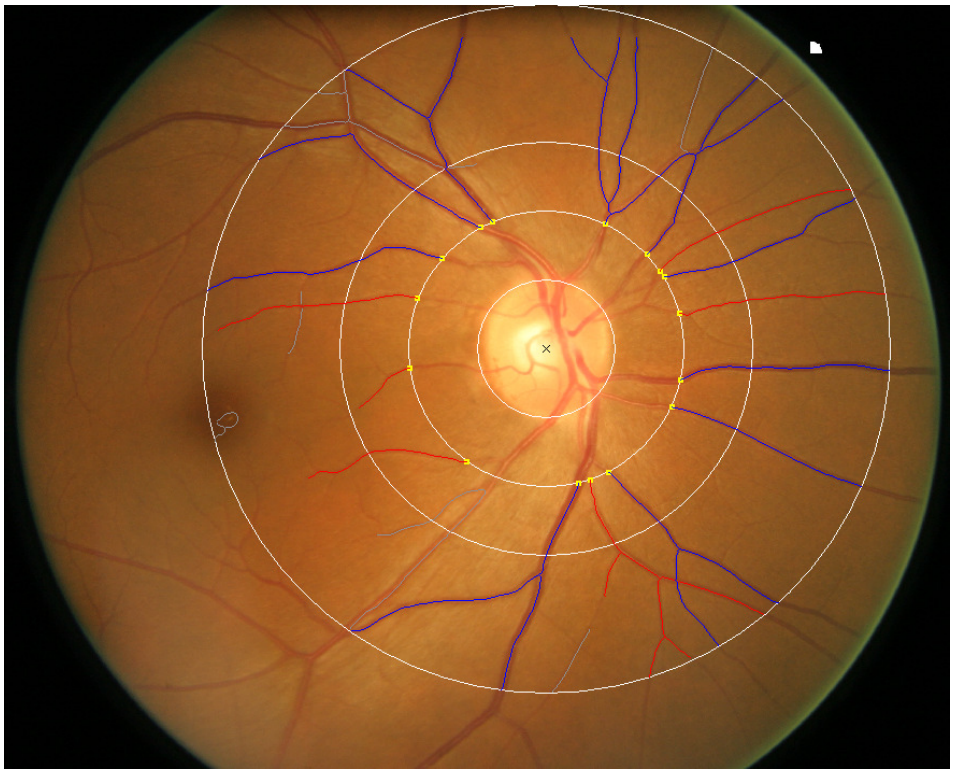
7.1 Motivation

A retinal image provides a snapshot of what is happening inside the human body. In particular, the state of the retinal vessels has been shown to reflect the cardiovascular condition of the body. Many measures to quantify retinal vascular structure and

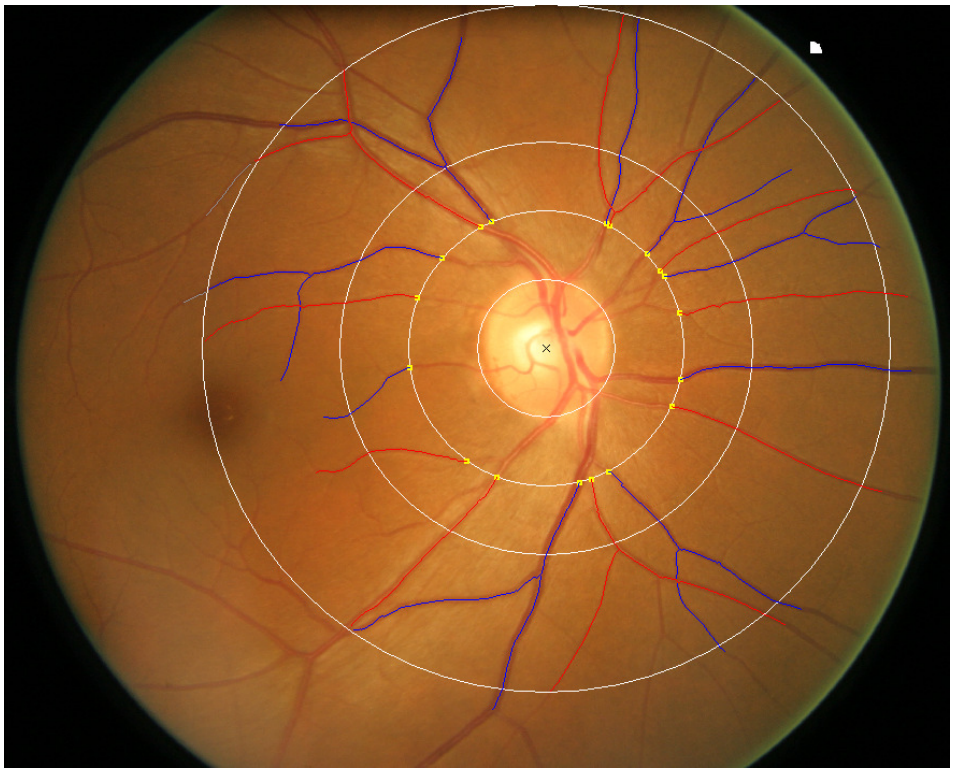
properties have been proposed (Patton et al., 2006; Cheung et al., 2012). Through population studies, some of these measures have been shown to provide good predictive capabilities for diseases such as high blood pressure, stroke, and hypertension (Wong et al., 2004b; Cheung et al., 2011a,b). For example, the central retinal artery equivalent (CRAE), a measure of the diameter of the six largest veins (by diameter) in the retinal image, has good correlation with hypertension (Wong et al., 2004a, 2006a; Liew et al., 2008), while the central retinal vein equivalent (CRVE), a measure of the diameter of the six largest artery, has stronger correlation with coronary heart disease and stroke (Wong et al., 2006b; McGeechan et al., 2009a,b). Likewise, the junctional exponent (JE), a measure of the relative diameters of parent and daughter blood vessels, can indicate diseased coronary arteries if there is deviation from the theoretical optimum value of 3 (Chapman et al., 2002; Witt et al., 2006). These measures require the accurate extraction of individual vessels in a retinal image and this involves: (a) *segmentation* of the vascular structure, (b) *separation* of vascular structure and linking segments into individual vessels, and (c) *classification* of vessels as arteries or veins.

Retina grading is the process where a trained human operator, a grader, makes use of a computer software to identify and extract properties of a retinal image. For the purpose of quantifying vascular structure, a grader works with the results of automated vascular extraction and correct them as necessary to provide high quality results. We have developed the Singapore Eye Vessel Assessment (SIVA) system that is the current state-of-the-art computer assisted system used in real world population studies involving thousands of patients (Cheung et al., 2010, 2011a,b, 2012). The advantages from a more accurate automated vascular extraction system are obvious, grader man hours may be reduced thereby increasing productivity. Furthermore, if automated vascular extraction is reasonably accurate, we may no longer require human graders when extracting measurements for certain purposes, e.g., for input to a noise tolerant disease classifier used to screen out high priority patients for early consultation with medical specialists.

The current SIVA system has room for improvement with respect to the extracted vascular structures before it can be readily used as a fully automated system. Inaccu-



(a) Initial extracted vascular structure



(b) Human graded result

Figure 7.1: Example of automated and human graded vascular structure in terms of vessel centre lines. The zone of interest is the ring spanning the inner most white circle to the outer most white circle. Yellow boxes are vessel root points. (a) shows initial result from the fully automated extraction with errors. (b) shows the result from a human grader after working on (a) where the grader followed a grading protocol that only connects the first bifurcation of each vessel.

racies in vascular segmentation affects the resulting measurements and may arise from noise in the image or the inherent difficulties in extracting vascular structures. Such inaccuracies at the segmentation stage are further compounded in vessel separation and classification. To illustrate, Figure 7.1 shows vascular structures extracted from an image. Figure 7.1a show the initial extraction results where there are errors due to separate vessels appearing close together and other complications. These errors are mostly between 10 to 2 o'clock from the centre of the optic disc marked with a cross. Figure 7.1b shows the same image corrected by a grader based on the initial extraction results. Hence, more work is required before a fully automated system may perform competitively with a human grader.

Reinforcement learning has been used in a number of real world applications for various medical image analysis problems. Sahba et al. (2008) reports an application in segmenting the prostrate from ultrasound images, Chitsaz and Seng (2009) investigates RL for segmentation of computed tomography (CT) images of the brain, and Wang et al. (2011) detects anatomical contours in magnetic resonance (MR) images. However, the retinal vascular structure has a more complex shape than these previous works. To the best of our knowledge, there are no RL applications in extracting the vascular structure from retinal images. Apart from segmentation, retinal image analysis requires the correct separation of individual vessels that branch (bifurcate) and cross each other, as well as accurate classification of vessels as arteries or veins. This complexity suggests that an RL solution will have to grapple with a very large state and action space common to multi-agent problems. Hence, the interest in attempting the multi-agent RL solutions presented in this thesis to address this problem domain.

7.2 Aims & Approach

This chapter proposes a multi-agent RL approach as a post processing step to the current automated extraction methods to improve the quality of the extracted vascular structures. Apart from handling the large state and action space, a multi-agent approach may

take advantage of coordination in editing complex structures. In contrast, a single agent approach will require actions to be taken over multiple time steps to achieve the same effect. Furthermore, parallelism may be exploited with multiple agents especially when editing vessels that are far apart.

Currently, human graders edit vascular structures in a feedback loop. That is, human graders make use of the displayed extracted vascular structures to edit the vessel segmentation. This in turn triggers a rerun of the routines to reconstruct the extracted vascular structures. The aim is to create an RL system that mimics the human grader, operating in a similar feedback loop. This approach is modular as it does not interfere with the individual algorithms used to perform various tasks. Furthermore, the system as a whole can benefit from improved algorithms in the future by simply replacing those algorithms. In contrast to the online setting in the earlier chapters of this thesis, the RL system will be trained in a offline phase on human graded vascular structures and evaluated against other human graded images. Furthermore, the RL system will have to transfer what it has learned from the training images into the testing images.

There are a few challenges in this application. First, a balance has to be struck when defining the agents between their capabilities (actions), and the amount of information (states) required to perform them. Then, the reward function has to be one that captures the concept of a correct vascular structure while enabling agents to learn low level actions effectively. Last, discerning predicates have to be devised for linear function approximation and coordination constraints (CCs) in the case of coordination guided RL (CGRL).

7.3 Computer Assisted Retinal Grading

Computer assisted retinal grading is carried out using the SIVA system shown in Figure 7.2. The general work flow for a grader for each retinal image is given in Figure 7.3 and it consists of the following automated steps.

1. *Optic Disc Detection:* Automated detection of the optic disc landmark using a

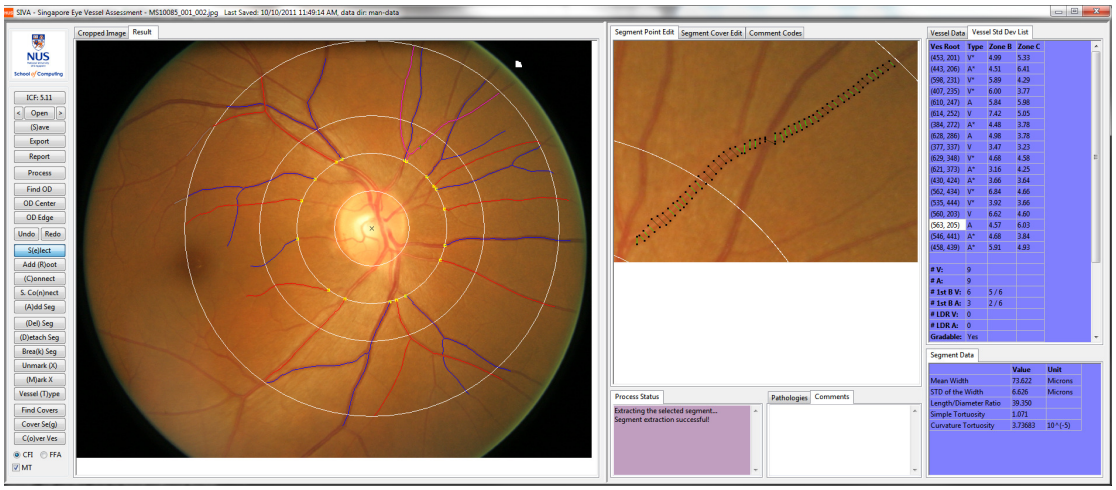


Figure 7.2: The SIVA System for computer assisted retinal grading.

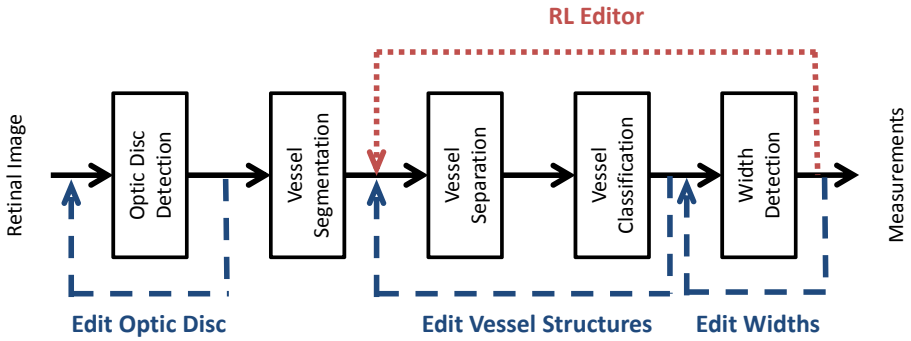


Figure 7.3: Work flow of the SIVA system. Feedback loops from human grader edits in dashed lines. Feedback loop for RL system in dotted lines.

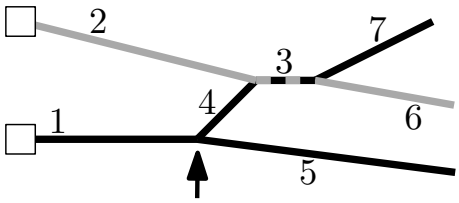


Figure 7.4: Example of bifurcation and crossover shared segments. There are two vessels, grey and black, with white boxes as their root points. The black arrow at the confluence of line segments 1, 4, 5 indicates a bifurcation (branching) while line segment 3 is shared between the two vessels.

modified template fitting method from Pallawala et al. (2004). Vessels will be measured within the zone of interest defined by the disc radius and location as shown by the white circles in Figure 7.1.

2. *Vessel Segmentation:* In this step we detect the centre lines of the vessels in the retinal image using an adaptation of a trench detection algorithm (Garg et al., 2007). Then, the lines are cropped to be within the zone of interest.

3. *Vessel Separation:* This step involves distinguishing which line segments belong to the same vessel. Line segments starting near the beginning of the zone of interest are marked as root segments shown as yellow boxes in Figure 7.1. From here the problem of linking up line segments that belong to the same vessel is cast as the optimization problem of finding the best forest of vessel trees from a graph. After this step, line segments either belong to some vessel, or are orphans (e.g., shown in grey in Figure 7.1a). Figure 7.4 shows a conceptual diagram of two separated vessels. The end result may include shared segments that are due to one vessel being partially occluded by another as depicted by line segment 3.
4. *Vessel Classification:* This step involves labelling the individual vessels as either a vein or an artery. A clustering approach based on the intensity of vessel pixels is used to classify the two types (classes) of vessels.
5. *Width Detection:* In this step, the system detects the width (diameter) of the vessels based on the centre lines. Bad width samples are automatically discarded using an optimization procedure. The vascular structure is now ready to be queried for measurements of interest.

The grader interacts with the system at various steps as shown by the three feedback loops represented using dashed lines in Figure 7.3, namely: adjusting the optic disc result, editing the vessel segmentation and classification, and modifying the width of the segments. In editing the vessel segmentation, graders may break or detach line segments from existing vessels, add or delete new line segments, (un)mark hints to vessel crossover locations for vessel separation, and add new vessel roots.

The primary interest is in correcting the vessel centre line segmentation and vessel classification. This is expressed through the dotted line feedback loop at the top in Figure 7.3. After these edits are made, the system goes through the automated process of vessel separation, classification (if new vessels are created) and subsequently, width detection. Similar to the human grading process, the separation of vessels is not modified. However, in contrast to the human grading process, we also do not modify the detected vessel widths and instead rely on automated width detection. The RL system

interacts with the other algorithms over time through this feedback loop resulting in a sequence of edit operations. The next section presents an RL system that learns to edit the vascular structures for improved extraction.

7.4 Retinal Grading As a Multi-Agent Markov Decision Process

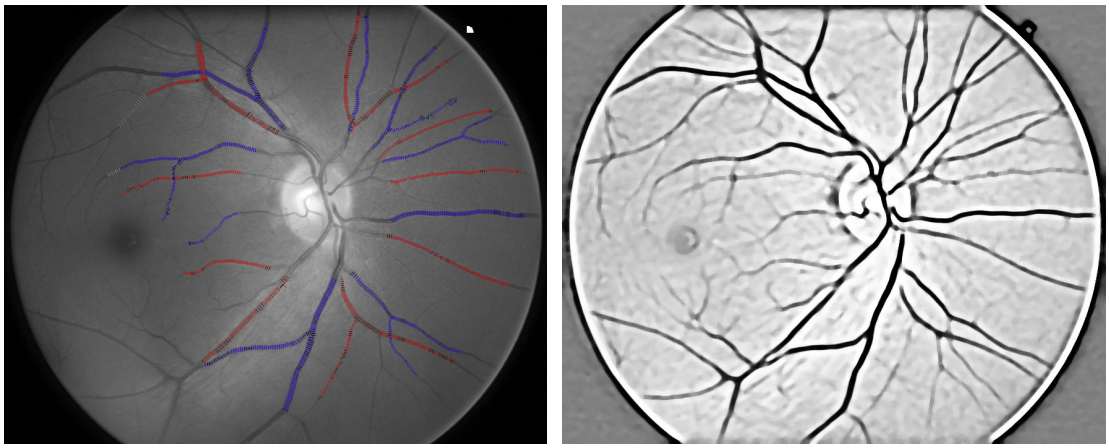
Before detailing the multi-agent Markov decision process (MDP), we first describe the formulation of editing the vascular structure as a multi-agent problem. The goal of the system is to produce a modified vascular structure that matches the *gold* standard, i.e., the vascular structure edited by a human grader. To do so, we treat the entire retinal image as a grid world and disperse a number of editor agents within it. These agents may move around various *locations of interest* in the image that are determined by the current state of the vascular structure. At each time step, each agent may take a movement action to move to another location of interest, or an edit action that modifies the vascular structure with respect to its position. The state transition is handled by the environment, i.e., the vascular extraction system, that processes the new positions of the agents, the edits made, and recreates the modified vascular structures by calling the vessel separation and width detection routines. At the end of a pre-determined number of time steps, the episode ends and a terminal reward is given to the agents.

The above multi-agent problem is formulated with a number of learning and implementation considerations. First, the number of mobile agents is fixed as opposed to identifying every location of interest as an agent. In the former, the quantity of agents do not vary with the state unlike the latter. This simplifies implementation and allows the user to limit the time required to compute the selection of joint actions by limiting the number of agents. Next, the formulation introduces sparsity into an otherwise massive state space. An agent editing a vessel in one part of an image need not be concerned with a completely separate vessel on the opposite side of the image. Furthermore, the agent may only make edits in its vicinity. This introduces a *local context* for

each agent's actions that provides a form of state-action space abstraction. Assuming that editing a vessel is invariant of its orientation, this local context reduces the number of learning parameters for the system as a whole by allowing the design of features for function approximation with respect to an agent's context. Last, by defining locations of interest to be the positions that agents may move to, learning is further directed to areas where editing the vascular structure should make the most impact on achieving the goal. This allows most of the time steps to be spent on edit actions instead of minute grid by grid movement actions that have little effect on goal achievement.

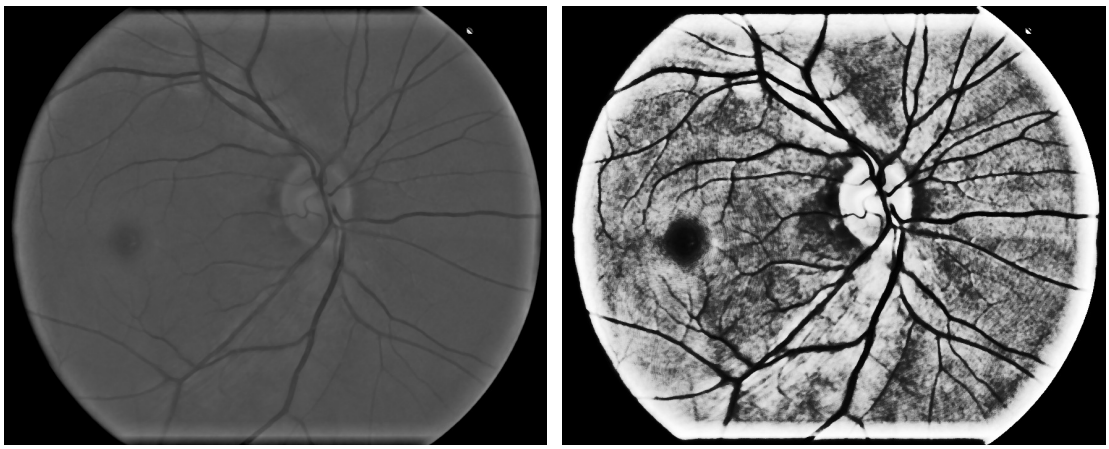
The motivation for using multiple agents as opposed to a single agent approach is an obvious one. To illustrate, revisit Figure 7.1. Notice that while there are vessels that are connected to each other due to vessel crossovers, there are entire groups of vessels that remain separate. Hence there are opportunities for parallelism. Furthermore, multiple agents permits the encoding of coordination within a time step that is akin to declarative predicates, whereas in a single agent case the step-by-step decomposition of procedural knowledge will be required.

The multi-agent, factored MDP formulation given in Section 2.1 page 13 is used to model the problem domain. As illustrated in Figure 7.3, the RL editor is a centralized controller of multiple agents. The RL editor interacts with the existing components for vascular extraction over time in a feedback loop. Edits made in one time step affects future edits. The Markov assumption is expected to hold in this case as all the information required to make an edit in one time step is in the present state. This holds in most circumstances, at least for human graders that are usually able to identify errors in the vascular structure based on its current state. The environment that processes the edits is a fully deterministic one. However, there is no deterministic transition model. Hence, the environment serves as a state generator where model-free RL may be employed to discover solutions. Furthermore, the problem is treated as an episodic MDP. Agents are allowed to take actions for a certain number of time steps. After which the episode terminates and a terminal reward is given. Rewards are zero for the other time steps. The next few sections describe the following: the state space, the actions that include



(a) Vessel widths

(b) Gradient map of intensity



(c) Illumination corrected image

(d) Histogram equalized

Figure 7.5: State information for vascular extraction. The widths in (a) change based on edits made by agents, but (b) and (d) do not. (d) is the histogram equalized image of (c).

the definition of the locations of interest, the reward function that attempts to quantify the degree of matching between the current vascular structure and the gold standard, and last, the details for our proposed RL solution.

7.4.1 State Space

The state space consists of two types of information. The first type of information is static and does not change based on actions taken by agents. This includes the intensity values in the retinal image, the optic disc, and the zone of interest. The environment that the agents operate in is limited to the zone of interest. In addition, we have also added an intensity gradient map (Kang et al., 2007), and a histogram equalized illumination corrected image. Respective examples of these images are shown in Figure 7.5b and

Figure 7.5d.

The second type of state information is dynamic, i.e., it varies due to agent actions. Besides the agent positions, we have the current vascular structure that includes: the root points of vessels, the vessel type (artery, vein, or orphaned), the centre lines of individual vessels as shown in Figure 7.1, the automated vessel width detection result as shown in Figure 7.5a, and a map of crossover hints. The last is a binary map that provides a hint to the vessel separation routine that a crossing of vessels is near some vessel pixel.

7.4.2 Actions

Each agent has a dynamic set of actions available depending on the current state and position it is in. Therefore, actions are limited to the local context an agent is in. By having multiple agents present in the system we may coordinate the actions between agents. The types of actions available to each agent are divided into two categories, namely: movement actions, and edit actions. Also, an *idle* action is always available for each agent that essentially does nothing.

First, we present the edit actions. Recall that a vessel is a tree of line segments. Let the vascular centre line structure be referred to as the *line image*. All edit actions may only take place on points in the line image. For example, the red, blue and grey lines in Figure 7.1 together constitute the line image. Agents may take the following actions with respect to their current position (two dimensional point) in the retinal image:

Add Root This action will mark a new line segment end point as a vessel root and is only allowed near the inner circle of the zone of interest. After a new root is added, the environment will create a new vessel based on it in the next state.

Break Segment This action removes the point in the line image that the agent is on, and any adjacent points to the agent in its 8-neighbourhood (see white pixels in Figure 7.6).

Detach Segment This action disconnects a line segment from the nearest junction to the agent as shown in Figure 7.7.

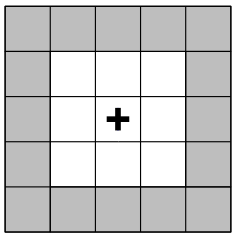


Figure 7.6: Example of 8-neighbourhood and the set of add segment actions. Each square is a pixel. The agent occupies the pixel with a cross. White pixels indicate the 8-neighbourhood of the agent. There are 15 Add Segment actions indicated by the grey pixels. Each Add Segment action results in a line drawn between the agent and one of the grey pixels.

(a) Before **(b)** After

Figure 7.7: Example of the detach action. Grey lines are points in the line image, the cross indicates the agent.

Add Segment This is a set of 15 actions that adds a line segment to the line image originating from the agent's position and ending at any of the 15 locations shown in the grey boxes in Figure 7.6. The agent will move to the end location.

Toggle Vessel Type This action changes the type of the vessel the agent is on from vein to artery and vice versa. It does nothing if the line segment is not part of a vessel, i.e., is an orphan segment.

Mark & Unmark Crossover These actions mark or unmark a point as a crossover hint for the vessel separation algorithm.

Central to the definition of the movement actions in each state are the locations of interest. As previously mentioned, agents move from one location of interest to the next. This narrows down the massive state space of the entire vascular structure to a smaller number of locations where we think the agent edit actions are best carried out. The following rules define points (pixel locations) on the vessel centre lines that are considered locations of interest. The locations of interest for the vessel centre lines of the image in Figure 7.1 is displayed in black in Figure 7.8.

1. The first and last point, i.e., the end points, of every line segment. Such points

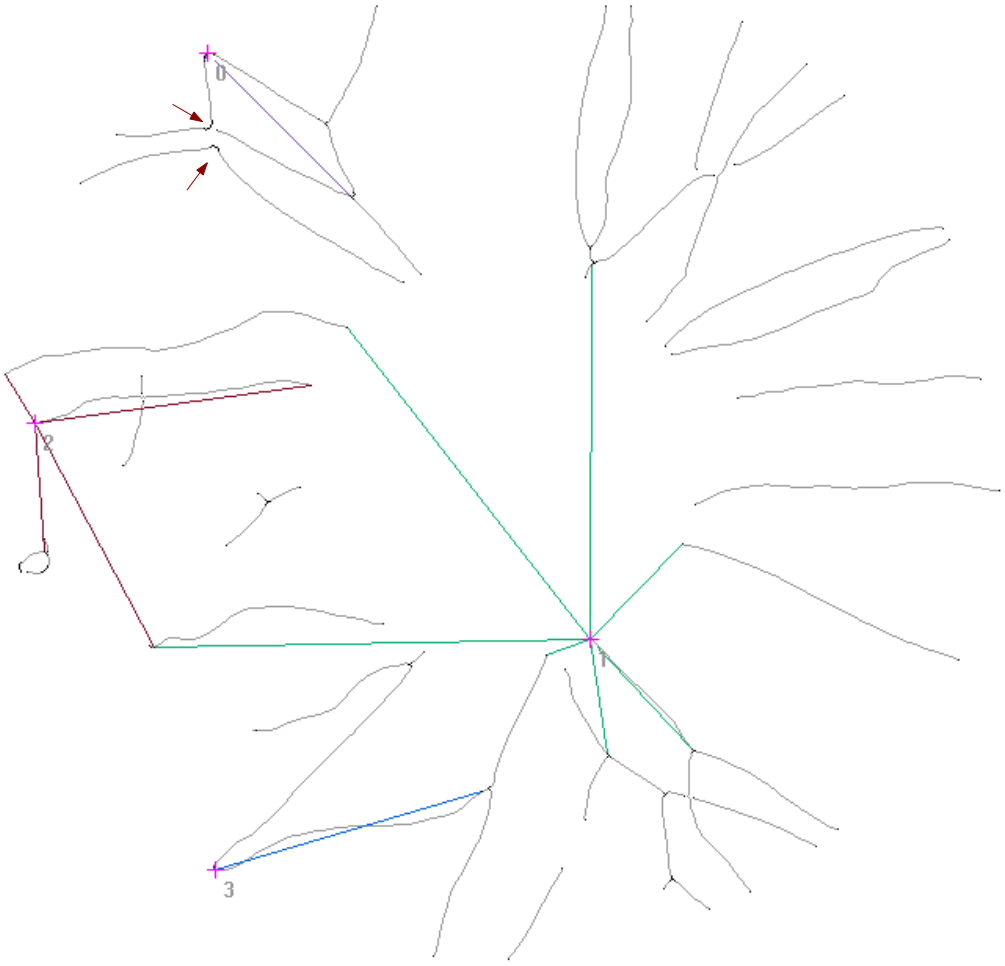


Figure 7.8: Locations of interest and movement model corresponding to the vessel centre lines in Figure 7.1. Grey lines are vessel points, black points are locations of interest, purple crosses with numbers are agents and other coloured lines emanating from each agent depicts the available movement actions. The arrows highlight locations of interest identified by the acute angle of the line segments.

may be candidates for the Add Root action or may be extended by Add Segment.

2. Junction points that are points on the line image that are at the confluence of three or more line segments. These points allow agents to destroy junctions using Break or Detach Segment actions. Furthermore, agents may mark or unmark such points as crossover hints to the vessel separation algorithm.
3. Each point on a line segment whereby the angle between lines rooted at the point is less than $120°$. Vessels seldom bend at acute angles. Hence such points may indicate erroneously linked segment lines due to noise and are candidates for Break Segment. Additionally, such points may be due to a missing line segment at a bifurcation (see the arrows in Figure 7.8) that the Add Segment actions may

remedy.

With the description of locations of interest, we now explain how to compute the movement actions using the following movement model. In each state, each agent extends a ray in one of the eight compass directions as shown by the white pixels surrounding the agent in Figure 7.6. The ray extends through the inner circle in the zone of interest but is bounded by the outer circle. If the ray encounters a location of interest, an action is added to the agent's set of movement actions that moves the agent to that location (point). If the ray encounters a point, p, on a segment line, the line is followed in both directions until each encounters a location of interest. Then, the nearest of the locations to p is added as a movement action. In all other cases, no movement actions will be added. Hence from this movement model, there are at most eight movement actions for each agent in every state.

In Figure 7.8, coloured lines emanating from each agent (purple cross) illustrate its movement actions in the current state based on the movement model described above. Typically the movement actions will allow an agent to move back and forth between locations of interest within connected groups of line segments that form vessels. However, agents may also cover great distances by moving to a new group of line segments as shown by the agent 1 in Figure 7.8. Together with the locations of interest, the movement model allows agents to quickly arrive at locations that usually require editing while establishing a local context for learning. Next, we proceed to define the reward function.

7.4.3 Reward Function

The goal of the agents is to produce a vascular structure similar to the gold standard so that it can be queried for measurements. As we are employing a centralized solution, we may design a global reward function for all agents. There are a few measures of performance from which the reward function can be derived. One measure is to define the reward based on the error of a particular vascular measurement of interest with the goal standard measurement. This approach is undesirable as there are multiple

measurements. Furthermore, many measurements are aggregates. A measurement with small error may be based on an incorrectly extracted vascular structure. Hence, the reward function is based directly on the quality of the vascular structure.

Quantifying the quality of the vascular structure with respect to a given gold standard requires some method to match each extracted vessels to an appropriate gold standard vessel. Consider vessels in the automated extraction of vascular structure in Figure 7.1a. Notice, at one o'clock from the optic disc centre, that the automated extraction may produce vessels that have line segments belonging to two different vessels in the gold standard in Figure 7.1b. Therefore, some criteria is needed to decide which vessel in the gold standard should the vessel to be scored against, i.e., matched. To do this, we turn to an idea in evaluating clustering solutions.

Clustering is the problem of grouping similar data points together. Many notions of similarity exists, as well as the definition of a group. While the former is immaterial for our purpose, the latter is important. We are interested in the evaluation of hard clusters where each data point definitively belongs to one group. A group is essentially a label for the data point. Since we have the gold standard, this is a problem of evaluation with an external criterion where the concept of cluster *purity* (Manning et al., 2008, Section 16.3) can be applied. Let a cluster be a set of data points. Given N data points, the set of clusters $\mathbf{X}$, and the set of gold standard clusters $\mathbf{Y}$, purity is defined as,

$$purity(\mathbf{X}, \mathbf{Y}) = \frac{1}{N} \sum_{X \in \mathbf{X}} \max_{Y \in \mathbf{Y}} |X \cap Y|. \tag{7.1}$$

The purity provides a score for each computed cluster by assigning (matching) it to the gold standard cluster where the majority of its data points come from. This score is the count the number of majority points and summing over all computed clusters gives the final purity score. Hence clusters close to the original will have a purity near one while the converse will result in a purity near zero. We illustrate this with an example.

Example 7.1 (Cluster Purity)**.** *Consider the clusters in Figure 7.9. There are two clusters in the gold standard, circle points and square points respectively. The dotted boxes*

(a) Poorer Clusters **(b)** Better Clusters

Figure 7.9: Example of cluster purity. Circle and squares are two gold standard clusters while the dotted boxes are the clustering results.

are the result of the computed clusters. There are a total of 10 points in the data set and the purity for Figure 7.9a is, $\frac{1}{10}[3+3] = 0.6$*, where cluster 1 is assigned to the square cluster with an intersection count of 3 and cluster 2 is assigned to the circle cluster with a count of 3. However, in Figure 7.9b the purity is* $\frac{1}{10}[4+4] = 0.8$ *corresponding to the better fit with the gold standard clusters.*

Armed with the concept of purity we are ready to adapt it to quantify the quality of the extracted vascular structure. Pixels from the same vessel are treated as a "cluster". With the widths of the vessels already detected, e.g., as shown in Figure 7.5a, the polygon of each vessel can be computed, consequently all pixels in the polygon can be identified. Let each artery vessel $X_A \in \mathbf{X}_A$ be represented by a set of pixels where pixels are from the polygon representing the vessel. Similarly, define each vein vessel $X_V \in \mathbf{X}_V$ as a set of points in its polygon, and all points in orphaned segments as the set X_S. Then, given the set of all extracted vessels $\mathbf{X} = \{\mathbf{X}_A, \mathbf{X}_V, X_S\}$, and the gold standard set of vessels $\mathbf{Y} = \{\mathbf{Y}_A, \mathbf{Y}_V, Y_S\}$, the purity of the extracted vessels is,

$$VesPurity(\mathbf{X}, \mathbf{Y}) = \frac{1}{N_{\mathbf{X}}} \left(\left[\sum_{i \in \{A,V\}} \sum_{X \in \mathbf{X}_i} \max_{Y_j \in \mathbf{Y}_i} |X_i \cap Y_j| \right] + |X_S \cap Y_S| \right) \tag{7.2}$$

where $N_{\mathbf{X}}$ is the total number of pixels in the extracted vessels' polygons.

A few explanations of the intuition behind Equation 7.2 are in order. First, Equation 7.2 is scaled by $N_{\mathbf{X}}$ as unlike Equation 7.1, the number pixels can defer from the gold standard. If many more pixels are present that are absent in the gold standard, purity will be reduced. Second, like cluster $purity$, the range of $VesPurity$ is in $[0, 1]$. A completely wrong vascular structure will result in a purity near zero while a vascular structure resembling the gold standard will result in purity near one. Third, all orphans

are grouped into one cluster and counted as long as the orphan's pixel appears in some gold standard orphan. This means that having more orphans than the gold standard lowers purity. Next, the vessels are treated like clusters but they further are split into three classes, arteries, veins, and orphans. This indicates that only arteries are matched with gold standard arteries and similarly for veins. Therefore, both a mislabelling of vessel type and erroneous linkage of between vessels will result in lower purity. Last, in making use of points from vessel polygons, the purity is implicitly biased towards large vessels that will be naturally represented by larger "clusters". This is deliberate as many vascular measurements of interest involve querying the larger vessels.

Using $VesPurity$ we propose two reward functions for the terminal reward. The first is the change in purity from the initial state. Let,

$$\Delta(\mathbf{X}_a, \mathbf{X}_b) = VesPurity(\mathbf{X}_a, \mathbf{Y}) - VesPurity(\mathbf{X}_b, \mathbf{Y}) \tag{7.3}$$

for any two vascular structure $\mathbf{X}_a$ and $\mathbf{X}_b$. Then given that the state s consists of the current extracted vascular structure $\mathbf{X}_s$, the reward is,

$$R_\Delta(s) = \begin{cases} \Delta(\mathbf{X}_s, \mathbf{X}_0) & \text{if } \Delta(\mathbf{X}_s, \mathbf{X}_0) < 0 \\ \zeta \cdot \Delta(\mathbf{X}_s, \mathbf{X}_0) & \text{otherwise} \end{cases} \tag{7.4}$$

where $\mathbf{X}_0$ is the vascular structure from the initial state s_0 and ζ is a scaling factor to bias positive outcomes. The reward function R_Δ gives a negative reward if the vascular structure has less purity than the initially extracted vascular structure. If the vascular structure has been improved (higher purity), we give a positive reward scaled by ζ to encourage positive outcomes. We used $\zeta = 10$ in our experiments.

The second reward function is the negative reward,

$$R_{-ve}(s) = VesPurity(\mathbf{X}_s, \mathbf{Y}) - 1. \tag{7.5}$$

This reward function has the range [-1, 0]. Vascular structures with higher purity results

in a reward near zero while those with low purity will receive a reward near -1.

7.4.4 Learning

Reinforcement learning in this application differs from the previous chapters of this thesis. While the previous chapters focused on online learning within the same environment, this application is a case of offline learning. Each retinal image initializes a different environment as the static state information shown in Figures 7.5b and 7.5d will not be the same, neither will the reward function as it is based on the gold standard and the initial state. Hence there is an additional element of transfer learning in this domain. Furthermore, learning is partially supervised as we make use of gold standard graded images to compute the reward for RL during training. However, while the gold standard result is known, the sequence of edit actions to achieve it is unknown. Hence RL is employed to learn such a sequence of actions.

The solution employs model-free TD learning and linear function approximation. The agents' local contexts allow the design of features with respect to the relative context they are in. To handle the problem of transfer learning, we have normalized and equalized the static state information shown in Figures 7.5b and 7.5d respectively. Furthermore, we make use of relational features (RFs) from predicates to generalize among local contexts. For example, let the predicate $MoveToSameVessel(a_x, a_y)$, from No. 68 in Section A.3.1 page 239, encode if two agents are moving to the same vessel. This predicate can be used regardless of the retinal image, vessel, and agents in question. Using such predicates we can encode meaningful information that is likely to generalize across different retinal images.

Similarly if two level CGRL is used, such predicates can define coordination constraints (CCs) that are equally meaningful for different retinal images. For example, we may have the predicate,

$$\begin{aligned} P0(a_x, a_y) :=& Move(a_x) \wedge Move(a_y) \wedge [vessel(a_x) = vessel(a_y)] \\ & \wedge [vessel(pointAfter(a_x)) \neq vessel(pointAfter(a_y))] \end{aligned} \tag{7.6}$$

where $vessel(a_x)$ is a function that returns the current vessel an agent is on, and $pointAfter(a_x)$ returns the point after the agent has moved. $P0$ encodes whether two agents that are on the same vessel will no longer be on the same vessel after moving. Such a predicate can be useful, both as an RF or a CC, to encourage agents to remain on the same vessel. A detailed listing of predicates used is given in Section A.3 page 239.

The multi-agent MDP formulation allows agents to coordinate their actions as described by the binary predicates. However, coordination is not necessary in every state. We define the following situations where agents must coordinate,

1. Agents are in the same connected component of the line image. This is because the edit actions by one agent is likely to affect the actions taken by another agent.
2. Agents that share at least one destination in their movement actions. This is a loose measure of proximity based on our movement model and can be used to prevent agents from colliding if desired.
3. Agents are within a fixed Euclidean distance. This is another measure of agent proximity.

In the cases where agents need not coordinate, they may select their actions individually. This speeds up joint action selection.

The online RL methods can be directly applied for the training phase that takes place offline. Our set up for offline learning is as follows. An episode consists of agents taking actions using an exploration policy in a retinal image and updating their value functions for a fixed number of steps. The initial positions of agents in the vascular structure are randomized so that agents may encounter different states more rapidly. A terminal reward is given at the end of the episode. Given T training images, we perform RL on each image once to give an *episode block* of T episodes. Then, we continue training by reiterating through the images. After training for a certain number of episode blocks, we evaluate the current value function by using a greedy policy on a separate set of test images. The learning process stops at any desired number of training episode blocks.

7.5 Experiments

The experiments make use of retinal images taken from the real world Singapore Malay Eye study (SiMES) data set (Foong et al., 2007). The vascular structures from the retinal images have been extracted by human graders using the SIVA system by following a fixed grading protocol to standardize decision making.

The experiments are conducted using the method given in Section 7.4.4 with four agents and twenty steps per episode. We compare the different variations of RL editors as shown in Table 7.1. Coordinated RL and CGRL learning methods, and variations in the MDP formulation such as different reward functions and multi-agent RL methods are also investigated. Exploration policies are ϵ-greedy and bucket elimination (see Algorithm 4.2 page 75) is used for action selection among the 4 agents.

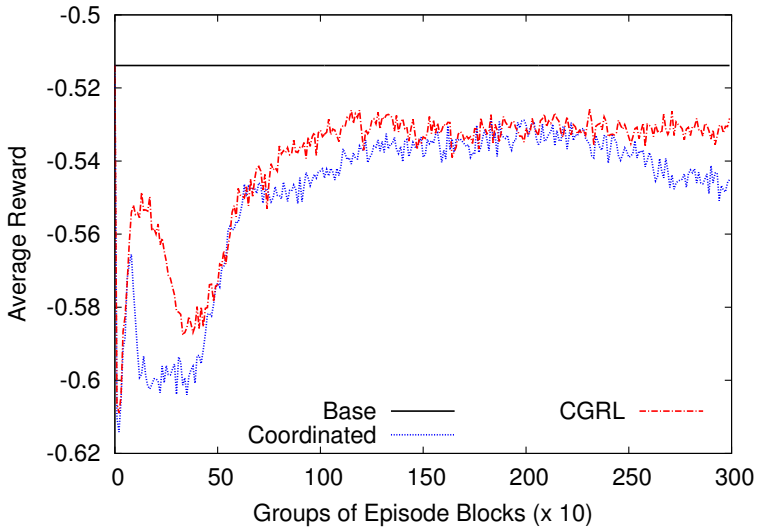
RL Editor	**RL Method**	**Reward Function**
Coordinated R_{Δ}	coordinated	R_{Δ}
CGRL R_{Δ}	CGRL	R_{Δ}
Coordinated R_{-ve}	coordinated	R_{-ve}
CGRL R_{-ve}	CGRL	R_{-ve}

Table 7.1: Various RL editors used to edit the retina's vascular structures.

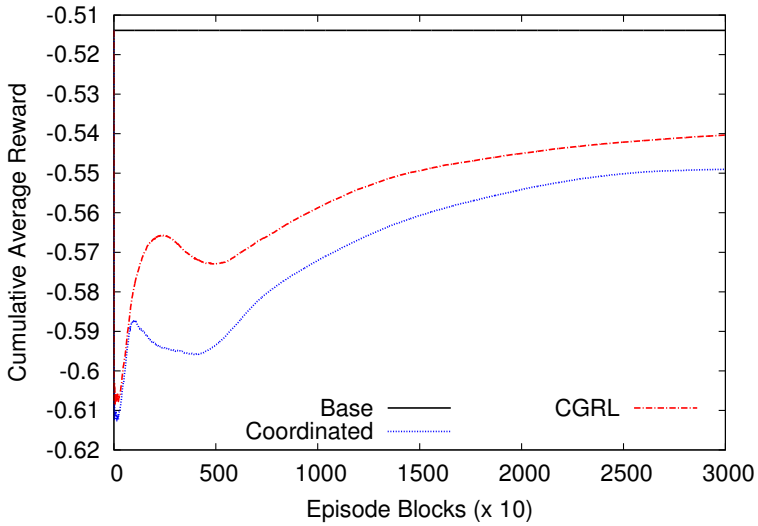
We analyse the results in a few ways. First, we present results to evaluate the various RL methods on the training set and a separate set of testing images. Second, we analyse results based on the actual impact on retinal measurements of interest. This allows us to compare the strengths of different methods. Third, we perform decile analysis on our results to yield insights into how the RL editors perform on images with varying errors in their vascular structure. Last, we show some sample edited images.

7.5.1 Learning Efficiency

This section presents the results of training on a set of ten retinal images (i.e., an episode block of length 10). First, we present plots for training performance on the training set that is akin to online learning. Then, we show plots for testing performance evaluated on five retinal images at intervals of ten episode blocks, i.e., evaluation is performed



(a) Average Reward



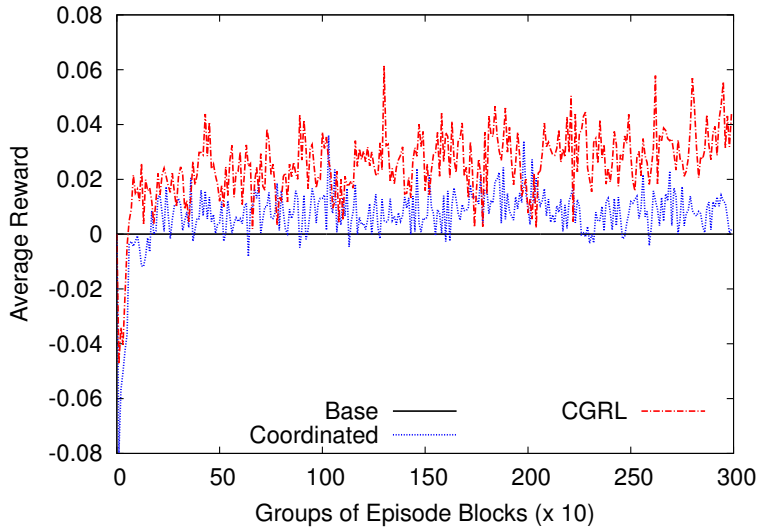
(b) Cumulative Average Reward

Figure 7.10: Training results using R_{-ve}. Each plot is an average of five runs.

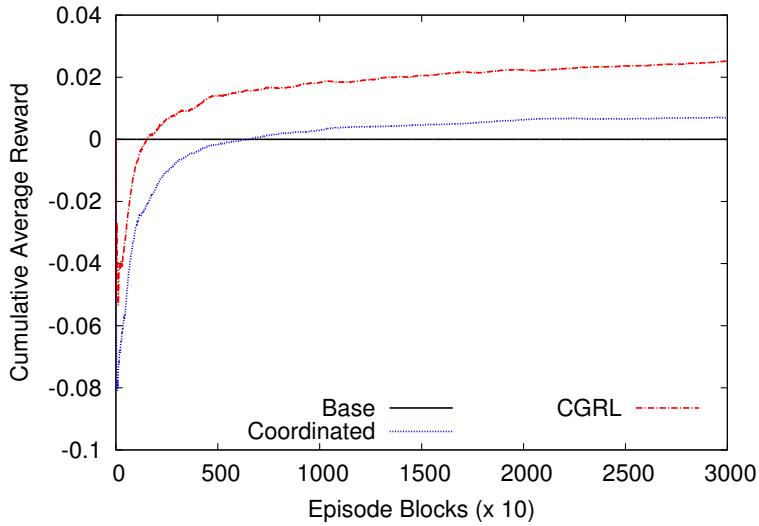
after 100 training episodes. Plots for testing performance show offline learning results. The purpose of this comparison is to analyse learning efficiency.

The first set of results are shown in Figures 7.10 and 7.11 for the reward functions R_{-ve} and R_{Δ} respectively. The RL editors are trained for 3000 episode blocks giving a grand total of 30,000 episodes using ϵ-greedy policies. Each plot is the average of five training runs, the black line labelled *Base* show the initial reward before any editing. All RL editors used the same RL parameters, $\gamma = 1$, $\alpha = 10^{-4}$, and ϵ starting at value 1 and decaying at a rate of 0.997 until remaining constant at 0.1.

From the plots, we see that CGRL outperforms Coordinated RL on the training data.



(a) Average Reward

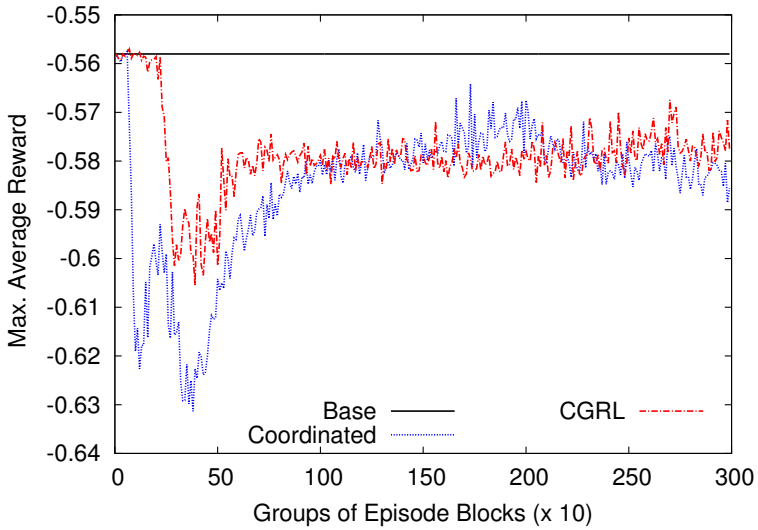


(b) Cumulative Average Reward

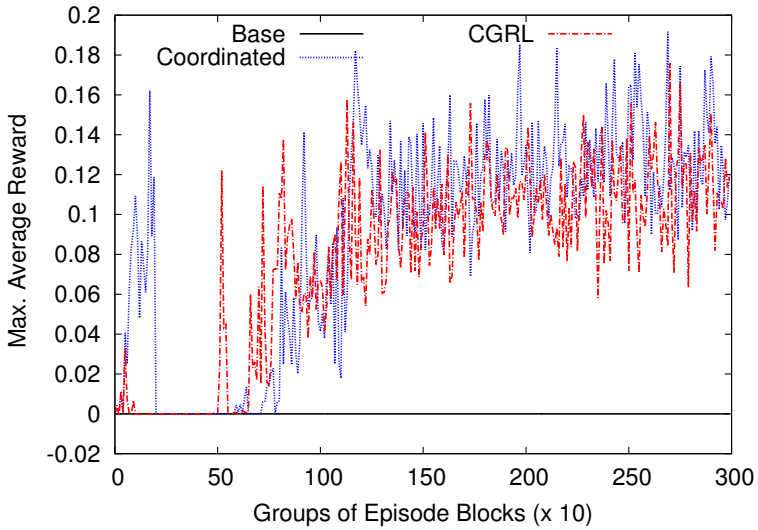
Figure 7.11: Training results using R_{Δ}. Each plot is an average of five runs.

This is as expected as the training results are similar to online learning. The oscillations in the plots are due to the constant α used. In terms of the reward function, RL editors using R_{-ve} do not perform better than the baseline reward. However, R_{Δ} is able to achieve rewards that exceed the baseline. This may be due to the exploration steps and unlike in the case where R_{Δ} is positive, R_{-ve} is not asymmetrically scaled. These results verify that the multi-agent RL methods are able to learn using our proposed MDP formulation.

For the next set of results, we evaluate the RL editors after every 10 training episode blocks (i.e., 100 episodes) on five testing images. RL editors are applied on each of the



(a) Using reward function R_{-ve}



(b) Using reward function R_{Δ}

Figure 7.12: Testing results where each plot is the maximum average out of five runs.

five testing images using the trained value functions with $\epsilon = 0$, i.e., the greedy policy. Each testing image is edited 20 times with random starting positions. The average of these $5 \times 20 = 100$ testing episodes are computed. There are five training runs for each RL editor. Since we are only interested in the best performer, we plot the maximum average from these five runs that is shown in Figure 7.12.

From Figure 7.12, we see that there are some similarities to the general trend between the testing and training results. However, the differences between the CGRL and Coordinated RL editors are less apparent. This may be due to the transfer learning issues that are present due to offline learning. Every retinal image presents a different

MDP environment as the static state values differs, e.g., the gradient map (see Figure 7.5b) differs between images. While in the training phase, CGRL performs better than Coordinated RL, this advantage may be lost to difficulties in transferring the learned results.

The results in this section has shown that the MDP formulation allows RL techniques to be applied to this problem domain. However, it tells us little of the impact on the retinal vascular measurements due to the different reward functions and RL methods. We investigate this further in the next section.

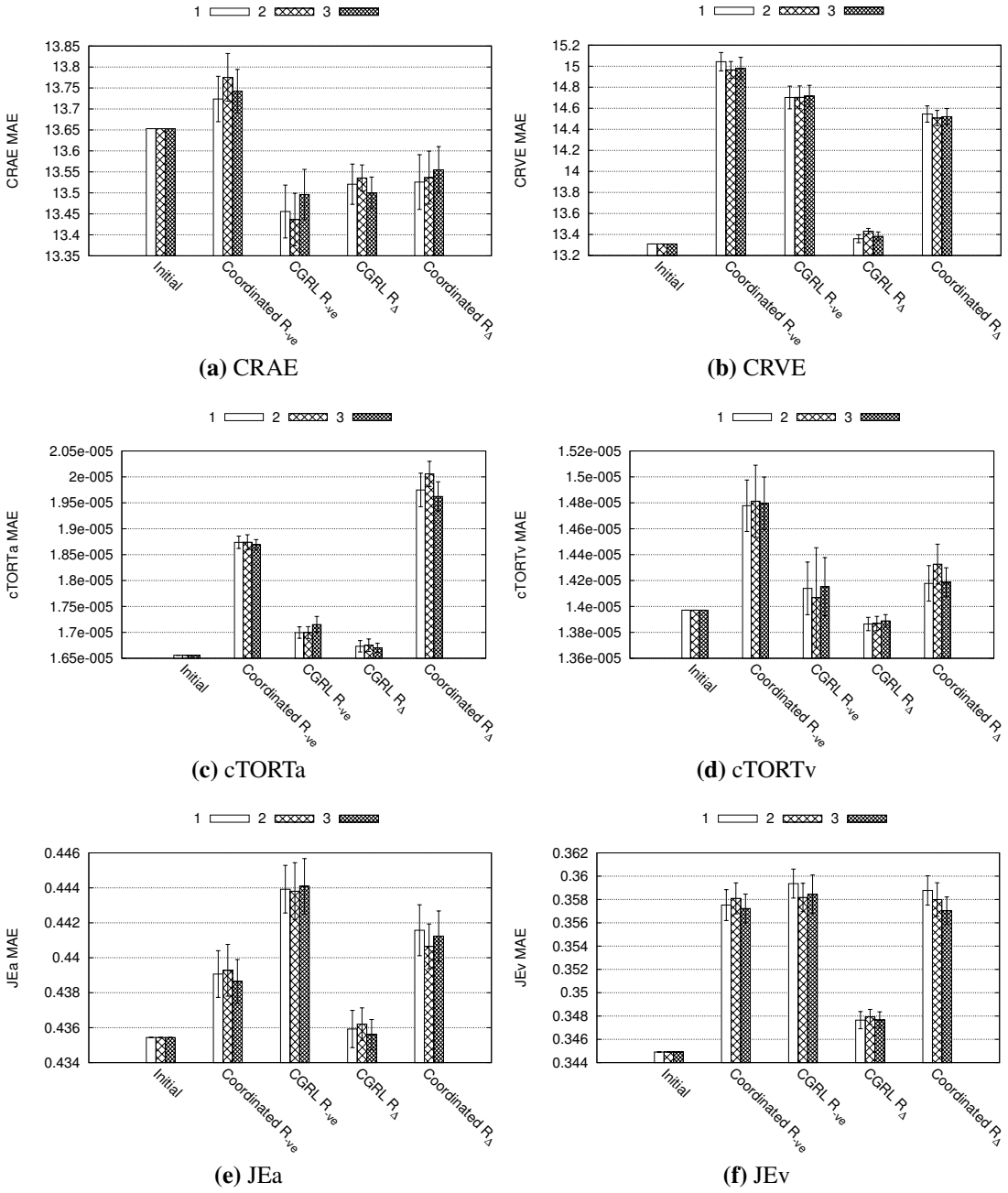
7.5.2 Comparing Problem Formulations

Improving the accuracy of vascular measurements is the true goal of our application. Therefore, in this section, we analyse the results based on the actual vascular measurements of interest by evaluating our trained RL editors on 900 retinal images with gold standard graded results. We compare variations of our approach based on the different reward functions by using the best performer out of the five runs for each RL editor, i.e., the run shown in Figure 7.12 at 3000 episode blocks (last point). Also, we study if applying our RL editor multiple times on the same retinal image in sequence yields any benefits. The six measurements of interest are:

CRAE & CRVE These two measurements are based on the aggregated widths (calibre) of the six largest arteries and six largest veins respectively.

cTORT The average curvature tortuosity of all arteries (cTORTa) and all veins (cTORTv) respectively. This measurement quantifies how tortuous are the vessel centre lines (Hart et al., 1997; Cheung et al., 2011b).

JE The average junctional exponent deviation of the six largest arteries (JEa) and six largest veins (JEv) respectively. This measurement requires vessels to be linked up correctly as the junctional exponent deviation is a ratio of the sum of the daughter segments' widths to the parent segment width at the first bifurcation of a vessel.



(a) CRAE **(b)** CRVE

(c) cTORTa **(d)** cTORTv

(e) JEa **(f)** JEv

Figure 7.13: MAE results for various retinal measurements. Lower is better. RL editors are applied up to three consecutive times indicated by 1, 2, 3. Each bar is the mean of 30 evaluations on 900 images, the error bars show $+1$ and -1 standard error.

These measurements are important as they have been shown to be correlated to a number of diseases as described earlier in Section 7.1.

The first series of plots in Figure 7.13 is based on the mean absolute error (MAE) of each measurement with the gold standard. The MAE is the mean of the absolute difference between the measurement of each image after editing and the measurement from the gold standard. RL editors are applied to each of the 900 images 30 times giving 30

sets of results. The mean MAE with its standard error is illustrated in Figure 7.13 for each measurement. In addition, the *initial* bars shows the MAE before any editing. We further investigate applying the RL editors on the same image multiple times consecutively. These are shown as the bars labelled 2 and 3 in the plots corresponding to the number of times the RL editors are applied. Ideal MAE values should be close to 0.

The second series of plots in Figure 7.14 show the Pearson correlation coefficient (PCC) between the edited images and the gold standard. Each bar in shows the mean PCC of the 30 sets of results along with the standard error. We have chosen PCC as a comparison as it is less affected by constant error factors. Furthermore, the PCC is usually used to correlate retinal vascular measurements with the risk of diseases. Ideal PCC values should be close to 1.

The results in Figure 7.13 show that certain measurements benefit from the RL editors more than others. Notably, there exists an RL editor that is able to reduce the MAE values of CRAE in Figure 7.13a and cTORTv Figure 7.13d. Furthermore, the CGRL R_Δ editor has generally lower MAE values than the other RL editors. For the PCC results in Figure 7.14, the RL editors result in lower PCC values for most measurements with the obvious exception of the CRAE in Figure 7.14a where all editors improve the PCC. This indicates that the RL editors may be most useful for improving the CRAE measurement. We notice that the initial PCC of CRAE in Figure 7.14a is the lowest among the other measurements in Figures 7.14b to 7.14d, indicating that it may be easier to improve an already poor measure. However, this is not the case for the JE measures from vessel bifurcations that have low initial PCC values shown in Figures 7.14e and 7.14f where the RL editors were not able to improve the PCC. This suggests that the RL editors are not adept at connecting correct branches to vessels.

Among the various RL editors, the CGRL R_Δ editor stands out the most as it either improves the PCC, or results in a PCC closest to the initial value. The same can be observed for the MAE results. From these results we conclude that the CGRL R_Δ is mostly superior to the rest of the RL editors. Furthermore, additional application of the RL editors beyond the first (bar labelled 1) do not yield significant benefits as seen in

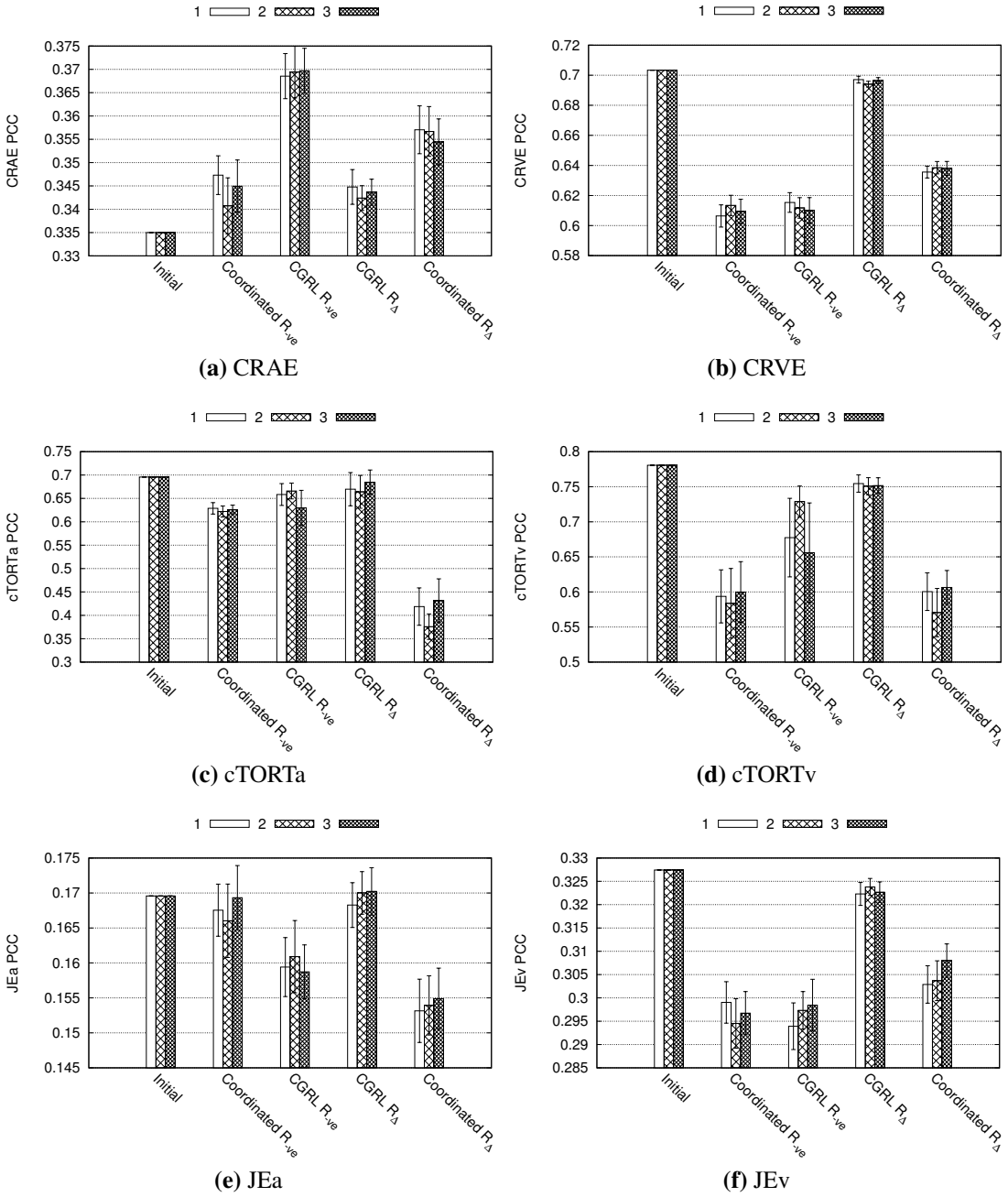


Figure 7.14: PCC results for various retinal measurements. Higher is better. RL editors are applied up to three consecutive times indicated by 1, 2, 3. Each bar is the mean of 30 evaluations on 900 images, the error bars show $+1$ and -1 standard error.

the bars labelled 2 and 3. The results thus far have only shown marginal improvement for certain measurements. For a more detailed analysis, we divide the 900 images into deciles in the next section.

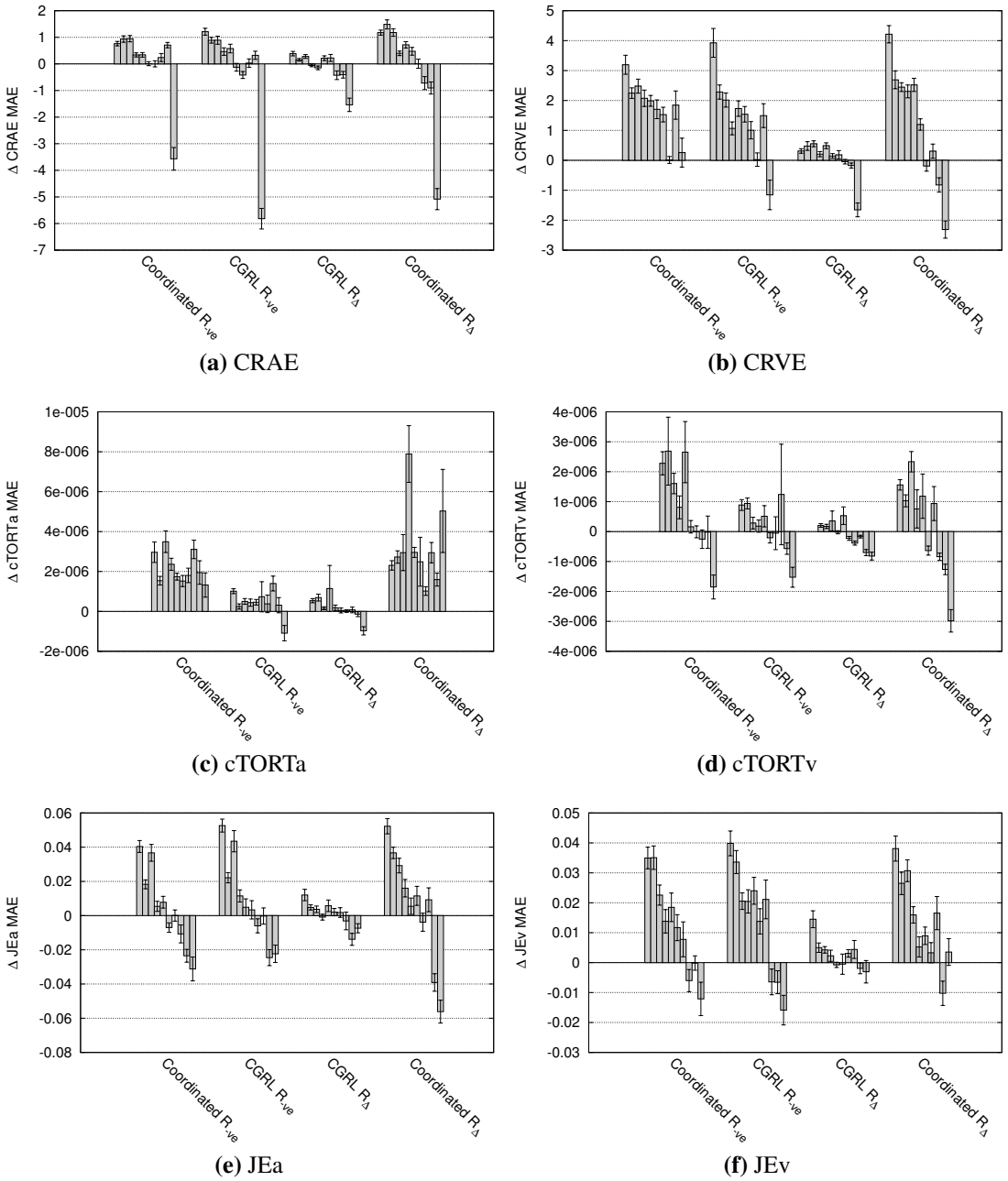
7.5.3 Decile Analysis of Measurement Quality

To gain a better insight into the impact of the quality of the initial extracted vascular structures on RL editor performance, we perform decile analysis on our results. For each measurement of interest, the 900 retinal images are ranked based on their absolute error between the initial automated vascular extraction and the gold standard. Then, the ranked images are partitioned into deciles of 90 images each. Subsequently, we analyse the performance in terms of MAE and PCC with respect to each decile. To better illustrate the results with respect to the initial extracted vascular structures, the changes in MAE and PCC with respect to the initial values for each decile are plotted.

The results for MAE and PCC are shown in the Figures 7.15 to Figure 7.16 respectively. For each plot, the left most bar is the decile with lowest absolute error (high quality) and the right most bar is the decile with the highest absolute error (low quality). As before, each bar plot is a mean of 30 evaluations for a decile and the error bars represent $+1$ and -1 standard error.

Figure 7.15 depicts the MAE by decile for the six measurements of interest. Generally, the RL editors increase the MAE for the high quality deciles and conversely, decreases the MAE for the low quality deciles. These suggest that the RL editors perform best on retinal images with low quality extracted vascular structures at the start. In particular, notice that the CGRL R_{Δ} editor is the most conservative. Its changes to the MAE for both high quality and low quality images are much less compared to the other RL editors. This may be due to the coordination constraints (CCs) present in CGRL that may bias the solution towards more conservative editing. For the other RL editors, such as Coordinated R_{Δ}, that greatly reduces error in the low quality deciles, more errors are introduced in the high quality deciles.

There are variations in quality between the artery and vein measurements. For example, the results of CGRL R_{Δ} editor for the CRAE in Figure 7.15a are similar to the results for CRVE in Figure 7.15b. However, this is not the case for the RL editors using R_{-ve}, where the MAE of the CRVE mostly increases throughout the deciles. This indicates that there may be spurious arteries (i.e., mislabeled veins) resulting from these



(a) CRAE **(b)** CRVE

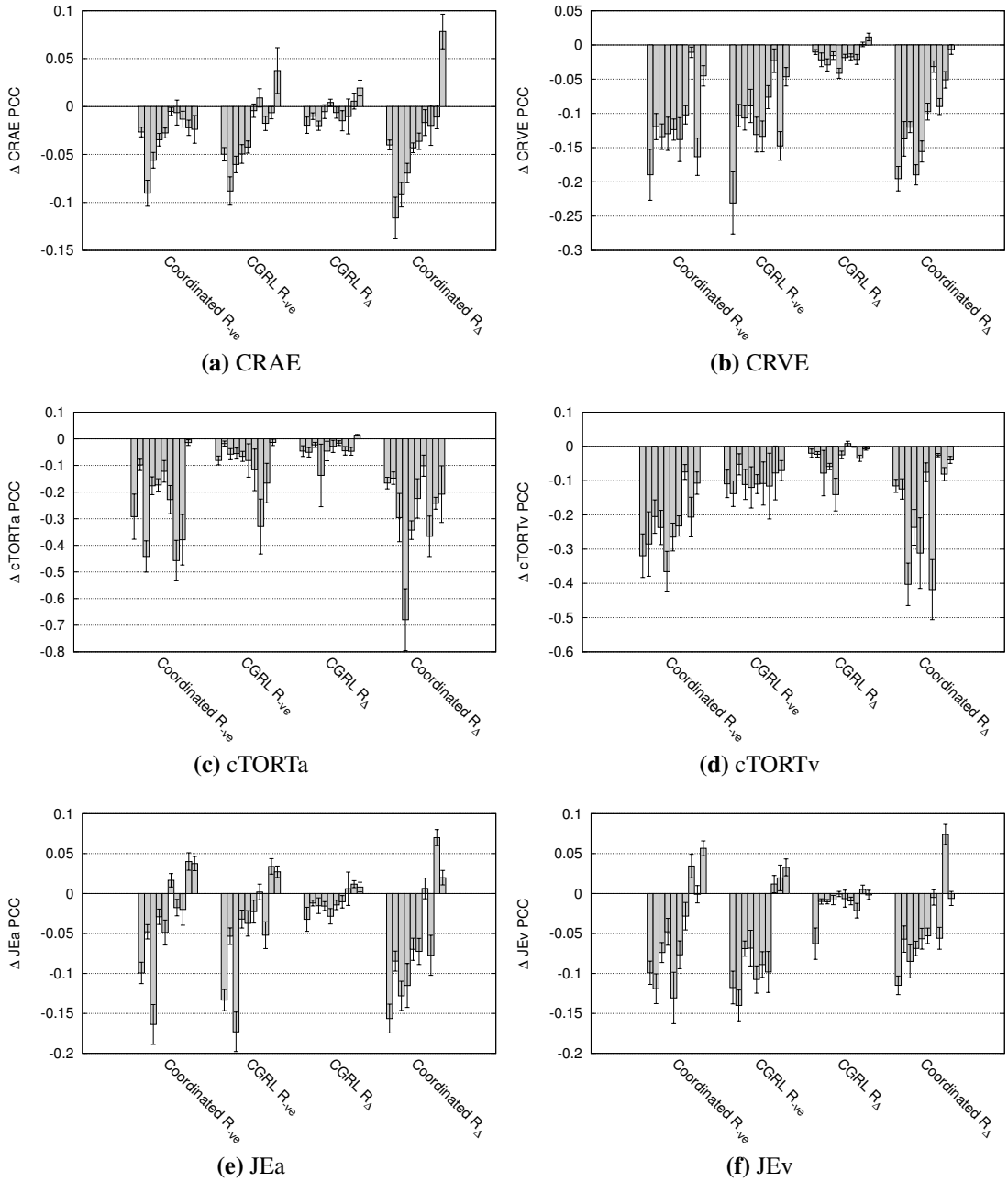
(c) cTORTa **(d)** cTORTv

(e) JEa **(f)** JEv

Figure 7.15: Change in MAE by decile for various measurements. Lower is better. Decile from lowest error on the left to highest error on the right.

RL editors. Hence, the CGRL R_Δ editor, is most consistent between the same artery and vein measurements for vessel widths. The results for the JE values in Figures 7.15e and 7.15f indicate that the RL editors are mostly able to improve these values for the low quality deciles for both arteries and veins.

Figure 7.16 illustrates the PCC results by decile for each of the measurements of interest. In general, the CGRL R_Δ editor is similarly conservative in the way it affects



(a) CRAE **(b)** CRVE

(c) cTORTa **(d)** cTORTv

(e) JEa **(f)** JEv

Figure 7.16: Change in PCC by decile for various measurements. Higher is better. Decile from lowest error on the left to highest error on the right.

the PCC values for most measurements. In terms of the measurements, the editors using R_{Δ} improve the PCC values in some proportion to the decrease in MAE for the lowest quality decile of the CRAE in Figure 7.16a. However, this is not as evident for the R_{-ve} editors. For the CRVE in Figure 7.16b, only the CGRL R_{Δ} editor increases the PCC for the lowest quality decile, suggesting that the other editors have mostly inaccurately edited veins. The tortuosity measurements in Figures 7.16c and 7.16d generally show

no improvement across most deciles, suggesting that the RL editors have not performed well for these measures. Last, notice that the PCC for the JE values in Figures 7.16e and 7.16f correspondingly improved as the MAE decreased for the low quality deciles.

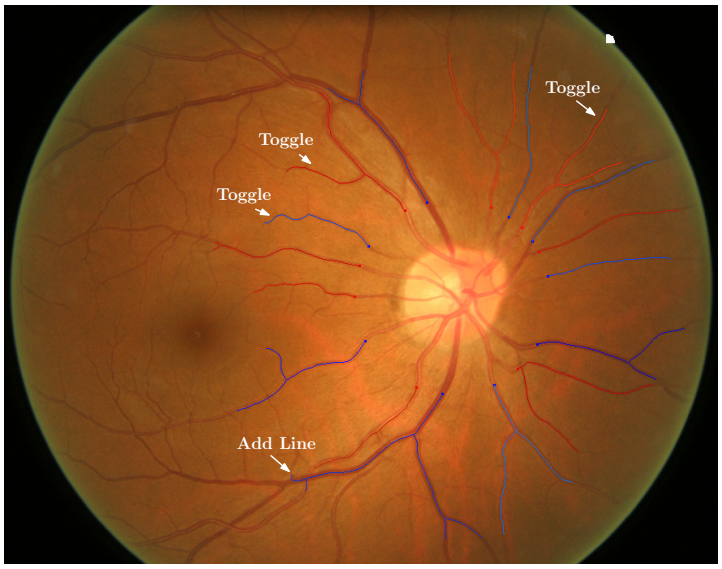
7.5.4 Example Edited Images

This section attempts to visualize the edited results from images based on their reward. The top 20 results in terms of reward for the CGRL R_{Δ} editor from the 30 evaluations of the 900 images are examined. As a sample, the top two images are displayed in Figures 7.17 and 7.18. Note that the extent of the vessels is limited to the zone of interest but the zone is omitted to reduce clutter. For each image, we present the initial extracted vascular structures, then, the edited vascular structures and last, the gold standard vascular structure edited by a trained human grader.

The human graders followed a grading protocol to standardize their decision making. This protocol is a set of rules for editing the vascular structure. In the SiMES study where the gold standard was obtained, the grading protocol only connected the first order bifurcation (branch) of each vessel, while disconnecting higher order bifurcations. Previously, this was depicted in Figure 7.1b and it can be further observed clearly in Figure 7.18c, where the second order bifurcations in the veins (in blue) at eleven and one o'clock from the optic disc centre are not connected.

From Figures 7.17 and 7.18, the most obvious changes made by the RL editor is the toggling of the vessels type. In the top half of Figure 7.17b above the optic disc, two veins have been changed to arteries and one small artery to a vein. These changes are correct when verified with Figure 7.17c. Similarly in Figure 7.18b, toggling the vessel types for three vessels resulted in more correct measurements. Further visual inspection of the other top 20 images reveals that most have toggled vessels but the other edit actions are not heavily used. This suggests that the Toggle Vessel Type action yielded the most reward.

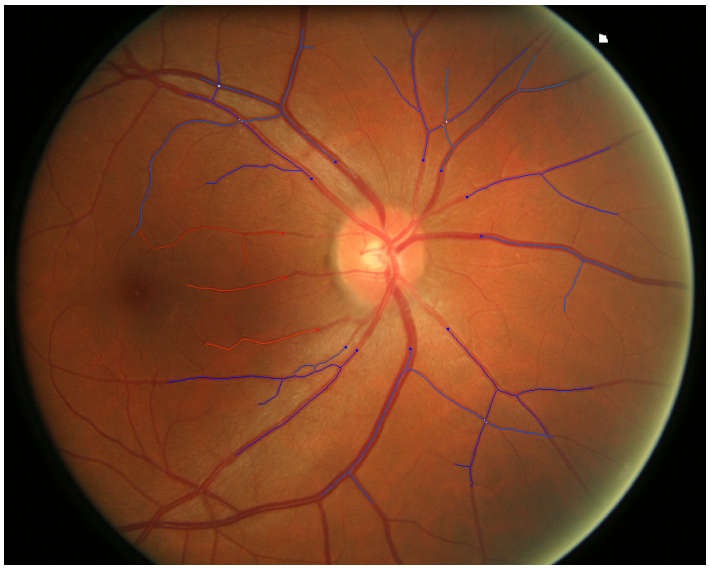
(a) Initial



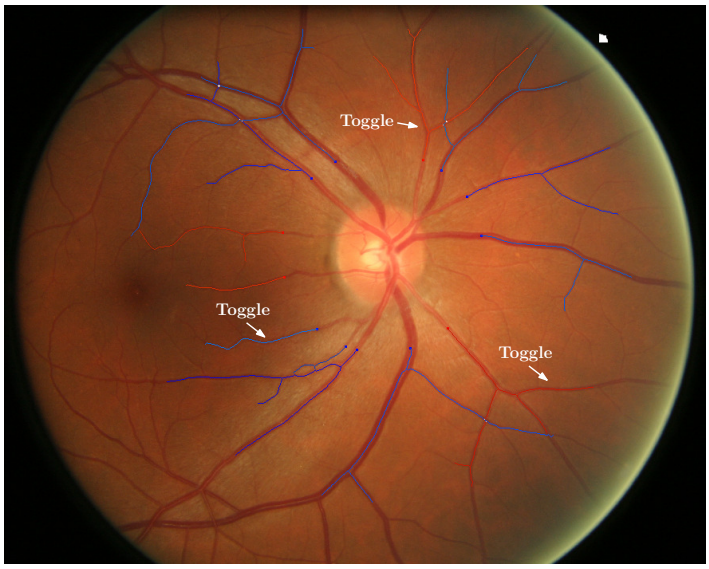
(b) Edited

(c) Gold Standard

Figure 7.17: Example edited retinal image 1. Final reward is 1.686.



(a) Initial



(b) Edited

(c) Gold Standard

Figure 7.18: Example edited retinal image 2. Final reward is 1.674.

7.6 Discussion

The next few sections discuss the experiment results and suggestions to improve the current application.

7.6.1 Learning & Problem Formulation

The overall results have shown that it is feasible to employ RL in this problem domain with the CGRL R_Δ editor showing the most promise. The CGRL R_Δ editor is more conservative than the other RL editors. This is likely due to a combination of coordination constraints and the R_Δ reward function that directed learning towards a more conservative solution. From the decile based analysis in Section 7.5.3, CGRL R_Δ reduces errors in the lowest quality decile albeit with a lesser reduction than the other RL editors. However, unlike the other RL editors, CGRL R_Δ does not introduce much higher errors into the higher quality deciles. Furthermore, it produces more balanced results between artery and veins for the same measurement.

The proposed MDP formulation has resulted in the Toggle Vessel Type action yielding the most improvement. This discovery can be explained by the definition of the reward function based on vessel purity in Equation 7.2. As vessels are considered to have higher purity if they correspond to the correct vessel type, the act of toggling a large vessel can result in large changes in the vessel purity. In contrast, other actions such as Add Segment modifies the vessel slightly and are unlikely to result in large changes in the resulting reward.

We have some suggestions to better encourage learning to use the other actions besides Toggle Vessel Type. The first, is to split the current problem formulation into two sub-problems to be solved in sequence: one for editing the vessels, and one for labelling the vessels. In this way, the Toggle Vessel Type actions will not dominate the reward function. Alternatively, ideas from multi-objective RL (Vamplew et al., 2011) may be attempted to prevent the accurate vessel type objective from dominating the reward signal. Last, is to refine the reward function into a more fine-grained one such

as an agent decomposed reward. Consequently, the agents will have a more precise sample of the reward for their individual actions.

7.6.2 Domain Related Issues

The RL editors have shown that it is possible to improve measurement quality for certain images. Of all the measurements of interest, improving the CRAE is the most promising as evidenced by the results in both Sections 7.5.2 and 7.5.3. The decile based MAE results and PCC results in Section 7.5.3 suggest that in general the CRAE and JE values for low quality images can be improved with our methods.

One obvious issue with the gold standard data used in this study is the grading protocol. As only the first bifurcations are connected in the gold standard, the results of RL editors will be penalized if higher order bifurcations are connected. Apart from directly impacting vessel purity, the MAE and PCC of the measurements of interest are also affected. This is most severe for the cTORT measurement as it is an average of all arteries or veins including the small vessels. In contrast, the width measurements, CRAE and CRVE, are less affected as they use the six largest arteries and veins, while the JE measures are least affected as they only measure the first order bifurcation of the six largest arteries and veins. This partly explains the poor PCC results for cTORT in Figures 7.16c and 7.16d. Results are expected to improve after training on more complete human graded vascular structures as they become available.

The better performance of the RL editors on the low quality deciles suggests that it may be useful to design a filter method to sieve out images where our existing vascular extraction methods will perform poorly. These images are most likely to be similar to the lowest quality decile where the RL editors can perform best. In a fully automated solution, we can apply the RL editor on only the poor images before measuring their vascular structures.

7.6.3 Related Work

We have presented an RL application that edits the result of automated vascular structures in a post-processing feedback loop. The edits interact sequentially with the other automated vessel separation, classification, and width detection algorithms. Hence this RL application is agnostic to the other algorithms used in the process of extracting vascular structures. In contrast, other related works that employ RL on other types of medical images do so directly within the segmentation algorithms (Sahba et al., 2008; Chitsaz and Seng, 2009; Wang et al., 2011) and hence, are more tightly coupled to those algorithms.

Abdul-Karim et al. (2005) made use of an RL system to automatically tune the parameters of vessel for a neurite segmentation algorithm. Our application differs in that it participates in the process of obtaining the vascular structure in a more direct way by iteratively modifying the input to other parts of the process chain to obtain the extracted vascular structures. Furthermore, our approach is independent of vessel segmentation.

Our work mostly differs from the other existing works in the complexity of the extracted structure that involves additional processing after segmentation. This gave rise to a large state and action space that we reformulated into a multi-agent RL problem. Therefore, we are able to handle greater complexity as opposed to the single agent formulation for the existing works.

7.7 Conclusion

This chapter presented a preliminary application of multi-agent RL for improving the extraction of vascular structures from images of the human retina. Subsequently, the vascular structure can be queried for measurements of interests that may be used as feature inputs to disease risk predictors or for population studies. To the best of our knowledge, this is the first attempt to apply multi-agent RL to extracting vascular structures from retinal images.

This work proposed a multi-agent MDP problem formulation that was inspired by the real world computer assisted process of manually editing vascular structures to correct inconsistencies. Explanations were provided for the design decisions that handle scalability in this large domain through the use of agent dependent local contexts. Experiment results on real world data set demonstrate the feasibility of the solution. However, there is room for improvement when the vascular measurements of interests are considered. The solution performs best for poor initially extracted vascular structures and is most useful for improving the CRAE measure. Enhancing the current solution involves: adjusting the MDP problem formulation, obtaining a more comprehensive data set as the gold standard, and developing a filter to identify poorly extracted vascular structures.

In the wider scope of this thesis, this chapter provides an example of a real world application to automating computer assisted tools that is formulated as a collaborative multi-agent RL. Hence, demonstrating the flexibility of the solutions developed in this thesis for various seemingly unrelated domains.

A Blank Page

Chapter 8

Conclusion

This thesis investigated reinforcement learning (RL) for collaborative multi-agent domains. It primarily focused on three main research challenges for online learning in these domains, namely:

1. *Exploration.* Multiple agents result in exponentially large state and action spaces to explore. The heavy exploration requirements result in less time to exploit in the online setting.
2. *Distribution.* Communication among agents are dynamic. This requires learning systems to be distributed among agents. Agents should be able to make effective use of communication when available.
3. *Model Complexity.* Many learning parameters may be required for multiple agents resulting in slow learning. Furthermore, models usually encode some expert knowledge. Simplifying model representation will improve applicability of proposed methods.

For each of these challenges specific gaps in current research were addressed. First, is the absence of an active involvement of coordination knowledge among agents for improving online exploration in large multi-agent problems. The second follows from the first but with respect to a distributed setting with dynamic communication restrictions. Third, there are few works that decentralize relational RL whereby a solution has the potential to greatly reduce model complexity. Addressing these gaps are the

contributions of this thesis.

8.1 Contributions

The main contribution of this thesis over existing works is the introduction of coordination constraints (CCs). CCs are fragments of expert coordination knowledge used to dynamically remove large parts of the joint action space from exploration. When to make use of CCs is a part of the overall RL system, i.e., the system learns to make use of CCs. An important benefit of this contribution is that encoding declarative CCs are similar to encoding propositional features based on predicates for function approximation. In fact, this similarity allows CCs to guide exploration in conjunction with representing value functions. The burden of encoding expert knowledge on the users of the proposed methods is reduced as existing predicates for features can be used as CCs and vice versa. Hence, CCs increase the effectiveness of the same predicate building blocks, that are commonly used to handle other aspects of multi-agent RL, by making greater use of them in RL.

The coordination guided reinforcement learning (CGRL) system was presented to learn when to make use of CCs to guide exploration of primitive actions. The first step developed the centralized CGRL method that has two levels. The top level learns to select CCs to activate. Activated CCs constrain the policy of the bottom level that selects the actual primitive actions to be taken in the environment. Learning an optimal solution to this augmented learning problem yields an optimal solution for the original problem. Temporal difference (TD) updates were used to learn value functions for both levels incrementally while the system interacts with the environment. CGRL may also be combined with relational TD learning through the use of relational features (RFs). Experiment results demonstrated that CGRL outperforms existing methods for coordinated RL for many different variations of simplified soccer and tactical real-time strategy (RTS) game domains. Hence, CGRL addresses the exploration and model complexity challenges for centralized multi-agent RL.

The second contribution dealt with the challenge of dynamic communication. In this situation, communication links among agents change over time and agents may be removed from the environment. The distributed CGRL system was developed to address these issues. The top and bottom level value functions were decomposed such that each agent learns its share, ensuring that no one agent is critical to the learning system. Instead of using global TD updates for learning, local updates were based on each agent's decomposed local functions and observation of its local reward. Communication between agents took place during action selection. The results on variations of soccer and tactical RTS games with dynamic communication showed that CCs improve learning over existing distributed RL methods. However, in addressing the issue of communication, distribution resulted in more learning parameters for the multi-agent system as a whole. This is because relation features cannot be employed directly like in the centralized system. As a result of this lack of relational generalization, the overall solution quality of distributed CGRL for domains like tactical RTS is reduced as compared to centralized CGRL. Overcoming these limitations led to the third contribution.

The third contribution investigated a distributed solution for relational TD learning. This mitigates the problem of model complexity in distributed CGRL and in general, distributed RL. Two types of relational generalization were used, namely, internal and external. Internal generalization proposed a scheme to develop agent decomposed value functions based on local agent based RFs. The sum of the agent decomposed value functions with suitably learned weights can be equivalent to the global value function used in centralized relational TD learning. Internal generalization allows an agent to generalize learning over its interactions with various agents. To further generalize learning among different groups of agents, external generalization was introduced. This is achieved using a message passing scheme to share learned parameters that share relational semantics. Experiment results show that relational TD learning integrates well with existing works in distributed RL and distributed CGRL; outperforming them in terms of learning performance while using a much reduced number of learning parameters. Furthermore, the learning performance of distributed CGRL with relational TD learning was com-

petitive with centralized approaches in certain domains. Thus, a reasonable trade-off between distribution requirements and model complexity was achieved.

The last contribution is a prototype application for automating a real world program for computer assisted image analysis of the human retina. The goal of this analysis is to extract the vascular structure from retinal images so that they can be queried for measurements. Such measurements may then be used for population studies or for disease predictions. A novel multi-agent MDP formulation was used to model the problem of interactively editing the vascular structure. Consequently, a reward function to capture the correctness of the structure was defined. Agents repeatedly modify the vascular structure like a human grader, triggering the existing system to re-separate and re-identify individual vessels. The approach was evaluated using data from a real world population study. The results demonstrated that it is feasible to employ multi-agent RL in this domain, and that there are some improvements to the measurement quality for certain measurements when the initial extracted vascular structure is poor. Hence, this preliminary solution demonstrates promise and verifies the wide applicability of collaborative multi-agent learning techniques.

8.2 Future Work

There are few main directions for future work. First, like most methods that introduce new uses for expert knowledge, the reliance on expert knowledge should be reduced. In the short term, plausible directions include automatically deciding on features to double up as CCs and automatically constructing top level features for these CCs. However, there is also a need to construct the CCs themselves. For this, note that CCs are coordination knowledge encoded by predicates. By deliberately using the same form of knowledge encoding as many works in the larger field of relational learning, future work for automated predicate construction can serve as a basis for developing methods to create the predicates for CCs and RFs. Likewise, developing methods to incrementally construct useful predicates online can benefit both RL and relational learning.

Another form of expert knowledge that will have wide benefits if reduced relates to the agent decomposition of the global reward signal. Good reward signals are crucial for RL. Agent decomposed signals can provide fine grained information to centralized approaches and are a requirement for distributed RL. To aid in applicability of RL methods, it will be useful to have automated methods to decompose global reward signals into individual agent signals such as the methods highlighted in Section 3.5 page 45. More so if this decomposition can be learned online or handle non-stationary environments. Providing such solutions will further reduce the overall reliance on expert knowledge in RL.

Second, much work remains in determining the strengths of various RL methods that rely on different forms of expert knowledge to improve online exploration. CGRL makes use of existing predicate based knowledge used in encoding features to provide CCs. This aids in its applicability for improving existing learning projects. Furthermore, CGRL is suitable for domains whenever a predicate feature encoding for function approximation is suitable. However, when choosing an RL method for new projects, there is little work that compares the strengths of existing works such as the task based or organizational based approaches described in Section 3.4 page 43. The main impediment to a fair comparison of these alternatives is the inherent difficulty in objectively quantifying the different forms of expert knowledge. Contemporary works that make use of probability models in Bayesian multi-agent RL (Teacy et al., 2012) and adaptively modifying reward shaping (Devlin and Kudenko, 2012) further add to the myriad of employable expert knowledge for improving exploration. Comparisons will also require well defined categories for various domains. However this may take much time to discover. Therefore, a more expedient avenue to evaluate various approaches is through the development of fusion methods where multiple ideas are integrated. For example, a hybrid task based CGRL can be compared with plain task based RL and CGRL. Similar difficulties were noted in the comparison of value function based RL and direct policy search in Kalyanakrishnan and Stone (2009) that also included a hybrid approach.

Third, there may be merit in determining if there is a relationship between the

model-free and model-based approaches to improving exploration. A number of model-based methods make use of uncertainty in the model to direct exploration concurrently with exploitation (Strens, 2000; Poupart et al., 2006; Epshteyn et al., 2008). In contrast, many model-free methods reviewed in this thesis makes use of some hierarchical concept to adjust explorations, e.g., through higher level actions or decisions that affect the actions, rewards, or value functions. Such hierarchical concepts to adjust exploration can be viewed as introducing some form of bias into a stochastic exploration policy. Investigating the link between hierarchical concepts and uncertainty in an approximate model may yield new insights to improve online exploration.

The fourth relates to distributing relational TD learning. Future work in this direction includes investigating an adaptive method to adjust the amount of relational generalization over time for domains where specialized roles are important, and automatically identifying the predicate based features that are suitable for relational generalization. Automatic construction of higher level predicates remains a crucial problem to address in multi-agent domains to reduce the dependency on background knowledge provided by human experts. This thesis demonstrated how relational generalization can work for CGRL that has higher level actions based on coordination constraints. This may provide insights as to how other multi-agent hierarchical RL methods that involve higher level actions (Marthi et al., 2005; Ghavamzadeh et al., 2006; Zhang et al., 2010) can be augmented with relational constructs.

Fifth, generalizing the methods in this thesis to more complex problem formulations will increase their applicability. This thesis explored variations of the multi-agent MDP formulation with respect to decentralization. However, adapting to more complex formulations, such as partially observable Markov decision processes and Markov games, will extend the proposed methods to the respective domains where the information of the state is a stochastic signal and where adversarial agents are present. In the case of Markov games, there may still be collaborative agents among adversarial agents. Hence, the idea of CCs may remain relevant for the collaborative agents. However, there will be a need to overcome the stronger communication restrictions and the

non-stationary environment that is common in Markov game formulations.

Finally, the sheer complexity of multi-agent learning and flexibility in its application ensures that research in this field has great potential as more complex and larger domains are encountered. There remains many exciting opportunities to apply agent learning to improve automation in real world computer assisted tools and autonomous robotics, as well as to provide a better entertainment experience in computer games. Indeed in time to come, autonomous and smart multi-agent systems may become essential tools in our daily lives.

A Blank Page

Bibliography

Abdul-Karim, M.-A., Roysam, B., Dowell-Mesfin, N., Jeromin, A., Yuksel, M., and Kalyanaraman, S. (2005). Automatic selection of parameters for vessel/neurite segmentation algorithms. *IEEE Transactions on Image Processing*, 14(9):1338 –1350.

Ahmadabadi, M. and Asadpour, M. (2002). Expertness based cooperative Q-learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 32(1):66 –76.

Andre, D. and Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, pages 119–125, Menlo Park, CA, USA. AAAI.

Apt, K. (2003). *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA.

Araabi, B., Mastoureshgh, S., and Ahmadabadi, M. (2007). A study on expertise of agents and its effects on cooperative Q-learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(2):398 –409.

Asgharbeygi, N., Stracuzzi, D. J., and Langley, P. (2006). Relational temporal difference learning. In Cohen, W. W. and Moore, A., editors, *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, volume 148 of *ACM International Conference Proceeding Series*, pages 49–56. ACM.

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Machine Learning (ICML)*, pages 30–37, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Balla, R.-K. and Fern, A. (2009). UCT for tactical assault planning in real-time strategy games. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, pages 40–45, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27:819–840.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Bianchi, R. A. C., Ribeiro, C. H. C., and Costa, A. H. R. (2007). Heuristic selection of actions in multiagent reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 690–696, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Buro, M. (2004). Call for AI research in RTS games. In *AAAI Workshop on AI in Games*, pages 139–141. AAAI Press.

Chapman, N., Dell'omo, G., Sartini, M. S., Witt, N., Hughes, A., Thom, S., and Pedrinelli, R. (2002). Peripheral vascular disease is associated with abnormal arteriolar diameter relationships at bifurcations in the human retina. *Clinical Science (Lond)*, 103(2):111–116.

Chen, G., Yang, Z., He, H., and Goh, K. M. (2005). Coordinating multiple agents via reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 10:273–328.

Cheung, C. Y.-L., Hsu, W., Lee, M. L., Wang, J. J., Mitchell, P., Lau, Q. P., Hamzah, H., Ho, M., and Wong, T. Y. (2010). A new method to measure peripheral retinal vascular caliber over an extended area. *Microcirculation*, 17(7):1–9.

Cheung, C. Y.-L., Tay, W. T., Mitchell, P., Wang, J. J., Hsu, W., Lee, M. L., Lau, Q. P., Zhu, A. L., Klein, R., Saw, S. M., and Wong, T. Y. (2011a). Quantitative and qualitative retinal microvascular characteristics and blood pressure. *Journal of Hypertension*, 29(7):1380–1391.

Cheung, C. Y.-L., Thomas, G., Tay, W., Ikram, K., Hsu, W., Lee, M. L., Lau, Q. P., and Wong, T. Y. (2012). Retinal vascular fractal dimension and its relationship with cardiovascular and ocular risk factors. *American Journal of Ophthalmology*, In Press.

Cheung, C. Y.-L., Zheng, Y., Hsu, W., Lee, M. L., Lau, Q. P., Mitchell, P., Wang, J. J., Klein, R., and Wong, T. Y. (2011b). Retinal vascular tortuosity, blood pressure, and cardiovascular risk factors. *Ophthalmology*, 118(5):812–818.

Chitsaz, M. and Seng, W. C. (2009). Medical image segmentation by using reinforcement learning agent. In *Proceedings of the International Conference on Digital Image Processing*, pages 216–219, Washington, DC, USA. IEEE Computer Society.

Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the 15th National Conference on Artificial Intelligence and 10th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI)*, pages 746–752.

Conde, T. and Thalmann, D. (2006). Learnable behavioural model for autonomous virtual agents: low-level learning. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 89–96, New York, NY, USA. ACM.

Cosatto, V., Liew, G., Rochtchina, E., Wainwright, A., Zhang, Y. P., Hsu, W., Lee, M. L., Lau, Q. P., Hamzah, H., Mitchell, P., Wong, T. Y., and Wang, J. J. (2010). Retinal vascular fractal dimension measurement and its influence from imaging variation: Results of two segmentation methods. *Current Eye Research*, 35(9):850–856.

Croonenborghs, T., Ramon, J., Blockeel, H., and Bruynooghe, M. (2007). Online learning and exploiting relational models in reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 726–731, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Croonenborghs, T., Tuyls, K., Ramon, J., and Bruynooghe, M. (2005). Multi-agent relational reinforcement learning. In Tuyls, K., Hoen, P. J., Verbeeck, K., and Sen, S., editors, *LAMAS*, volume 3898 of *Lecture Notes in Computer Science*, pages 192–206. Springer.

Dechter, R. (1996). Bucket elimination: a unifying framework for processing hard and soft constraints. *ACM Computing Surveys*, 28.

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85.

Devlin, S. and Kudenko, D. (2012). Dynamic potential-based reward shaping. In *Proceedings of The 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 433–440.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.

Džeroski, S., De Raedt, L., and Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, 43(1/2):7–52.

Epshteyn, A., Vogel, A., and DeJong, G. (2008). Active reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, ICML, pages 296–303, New York, NY, USA. ACM.

Ferreira, E. and Khosla, P. (2000). Multi agent collaboration using distributed value functions. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, pages 404–409.

Foong, A. W. P., Saw, S.-M., Loo, J.-L., Shen, S., Loon, S.-C., Rosman, M., Aung, T., Tan, D. T. H., Tai, E. S., and Wong, T. Y. (2007). Rationale and methodology for a population-based study of eye diseases in Malay people: The Singapore Malay eye study (simes). *Ophthalmic Epidemiol*, 14(1):25–35.

Fox, M. and Poole, D., editors (2010). *Proceedings of the 24th AAAI Conference on Artificial Intelligence, Atlanta, Georgia, USA, July 11-15*. AAAI Press.

Garg, S., Sivaswamy, J., and Chandra, S. (2007). Unsupervised curvature-based retinal vessel segmentation. In *Proceedings of the 4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro.*, pages 344–347.

Geibel, P. (2006). Reinforcement learning for MDPs with constraints. In Fürnkranz, J., Scheffer, T., and Spiliopoulou, M., editors, *Proceedings of the 17th European Conference on Machine Learning (ECML)*, volume 4212 of *Lecture Notes in Computer Science*, pages 646–653. Springer.

Geibel, P. and Wysotzki, F. (2005). Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 24(1):81–108.

Gelly, S. and Silver, D. (2007). Combining online and offline knowledge in UCT. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 273–280, New York, NY, USA. ACM Press.

Ghavamzadeh, M., Mahadevan, S., and Makar, R. (2006). Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 13(2):197–229.

Guestrin, C., Koller, D., Gearhart, C., and Kanodia, N. (2003). Generalizing plans to new environments in relational MDPs. In Gottlob, G. and Walsh, T., editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1003–1010. Morgan Kaufmann.

Guestrin, C., Koller, D., and Parr, R. (2001). Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems 14*, pages 1523–1530. The MIT Press.

Guestrin, C., Lagoudakis, M. G., and Parr, R. (2002). Coordinated reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, pages 227–234, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Hart, W. E., Goldbaum, M., Kube, P., and Nelson, M. R. (1997). Automated measurement of retinal vascular tortuosity. In *Proceedings of the American Medical Informatics Association Fall Symposium*, pages 459–463.

Hsu, W., Lau, Q. P., and Lee, M. L. (2009). Detecting aggregate incongruities in XML. In Zhou, X., Yokota, H., Deng, K., and Liu, Q., editors, *Proceedings of the 14th International Conference on Database Systems for Advanced Applications (DASFAA)*, volume 5463 of *Lecture Notes in Computer Science*, pages 601–615. Springer.

Jong, N. K., Hester, T., and Stone, P. (2008). The utility of temporal abstraction in reinforcement learning. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 299–306, Richland, SC. IFAAMAS.

Judah, K., Roy, S., Fern, A., and Dietterich, T. G. (2010). Reinforcement learning via practice and critique advice. In Fox and Poole (2010).

Kalyanakrishnan, S. and Stone, P. (2009). An empirical analysis of value function-based and policy search reinforcement learning. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, volume 2 of *AAMAS*, pages 749–756, Richland, SC. IFAAMAS.

Kang, H., Lee, S., and Chui, C. K. (2007). Coherent line drawing. In *Proceedings of the 5th International Symposium on Non-Photorealistic Animation and Rendering*, pages 43–50, New York, NY, USA. ACM.

Kersting, K. and De Raedt, L. (2004). Logical Markov decision programs and the convergence of logical TD(λ). In Camacho, R., King, R., and Srinivasan, A., editors, *Inductive Logic Programming*, volume 3194 of *Lecture Notes in Computer Science*, pages 103–116. Springer Berlin / Heidelberg.

Kok, J. R., Jan, P., Bram, H., and Vlassis, B. N. (2005). Utile coordination: Learning interdependencies among cooperative agents. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 29–36.

Kok, J. R. and Vlassis, N. (2004). Sparse cooperative Q-learning. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, pages 61–68, New York, NY, USA. ACM.

Kok, J. R. and Vlassis, N. (2006). Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7:1789–1828.

Konidaris, G. and Barto, A. (2007). Building portable options: skill transfer in reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 895–900, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Kschischang, F., Frey, B., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498 –519.

Kuyer, L., Whiteson, S., Bakker, B., and Vlassis, N. (2008). Multiagent reinforcement learning for urban traffic control using coordination graphs. In *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I*, ECML PKDD '08, pages 656–671. Springer-Verlag.

Lagoudakis, M. G. and Parr, R. (2001). Model-free least-squares policy iteration. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems*, pages 1547–1554. MIT Press.

Larrosa, J. and Dechter, R. (2003). Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. *Constraints*, 8(3):303–326.

Lau, Q. P., Hsu, W., and Lee, M. L. (2008). Deepdetect: An extensible system for detecting attribute outliers & duplicates in XML. In Chan, C.-Y., Chawla, S., Sadiq, S., Zhou, X., and Pudi, V., editors, *Data Quality and High-Dimensional Data Analysis: Proceedings of the DASFAA 2008 Workshops*, pages 6–20. World Scientific.

Lau, Q. P., Hsu, W., Lee, M. L., Mao, Y., and Chen, L. (2007). Prediction of cerebral aneurysm rupture. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 350–357. IEEE Computer Society.

Lau, Q. P., Lee, M. L., and Hsu, W. (2011). Distributed coordination guidance in multiagent reinforcement learning. In *Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 456–463. IEEE Computer Society.

Lau, Q. P., Lee, M. L., and Hsu, W. (2012). Coordination guided reinforcement learning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pages 215–222. IFAAMAS.

Lau, Q. P., Lee, M. L., and Hsu, W. (2013). Distributed relational temporal difference learning. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. IFAAMAS.

Lauer, M. and Riedmiller, M. A. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pages 535–542, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Liew, G., Sharrett, A. R., Wang, J. J., Klein, R., Klein, B. E. K., Mitchell, P., and Wong, T. Y. (2008). Relative importance of systemic determinants of retinal arteriolar and venular caliber: the atherosclerosis risk in communities study. *Arch Ophthalmol*, 126(10):1404–1410.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Marthi, B. (2007). Automatic shaping and decomposition of reward functions. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 601–608, New York, NY, USA. ACM.

Marthi, B., Latham, D., Russell, S., and Guestrin, C. (2005). Concurrent hierarchical reinforcement learning. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 779–785, Edinburgh, Scotland.

Marthi, B., Russell, S. J., and Andre, D. (2006). A compact, hierarchical Q-function decomposition. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 332–340. AUAI Press.

McGeechan, K., Liew, G., Macaskill, P., Irwig, L., Klein, R., Klein, B. E. K., Wang, J. J., Mitchell, P., Vingerling, J. R., de Jong, P. T. V. M., Witteman, J. C. M., Breteler, M. M. B., Shaw, J., Zimmet, P., and Wong, T. Y. (2009a). Prediction of incident stroke events based on retinal vessel caliber: a systematic review and individual-participant meta-analysis. *Am J Epidemiol*, 170(11):1323–1332.

McGeechan, K., Liew, G., Macaskill, P., Irwig, L., Klein, R., Klein, B. E. K., Wang, J. J., Mitchell, P., Vingerling, J. R., Dejong, P. T. V. M., Witteman, J. C. M., Breteler, M. M. B., Shaw, J., Zimmet, P., and Wong, T. Y. (2009b). Meta-analysis: retinal vessel caliber and risk for coronary heart disease. *Ann Intern Med*, 151(6):404–413.

Mehta, N., Ray, S., Tadepalli, P., and Dietterich, T. (2008). Automatic discovery and transfer of MAXQ hierarchies. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 648–655, New York, NY, USA. ACM.

Melo, F. S., Meyn, S. P., and Ribeiro, M. I. (2008). An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 664–671, New York, NY, USA. ACM.

Neapolitan, R. E. (2003). *Learning Bayesian Networks*. Prentice Hall.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning (ICML)*, pages 278–287, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Ng, A. Y. and Russell, S. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pages 663–670, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Pallawala, P., Hsu, W., Lee, M., and Eong, K.-G. (2004). Automated optic disc localization and contour detection using ellipse fitting and wavelet transform. In Pajdla, T. and Matas, J., editors, *Computer Vision - ECCV 2004*, volume 3022 of *Lecture Notes in Computer Science*, pages 139–151. Springer Berlin / Heidelberg.

Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

Parr, R. and Russell, S. (1997). Reinforcement learning with hierarchies of machines. In Jordan, M. I., Kearns, M. J., and Solla, S. A., editors, *Advances in Neural Information Processing Systems*, volume 10, pages 1043–1049. The MIT Press.

Patton, N., Aslam, T. M., MacGillivray, T., Deary, I. J., Dhillon, B., Eikelboom, R. H., Yogesan, K., and Constable, I. J. (2006). Retinal image analysis: Concepts, applications and potential. *Progress in Retinal and Eye Research*, 25:99–127.

Perkins, T. J. and Precup, D. (2002). A convergent form of approximate policy iteration. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 1595–1602. MIT Press.

Peshkin, L., Kim, K.-E., Meuleau, N., and Kaelbling, L. P. (2000). Learning to cooperate via policy search. In *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 489–496. Morgan Kaufmann.

Petcu, A. and Faltings, B. (2005). A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 266–271, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Ponsen, M., Croonenborghs, T., Tuyls, K., Ramon, J., Driessens, K., van den Herik, J., and Postma, E. (2010). Learning with whom to communicate using relational reinforcement learning. In Babuka, R. and Groen, F., editors, *Interactive Collaborative Information Systems*, volume 281 of *Studies in Computational Intelligence*, pages 45–63. Springer Berlin / Heidelberg.

Poupart, P., Vlassis, N., Hoey, J., and Regan, K. (2006). An analytic solution to discrete Bayesian reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 697–704, New York, NY, USA. ACM.

Proper, S. and Tadepalli, P. (2009). Solving multiagent assignment Markov decision processes. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1 of *AAMAS*, pages 681–688, Richland, SC. IFAAMAS.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR, 166. Engineering Department, Cambridge University.

Russell, S. J., Norvig, P., Candy, J. F., Malik, J. M., and Edwards, D. D. (1996). *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Russell, S. J. and Zimdars, A. (2003). Q-decomposition for reinforcement learning agents. In Fawcett, T. and Mishra, N., editors, *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pages 656–663. AAAI Press.

Sahba, F., Tizhoosh, H. R., and Salama, M. M. (2008). A reinforcement agent for object segmentation in ultrasound images. *Expert Systems with Applications*, 35(3):772 – 780.

Sallans, B. and Hinton, G. E. (2004). Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, 5:1063–1088.

Sánchez, M., Meseguer, P., and Larrosa, J. (2004). Improving the applicability of adaptive consistency: Preliminary results. In Wallace, M., editor, *Principles and Practice of Constraint Programming (CP)*, volume 3258 of *Lecture Notes in Computer Science*, pages 757–761. Springer.

Schneider, J. G., Wong, W.-K., Moore, A. W., and Riedmiller, M. A. (1999). Distributed value functions. In *Proceedings of the 16th International Conference on Machine Learning (ICML)*, ICML, pages 371–378, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Silver, D., Sutton, R., and Müller, M. (2007). Reinforcement learning of local shape in the game of Go. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1053–1058, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Silver, D., Sutton, R., and Müller, M. (2008). Sample-based learning and search with permanent and transient memories. In McCallum, A. and Roweis, S., editors, *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 968–975. Omnipress.

Stone, P. and Sutton, R. S. (2001). Scaling reinforcement learning toward RoboCup soccer. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pages 537–544. Morgan Kaufmann, San Francisco, CA.

Stranders, R., Fave, F. M. D., Rogers, A., and Jennings, N. R. (2010). A decentralised coordination algorithm for mobile sensors. In Fox and Poole (2010), pages 874–880.

Strens, M. (2004). Relational spatial features in reinforcement learning of multi-agent search strategies. In *Proceedings of the ICML04 Workshop on Relational Reinforcement Learning*.

Strens, M. J. A. (2000). A Bayesian framework for reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pages 943–950, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

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

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. MIT Press.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. (2009a). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 993–1000, New York, NY, USA. ACM.

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

Sutton, R. S., Szepesvári, C., and Maei, H. R. (2009b). A convergent $O(n)$ temporal-difference algorithm for off-policy learning with linear function approximation. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 1609–1616.

Tadepalli, P., Givan, R., and Driessens, K. (2004). Relational reinforcement learning: An overview. In *Proceedings of the ICML04 Workshop on Relational Reinforcement Learning*, pages 1–9.

Teacy, W. T. L., Chalkiadakis, G., Farinelli, A., Rogers, A., Jennings, N. R., McClean, S., and Parr, G. (2012). Decentralized Bayesian reinforcement learning for online agent collaboration. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pages 417–424. IFAAMAS.

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219.

Tsitsiklis, J. N. and Roy, B. V. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., and Dekker, E. (2011). Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84:51–80.

Wainwright, M., Jaakkola, T., and Willsky, A. (2004). Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. *Statistics and Computing*, 14(2):143–166.

Wang, L., Merrifield, R., and Yang, G.-Z. (2011). Reinforcement learning for context aware segmentation. In Fichtinger, G., Martel, A., and Peters, T., editors, *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 6893 of *Lecture Notes in Computer Science*, pages 627–634. Springer Berlin / Heidelberg.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK.

Wilson, A., Fern, A., Tadepalli, P., and Ray, S. (2007). Multi-task reinforcement learning: a hierarchical Bayesian approach. In Ghahramani, Z., editor, *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 1015–1022. Omnipress.

Witt, N., Wong, T. Y., Hughes, A. D., Chaturvedi, N., Klein, B. E., Evans, R., McNamara, M., Thom, S. A. M., and Klein, R. (2006). Abnormalities of retinal microvascular structure and risk of mortality from ischemic heart disease and stroke. *Hypertension*, 47(5):975–981.

Wong, T. Y., Islam, F. M. A., Klein, R., Klein, B. E. K., Cotch, M. F., Castro, C., Sharrett, A. R., and Shahar, E. (2006a). Retinal vascular caliber, cardiovascular risk factors, and inflammation: the multi-ethnic study of atherosclerosis (mesa). *Invest Ophthalmol Vis Sci*, 47(6):2341–2350.

Wong, T. Y., Kamineni, A., Klein, R., Sharrett, A. R., Klein, B. E., Siscovick, D. S., Cushman, M., and Duncan, B. B. (2006b). Quantitative retinal venular caliber and risk of cardiovascular disease in older persons: the cardiovascular health study. *Arch Intern Med*, 166(21):2388–2394.

Wong, T. Y., Klein, R., Sharrett, A. R., Duncan, B. B., Couper, D. J., Klein, B. E. K., Hubbard, L. D., Nieto, F. J., and in Communities Study, A. R. (2004a). Retinal arteriolar diameter and risk for hypertension. *Ann Intern Med*, 140(4):248–255.

Wong, T. Y., Knudtson, M. D., Klein, R., Klein, B. E. K., Meuer, S. M., and Hubbard, L. D. (2004b). Computer-assisted measurement of retinal vessel diameters in the beaver dam eye study: methodology, correlation between eyes, and effect of refractive errors. *Ophthalmology*, 111(6):1183–1190.

Wu, J. and Durfee, E. H. (2005). Automated resource-driven mission phasing techniques for constrained agents. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 331–338, New York, NY, USA. ACM.

Yue, B. and de Byl, P. (2006). The state of the art in game AI standardisation. In *Proceedings of the 2006 International Conference on Game Research and Development (CyberGames)*, pages 41–46, Murdoch University, Australia, Australia. Murdoch University.

Zhang, C., Abdallah, S., and Lesser, V. (2009). Integrating organizational control into multi-agent learning. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 2 of *AAMAS*, pages 757–764, Richland, SC. IFAAMAS.

Zhang, C., Lesser, V., and Abdallah, S. (2010). Self-organization for coordinating decentralized reinforcement learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1 of *AAMAS*, pages 739–746, Richland, SC. IFAAMAS.

Zhang, C. and Lesser, V. R. (2010). Multi-agent learning with policy prediction. In Fox and Poole (2010), pages 927–934.

Zhang, N. L. and Poole, D. (1996). Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328.

A Blank Page

Appendix A

Implementation Details

This chapter contains implementation details such as predicates that are used to create features for function approximation and coordination constraints. Code for the experiments in this thesis was written in C++, compiled as 32-bit binaries, and ran on Linux.

In the functions and predicates, we refer to a controllable agent as a friendly unit a_i, an enemy as e_i, and either type of unit as u_i. We presume that these variables include both the state and action of the agents they refer to. To simplify, we have written functions and predicates with the action variables (agents) only. A *unit* refers to some agent entity, e.g. a soccer player or a marine.

To facilitate listing we define the set conjunction operator $\bigwedge$ for two sets of predicates with the same bound variables that returns a set of conjunctions of those predicates. For example, given the sets of predicates,

$$P1 = \{p1_1(a_i), p1_2(a_i), ..., p1_m(a_i)\}, \text{ and } P2 = \{p2_1(a_i), p2_2(a_i), ..., p2_n(a_i)\},$$

then $P1 \bigwedge P2$ is the set of predicates formed by conjuncting the values in every tuple of the set product $P1 \times P2$, i.e.,

$$P1 \bigwedge P2 = \begin{Bmatrix} p1_1(a_i) \wedge p2_1(a_i) & p1_1(a_i) \wedge p2_2(a_i) & \cdots & p1_1(a_i) \wedge p2_n(a_i) \\ p1_2(a_i) \wedge p2_1(a_i) & p1_2(a_i) \wedge p2_2(a_i) & \cdots & p1_2(a_i) \wedge p2_n(a_i) \\ \vdots & \vdots & \ddots & \vdots \\ p1_m(a_i) \wedge p2_1(a_i) & p1_m(a_i) \wedge p2_2(a_i) & \cdots & p1_m(a_i) \wedge p2_n(a_i) \end{Bmatrix}.$$

Where each conjuncted predicate's variable, a_i, is the same. The same applies to sets of predicates involving multiple agents.

For the centralized methods, function approximation for the entire learning system as a whole does not contain duplicate features. For the distributed methods, each U_i, W_i or Q_i function contains all the non-predicate based features and all the predicate based features that agent i is involved in. Hence in the global sense, pairwise features may be duplicated. For example, in distributed CGRL for RTS, marines 1 and 2 will each have a copy of the $PairAttack(a_1, a_2)$ propositional feature and a corresponding weight to learn.

A.1 Simplified Soccer Game

A.1.1 Base Predicates & Functions

The following are basic functions:

1. $distance(u_i, u_j)$ is the Manhattan distance between two soccer players.
2. P_{ball} A function of the state that returns the player object that has the ball.
3. $nearestEnemy(a_i)$ A function of the state that returns the nearest enemy player object.
4. $pass(a_i)$ A function that returns the player object that player a_i action passes to, otherwise returns null player.
5. $intercept(a_i)$ A function that returns the enemy player object that player a_i after taking a move action will collide with. Null if no such enemy or player is not moving.
6. $moveTo(a_i)$ A function that returns the player object with new positional coordinates if the action was successful, otherwise the player object with current positional coordinates is returned.

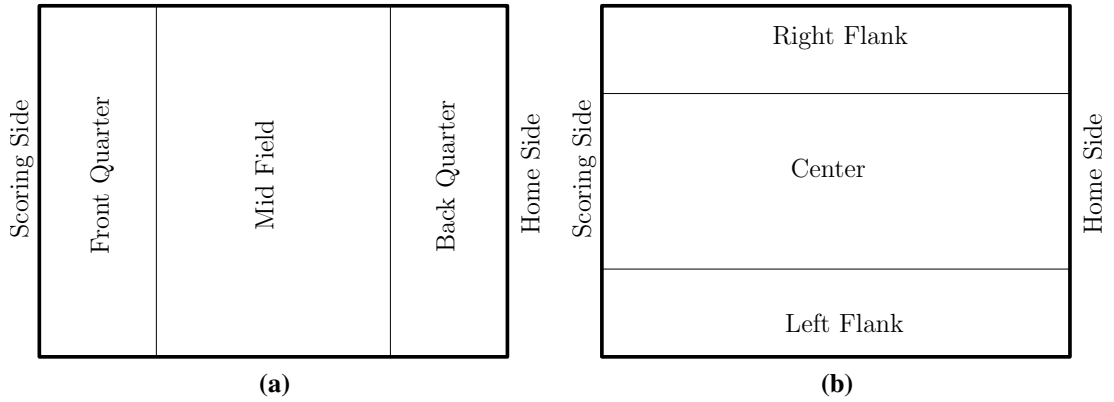


Figure A.1: Soccer field positions

Let $\mathcal{A}$ be the set of friendly controlled players and $\mathcal{E}$ be the set of opposing enemy players. The following are state only base level predicates with reference to the parts of the soccer field given in Figure A.1.

7. $HasBall(u_i) := u_i = P_{ball}$ the player has the ball
8. $BackQuarter(u_i)$ player is in the quarter of the field nearer to its home goal.
9. $MidField(u_i)$ player is in the middle half of the field, from a quarter to three quarters.
10. $FrontQuarter(u_i)$ player is in the quarter of the field nearer to its scoring goal.
11. $RightFlank(u_i)$ player is in the right flank.
12. $LeftFlank(u_i)$ player is in the left flank.
13. $Center(u_i)$ player is in the center.
14. $Flank(u_i) := RightFlank(u_i) \vee LeftFlank(u_i)$.
15. $EnemyWithin_d(a_i) := distance(a_i, nearestEnemy(a_i)) \leq d$, player is $\leq d$ distance to an enemy player.
16. $TeamHasBall(\mathcal{T})$ the team, $\mathcal{A}$ or $\mathcal{E}$, has the ball

17. $TeamOffensive(\mathcal{T}) := \forall u_i \in \mathcal{T}, \neg BackQuarter(u_i)$ the entire team of players $\mathcal{T}$ are not in the back quarter of the field.
18. $TeamDefensive(\mathcal{T}) := \forall u_i \in \mathcal{T}, \neg FrontQuarter(u_i)$ the entire team of players $\mathcal{T}$ are not in the front quarter of the field.

The following are predicates based on player's actions.

19. $Pass(a_i)$ player is passing the ball.
20. $IsPass(a_i, a_j) := Pass(a_i) \wedge pass(a_i) = a_j$.
21. $Forward(a_i)$ action taken moves player strictly closer to scoring side.
22. $Backward(a_i)$ action taken moves player strictly closer to home side.
23. $Left(a_i)$ action taken moves player strictly closer to the left of the field.
24. $Right(a_i)$ action taken moves player strictly closer to the right of the field.
25. $MoveToScoring(a_i)$ player moving towards scoring side.
26. $MoveToHome(a_i)$ player moving towards home side.
27. $MoveToBall(a_i)$ player moving closer to ball location, false if player has ball.
28. $MoveFromBall(a_i)$ player moving further from ball location, false if player has ball.
29. $Shoot(a_i)$ action shoots.
30. $PassFront(a_i)$ player is passing the ball to a receiver that is in front of the player.
31. $PassBack(a_i)$ player is passing the ball to a receiver that is behind the player.
32. $PassSide(a_i)$ player is passing the ball to a receiver either left or right to the player.
33. $MoveWithin_d(a_i) := EnemyWithin_d(moveTo(a_i))$ player moving to $\leq d$ distance to enemy.

A.1.2 Bottom Level Predicates

The following are predicates used as features for bottom level function approximation for U and Q. First, we list the predicates based on only the state:

34. $BackQuarter(P_{ball})$
35. $FrontQuarter(P_{ball})$
36. $MidField(P_{ball})$
37. $TeamDefensive(\mathcal{A})$
38. $TeamOffensive(\mathcal{A})$
39. $TeamDefensive(\mathcal{E})$
40. $TeamOffensive(\mathcal{E})$
41. $TeamHasBall(\mathcal{A})$
42. $TeamHasBall(\mathcal{E})$

Second, the list of features based on single player actions (unary) in the team:

43. Base predicates 21 to 32 used directly.
44. $CollideBorder(a_i)$ player will collide with the environment boundary.
45. $CollideEnemy(a_i)$ player will move to a square that contains an enemy, note that player will loss ball if it has a ball or will intercept an enemy player's ball.
46. $MoveWithin_1(a_i)$.
47. $MoveWithin_2(a_i)$.
48. $NoShoot(a_i) := HasBall(a_i) \wedge \neg Shoot(a_i)$.

49. $BallCollideEnemy(a_i) := HasBall(a_i) \wedge CollideEnemy(a_i)$.
50. $Intercept(a_i) := intercept(a_i) \neq \emptyset \wedge HasBall(intercept(a_i))$.
51. $\neg Intercept(a_i)$.
52. Not passing or shooting when enemy is in the next grid square.

$$\begin{aligned} &NotBallActionNextToEnemy(a_i) \\ :=&HasBall(a_i) \wedge \neg Shoot(a_i) \\ &\wedge \neg Pass(a_i) \wedge distance(a_i, nearestEnemy(a_i)) \leq 1. \end{aligned}$$

53. $PassFrontQuarter(a_i) := IsPass(a_i) \wedge FrontQuarter(pass(a_i))$.
54. $PassBackQuarter(a_i) := IsPass(a_i) \wedge BackQuarter(pass(a_i))$.
55. $PassMidField(a_i) := IsPass(a_i) \wedge MidField(pass(a_i))$.
56. $PassLeftFlank(a_i) := IsPass(a_i) \wedge LeftFlank(pass(a_i))$.
57. $PassRightFlank(a_i) := IsPass(a_i) \wedge RightFlank(pass(a_i))$.
58. $PassCenter(a_i) := IsPass(a_i) \wedge Center(pass(a_i))$.
59. $PassEnemyNear(a_i) := IsPass(a_i) \wedge EnemyWithin_1(a_i)$.
60. $MoveToEnemyBall(a_i) := TeamHasBall(\mathcal{E}) \wedge MoveToBall(a_i)$.
61. $MoveFromEnemyBall(a_i) := TeamHasBall(\mathcal{E}) \wedge MoveFromBall(a_i)$.
62. Move next to an enemy player with the ball. Let the statement

$$E := nearestEnemy(moveTo(a_i)).$$

Then,

$$\begin{aligned} &NearEnemyBall(a_i) \\ :=&nearestEnemy(moveTo(a_i)) = P_{ball} \wedge MoveWithin_1(a_i). \end{aligned}$$

63. Let M_a be the set of positional predicates 8–13. Let M_b be the set of movement related predicates 21–26, and 44–47. We use unary predicates in the set $\{HasBall(a_i)\} \cup M_a \bigwedge M_b$.
64. Let M_c be the set of predicates 27–32, and 48–62. We use unary predicates in the set $M_a \bigwedge M_c$.

Third, the list of predicates based on pairwise players' action in the team:

65. $Collide(a_i, a_j)$ players will move to the same square.
66. $BallCollide(a_i, a_j) := [HasBall(a_i) \vee HasBall(a_j)] \wedge Collide(a_i, a_j)$.
67. Moving nearer when one of the players has the ball.

$$\begin{aligned} &BallNearer(a_i, a_j) \\ :=\ &[HasBall(a_i) \vee HasBall(a_j)] \\ &\wedge distance(a_i, a_j) < distance(moveTo(a_i), moveTo(a_j)) \end{aligned}$$

68. Moving further when one of the players has the ball.

$$\begin{aligned} &BallFurther(a_i, a_j) \\ :=\ &[HasBall(a_i) \vee HasBall(a_j)] \\ &\wedge distance(a_i, a_j) > distance(moveTo(a_i), moveTo(a_j)) \end{aligned}$$

69. Moving towards enemy with ball.

$$EnemyBallNearer(a_i, a_j) := TeamHasBall(\mathcal{E}) \\ \wedge MoveToBall(a_i) \wedge MoveToBall(a_j)$$

70. Moving away from enemy with ball.

$$EnemyBallFurther(a_i, a_j) := TeamHasBall(\mathcal{E}) \\ \wedge MoveFromBall(a_i) \wedge MoveFromBall(a_j)$$

71. Moving back to midfield from front quarter.

$$Defensive(a_i, a_j) := FrontQuarter(a_i) \wedge FrontQuarter(a_j) \\ \wedge MidField(moveTo(a_i)) \wedge MidField(moveTo(a_j))$$

72. Moving forward to midfield from back quarter.

$$Offensive(a_i, a_j) := BackQuarter(a_i) \wedge BackQuarter(a_j) \\ \wedge MidField(moveTo(a_i)) \wedge MidField(moveTo(a_j))$$

73. Moving forward along the flanks.

$$FlankOffensive(a_i, a_j) := Flank(a_i) \wedge Flank(a_j) \\ \wedge Forward(a_i) \wedge Forward(a_j)$$

74. Moving backward along the flanks.

$$FlankDefence(a_i, a_j) := Flank(a_i) \wedge Flank(a_j) \\ \wedge Backward(a_i) \wedge Backward(a_j)$$

75. Joint interception,

$$JointIntercept(a_i, a_j) := Intercept(a_i) \wedge Intercept(a_j).$$

76. $\neg JointIntercept(a_i, a_j)$.
77. $Less4(a_i, a_j) := distance(moveTo(a_i), moveTo(a_j)) \leq 4$.
78. $Less8(a_i, a_j) := distance(moveTo(a_i), moveTo(a_j)) \leq 8$.
79. $GoodPass(a_i, a_j) := IsPass(a_i, a_j) \wedge \neg MoveWithin_1(a_i)$.
80. $BadPass(a_i, a_j) := IsPass(a_i, a_j) \wedge MoveWithin_1(a_i)$. This is a redefinition of Example 4.6 page 81.
81. Not moving together towards enemy with ball.

$$NotPairBlock(a_i, a_j) := TeamHasBall(\mathcal{E}) \\ \wedge [distance(moveTo(a_i), P_{ball}) > distance(a_i, P_{ball}) \\ \vee distance(moveTo(a_j), P_{ball}) > distance(a_j, P_{ball})]$$

82. Not jointly moving closer to intercept when within distance of 2.

$$\begin{aligned} & NotMoveToIntercept(a_i, a_j) \\ := \; & TeamHasBall(\mathcal{E}) \\ & \wedge distance(a_i, P_{ball}) \leq 2 \\ & \wedge distance(a_j, P_{ball}) \leq 2 \\ & \wedge distance(moveTo(a_i), P_{ball}) < distance(a_i, P_{ball}) \\ & \wedge distance(moveTo(a_j), P_{ball}) < distance(a_j, P_{ball}) \end{aligned}$$

A.1.3 Coordination Constraints

Constraints are the negation of selected predicates. The following are static CCs in addition to obvious constraints (e.g. shooting or passing without the ball):

83. $BackQuarter(a_i) \wedge Shoot(a_i)$, this comes from 63;
84. $BallCollideEnemy(a_i)$,
85. $Collide(a_i, a_j)$,
86. $BallCollide(a_i, a_j)$.

The set of dynamic single agent CCs, C_1, used are:

87. $HasBall(a_i) \wedge Backward(a_i)$ from 63,
88. $HasBall(a_i) \wedge MoveWithin_1(a_i)$ from 63,
89. $NotBallActionNextToEnemy(a_i)$,
90. $MoveFromEnemyBall(a_i)$,
91. $FrontQuarter(a_i) \wedge NoShoot(a_i)$ from 64,
92. $MidField(a_i) \wedge NoShoot(a_i)$ from 64,
93. $\neg Intercept(a_i)$,

The dynamic pairwise agent CCs, C_2, used are:

94. $BadPass(a_i, a_j)$,
95. $NotPairBlock(a_i, a_j)$,
96. $NotMoveToIntercept(a_i, a_j)$.

A.1.4 Top Level Predicates

The top level predicates for features are programatically generated using state only predicates and the $Activated(c)$ predicate that returns true if a top level CC is activated. Let T_0 be the predicates in 34–42 that involve only the state, $T_1 = \{HasBall(a_i)\} \cup M_a$ from 63, $T_{2,1}$ is the set of disjunction of the predicates in T_1, e.g., $MidField(a_i) \vee MidField(a_j)$, and $T_{2,2}$ is the set of predicates $\{HasBall(a_i) \vee HasBall(a_j), Flank(a_i) \wedge Flank(a_j)\}$. Further let $\mathcal{C}_1 = \{Activated(c) \mid c \in C_1\}$ and $\mathcal{C}_2 = \{Activated(c) \mid c \in C_2\}$. The following top level predicates are generated:

97. $T_0 \bigwedge \mathcal{C}_1$,
98. $T_1 \bigwedge \mathcal{C}_1$,
99. $T_0 \bigwedge \mathcal{C}_2$,
100. $T_{2,1} \bigwedge \mathcal{C}_2$,
101. $T_{2,2} \bigwedge \mathcal{C}_2$.

A.2 Tactical Real Time Strategy

A.2.1 Base Predicates & Functions

The following are basic functions.

1. $nearestEnemy(a_i)$ Return the nearest enemy unit object.
2. $nearestFriend(a_i)$ Return the nearest friendly unit object.
3. $target(a_i)$ Return the enemy unit object that is the current target of an attack action taken by the unit. If the unit is not attacking, the unique null enemy object is returned.
4. $distance(a_i, e_j)$ Euclidean distance between a friendly and enemy unit after friendly unit's action is taken. The distance is scaled over the diagonal length of the map.
5. $distance(a_i, a_j)$ Euclidean distance between two friendly units after their actions are taken. The distance is scaled over the diagonal length of the map.
6. $targetDamage(a_i)$ Returns the health points lost in percentage of the current attack target of unit a_i or zero if a_i is not taking an attack action.
7. E_{iso} A function of the state that returns the most isolated enemy that is furthest away from its other teammates.
8. $health(u_i)$ Returns the health of a unit in a percentage.
9. $weaker(u_i, u_j)$ Returns the unit object with lower health.

The following predicates were used to build (e.g., through conjunction) more complex predicates used as features. Predicates are functions that return a boolean value in $\{0, 1\}$.

10. $BoundaryCollide(a_i)$ Action taken by unit will cause it to collide with some boundary.
11. $EnemyCollide(a_i)$ Action taken by unit will cause it to collide with some enemy.
12. $TargetOutRange(a_i)$ Unit is taking attack action and target of attack is out of range of the unit's weapons.
13. $Stoning(a_i)$ Unit is taking *noop* action while within range of some enemy's weapon.
14. $Idling(a_i)$ Unit is taking *noop* action.
15. $PairCollide(a_i, a_j)$ Units a_i and a_j will collide after taking their respective actions.
16. $MoveToIsolated(a_i)$ Moving towards the isolated enemy.
17. $NoAttack(a_i)$ Unit is not attacking any enemy within range when there is at least one enemy within range of the unit's weapon.
18. $Closer(a_i, e_i)$ Unit takes an action that moves it closer to enemy e_i.
19. $Further(a_i, e_i)$ Unit takes an action that moves it further from enemy e_i.
20. $Attacked(a_i)$ Unit is within some enemy's attack range.
21. $IsAttack(a_i)$ Unit takes some attack action.
22. $TargetInRange(a_i)$ Unit takes attack action (i.e. $IsAttack(a_i)$ is true) and the target is in range of the unit's weapon.
23. $SameNearestEnemy(a_i, a_j) := nearestEnemy(a_i) = nearestEnemy(a_j)$ Both units are closest to the same nearest enemy unit.
24. $HealthWithin_{h_1,h_2}(u_i)$ Health of unit(friendly or enemy) in percentage is between $(h_1, h_2]$.

A.2.2 Bottom Level Predicates

In this section we list the bottom level predicates used as features for function U, they are also the features for flat RL's action value function Q. The following predicate based features were constructed using base predicates and functions. These predicates are either used as propositional features by binding specific agents to their parameters, or relational features by summing over various possible bindings.

25. These base predicates are used directly as features:
$Stoning(a_i), Idling(a_i), PairCollide(a_i, a_j)$.
26. $NotAligned(a_i, a_j)$ as defined in Equation 6.1 page 142.
27. Both units a_i and a_j take attack actions targeting the same enemy unit that is within attack range of both of them:

$$PairAttack(a_i, a_j) := IsAttack(a_i) \wedge IsAttack(a_j) \wedge target(a_i) = target(a_j)$$

28. Units a_i and a_j will have Euclidean distance within the range $(l, h]$ after their actions are taken,

$$PairDist_{l,h}(a_i, a_j) := l < distance(a_i, a_j) \leq h,$$

The following ranges were used: $(0, 20]$, $(20, 40]$, $(40, 60]$, and $(60, \infty]$.
29. The following encode knowledge with reference to isolated enemies:
    (a) $Iso1(a_i) := health(a_i) \geq health(E_{iso}) \wedge Closer(a_i, E_{iso})$
    (b) $Iso2(a_i) := health(a_i) < health(E_{iso}) \wedge Closer(a_i, E_{iso})$
    (c) $Iso3(a_i) := health(a_i) \geq health(E_{iso}) \wedge Further(a_i, E_{iso})$
    (d) $Iso4(a_i) := health(a_i) < health(E_{iso}) \wedge Further(a_i, E_{iso})$
    (e) $PairIso1(a_i, a_j) := Iso1(a_i) \wedge Iso1(a_j)$
    (f) $PairIso2(a_i, a_j) := Iso2(a_i) \wedge Iso2(a_j)$
    (g) $PairIso3(a_i, a_j) := Iso3(a_i) \wedge Iso3(a_j)$
    (h) $PairIso4(a_i, a_j) := Iso4(a_i) \wedge Iso4(a_j)$
30. Let the set $H = \{HealthWithin_{h_1,h_2}(a_i)\}$ for the health ranges: $(-\infty, 0.25]$, $(0.25, 0.5]$, $(0.5, 0.75]$ and $(0.75, 1.0]$. Let the set,

$$P1 = \left\{ \begin{array}{cc} NoAttack(a_i), & Closer(a_i, nearestEnemy(a_i)), \\ Attacked(a_i), & Further(a_i, nearestEnemy(a_i)), \\ TargetInRange(a_i) & \end{array} \right\}$$

We use the conjuncted predicates in the set $H \bigwedge P1$ as features.
31. Two agents moving closer to same nearest enemy:

$$\begin{aligned} PairCloser(a_i, a_j) :=& SameNearestEnemy(a_i, a_j) \\ & \wedge Closer(a_i, nearestEnemy(a_i)) \\ & \wedge Closer(a_j, nearestEnemy(a_i)) \end{aligned}$$

32. Two agents attacking the same target:

$$\begin{aligned} PairAttack(a_i, a_j) :=& TargetInRange(a_i) \wedge TargetInRange(a_j) \\ & \wedge target(a_i) = target(a_j) \end{aligned}$$

33. Only one of two agents that share a target in range is attacking that target:

$$OneOfTwoAttack(a_i, a_j) := [TargetInRange(a_i) \vee TargetInRange(a_j)] \wedge \neg PairAttack(a_i, a_j)$$

34. A weaker marine is in front of the other and moving closer to a shared nearest enemy:

$$WeakerInFrontAndCloser(a_i, a_j) := SameNearestEnemy(a_i, a_j) \wedge Closer(weaker(a_i, a_j), nearestEnemy(a_i))$$

35. Not attacking an enemy together whose health is in the range $(h_1, h_2]$:

$$NotPairAttack_{h_1,h_2}(a_i, a_j) := \neg[PairAttack(a_i, a_j) \wedge HealthWithin_{h_1,h_2}(target(a_i))]$$

for the health ranges: $(-\infty, 0.25]$, $(0.25, 0.5]$, $(0.5, 0.75]$ and $(0.75, 1.0]$.

Apart from the predicates listed above, we also use the following non-predicate based features.

36. Base function used: $targetDamage(a_i)$.
37. $SimpleUnitDiff$ Difference in number of player's units and enemy units scaled by the total number of units in the game at the beginning.
38. $TotalHealthFriendly$ Percentage total health (hit) points of enemy.
39. $TotalHealthEnemy$ Percentage total health (hit) points of enemy.
40. $AveFriendlyHealth$ The average friendly marines' health points.
41. $AveEnemyHealth$ The average enemy marines' health points.

A.2.3 Coordination Constraints

Constraints are the negation of selected predicates. The following are used as static CCs: $TargetOutRange(a_i)$, $BoundaryCollide(a_i)$, $EnemyCollide(a_i)$, and $PairCollide(a_i, a_j)$. The single agent (unary) dynamic CCs used are based on Section A.2.2 Feature 30:

42. $c1(a_i) := HealthWithin_{-\infty,0.25}(a_i) \wedge NoAttack(a_i)$
43. $c2(a_i) := HealthWithin_{0.25,0.5}(a_i) \wedge NoAttack(a_i)$
44. $c3(a_i) := HealthWithin_{0.5,0.75}(a_i) \wedge NoAttack(a_i)$
45. $c4(a_i) := HealthWithin_{0.75,1.0}(a_i) \wedge NoAttack(a_i)$
46. $c5(a_i) := HealthWithin_{0.5,0.75}(a_i) \wedge Further(a_i, nearestEnemy(a_i))$
47. $c6(a_i) := HealthWithin_{0.75,1.0}(a_i) \wedge Further(a_i, nearestEnemy(a_i))$

Finally, the pairwise agent (binary) dynamic CCs used are:

48. $c7(a_i, a_j) := NotAligned(a_i, a_j)$
49. $c8(a_i, a_j) := WeakerInFrontAndCloser(a_i, a_j)$

A.2.4 Top Level Predicates

The dynamic CCs listed in Section A.2.3 are action variables for the top level. Each top level action variable may be activate (1) or deactivate (0). The basic state-only predicates of the top level function are: $SimpleUnitDiff$, $TotalHealthFriendly$, $AveFriendlyHealth$, and $AveEnemyHealth$.

We define a few more predicates based on only the state:

50. $HasEnemyInRange(a_i)$ Unit has some enemy in range of its weapon.
51. $NoEnemyInRange(a_i) := \neg HasEnemyInRange(a_i)$.
52. $DistanceWithin_{d_1,d_2}(u_i, u_j) := d_1 < distance(u_i, u_j) \leq d_2$. Distance of two units is within some range $(d_1, d_2]$.
53. Both units have health within a range $(h_1, h_2]$:

$$\begin{aligned} PairHealthWithin_{h_1,h_2}(u_i, u_j) :=& HealthWithin_{h_1,h_2}(u_i) \\ & \wedge HealthWithin_{h_1,h_2}(u_j). \end{aligned}$$

54. Either unit has its nearest enemy within some range,

$$\begin{aligned} NearestIn_{d_1,d_2}(a_i, a_j) :=& DistanceWithin_{d_1,d_2}(a_i, nearestEnemy(a_i)) \\ & \vee DistanceWithin_{d_1,d_2}(a_j, nearestEnemy(a_j)) \end{aligned}$$

We describe the other predicates used to approximate W by constructing sets to conjunct. Let the set of all unary CCs be C_i and binary CCs be $C_{i,j}$. Let $Activated(c)$ be true if the CC, c is activated. The set of all unary CC activated predicates is $\mathcal{C}_i = \{Activated(c) \mid c \in C_i\}$ and for binary CCs $\mathcal{C}_{i,j} = \{Activated(c) \mid c \in C_{i,j}\}$. The other top level predicates are:

55. The following set is set conjuncted ($\bigwedge$) with $\mathcal{C}_i$ and $\mathcal{C}_{i,j}$:

$$\left\{ \begin{matrix} TotalHealthFriendly \geq TotalHealthEnemy, \\ TotalHealthFriendly < TotalHealthEnemy \end{matrix} \right\}.$$

56. The set of $\{NoEnemyInRange(a_i), HasEnemyInRange(a_i)\}$ is conjuncted with the set of $Activated$ for unary CCs 42 to 45.
57. $HealthWithin_{-\infty,0.25}(a_i) \wedge Activated(c1)$
58. $HealthWithin_{0.25,0.5}(a_i) \wedge Activated(c2)$
59. $HealthWithin_{0.5,0.75}(a_i) \wedge Activated(c3)$
60. $HealthWithin_{0.75,1.0}(a_i) \wedge Activated(c4)$
61. The following set is set conjuncted ($\bigwedge$) with $\mathcal{C}_{i,j}$:

$$\left\{ \begin{matrix} DistanceWithin_{-\infty,20}(a_i, a_j), & DistanceWithin_{20,40}(a_i, a_j), \\ DistanceWithin_{40,60}(a_i, a_j), & DistanceWithin_{60,\infty}(a_i, a_j), \\ NearestIn_{-\infty,20}(a_i, a_j), & NearestIn_{20,40}(a_i, a_j), \\ NearestIn_{40,60}(a_i, a_j), & NearestIn_{60,\infty}(a_i, a_j), \\ PairHealthWithin_{-\infty,0.25}, & PairHealthWithin_{0.25,0.5}, \\ PairHealthWithin_{0.5,0.75}, & PairHealthWithin_{0.75,1.0} \end{matrix} \right\}.$$

A.3 Automated Retinal Image Analysis

This section details the predicates used for relational features in Chapter 7 page 171. As RL is centralized, to reduce clutter we omit the global state variable when action variables are present in the function or predicates.

A.3.1 Base Predicates & Functions

Let $Ves(s)$ be a set of all vessels in state s, $Art(s) \subseteq Ves(s)$ be the set of arteries and $Ven(s) \subseteq Ves(s)$ be the set of all veins. We use the variable $Vset$ to denote one of the functions Ves, Art, Ven that returns a set of vessels. Further let p be the variable denoting some point in the retinal centre line image. In every state, the agent's state s_i (assumed to be present when a_i is present) is also treated as a point since agents can only occupy one point as its current position. The following are basic functions:

1. $num_{Vset}(s) = Vset(s)$ Returns the number of vessels in $Vset$ in the current state.
2. $fractalDim_{Vset}(s)$ Returns the box counting fractal dimension of the given set of vessels.
3. $density_{Vset}(s)$ Returns the number of pixels in the centre line of the vessels given by $Vset$ divided by square root of the area in the ring that forms the zone of interest.
4. $percent_{Vset}(s)$ Returns the sum of centre line points in the vessels given by $Vset$ divided by all centre line points (that includes those of orphaned line segments).
5. $pointAfter(a_i)$ Return a point in the image that an agent i should be after it has taken its action.
6. $vessel(p)$ Returns a vessel object at a point is on.
7. $segment(p)$ Returns a line segment object a point is on.
8. $component(p)$ Returns a connected centre line component object from the line image that a point is on.
9. $type(p)$ Returns the type object of the vessel that a point is on.
10. $intensity(p)$ The value at p from the illumination corrected map (see Figure 7.5d page 180).
11. $gradient(p)$ The value at p from the gradient map (see Figure 7.5b page 180).
12. $rootIntensity(p)$ The mean value of the pixels at the root segment of the vessel at point p from the illumination corrected map.
13. $diffIntensity(a_i) = intensity(a_i) - intensity(pointAfter(a_i))$.
14. $diffGradient(a_i) = gradient(a_i) - gradient(pointAfter(a_i))$.
15. $width(p)$ The value of the vessel width at point p.
16. $diffWidth(a_i) = width(a_i) - width(pointAfter(a_i))$.
17. $widthShift(p)$ The absolute difference between the distance of the centre line point p to each end of the vessel width.
18. $widthShiftAfter(a_i) = widthShift(pointAfter(a_i))$.
19. $diffWidthShift(a_i) = |widthShift(a_i) - widthShiftAfter(a_i)|$
20. $localDensity(p)$ The number of pixels in the centre line of the vessels in a square of side 21 pixels centred on point p divided by 10.
21. $widthAngle(p)$ The angle orientation of the vessel width at point p.
22. $diffWidthAngle(a_i) = |widthAngle(a_i) - widthAngle(pointAfter(a_i))|$.
23. $lineAngle(p)$ The angle between the lines ending at point p in the line image.

24. $boxDist(p_i, p_j) = \min\{|x(p_i) - x(p_j)|, |y(p_i) - y(p_j)|\}$ The length of the side of the smallest square that includes the two points.
25. $l2norm(p_i, p_j)$ The Euclidean distance between the points p_i and p_j.

The following are action predicates:

26. $Idle(a_i)$ Agent i is taking an idle action that does nothing.
27. $Move(a_i)$ Agent i is taking a move action.
28. $Edit(a_i)$ Agent i is taking an edit action, i.e., any action that is not $Move(a_i)$ and not $Idle(a_i)$.
29. $Line(a_i)$ Agent i is taking any one of the 16 add segment line actions.
30. $Break(a_i)$ Agent i is taking a break action.
31. $Detach(a_i)$ Agent i is taking a detach action.
32. $AddRoot(a_i)$ Agent i is taking add root action.
33. $Mark(a_i)$ Mark crossover action.
34. $Unmark(a_i)$ Unmark crossover action.
35. $Toggle(a_i)$ Toggle vessel type action.
36. $BreakOrDetach(a_i) := Break(a_i) \vee Detach(a_i)$.

The following are basic or parametrized state predicates used to build more complex predicates. Note that where p is used, the variable is some point that can be substituted by an agent's state indicating the point the agent is on.

37. $Within_{f,h_1,h_2}(s) := f(s) \geq h_1 \wedge f(s) < h_2$ Returns true if the return value of function f is in the range $[h_1, h_2)$.
38. $AgentWithin_{f,h_1,h_2}(a_i) := f(a_i) \geq h_1 \wedge f(a_i) < h_2$ Returns true if the return value of function f with respect to agent i is in the range $[h_1, h_2)$.
39. The 67 $AgentWithin$ predicates with given parameters in Table A.1.

Next are basic predicates with respect to a point or agent location. Let $P_{unit}(p)$ denote the set of the following predicates:

40. $OnSegment(p)$ Agent is on some line segment.
41. $OnRoot(p)$ Agent is on a root segment.
42. $OnVein(p)$ Agent is on a vein.
43. $OnArtery(p)$ Agent is on an artery.
44. $OnShared(p)$ Agent is on a crossover segment, e.g., line segment 3 in Figure 7.4 page 176.
45. $OnOrphan(p)$ Agent is on an orphan (non-vessel) line segment.
46. $OnJunction(p)$ Agent is on a junction pixel that is a confluence of more than 2 line segments.
47. $OnAdjJunction(p)$ Agent is next to a junction pixel.
48. $OnLineEnd(p)$ Agent is on the end point of a line segment.
49. $OnFirstOrder(p)$ Agent is on the first order branch of a vessel.
50. $OnSecondOrderMore(p)$ Agent is on a second or higher order branch of a vessel.
51. $OnNearRoot(p)$ Agent is on a root pixel or on a vessel pixel next to a root pixel.
52. $OnSameTypeAsOther(p)$ Agent is on a vessel that crosses another vessel that is of the same type.
53. $OnAltTypeAsAdj(p)$ Agent is on a vessel with a different type from its left and right vessel in clockwise root point order.

f	$[h_1,$	$h_2)$	# of predicates
$intensity$	0	21	4
	21	41	
	41	61	
	61	255	
$gradient$	0	21	4
	21	41	
	41	61	
	61	255	
$rootIntensity$	0, 16, 32, ..., 240	16, 32, 48, ..., 256	16
$diffIntensity$, $diffGradient$	-255	-25	14
	-25	-15	
	-15	-5	
	-5	6	
	6	16	
	16	26	
	26	256	
$width$	0	1	6
	1	5	
	5	10	
	10	15	
	15	20	
	20	∞	
$diffWidth$	0	5	3
	5	10	
	10	∞	
$widthShift$, $widthShiftAfter$, $diffWidthShift$	0	1	15
	1	2	
	2	4	
	4	6	
	6	∞	
$diffAngle$	0	$\frac{\pi}{8}$	5
	$\frac{\pi}{8}$	$\frac{\pi}{4}$	
	$\frac{\pi}{4}$	$\frac{3\pi}{8}$	
	$\frac{3\pi}{8}$	$\frac{\pi}{2}$	
	$\frac{\pi}{2}$	π	
$localDensity$	0.0	0.2	3
	0.2	0.4	
	0.4	1.0	

Table A.1: Parameters used for predicate $AgentWithin_{f,h_1,h_2}$

54. $OnSameTypeAsAdj(p)$ Agent is on a vessel with the same type as its left and right vessel in clockwise root point order.
55. Agent is on vessel that is same type as one and only one of its left or right vessel in clockwise root point order.

$$\begin{aligned} OnSameTypeAsOneAdj(p) :=& \neg OnAltTypeAsAdj(p) \\ & \wedge \neg OnSameTypeAsAdj(p) \end{aligned}$$

56. $OnLoop(p)$ Agent is on a centre line that forms a loop.

The following are basic predicates involving two agents:

57. Two agents are within range to link up segment lines, $CanLinkUp(a_i, a_j) := boxDist(a_i, a_2) \leq 5$.

58. $OnHigherOrder(a_i, a_j)$ Agent i is on a higher order branch than agent j when both are on the same vessel.
59. On different vessels,

$$OnDiffVessel(a_i, a_j) := OnVessel(a_i) \wedge OnVessel(a_j) \\ \wedge vessel(a_i) \neq vessel(a_j)$$

60. On different segments,

$$OnDiffSegment(a_i, a_j) := OnSegment(a_i) \wedge OnSegment(a_j) \\ \wedge segment(a_i) \neq segment(a_j)$$

61. On different vessel types,

$$OnDiffType(a_i, a_j) := OnDiffVessel(a_i, a_j) \wedge type(a_i) \neq type(a_j)$$

62. $OnSameComponent(a_i, a_j) := component(a_i) = component(a_j)$.
63. $OnSameSegment(a_i, a_j) := OnSegment(a_i) \wedge segment(a_i) = segment(a_j)$.
64. $OnSameVessel(a_i, a_j) := OnVessel(a_i) \wedge vessel(a_i) = vessel(a_j)$.
65. $AreAdjacent(a_i, a_j) := boxDist(a_i, a_j) \leq 1$.
66. $OnAdjacentSegment(a_i, a_j)$ True if both agents are on segments and there exists one point in each segment that have a $boxDist$ of at least 1.
67. $OnAdjacentVessel(a_i, a_j)$ True if both agents are on vessels and there exists on point in each vessel that have a $boxDist$ of at least 1.
68. Moving to the same vessel,

$$MoveToSameVessel(a_i, a_j) := OnVessel(pointAfter(a_i)) \\ \wedge vessel(pointAfter(a_i)) = vessel(pointAfter(a_j))$$

69. Moving to the same segment,

$$MoveToSameSegment(a_i, a_j) := OnSegment(pointAfter(a_i)) \\ \wedge segment(pointAfter(a_i)) = segment(pointAfter(a_j))$$

70. Moving to same component,

$$MoveToSameComponent(a_i, a_j) := \\ \wedge component(pointAfter(a_i)) = component(pointAfter(a_j))$$

71. Moving to the different vessel,

$$MoveToDiffVessel(a_i, a_j) := \\ OnVessel(pointAfter(a_i)) \wedge OnVessel(pointAfter(a_j)) \\ \wedge vessel(pointAfter(a_i)) \neq vessel(pointAfter(a_j))$$

72. Moving to different segment,

$$MoveToDiffSegment(a_i, a_j) := \\ OnSegment(pointAfter(a_i)) \wedge OnSegment(pointAfter(a_j)) \\ \wedge segment(pointAfter(a_i)) \neq segment(pointAfter(a_j))$$

73. Moving to different component,

$$MoveToDiffComponent(a_i, a_j) := \\ component(pointAfter(a_i)) \neq component(pointAfter(a_j))$$

74. $MoveToHigherOrder(a_i, a_j)$ Agent i will be on a higher order branch of the same vessel as agent j after their actions are taken.

A.3.2 Bottom Level Predicates

f	$[h_1,$	$h_2)$	# of predicates
num_{Vset}	0	3	9
	3	6	
	6	∞	
$fractalDim_{Vset}$	0.00	1.00	18
	1.00	1.25	
	1.25	1.50	
	1.50	1.75	
	1.75	2.00	
	2.00	∞	
$density_{Vset}$	0.000	0.025	15
	0.025	0.050	
	0.050	0.075	
	0.075	0.100	
	0.100	1.000	
$percent_{Vset}$	0.0	0.2	15
	0.2	0.4	
	0.4	0.6	
	0.6	0.8	
	0.8	1.0	

Table A.2: Parameters used for predicate $Within_{f,h_1,h_2}(s)$. The number of predicates takes into account predicates for each $Vset$ namely Ves, Art, Ven.

There are a total of 573 predicates used for bottom level features of which 58 are global, 375 are unary that depend on single agents, and 140 are binary that depend on two agents. The following are *nullary* predicates that are not based on any agent's actions but the current global state.

75. $IsBig6(s)$ Returns true if there are at least six arteries and at least six veins.
76. 57 $Within$ predicates are created, one for each of the entries in Table A.2 for each of the possible values of $Vset$.

The following predicates are based on a single (unary) agent's action and the state.

77. $Idle(a_i) \wedge \neg IsBig6(s)$

78. $BreakWillOrphan(a_i)$ An agent carrying out a Break or Detach orphan while on a vessel will result in an orphaned segment.
79. $Break(a_i) \wedge lineAngle(a_i) < 85°$. Breaking at a line point with a small angle.
80. $SmallAngleLink(a_i)$ An agent performing a Line action from one line's end point to another line's end point will result in a small angle of $< 85°$ between the lines.
81. $SmallAngleExtension(a_i)$ An agent performing a Line action at a line's end point results in a line angle of $< 85°$, i.e., a 'V' shaped line.
82. $FormNearbyBranch(a_i)$ An Agent performing a Line action that will create a new junction that it is near an existing junction.
83. The set of predicates $\{Move(a_i), Line(a_i)\}$ is conjuncted ($\bigwedge$) with each of the following set of predicates,

$$\left\{\begin{array}{ccc} component(a_i) & = & component(pointAfter(a_i)), \\ component(a_i) & \neq & component(pointAfter(a_i)), \\ vessel(a_i) & = & vessel(pointAfter(a_i)), \\ vessel(a_i) & \neq & vessel(pointAfter(a_i)), \\ OnVessel(a_i) & \wedge & OnCross(pointAfter(a_i)), \\ \neg OnVessel(a_i) & \wedge & OnCross(pointAfter(a_i)), \\ OnJunction(pointAfter(a_i)), & & OnAdjJunction(pointAfter(a_i)), \\ \neg OnJunction(pointAfter(a_i)), & & OnVessel(pointAfter(a_i)), \\ OnLineEndPoint(pointAfter(a_i)), & & \end{array}\right\}$$

84. The predicate $Move(a_i) \vee Idle(a_i)$ is conjuncted with each of the predicates in Table A.2.
85. The predicate in $\{AddRoot(a_i), Toggle(a_i)\}$ are conjuncted ($\bigwedge$) with the set of $Within$ predicates from the entry num_{Vset} in Table A.2.
86. The predicates in $\{Line(a_i), Move(a_i)\}$ are conjuncted ($\bigwedge$) with the set of $AgentWithin$ predicates from the entries $width$, $widthShift$, $widthShiftAfter$, and $diffWidthShift$ in Table A.1.
87. The predicates in $\{Unmark(a_i), Mark(a_i), AddRoot(a_i), Break(a_i), Line(a_i), Move(a_i)\}$ are conjuncted ($\bigwedge$) with the set of $AgentWithin$ predicates from the entries $intensity$ and $gradient$ in Table A.1.
88. The predicates created from,

$$\{Toggle(a_i)\} \bigwedge \{OnVein(a_i), OnArtery(a_i)\}$$
$$\bigwedge \left\{\begin{array}{cc} OnSameTypeAsOther(a_i), & OnAltTypeAsAdj(a_i) \\ OnSameTypeAsAdj(a_i), & OnSameTypeAsOneAdj \end{array}\right\}$$

89. The predicates in $\{Move(a_i), Line(a_i)\}$ are conjuncted ($\bigwedge$) with the set of $AgentWithin$ predicates from the entries $diffIntensity$, $diffGradient$.
90. $Line(a_i) \wedge OnLineEnd(a_i) \wedge OnLineEnd(pointAfter(a_i))$
91. The predicate $Line(a_i)$ is conjuncted with each of the $AgentWithin$ predicates from the entry $diffAngle$ in Table A.1.
92. The predicates created from,

$$\{AddRoot(a_i), Mark(a_i), Unmark(a_i)\} \bigwedge$$
$$(\{OnFirstOrder(a_i), OnSecondOrderMore(a_i)\} \cup P_{width} \cup P_{widthShift})$$

where P_{width} and $P_{widthShift}$ are the sets of $AgentWithin$ predicates from the $width$ and $widthShift$ entries in Table A.1 respectively.

93. The predicates created from,

$$\{Break(a_i), Detach(a_i), Line(a_i)\} \bigwedge (P_{localDensity} \cup [P_{unit}(a_i) - \{OnVessel(a_i), OnSegment(a_i)\}])$$

where $P_{localDensity}$ is the set of $AgentWithin$ predicates from the $localDensity$ entry in Table A.1, and $P_{unit}(a_i)$ is the set of predicates 40 to 56 with respect to a_i.

Let P_{on2} be the set of predicates from 62 to 67, P_{move2} be the set of predicates from 68 to 74, P_{diff2} be the set of predicates from 59 to 61, and

$$P_{other2} = \left\{ \begin{array}{c} OnDiffVessel(a_i, a_j) \wedge OnSameComponent(a_i, a_j), \\ OnDiffSegment(a_i, a_j) \wedge OnSameComponent(a_i, a_j), \\ OnDiffType(a_i, a_j) \wedge OnSameComponent(a_i, a_j), \\ OnDiffSegment(a_i, a_j) \wedge OnSameVessel(a_i, a_j) \end{array} \right\}$$

The following are predicates for features involving two agents, i.e., they contain knowledge of coordination.

94. Predicates created by $\{Line(a_i) \wedge Line(a_j)\} \bigwedge [P_{on2} \cup P_{move2} \cup P_{other2}]$.
95. Predicates created by $\{Move(a_i) \wedge Move(a_j)\} \bigwedge [P_{on2} \cup P_{move2} \cup P_{other2}]$.
96. Predicates created by $\{Move(a_i) \wedge Move(a_j)\} \bigwedge [P_{on2} \cup P_{diff2}] \bigwedge P_{move2}$.
97. Predicates created by,

$$\{Break(a_i) \wedge Break(a_j), Detach(a_i) \wedge Detach(a_j)\} \bigwedge [P_{on2} \cup P_{other2}]$$

98. Agents are linking up to form a line that does not join at a unrealistic angle,

$$\begin{aligned} LinkedLine(a_i, a_j) := & \; Line(a_i) \wedge Line(a_j) \\ & \wedge lineAngle(pointAfter(a_i)) < 90^{\circ} \\ & \wedge AreAdjacent(pointAfter(a_i), pointAfter(a_j)) \end{aligned}$$

99. $FormJunction(a_i, a_j)$ Line actions taken by two agents are forming a junction.
100. $NotFormJunction(a_i, a_j)$ Line actions taken by two agents are not forming a junction.
101. Two agents that can link lines up but are not doing so,

$$\begin{aligned} NotLinking(a_i, a_j) := & \; Line(a_i) \wedge Line(a_j) \\ & \wedge CanLinkUp(a_i, a_j) \wedge \neg LinkedLine(a_i, a_j) \end{aligned}$$

102. Agents are moving nearer,

$$\begin{aligned} & MoveNearer(a_i, a_j) := \\ & \quad \wedge l2norm(pointAfter(a_i), pointAfter(a_j)) < l2norm(a_i, a_j) \end{aligned}$$

103. Agents are moving further,

$$MoveFurther(a_i, a_j) := \\ \wedge\, l2norm(pointAfter(a_i), pointAfter(a_j)) > l2norm(a_i, a_j)$$

104. Move out of coordination range where C_{max} is the coordination range constant,

$$MoveOutOfCoord(a_i, a_j) := l2norm(a_i, a_j) \leq C_{max} \\ \wedge\, l2norm(pointAfter(a_i), pointAfter(a_j)) > C_{max}$$

105. $FormNearbyBranches(a_i, a_j)$ Two agents are forming nearby branches that are close together.
106. If either of two agents is breaking or detaching a higher order branch of a vessel while the other is performing an edit action on a lower order branch of the vessel.

$$BreakDetachHigherOrder(a_i, a_j) := \\ [BreakOrDetach(a_i) \wedge Edit(a_j) \wedge OnHigherOrder(a_i, a_j)] \\ \vee\, [BreakOrDetach(a_j) \wedge Edit(a_i) \wedge OnHigherOrder(a_j, a_i)]$$

107. $BreakDetachWillOrphan(a_i, a_j)$ True if two break or detach actions by two agents are required to create a new orphaned segment, but a single action by either agent will not.
108. $BreakDetachCross(a_i, a_j)$ True if $BreakDetachWillOrphan(a_i, a_j)$ is true and the orphan segment was formally a crossover shared segment, i.e., line segment 3 in Figure 7.4 page 176.

A.3.3 Coordination Constraints

For static constraints, we incorporate constraints to disallow illegal actions. For example, breaking a line segment when there is none where the agent is, toggling the vessel type for an orphaned segment, and adding a new root point in the middle of a line segment instead of at the end. We also disallow agents from colliding. In addition, the following predicates are used as static constraints for both coordinated RL and CGRL.

109. Predicate 81 to disallow creating 'V' kinks in line segments.
110. $Line(a_i) \wedge AgentWithin_{width,20,\infty}(a_i)$ created at 86.
111. $Line(a_i) \wedge AgentWithin_{diffAngle,\frac{\pi}{2},\pi}(a_i)$ created at 86.
112. $Line(a_i) \wedge AgentWithin_{diffWidth,10,\infty}(a_i)$ created at 86.
113. $Line(a_i) \wedge AgentWithin_{widthShift,6,\infty}(a_i)$ created at 86.
114. $Line(a_i) \wedge AgentWithin_{widthShiftAfter,6,\infty}(a_i)$ created at 86.
115. $Line(a_i) \wedge AgentWithin_{localDensity,0.4,1}(a_i)$ created at 93.
116. $Break(a_i) \wedge OnNearRoot(a_i)$ created at 93.
117. $AddRoot(a_i) \wedge AgentWithin_{widthShift,6,\infty}(a_i)$ created at 92.
118. $Detach(a_i) \wedge Detach(a_j) \wedge OnSameSegment(a_i, a_j)$ created at 97.

The following predicates are used as dynamic constraints for CGRL.

119. $Move(a_i) \wedge component(a_i) = component(pointAfter(a_i))$ created at 83.
120. $Move(a_i) \wedge vessel(a_i) = vessel(pointAfter(a_i))$ created at 83.
121. $Line(a_i) \wedge component(a_i) = component(pointAfter(a_i))$ created at 83.

122. $Line(a_i) \wedge vessel(a_i) = vessel(pointAfter(a_i))$ created at 83.
123. $Line(a_i) \wedge AgentWithin_{diffWidth,5,10}(a_i)$ created at 86.
124. $Line(a_i) \wedge AgentWithin_{widthShift,4,6}(a_i)$ created at 86.
125. $Line(a_i) \wedge AgentWithin_{widthShiftAfter,4,6}(a_i)$ created at 86.
126. $Line(a_i) \wedge AgentWithin_{diffWidthShift,4,6}(a_i)$ created at 86.
127. $Line(a_i) \wedge AgentWithin_{localDensity,0.2,0.4}(a_i)$ created at 93.
128. $Break(a_i) \wedge AgentWithin_{localDensity,0,0.2}(a_i)$ created at 93.
129. $Detach(a_i) \wedge OnFirstOrder(a_i)$ created at 93.
130. $Detach(a_i) \wedge OnRoot(a_i)$ created at 93.
131. $Toggle(a_i) \wedge OnVein(a_i) \wedge OnSameTypeAsOther(a_i)$ created at 88.
132. $Toggle(a_i) \wedge OnVein(a_i) \wedge OnAltTypeAsOther(a_i)$ created at 88.
133. $Toggle(a_i) \wedge OnArtery(a_i) \wedge OnSameTypeAsOther(a_i)$ created at 88.
134. $Toggle(a_i) \wedge OnArtery(a_i) \wedge OnAltTypeAsOther(a_i)$ created at 88.
135. Predicates 80 and 82.
136. Binary predicates 101 to 106.
137. Binary predicate $Move(a_i) \wedge Move(a_j) \wedge OnSameComponent(a_i, a_j)$ $\wedge MoveToDiffComponent(a_i, a_j)$ created at 96.
138. Binary predicate $Move(a_i) \wedge Move(a_j) \wedge OnSameVessel(a_i, a_j)$ $\wedge MoveToDiffVessel(a_i, a_j)$ created at 96.
139. Binary predicate $Move(a_i) \wedge Move(a_j) \wedge OnSameSegment(a_i, a_j)$ $\wedge MoveToDiffSegment(a_i, a_j)$ created at 96.
140. Binary predicate $Detach(a_i) \wedge Detach(a_j) \wedge OnDiffSegment(a_i, a_j)$ $\wedge OnSameVessel(a_i, a_j)$ created at 97.

A.3.4 Top Level Predicates

The top level predicates are created from bottom level and basic predicates by conjuncting selected predicates with the $Activated(c)$ predicate that indicates if a CC is activated or not. In addition, nullary predicates from 75 and 76 are used as features for the top level value function.

Let C_1 be the set of unary CCs from predicates 119 to 135, and C_2 be the set of binary CCs from predicates 136 to 140. We define three sets of top level predicates, nullary (P_0), unary (P_1), and binary (P_2). Nullary and unary predicates are conjuncted with $Activated$ for CCs in C_1. While all three sets are conjuncted with CCs in C_2 where in the case of unary predicates in P_1, they are used for both agents. For example, for some binary CC, c_{ij} and unary predicate $AgentWithin_{intensity,0,20}$, a top level predicate will be,

$$AgentWithin_{intensity,0,20}(a_i) \wedge AgentWithin_{intensity,0,20}(a_j) \wedge Activated(c_{ij})$$

141. P_0: Predicates from 75 and 76.
142. P_1: Predicates from 40 to 56. $AgentWithin$ predicates from the entries of $intensity$, $gradient$, $width$, $localDensity$, and $widthShift$ in Table A.1.
143. P_2: Predicates from 62 to 67. Predicates from the set P_{other2} defined before binary predicates in Section A.3.2.