CONCURRENT DECISION MAKING IN MARKOV DECISION PROCESSES

A Dissertation Presented

by

KHASHAYAR ROHANIMANESH

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2006

Department of Computer Science

CONCURRENT DECISION MAKING IN MARKOV DECISION PROCESSES

A Dissertation Presented

by

KHASHAYAR ROHANIMANESH

Approved as to style and content by:

Sridhar Mahadevan, Chair

Andrew G. Barto, Member

Roderic A. Grupen, Member

Weibo Gong, Member

W. Bruce Croft, Department Chair
Department of Computer Science

To my parents

ACKNOWLEDGMENTS

I believe that words are such a weak media to express our deep emotions and thoughts. Honestly, I am not able to find the words to express how I am thankful to my parents, and I find it really impossible to describe how much I feel blessed to have such a lovely and beautiful people in my life. Throughout my academic years, although being far from them, there was not a moment that I did not feel their presence in my life. Their constant support, encouragement, and endless love has been the main reason that enabled me to thrive even under immense pressure that I had to face in a period of my life. I bow down to them and thank them for all the sacrifices they made. I am also thankful to my sister Samaneh Rohanimanesh for her indefinite support and love throughout my academic years.

I am immensely grateful to my advisor Sridhar Mahadevan. Sridhar's constant support and guidance has played an important role in my academic education to evolve as a better researcher. I must thank him for his patience for giving me freedom to explore new ideas on my own which helped me to become a self-reliant researcher. Sridhar is truly a wonderful role model who has taught me how to identify fundamental problems and how to study them from many different points of view. Aside from Sridhar's academic influence, I have found him a wonderful human being. I am in huge debt to Sridhar for his constant support, for the time I was away from school due to the long visa delay. Without his generous support I would not have been able to complete this work at this time. Thank you Sridhar.

Throughout my academic years at UMASS I was also privileged to have the opportunity to collaborate with Andrew Barto. Taking Andrew's Reinforcement Learning course has been a memorable highlight in my academic education. I had many insightful conversations with Andrew that significantly influenced several fundamental ideas that I investigated in my dissertation. Andrew is truly an outstanding and visionary researcher and his style of research has significantly influenced me to become a better researcher. Andrew is also a wonderful human being, and I am also in huge debt to Andrew for his constant support and care, for the time I was away from school due to the long visa delay. It was a great pleasure for me to have him as a member of my thesis committee.

I am also grateful to my other members of thesis committee for their support, inspiration and assistance. I was privileged to have the opportunity to collaborate with Roderic Grupen. Roderic is an outstanding researcher who showed me to several insightful directions in robotics that inspired some of the fundamental ideas in this dissertation. Roderic's style of research taughtt me how to look at a problem from a broader perspective that would bridge between several inter-disciplinary areas such as robotics and machine learning. I also would like to thank him for his constant support and his faith in this work that inspired me to continue my line of research in this direction. I am also thankful to Weibo Gong for inspiring conversations.

I am immensely indebted to Andrew McCallum. It has been a thrilling experience working with Andrew in the past three years. I learned so many things from Andrew and he also has been a role model in my research. Andrew helped me to become an inter-disciplinary researcher and has been a major inspiration for developing a strong background in the area of probabilistic reasoning. I am so grateful to Andrew for his unbelievable support and patience during the time I was away from school due to the long visa delay, and also during the period I was completing this dissertation. Thank you Andrew.

I am also deeply grateful to Sam Roweis for his steady support during my stay in Canada due to the long visa delay. Sam generously accepted me in his research group at the University of Toronto, and made every effort to make me feel part of the machine learning group. It was also a privilege to have the opportunity to partly collaborate with Sam during my stay in Canada. I also would like to thank all the members of the machine learning group at the University of Toronto for their support and insightful discussions. In particular, I would like to thank Nathan Srebro for his useful comments and discussion.

I am grateful for the support and encouragement of many people who contributed to the development of the ideas in this thesis. I especially thank my old friend Mohammad Ghavamzadeh, Georgios Theocharous, Robert Platt, Mike Rosenstein, and Ashvin Shah for many insightful conversations. I thank Rob Platt for helpful discussions that inspired some of the ideas investigated in this dissertation. I am also very thankful to Ashvin Shah for many very helpful conversations, and also his constant support as a wonderful friend.

I am fortunate to have been part of the Autonomous Learning Laboratory , the Laboratory for Perceptual Robotics, and also Information Extraction and Synthesis Laboratory with many talented members as resources for ideas and debate. I am especially grateful for many helpful discussions with Charles Sutton, Oliver Brock, Mohammad Ghavamzadeh, Mike Rosenstein, Anders Jonsson, Ozgur Simsek, Alicia P. Wolfe, Ted Perkins, Rob Platt, Balaraman Ravindran, Victoria Manfredi, Sarah Osentoski, Steve Hart, Aron Culotta, Marc Pickett, Andy Fagg, Amy McGovern, and Brendan Burns. I would like to thank other members of of the Autonomous Learning Laboratory for their constant support and friendship. Thank you Collin Barringer, Jad Davis, Jeffrey Johns, George Konidaris, Kimberly Ferguson, Suchi Saria, Andrew Stout, and Chris Vigorito.

I am also grateful to the members of Autonomous Agent Laboratory at Michigan State University for many insightful discussions: Natalia Hernandez Gardiol, Rajbala Makar, Silviu Minut, Mohammad Ghavamzadeh, Georgios Theocharous, and Shailesh Shailesh.

I am grateful for the support and encouragement of many people during my undergraduate studies at University of Tehran, and graduate studies at Michigan State University. In particular I am thankful to Abdol-Hossein Esfahanian whose faith in my capabilities made it possible for me to seek higher education in United States of America. I am also thankful to Anil Jain who also has been as a role model for my research. I am also thankful to Ali. M. Eydgahi for his support and inspiration during my undergraduate studies.

During my academic years I was lucky to be surrounded by several beautiful souls whose constant support and encouragement enabled me to complete this work. I must thank my wonderful friend Laurel Crosby for her indefinite support and encouragement, and above all for her precious friendship. Laurel has been an inspiration in my life as a wonderful human being from whom I learned to be a stronger person. I would like to thank George Bissias – a wonderful friend and soul-mate – who constantly supported me in my life's ups and downs. I am also very grateful to one of my best friends Kristine Comartin for her wonderful friendship. Her unbelievable support immensely helped me to concentrate and complete this work. I also would like to thank my old friend Masoud Sadjadi for his constant support. I am also grateful to Hamed Fallahnia. Hamed hosted me during my stay in Canada and I am not able to describe how patiently he observed me, supported me, and encouraged me. Thank you Hamed for your patience and all the wonderful moments that I shared with you.

I am also deeply thankful to the staff community of the department of computer science at UMASS. In particular I am thankful to Sharon Mallory, Pauline Hollister, Laurie Downey, and Kate Moruzzi, for their constant support and patience during the time I was away from school due to a long visa delay.

This research was supported in part by the National Science Foundation under grant #ECS-0218125, and under the DARPA grant #DAANO2-98-C-4025. This work was also supported in part by the Center for Intelligent Information Retrieval and in part by The Central Intelligence Agency, the National Security Agency and National Science Foundation under NSF grant #IIS-0326249. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsor.

ABSTRACT

CONCURRENT DECISION MAKING IN MARKOV DECISION PROCESSES

FEBRUARY 2006

KHASHAYAR ROHANIMANESH

B.Sc., UNIVERSITY OF TEHRAN

M.Sc., MICHIGAN STATE UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Sridhar Mahadevan

This dissertation investigates concurrent decision making and coordination in systems that can simultaneously execute multiple actions to perform tasks more efficiently. Concurrent decision-making is a fundamental problem in many areas of robotics, control, and computer science. In the field of Artificial Intelligence in particular, this problem is recognized as a formidable challenge. By concurrent decision making we refer to a class of problems that require agents to accomplish long-term goals by concurrently executing multiple activities. In general, the problem is difficult to solve as it requires learning and planning with a combinatorial set of interacting concurrent activities with uncertain outcomes that compete for limited resources in the system.

The dissertation presents a general framework for modeling the concurrent decision making problem based on semi-Markov decision processes (SMDPs). Our approach is based on a centralized control formalism, where we assume a central control mechanism initiates, executes and monitors concurrent activities. This view also captures the type of concurrency that exists in single agent domains, where a single agent is capable of performing multiple activities simultaneously by exploiting the degrees of freedom (DOF) in the system. We present a set of coordination mechanisms employed by our model for monitoring the execution and termination of concurrent activities. Such coordination mechanisms incorporate various natural activity completion mechanisms based on the individual termination of each activity. We provide theoretical results that assert the correctness of the model semantics which allows us to apply standard SMDP learning and planning techniques for solving the concurrent decision making problem.

SMDP solution methods do not scale to concurrent decision making systems with large degrees of freedom. This problem is a classic example of the curse of dimensionality in the action space, where the size of the set of concurrent activities exponentially grows as the system admits more degrees of freedom. To alleviate this problem, we develop a novel decision theoretic framework motivated by the coarticulation phenomenon investigated in speech and motor control research. The key idea in this approach is based on the fact that in many concurrent decision making problems, the overall objective of the problem can be viewed as concurrent optimization of a set of interacting and possibly simpler subgoals of the problem for which the agent has gained the necessary skills to achieve them. We show that by applying coarticulation to systems with excess degrees of freedom, concurrency is naturally generated. We present a set of theoretical results that characterizes the efficiency of the concurrent decision making based on the coarticulation framework when compared to the case in which the agent is allowed to only execute activities sequentially (i.e., no coarticulation).

We also present a set of techniques for scaling the coarticulation framework to large domains. We develop tractable approximate algorithms suitable for such domains capable of executing many activities in parallel. We empirically evaluate our algorithms in a set of simulated domains ranging from an agent navigating in a grid world performing concurrent activities, to a simulated robot with multiple degrees of freedom that is capable of performing tasks concurrently.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Evolution in humans and other animals has led to a complex body with more degrees of freedom than are needed to perform any particular task. Such redundancy affords flexible and adaptable motor behavior (Bernstein, 1967) when all degrees of freedom can coordinate to contribute to task performance by enabling agents to carry out multiple concurrent activities, or by allowing them to commit to multiple tasks simultaneously.

Humans in particular are very efficient at performing tasks concurrently. Concurrent coordination and control takes place at different levels of the human control system. For example, when grasping an object, low level movements of our arms involves simultaneous contraction of multiple muscles that control the arm joints. At higher levels of control, we can initiate more complex activities along with the other ones that are already in progress. In some problems concurrency can be viewed as a means of enhancing planned activities by allowing them to overlap for further optimization of costs, improving over the optimal sequential solutions. In such problems we overlap activities in order to achieve goals faster where the optimization objective is the time required to achieve goals.

All such coordination must be done in the face of uncertainty; uncertainty about the state of the environment as well as uncertainty about the effects of the activities, and also uncertainty about the outcomes of multiple co-occurring activities. As an example, consider an episode of our everyday life which may involve getting up in the morning, making breakfast, driving to work, shopping and so on. When making breakfast, we may start the coffee maker, and as the coffee is preparing, begin boiling eggs. While these two processes are in progress, we might empty the dish-washer and turn on the radio for the morning news. Some time later, perhaps we notice that the eggs are almost ready, and thus we might begin toasting bread. While waiting for all of these processes to complete, we may start arranging the table in order to serve the breakfast. After breakfast, as we get ready to drive to work, the phone might ring and a friend of ours may ask for a ride to work. As a result, we may modify our plan in order to commit to this new objective by selecting a path that intersects to our friend's house en route to work. From the above example, we can observe several distinctive properties of this class of problems. First, concurrent activities compete for the limited amount of resources in the system; second, activities that run in parallel often do not terminate at the same time; and third, it can be conjectured that concurrency emerges due to the multi-objective optimization nature of the problem.

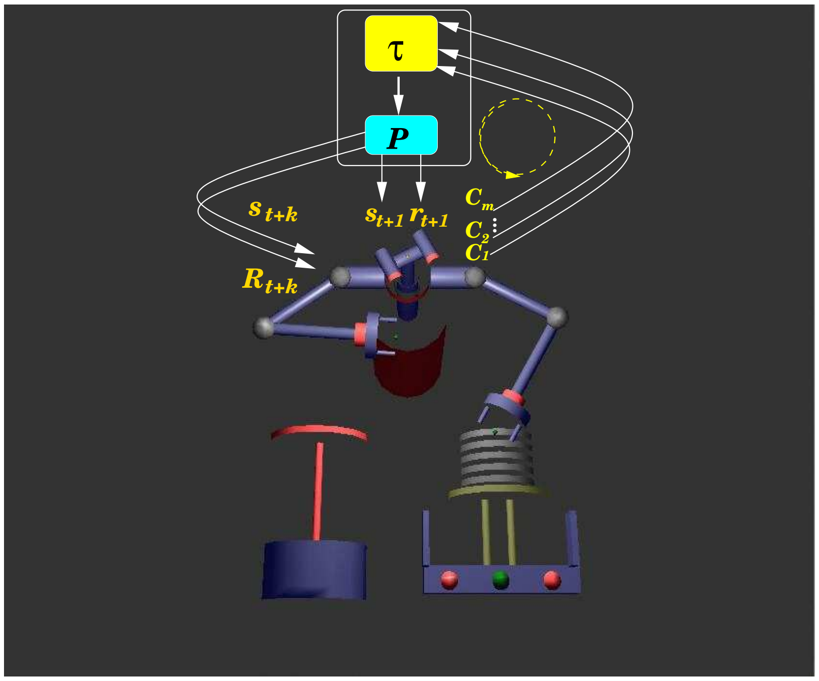


Figure 1.1. An abstract view of the concurrent decision making problem. At time t, the agent executes a set of activities $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ concurrently in state $\mathbf{s}_t$. A concurrent action termination mechanism τ determines when to terminate the concurrent action. Upon termination at some random time $\mathbf{t} + k$, the agent observes the new state $\mathbf{s}_{t+k}$ according to the transition function $\mathcal{P}$ and receives the multi-step reward $\mathbf{R}_{t+1}$.

Concurrent decision making can be defined as a class of problems that involve a decision making process in which the agent's goal is to behave optimally, either by concurrently executing multiple activities, or by optimizing multiple objectives simultaneously. More specifically, we assume that an agent inhabits an environment and has access to a set of previously acquired skills (Singh et al., 2004) $\zeta = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$, some of which can be executed concurrently (Figure 1.1). The agent and environment interact at discrete time steps $\mathbf{t}$. At time step $\mathbf{t}$ the agent observes the current state of the environment $\mathbf{s}_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states. The agent then selects a subset of activities $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ and initiates them in the current state $\mathbf{s}_t$. At every primitive time step, the agent transitions to the next state $\mathbf{s}_{t+1}$ according to a transition function $\mathcal{P}$, and receives single step reward $\mathbf{r}_{t+1}$. A concurrent action termination mechanism (the box labeled τ in Figure 1.1) determines when to terminate the set of activities that are running in parallel. Upon termination at some random time $\mathbf{t} + k$, the agent receives a multi-step accumulated reward $\mathcal{R}_{t+k} \in \mathbb{R}$, and observes the next state of the environment $\mathbf{s}_{t+k}$. The agent's goal is to compute a mapping from states to a set of activities in order to maximize the total amount of reward it receives over the long run.

Concurrent decision making problem is challenging for several reasons:

- The world is stochastic and the uncertain outcomes of executing multiple activities concurrently makes this problem difficult. In sequential decision making, the agent would simply select the next action when the current action being executed terminates. However, in concurrent decision making, often when a set of concurrent activities are executed, one or more activities may terminate before the rest (in the above example, while we are emptying the dish-washer, the eggs may become ready to serve). The optimality of the agent's behavior varies depending on how the semantics of concurrent action termination are defined.

- The agent has access to a limited set of resources in the system. Thus resource conflict may easily happen when a set of interactive activities are running in parallel.

- The *curse of dimensionality* incurs a combinatorial space of concurrent activities. In general, the agent can execute any subset of its acquired skills concurrently, which causes the space of its available concurrent activities to become exponential in the set of primary activities available to the agent.

Various forms of concurrency have been studied in many areas of science and engineering, such as computer systems, robotics and control, artificial intelligence, and machine learning (Figure 1.2). For example, in computer systems, concurrency has been studied in the contexts of computer architecture (i.e., pipeline architectures (Hennessy and Patterson, 1990)), operating systems (i.e., parallel processing (Tanenbaum and Woodhull, 1997)) and concurrent programming (i.e., multi-user database systems (Andrews and Elliott, 1991)).

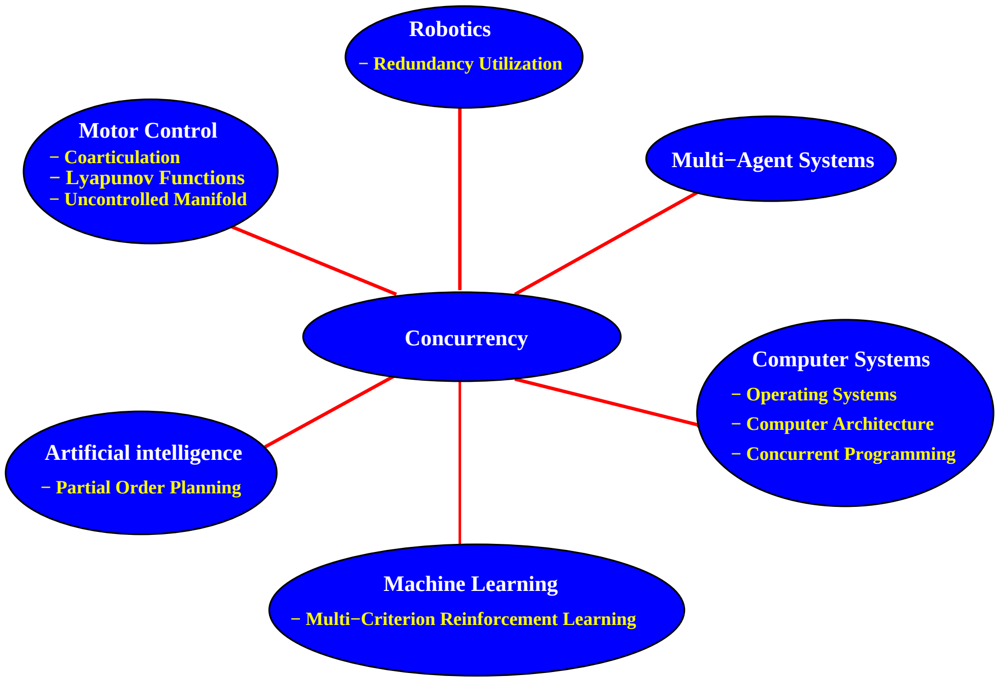


Figure 1.2. Concurrency has a long history in many areas of science and engineering, such as computer systems, robotics and control, artificial intelligence, and machine learning.

In robotics, concurrency is primarily introduced in the context of multi-objective control by exploiting the redundancy in the system kinematics (Craig, 1989; Nakamura, 1991). The common trend is to approximate the overall task in terms of concurrent optimization of a set of sub-tasks, based on their respective degrees of significance. This approach has been studied extensively in many robotics tasks such as obstacle avoidance (Khatib and Maitre, 1978; Nakamura, 1991; Grupen, 2006), avoiding mechanical joint limits (Liegeois, 1977), multi-legged walking (Huber et al., 1996), coordinating multiple communicating mobile robots (Sweeney et al., 2002), robot grasping (Platt et al., 2002), and robust finger gaits in object manipulation (Huber and Grupen, 2002b). Similarly, in motor control research concurrency is primarily investigated in the context of redundancy utilization, or equivalently, the *uncontrolled manifold* approach (Pellionisz and Llinas, 1985; Saltzman and Kelso, 1987; Mussa-Ivaldi et al., 1988; Todorov and Jordan, 2002), and in the context of coarticulation (Jordan and Rosenbaum, 1989; Jordan, 1990; Soechting and Flanders, 1992; Hoff and Arbib, 1993; Engel et al., 1997; Wiesendanger and Serrien, 2001; Johnson and Grafton, 2003; Breteler et al., 2003; Cohen and Rosenbaum, 2004; Baader et al., 2005; Perkins, 2002). Most of these ideas, however, have been investigated only in continuous domains and do not address concurrent decision making in discrete domains. Also, in general the problems studied by such approaches mostly involve lower level of control and do not pertain to higher level reasoning and planning as it takes place in humans.

Concurrent decision making also has a long history in machine learning in the context of decision making under uncertainty based on Markov decision processes (MDPs) when the actions are considered to be structured (Boutilier and Goldszmidt, 1995; Boutilier and Dearden, 1996; Singh and Cohn, 1998; Dean et al., 1998; Boutilier et al., 1999; Guestrin;, 2003; Russell and Zimdars, 2003; Younes and Simmons, 2004; Mausam and S.Weld, 2004; Marthi et al., 2005; Mausam and Weld, 2005). Most of these approaches, however, ignore the temporal properties of such problems, and do not address learning and planning with activities that take various amounts of time for completion (e.g., activities modeled as temporally extended actions (Precup, 2000)).

Multi-agent systems (Weiss, 2000) by definition involve concurrent decision making as they consider several agents acting in the environment simultaneously. The most common setting in multi-agent systems is based on the distributed-control paradigm. In such systems, the main challenge arises from the distributed control problem, limited access of the agents to the state of the environment and other agents, and noise in communication among agents. Despite the large body of work on distributed multi-agent systems, very little research addresses concurrent decision making in centralized control systems where one assumes a central control mechanism initiates, executes and monitors concurrent activities.

There has also been an extensive study of action representation and reasoning with actions in situation calculus, temporal logic and reasoning, and planning. Certain aspects of these areas address the specification and synthesis of concurrent actions. The idea of incorporating partial-order planning (Sacerdoti, 1975, 1977) to generate parallel execution plans has been studied since the early days of planning (Vilain et al., 1989; Veloso et al., 1991; Regnier and Fade, 1991; Knoblock, 1994; Boutilier and Brafman, 2001b,a). Most of these approaches, however, do not address stochastic domains.

1.1 Research Summary

In the broadest sense, the goal of our research is to develop a general decision theoretic framework for studying the concurrent decision making problem. In particular we are interested in developing learning and planning algorithms that alleviate some of the challenges that we face in solving this class of problems.

In summary, in the course of this research, we carry out the following major steps: we first focus on modeling aspects of the concurrent decision making problem, and develop a general decision theoretic model for monitoring execution and termination of concurrent activities. Next, we address the curse of dimensionality that arises in our model. We introduce a framework that is in spirit related to the concept of *coarticulation* in motor control research and demonstrate how this approach can alleviate the curse of dimensionality in concurrent decision making. However, we observe that learning basic models for performing coarticulation may still suffer from the curse of dimensionality. Thus, in the last part of our research, we present a set of approximation methods for performing coarticulation with a tractable complexity. These steps are further elaborated in the following sections.

1.1.1 Concurrent Action Model

Our approach to modeling concurrent decision making is based on semi-Markov decision processes (SMDPs) (Howard, 1971; Puterman, 1994), a widely studied probabilistic decision making paradigm. More formally, we introduce the *concurrent action model (CAM)* (Rohanimanesh and Mahadevan, 2001, 2002), where we primarily consider a system capable of performing multiple activities concurrently. We use the terminology *multi-action* to refer to a set of activities running in parallel. One can think of each activity either as a single step action, or a more complex skill acquired by the agent, modeled as a *temporally extended action* (e.g., a closed loop policy over single step actions (Sutton et al., 1999)). A centralized control architecture is also used for initiating, monitoring and termination handling of the concurrent activities.

One important problem in CAMs is to define the semantics of concurrent action termination, $\mathcal{T}$. When several activities run in parallel, some of them may terminate earlier, while the remaining activities continue to run. In this case how can one characterize the completion of a concurrent action? How does the choice of concurrent termination mechanism influence the optimality of the agent's behavior? We need to precisely define the

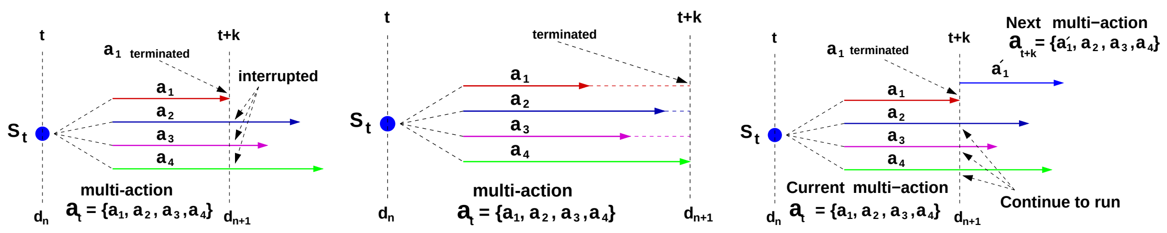


Figure 1.3. Left: T_{any} termination scheme. A concurrent action terminates when any of the primary actions terminates, Middle: T_{all} termination scheme. A concurrent action terminates when all of the primary actions terminate. Right: $T_{continue}$ termination scheme. A concurrent action terminates when any of the primary actions terminates, but those primary action that did not terminate will continue execution.

decision process based on different termination scenarios. To that end, we introduce three natural concurrent action termination mechanisms, namely T_{any} (Figure 1.3 (Left)), T_{all} (Figure 1.3 (Middle)) and $T_{continue}$ (Figure 1.3 (Right)). In T_{any} termination, a concurrent action terminates when any of the primary actions currently being executed terminates. At this point the agent initiates the next concurrent action, ignoring the fact that some of the primary actions have not yet terminated. If the next concurrent action that an agent executes chooses those primary actions that did not terminate, then they continue to run, otherwise they will be interrupted. In the breakfast example, when the eggs are ready to serve, we may interrupt other activities, such as watching television, and start serving the eggs.

In T_{all} termination, a concurrent action terminates when all of the primary actions terminate. In the breakfast example, we may wait, until both boiling eggs and toasting bread processes are all done, before we start serving them. The last termination mechanism (i.e, $T_{continue}$) terminates concurrent activities in a manner similar to T_{any}. However, it lets the activities that are still in progress continue running (without interrupting them), while the agent may also initiate new activities along with them. $T_{continue}$ termination mechanism is indeed very natural and can be observed in many real world problems. In the breakfast example, if the coffee becomes ready before the eggs are done, we let the process of boiling the eggs to continue until the eggs become ready, and may initiate a new activity, such as arranging the table.

Returning to the breakfast example, consider modeling this problem using a CAM, where the states of the problem may involve features such as whether or not the coffee is ready, status of the boiling eggs, and whether or not a friend has called for a ride. By primary actions, we refer to the set of basic abilities of the agent that can be executed along with other activities, such as boiling the eggs or making coffee, and so on. Note that each primary action may take a different amount of time to complete and also its termination is stochastic (for example the amount of time required to boil the eggs is different from the amount of time required to make coffee, and it also varies depending on the type of the pot and the heat level). A concurrent action is then formed by executing two or more primary actions simultaneously, such as boiling the eggs and making coffee. At every step, the agent may also initiate a set of new primary actions along with the other activities that are in progress. For example the agent may turn on the radio while the above activities are in progress.

Modeling concurrent decision making based on SMDPs allows us to cast this problem in a very general setting, inherited from the ability of SMDPs to model a large class of decision making problems. Furthermore, we can incorporate reinforcement learning (RL) methods, that are often employed with SMDP models of the problem. Reinforcement learning (Sutton and Barto, 1998) refers to a class of learning problems in which the agent learns to act through trial-and-error interaction with a dynamic environment. This is a more realistic view of the agent-environment interaction where the agent may not have access to the complete model of the environment.

In this dissertation, we formally establish that CAMs with a set of termination mechanisms as described above, are one realization of SMDPs, and thus standard SMDP learning and planning techniques can be applied to solve them. In succession, we extensively analyze each of the concurrent action termination mechanisms and present a set of theorems that characterize the optimality of the agent's behavior based on the various termination mechanisms. We then compare them with the sequential model that allows only one action to be executed at a time. We also present empirical results in a simple grid world domain that involves concurrent decision making.

1.1.2 Coarticulation Framework

Although CAMs provide an abstract framework for modeling concurrent decision making, the standard SMDP learning and planning methods for solving such problems may be intractable due to the combinatorial set of concurrent activities to which the agent has access. Recall that concurrent actions are formed by executing two or more primary actions and thus the total number of concurrent actions is exponential in the number of primary actions (the maximal concurrent action set is the power set of the primary actions). Although in practice not every configuration of primary actions can be executed concurrently, we cannot assume any feasible bound on the number of possible concurrent actions a priori. It is known that the complexity of planning in MDPs and the complexity of the near optimal reinforcement learning methods are polynomial in the set of states and actions (Papadimitriou and Tsitsiklis, 1987; Kearns and Singh, 1998). This renders the complexity of learning and planning in CAMs exponential in the number of primary actions. Thus in general finding the exact solution for CAMs using standard SMDP learning and planning methods is a hard problem.

Our approach to solving this problem (Rohanimanesh et al., 2004a,b) is based on the fact that many real-world concurrent decision making problems can be viewed in terms of concurrent optimization of a set of prioritized subgoals of the problem (Huber, 2000; Grupen, 2006). In general – especially in multi-objective optimization – it is intuitive to approximate the overall objective of the problem in terms of a set of sub-utility functions, each associated with a subgoal of the problem. This view has a long history in *multi-attribute utility theory* (Keeney and Raiffa, 1993), and in particular seems very suitable for the concurrent decision making problem. It also relates to the *multi-criterion Reinforcement Learning* (MRL) (Gabor et al., 1998) approach in which the reward signal is expressed as a vector whose elements describe a local reward signal associated with one of the objectives of the problem.

This view is observed in our daily activities, where by exploiting many degrees of freedom (DOF) in our body (e.g., arms, legs, eyes, etc), we are able to simultaneously commit to several tasks and as a result generate concurrent plans. A familiar example is a driving task which may involve subgoals such as safely navigating the car, talking on a cell phone, and drinking coffee, with the first subgoal taking precedence over the others. Having the benefit of extra DOF in our body, we are able to simultaneously commit to multiple subgoals. For example we can control the wheels by the left arm and use the right arm to answer the cell phone or drink coffee.

Abusing the terminology from speech and motor control research, by *coarticulation* we refer to the general class of problems in which an agent simultaneously commits to multiple objectives. The key idea in our approach is based on the fact that in many goal-oriented activities – in addition to the optimal policy – there often exists a *redundant* set of *ascending* (and possibly sub-optimal) policies that guarantee achieving the goal with a cost of a slight deviation from optimality. Ascendancy is a property of a policy that guarantees making progress toward the goal of the problem at every step. Such flexibility enables the agent to simultaneously commit to multiple subgoals. This model is also natural in a sense that it enables the agent to reuse the learned activities for solving new tasks by slightly modifying its policies, instead of starting a new learning problem for every possible combination of goals that it may face in future.

It is worth noting that the notion of policy redundancy also captures a form of redundancy that is not explicitly based on the degrees of freedom in the system. To illustrate this, consider the simple example in Figure 1.4 where an agent is located in the lower left corner of the environment and is planning to navigate to a refrigerator located in the upper right corner. The agent can execute one of the four navigation actions *Up*, *Left*, *Down* and *Right* at every state. There is also a phone located in the environment which starts ringing as the agent gets ready to move to the refrigerator. The agent can choose between two redundant optimal paths (i.e., p_1 and p_2) in order to get to the refrigerator, but it chooses the path p_2 in order to answer the phone while approaching the refrigerator. In this example, the agent has only one degree of freedom with respect to its set of actions, since it can only execute one navigation action at a time.

Interestingly, there exists a form of policy redundancy that emerges in systems with multiple degrees of freedom. This type of redundancy generally stems from the fact that not all degrees of freedom are involved when performing an activity. In the above example,

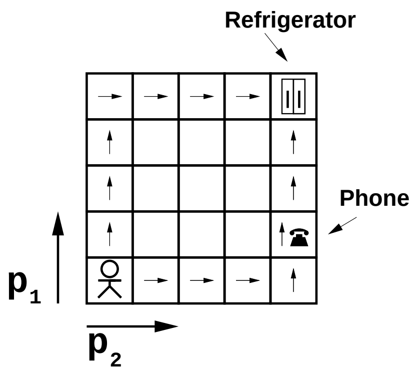


Figure 1.4. An example of policy redundancy: the agent can choose from two equally optimal paths (i.e., p_1 and p_2) in order to get to the refrigerator, but it chooses the path p_2 in order to answer the ringing phone while moving toward the refrigerator.

if the agent has more degrees of freedom, such as arms, more redundant policies can be found. These include, navigating to the goal state through the path p_2 and moving the arms upward, or navigating to the goal state through the path p_2, moving the left arm upward, and moving the right arm downward. This type of policy redundancy serves as a basis of performing coarticulation as we describe later in Chapter 4.

We argue that coarticulation is a natural way for generating parallel execution plans for several reasons. First, many concurrent decision making problems can actually be viewed as concurrent optimization of a set of prioritized subgoals, in which the agent manages its DOF to simultaneously commit to those subgoals. Second, because of the multiple DOF in the system, learned activities offer more flexibility in terms of the range of ascending policies associated with them. For example in the driving task, while the best policy for driving the car would be to control the wheel using both arms, by exploiting the extra DOF in our body we can perform the same task sub-optimally by engaging one arm for turning the wheel and releasing the other arm for committing to the other subgoals of lower priority (such as drinking coffee). However, the key advantage of coarticulation in concurrent decision making lies in its efficient search in the exponential space of concurrent actions. The action selection mechanism in this approach is restricted to those policies that ascend the value functions associated with each activity. This interactive search enables the agent to perform the search in a much smaller space of concurrent actions with a controllable cost in optimality.

To further motivate our approach, consider the life span of an agent. The agent is constantly performing learning and planning based on the set of current and future tasks introduced in the system. The agent also constantly acquires new skills from each learning experience (Iba, 1988; McGovern and Barto, 2001; Pickett and Barto, 2002; Şimşek and Barto, 2004). Even when the agent is not faced with a specific task, it may continue acquiring useful task-independent skills by exploring the environment (Singh et al., 2004; Şimşek et al., 2005). Such skills are efficiently incorporated by the agent when it is faced with a new task during its lifespan.

We can think of such acquired skills as the basic blocks of coarticulation (similar to phonemes in speech synthesis, or reaching to grasp skill in human). It can be observed that we are able to efficiently combine such skills when faced with a new task, however we are able to modify our skills in a context-sensitive fashion. In other words we are coarticulating among our skills in order to produce a more natural course of actions for performing the task more efficiently.

This is illustrated well in the driving example, where one possesses acquired skills such as navigating, drinking coffee, talking on a cell phone, etc. One can perform such tasks in parallel without severely affecting the overall performance. We can observe a form of coarticulation in this example that exploits the many DOF in our bodies to generate concurrent plans. Among different ways that we have learned to steer the wheels, we choose the one that enables us to commit to other subtasks, such as drinking coffee. In other words, we coarticulate between the two subtasks. For example we can steer the wheels by one hand, two hands, or for a short amount of time we can release the wheel completely if the road is clear and straight.

More technically, we can think of each acquired skill as a temporally extended action (Precup, 2000) in a semi-Markov Decision Process. Each skill models an activity that optimizes a subgoal of some sort. We assume that the agent has access to a set of learned activities modeled by a set of SMDP *controllers* $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$ each achieving a subgoal ω_i from the set of subgoals $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$. We further assume that the agent-environment interaction is an episodic task where, at the beginning of each episode, a subset of subgoals $\omega \subseteq \Omega$ are introduced to the agent, where subgoals are ranked according to some priority ranking system. The agent is to devise a a policy by coarticulating among the subgoals such that it simultaneously commits to them according to their degree of significance. In general, optimal policies of controllers do not offer flexibility required in order to commit to many subtasks. However, there exists a special class of admissible near-optimal policies that guarantee making progress toward the goal in every state of the problem. Given a controller, an admissible policy is either an optimal policy, or a policy that *ascends* the optimal state-value function associated with the controller (i.e., on average leads to states with higher values), and is not far from the optimal policy. The ascendancy property of policies is inspired by the Lyapunov constraints for action selection in MDPs introduced by Perkins (2002); Perkins and Barto (2001b,a).

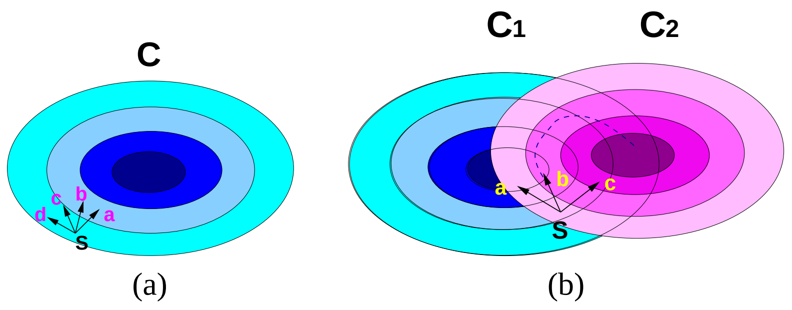


Figure 1.5. (a) actions **a**, **b**, and **c** are ascending on the state-value function associated with the controller $\mathcal{C}$, while action **d** is descending; (b) action **a** and **c** ascend the state-value function $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively, while they descend on the state-value function of the other controller. However action **b** ascends the state-value function of both controllers.

To illustrate this idea, consider Figure 1.5(a) showing a two dimensional state-value function. Regions with darker colors represents states with higher values. Assume that the agent is currently in state marked $\mathbf{s}$. The arrows show the direction of state transition as a result of executing different actions, namely actions $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$. The first three actions lead the agent to states with higher values, in other words they ascend the state-value function, while action $\mathbf{d}$ descends it. Figure 1.5(b) shows how introducing admissible policies enables simultaneous solution of multiple subgoals. In this figure, actions $\mathbf{a}$ and $\mathbf{c}$ are optimal in controllers $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively, but they both descend the state-value function of the other controller. However if we allow actions such as action $\mathbf{b}$, we are guaranteed to ascend both value functions, with a slight degradation in optimality. In this, example by choosing action $\mathbf{b}$ we are coarticulating between both tasks while the first task takes precedence over the second task.

In this dissertation, we investigate this class of policies and, in particular, introduce ϵ-redundant controllers as the basic blocks of the coarticulation in MDPs. An ϵ-redundant controller represents a class of ascending policies that deviate from the optimal policy associated with the controller by a user controllable factor ϵ. The larger the ϵ factor, the broader the class of ascending policies that the controller represents, and hence the more flexibility each controller offers for the coarticulation problem. We also present a coarticulation algorithm for performing coarticulation among a set of subtasks, each associated with an ϵ-redundant controller. We show how our coarticulation algorithm can reduce the impact of the curse of dimensionality in the action space. One major contribution of this dissertation is a set of theoretical results analyzing various aspects of our coarticulation framework. We present results characterizing various properties of ϵ-redundant controllers. We also present a set of conditions under which we can theoretically prove when a concurrent policy performs strictly better than a purely sequential policy. Finally, we present our empirical results in a simulated grid world problem.

1.1.3 Approximate Solutions for Scaling Coarticulation to Large Problems

Although we demonstrate that our coarticulation algorithm performs a more efficient search in the space of concurrent actions, the algorithms for computing the class of ascending policies in ϵ-redundant controllers still suffer from the curse of the dimensionality in large domains. Thus, efficient algorithms for computing the set of ascending policies in such controllers, and also scalable algorithms for the action selection problem are required.

We present a set of approximation techniques in order to address this problem (Rohanimanesh and Mahadevan, 2005). We assume that the optimal state-action value function associated with each controller can be approximated using linear function approximation techniques (Bertsekas and Tsitsiklis, 1997; Sutton and Barto, 1998). We then present an approximate algorithm based on non-serial dynamic programming methods (Dechter, 1999) for computing the top **h** best actions in every state. We show that our algorithm has a computational complexity logarithmic in **h** and exponential in the *network width* (Dechter, 1999) induced by the structure of the approximate linear additive value function.

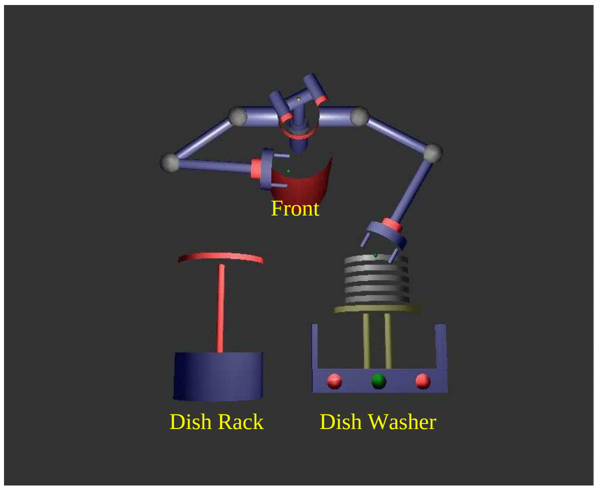


Figure 1.6. A simulated robot for object manipulation task. The task of the robot is to empty the dishes from the dish-washer and stack them into the dish-rack.

We use a simulated robot for evaluating our coarticulation approach. Figure 1.6 shows a robot with three degrees of freedom, namely, the *eyes*, the *left arm*, and the *right arm*. The robot's task is to empty the dish-washer and stack the dishes in the dish-rack. Note that the robot can exploit its DOFs and perform the task more efficiently. For example, while the robot uses its right arm to stack a dish that has already been picked up, it can transport its left arm to the dish-washer in order to pick a different dish concurrently. We incorporate all the ideas that we presented throughout this dissertation for solving this problem and show how our coarticulation framework can be used naturally to generate parallel execution plans. Our empirical evaluation includes results both on the performance of the coarticulation method versus non-concurrent plans, and also the accuracy and the performance of our approximate algorithms.

1.2 Overview of Technical Contributions

The main contributions of this dissertation are summarized as follows:

Concurrent Action Model

- We formally introduce and describe the concurrent action model (CAM). This model is derived from the semi-Markov decision process model, where the notion of action is generalized to include sets of actions (that will be executed concurrently). We introduce a set of concurrent action termination mechanisms and establish that the semantics of the model under such termination mechanisms are well defined.

- We extensively analyze each termination mechanism and present a set of theorems that characterize the optimality of the policies learned under each of them. We also provide experimental results in a simulated domain supporting our theoretical results.

A Model of Coarticulation for Solving the Concurrent Decision Making Problem

- We formally introduce the concept of coarticulation in MDPs and demonstrate how it can be used in order to alleviate the curse of dimensionality in the combinatorial space of concurrent actions.

- We introduce ϵ-redundant controllers as the basic block of the coarticulation framework. We define ascending policies and investigate their properties.

- We present a coarticulation algorithm for solving a concurrent decision making problem approximated in terms of a set of prioritized subtasks, each associated with an ϵ-redundant controller.

- We present an extensive set of theoretical results characterizing various properties of the ϵ-redundant controllers, and also a set of theoretical results establishing bounds on the optimality of the coarticulation algorithm.

- We empirically demonstrate the performance of our coarticulation method in a simulated grid-world domain.

Scaling the Coarticulation Framework to Large Domains

- We introduce an approach for scaling the coarticulation framework to large problems. We present a tractable approximate algorithm for computing the ascending policies in ϵ-redundant controllers.

- We empirically demonstrate all of the methods studied in this thesis on a simulated robot performing a concurrent decision making task. Our experimental results demonstrate how coarticulation can be viewed as one natural way for generating parallel execution plans. We also present a set of results that measure the performance and the accuracy of the approximate algorithm in larger domains.

1.3 Outline

The rest of this dissertation is organized as follows:

Chapter 2: This chapter provides the background material for this dissertation. We start by overviewing Markov decision processes (MDPs), and review some key ideas and solution methods. Next we overview the reinforcement learning (RL) problem and briefly explore the standard solution methods for solving this problem. Finally, discrete time semi-Markov decision Processes (SMDPs) are introduced, and the *options* framework is presented as a special case of SMDPs.

Chapter 3: We introduce CAMs and provide an in depth study of the various coordination mechanisms employed by the model. A set of theoretical results on the correctness of the model semantics, and the optimality of concurrent decision making based on the various concurrent action termination mechanisms is presented. We also present an empirical evaluation of this model in a simulated grid world domain.

Chapter 4: We present a model of coarticulation for alleviating the curse of dimensionality in CAMs. We begin by reviewing the historical development of the coarticulation phenomenon in other communities such as speech and control. Then we formally define ϵ-redundant controllers as the basic block of the coarticulation framework, and present an algorithm for performing coarticulation. An extensive set of theoretical results characterizing such controllers, and also provide theoretical bounds on the performance of the coarticulation method are presented. Finally, we detail a set of empirical results in a simulated grid world domain.

Chapter 5: We demonstrate how we can scale the coarticulation framework to large domains. We present a tractable approximate algorithm for computing the class of ascending policies in ϵ-redundant controllers. We then empirically demonstrate the performance of our coarticulation method in a simulated grid world domain in the context of a concurrent decision making object manipulation task.

Chapter 6: We summarize the dissertation and discuss some directions for future research.

CHAPTER 2
BACKGROUND

The development of our model for concurrent decision making is a multidisciplinary endeavor, with a considerable amount of related literature. The goal of this chapter is to provide the necessary background and framework for describing the basic research in this dissertation. Consequently, the following review is somewhat general with additional details deferred to later chapters. For a more comprehensive introduction, readers may also refer to standard texts such as (Bertsekas and Tsitsiklis, 1997; Sutton and Barto, 1998).

2.1 Markov Decision Processes

Markov decision processes (MDPs) (Puterman, 1994) model the sequential decision making problem, where a learning *agent* interacts with an *environment* at some discrete time scale, $t = 0, 1, 2, \ldots$. On each time step t the agent observes the current state of the environment, $s_t \in \mathcal{S}$, and selects an action, a_t. In response to each action, a_t, the environment produces a delayed reward, $r_{t+1} \in \mathcal{R}$, and a next state s_{t+1}. Formally an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function with $\mathcal{P}(s, a, s')$ (alternatively represented as $\mathcal{P}^a_{ss'}$) being the probability of transition from state s to state s' when action a is executed, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the expected reward function, with $\mathcal{R}(s, a)$ being the expected reward for taking action a in state s. Let $\mathcal{A}_s \subset \mathcal{A}$ denote the set of actions admissible in state s. In this work we assume that $\mathcal{S}$ and $\mathcal{A}$ are finite.

A *stochastic Markov policy* π is defined as a mapping $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, where $\pi(s, a)$ gives the probability of selecting action a in state s. For any policy π, the *value* of a state s under π is defined as the expected discounted infinite-horizon sum of rewards from state s. The *state-value* function $\mathcal{V}^\pi$ defines a mapping from states to their values under the policy π:

$$\begin{aligned}\mathcal{V}^\pi(s) &= E_\pi\{r_{t+1} + \gamma r_{t+2} + \ldots | s_t = s\} \\ &= E_\pi\{r_{t+1} + \gamma \mathcal{V}^\pi(s_{t+1}) | s_t = s\} \\ &= \sum_{a \in \mathcal{A}_s} \pi(s,a) \left[\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')\mathcal{V}^\pi(s')\right]\end{aligned} \tag{2.1}$$

where $0 \leq \gamma < 1$ is a discount factor. The objective of the learning agent is to learn an optimal policy that maximizes the expected discounted future reward. The *optimal* state-value function gives the value of each state under the optimal policy:

$$\begin{aligned}\mathcal{V}^*(s) &= \max_\pi \mathcal{V}^\pi(s) \\ &= \max_{a \in \mathcal{A}_s} E\{r_{t+1} + \gamma \mathcal{V}^*(s_{t+1}) | s_t = s, a_t = a\} \\ &= \max_{a \in \mathcal{A}_s} \left[\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')\mathcal{V}^*(s')\right]\end{aligned} \tag{2.2}$$

Any policy that achieves the maximum in Equation 2.2 is by definition an optimal policy. Equations 2.1 and 2.2 are referred to as *Bellman equations* , which recursively relate value functions to themselves.

Similarly, we can define Bellman equations for state-action pairs. The *state-action value function* $\mathcal{Q}^\pi$ for a policy π is the mapping from state-action pairs to their values, and gives the expected value of the sum of future rewards starting from state s, taking action a, and following π thereafter:

$$\begin{aligned}\mathcal{Q}^\pi(s,a) &= E_\pi\{r_{t+1} + \gamma r_{t+2} + \ldots | s_t = s, a_t = a\} \\ &= \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')\mathcal{V}^\pi(s') \\ &= \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') \sum_{a' \in \mathcal{A}_{s'}} \pi(s',a')\mathcal{Q}^\pi(s',a')\end{aligned}$$

and the optimal action-value function $\mathcal{Q}^*$ satisfies:

$$\begin{aligned}\mathcal{Q}^*(s) &= \max_{\pi} \mathcal{Q}^{\pi}(s,a) \\ &= \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') \left[R(s,a) + \gamma \max_{a' \in A_{s'}} \mathcal{Q}^*(s',a') \right] \\ &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') \max_{a' \in A_{s'}} \mathcal{Q}^*(s',a')\end{aligned} \tag{2.3}$$

The Bellman equations 2.2 and 2.3 are related by:

$$\mathcal{V}^*(s) = \max_{a} \mathcal{Q}^*(s,a) \tag{2.4}$$

known as the *Bellman optimality equation*. Typically, a solution to a MDP problem is found by solving the Bellman optimality equation 2.4. An alternative way of defining the optimal value function is based on the *Bellman operator* $\mathcal{T}^*$ defined as:

$$\mathcal{T}^* \ \mathcal{V}^{\pi}(s) = \max_{a} \mathcal{Q}^{\pi}(s,a) \tag{2.5}$$

The optimal value function $\mathcal{V}^*$ is the fixed point $\mathcal{V}^* = \mathcal{T}^* \mathcal{V}^*$.

2.2 Reinforcement Learning

Reinforcement learning (RL) (Sutton and Barto, 1998) is a computational framework for solving the sequential decision making problem. RL is distinguished from other computational approaches by its emphasis on learning from direct interaction with an environment. When a sequential decision making problem is modeled as an MDP, RL algorithms try to approximate the optimal value function. Many RL algorithms are instances of the *temporal difference* (TD) learning (Sutton, 1988) where an agent learns and updates estimates of the value function directly from raw experience without a model of the environment's dynamics. One important feature of TD methods is that they update the estimates based in part on other learned estimates, without waiting for a final outcome.One of the more popular TD methods is known as *Q-learning* (Watkins, 1998) that seeks to approximate the optimal action-value function from an experience that involves a transition from the current state s to the next state s' as a result of executing action a in state s and observing a reward of r:

$$\mathcal{Q}^*(s,a) \leftarrow \mathcal{Q}^*(s,a) + \alpha \left[r + \gamma \max_{a' \in \mathcal{A}(s')} \mathcal{Q}^*(s',a') - \mathcal{Q}^*(s,a) \right] \tag{2.6}$$

where $0 < \alpha \leq 1$ is the learning rate.

2.3 Discrete-Time Semi-Markov Decision Processes

Discrete-time semi-Markov decision processes (SMDPs) (Howard, 1971; Dietterich, 2000) are extension of MDPs that model actions that can take variable amount of time to complete. Formally a discrete-time SMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{N} \rightarrow [0,1]$ is the transition probability function with $\mathcal{P}(s,a,s',n)$ being the probability of transition from state s to state s' in n time steps when action a is executed. In this model $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the expected reward function, with $\mathcal{R}(s,a)$ being the expected reward for taking action a in state s. Let $\mathcal{A}_s \subset \mathcal{A}$ denote the set of actions admissible in state s.

For a stochastic policy π, the *value* of a state s under π is defined as the expected discounted infinite-horizon rewards from state s and the *state-value* function $\mathcal{V}^\pi$ defines a mapping from states to their values under the policy π:

$$\begin{aligned} \mathcal{V}^\pi(s) &= E_\pi\{r_{t+1} + \gamma r_{t+2} + \ldots | s_t = s\} \\ &= \sum_{a \in \mathcal{A}_s} \pi(s,a) \left[\mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \sum_{k=1}^{\infty} \gamma^k \, \mathcal{P}(s,a,s',k) \, \mathcal{V}^\pi(s') \right] \end{aligned} \tag{2.7}$$

where $0 \leq \gamma < 1$ is a discount factor. The objective of the learning agent is to learn an optimal policy that maximizes the expected discounted future reward. The *optimal* state-value function gives the value of each state under the optimal policy:

$$\begin{aligned}\mathcal{V}^*(s) &= \max_{\pi} \mathcal{V}^{\pi}(s) \\ &= \max_{a \in \mathcal{A}_s} \left[\mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \sum_{k=1}^{\infty} \gamma^k \; \mathcal{P}(s,a,s',k) \; \mathcal{V}^*(s') \right]\end{aligned} \tag{2.8}$$

Any policy that achieves the maximum in Equation 2.2 is by definition an optimal policy. Similarly, we can define Bellman equations for state-action pairs. The *action value function* $\mathcal{Q}^{\pi}$ for a policy π is the mapping from state-action pairs to their values, and gives the expected value of the sum of future rewards starting from state s, taking action a, and following π thereafter:

$$\begin{aligned}\mathcal{Q}^{\pi}(s,a) &= E_{\pi}\{r_{t+1} + \gamma r_{t+2} + \ldots | s_t = s, a_t = a\} \\ &= \mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \sum_{k=1}^{\infty} \gamma^k \; \mathcal{P}(s,a,s',k) \; \sum_{a' \in \mathcal{A}_{s'}} \pi(s',a') \mathcal{Q}^{\pi}(s',a')\end{aligned}$$

and the optimal action-value function $\mathcal{Q}^*$ satisfies:

$$\begin{aligned}\mathcal{Q}^*(s) &= \max_{\pi} \mathcal{Q}^{\pi}(s,a) \\ &= \mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \sum_{k=1}^{\infty} \gamma^k \; \mathcal{P}(s,a,s',k) \; \max_{a' \in A_{s'}} \mathcal{Q}^*(s',a')\end{aligned} \tag{2.9}$$

SMDP learning methods, such as *SMDP Q-learning* (Bradtke and Duff, 1995) extend the Q-learning (Equation 2.6) taking into account the duration of each action:

$$\mathcal{Q}(s,a) \leftarrow \mathcal{Q}(s,a) + \alpha \left[\mathbf{r} + \gamma^k \max_{a' \in A_{s'}} \mathcal{Q}(s',a') - \mathcal{Q}(s,a) \right] \tag{2.10}$$

where **r** is the reward received upon completion of the action a, and k is the duration of the action a.

2.3.1 Options

Options (Sutton et al., 1999) are one realization of SMDPs that generalize the primitive actions in MDPs to model temporally extended courses of action. Options consist of three components: a policy $\pi : \mathcal{S} \times A \rightarrow [0, 1]$, a termination condition $\beta : \mathcal{S} \rightarrow [0, 1]$, and an initiation set $\mathcal{I} \subseteq \mathcal{S}$, where I denotes the set of states $s \in \mathcal{S}$ in which the option can be initiated. Note that we can restrict the scope of application of a particular option by controlling the initiation set and the termination condition. For any state s, if option π is taken, then primitive actions are selected based on π until it terminates according to β. An option O is a *Markov option* if its policy, initiation set and termination condition depend stochastically only on the current state $s \in \mathcal{S}$.

Given a set of options O, let O_s denote the set of options in O that are available in each state $s \in \mathcal{S}$ according to their initiation set. O_s resembles A_s in the standard reinforcement learning framework, in which A_s denotes the set of primitive (single step) actions. Similarly, we introduce *policies over options*. For a decision epoch d_t, the Markov policy over options $\mu : \mathcal{S} \times O \rightarrow [0, 1]$ selects an option $o_t \in O$, according to the probability distribution $\mu(s_t, .)$. The option o_t is then initiated in s_t until it terminates at a random time $t + k$ in some state s_{t+k} according to the termination condition, and the process repeats in s_{t+k}. For an option $o \in O$, and for any state $s \in S$, let $\varepsilon(o, s, t)$ denote the event of o being initiated in state s at time t. The total discounted reward accrued by executing option o in any state $s \in S$ is defined as:

$$\mathcal{R}_s^o = E\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{k-1} r_{t+k} \mid \varepsilon(o, s, t)\}$$

where $t + k$ is the random time at which o terminates. Also let $\mathcal{P}^o(s, s', k)$ denote the *pseudo*-probability [1] that the option o is initiated in state s and terminates in state s' after k steps. Then

$$\mathcal{P}^o_{ss'} = \sum_{k=1}^{\infty} \mathcal{P}^o(s, s', k)\gamma^k$$

Given the reward and state transition model of option o, we can write the Bellman equation for the value of a general policy μ as :

$$\mathcal{V}^\mu(s) = \sum_{o \in O_s} \mu(s, o) \left[\mathcal{R}^o_s + \sum_{s'} \mathcal{P}^o_{ss'} \mathcal{V}^\mu(s') \right]$$

Similarly we can write the “option-value” Bellman equation for the value of an option o in state s as

$$\mathcal{Q}^\mu(s, o) = \mathcal{R}^o_s + \sum_{s'} \mathcal{P}^o_{ss'} \sum_{o' \in O_{s'}} \mu(s', o') \mathcal{Q}^\mu(s', o')$$

and the corresponding *optimal* Bellman equations are as follows:

$$\mathcal{V}^*_O(s) = \max_{o \in O_s} \left[\mathcal{R}^o_s + \sum_{s'} \mathcal{P}^o_{ss'} \mathcal{V}^*_O(s') \right]$$
$$\mathcal{Q}^*_O(s, o) = \mathcal{R}^o_s + \sum_{s'} \mathcal{P}^o_{ss'} \max_{o' \in O_{s_{t+k}}} \mathcal{Q}^*(s', o')$$

We can use *synchronous value iteration* (SVI) (Bellman, 1957; Puterman, 1994; Sutton and Barto, 1998) to compute $\mathcal{V}^*_O(s)$ and $\mathcal{Q}^*_O(s, o)$, which iterates the following step for every state $s \in S$:

[1]This quantity is not a true probability distribution for $\gamma \neq 1$. Thus we use the term *pseudo*-probability to refer to it. In fact it can be shown that it induces a *sub-stochastic* process in the SMDP level.

$$\mathcal{V}_t(s) = \max_{o \in O_s} \left[\mathcal{R}_s^o + \sum_{s'} \mathcal{P}_{ss'}^o \mathcal{V}_{t-1}(s') \right]$$

$$\mathcal{Q}_t(s,o) = \mathcal{R}_s^o + \sum_{s'} \mathcal{P}_{ss'}^o \max_{o' \in O_{s'}} \mathcal{Q}_{t-1}(s', o')$$

Alternatively, if the option model is unknown, we can estimate $\mathcal{Q}_O^*(s,o)$ using SMDP *Q-learning*, by doing sample backups after the termination of each option o, which transitions from state s to s' in k steps with cumulative discounted reward **r** :

$$\mathcal{Q}(s,o) \leftarrow \mathcal{Q}(s,o) + \alpha \left[\mathbf{r} + \gamma^k \max_{o' \in O_{s'}} \mathcal{Q}(s', o') - \mathcal{Q}(s,o) \right]$$

2.3.2 Subgoal Options

Formally, a subgoal option Precup (2000) of an MDP $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ is defined by a tuple $\mathcal{C} = \langle \mathcal{M}_\mathcal{C}, \mathcal{I}, \beta \rangle$. The MDP $\mathcal{M}_\mathcal{C} = \langle \mathcal{S}_\mathcal{C}, \mathcal{A}_\mathcal{C}, \mathcal{P}_\mathcal{C}, \mathcal{R}_\mathcal{C} \rangle$ is the option MDP induced by the option $\mathcal{C}$ in which $\mathcal{S}_\mathcal{C} \subseteq \mathcal{S}$, $\mathcal{A}_\mathcal{C} \subseteq \mathcal{A}$, $\mathcal{P}_\mathcal{C}$ is the transition probability function induced by $\mathcal{P}$, and $\mathcal{R}_\mathcal{C}$ is chosen to reflect the subgoal of the option. The policy component of such options are the solutions to the option MDP $\mathcal{M}_\mathcal{C}$ associated with them.

CHAPTER 3

CONCURRENT DECISION MAKING: A GENERAL FRAMEWORK

In this chapter we present a general decision theoretic framework for modeling the concurrent decision making problem. We are interested in developing a model that allows concurrent execution of various activities. We describe a set of concurrent coordination mechanisms and present a set of theoretical results asserting the correctness of the model semantics. We also theoretically study various concurrent coordination mechanisms and present a set of theoretical results that evaluate the class of optimal policies under each concurrent coordination mechanism. Our theoretical results are complemented by an experimental study using a simulated navigation task. The experiments illustrate the trade-offs between optimality and convergence speed, and the advantages of concurrency over sequentiality.

3.1 Concurrent Action Model

Building on SMDPs (Howard, 1971; Puterman, 1994), we introduce the *Concurrent Action Model (CAM)* (Rohanimanesh and Mahadevan, 2001, 2002) as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{T})$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of *primary* actions, $\mathcal{P}$ is a transition probability distribution $\mathcal{S} \times \wp(\mathcal{A}) \times \mathcal{S} \times \mathbb{N} \rightarrow [0, 1]$, where $\wp(\mathcal{A})$ is the power-set of the primary actions and $\mathbb{N}$ is the set of natural numbers, and $\mathcal{R}$ is the reward function mapping $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Here, a concurrent action is simply represented as a set of primary actions (hereafter called a *multi-action*), where each primary action is either a single step action, or a *temporally extended action* (e.g., modeled as a closed loop policy over single step actions, see (Sutton et al., 1999)). $\mathcal{T}$ is a concurrent action termination mechanism. Note that when a set of concurrent activities are being executed, some of them may terminate before others. $\mathcal{T}$ defines the semantics of the concurrent action termination event.

We denote the set of multi-actions that can be executed in a state s by $\mathcal{A}_s$ (note that this also includes primary actions, since a primary action is essentially a multi-action of cardinality one). In practice, this function can capture *resource* constraints that limit how many actions an agent can execute in parallel. Thus, the transition probability distribution in practice may be defined over a much smaller subset than the power-set of primary actions. Hereafter we use bold face letters (e.g., $\mathbf{a}$) to refer to a multi-action and use the normal letters to refer to the primary actions present in $\mathbf{a}$ (e.g., $a \in \mathbf{a}$).

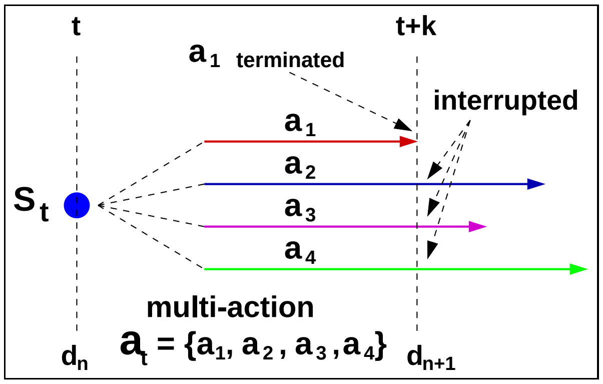


Figure 3.1. T_{any} termination scheme. The termination occurs when *any* of the concurrent activities terminate.

An important problem is to understand how to define decision epochs for concurrent processes, since the primary actions in a multi-action may not terminate at the same time. The event of termination of a multi-action (i.e., $\mathcal{T}$) can be defined in many ways. We have considered three natural ways of terminating a multi-action:

$$\mathcal{T} = \{T_{any}, T_{any}, T_{continue}\}$$

These termination schemes are illustrated in Figures 3.1, 3.2, and 3.3. In the T_{any} termination scheme (Figure 3.1), the next decision epoch is when the first primary action within the multi-action currently being executed terminates, where the rest of the primary actions that did not terminate naturally are interrupted (the notion of interruption is similar to (Sutton et al., 1999)). In the T_{all} termination scheme (Figure 3.2), the next decision epoch is the earliest time at which all the primary actions within the multi-action currently being executed have terminated.

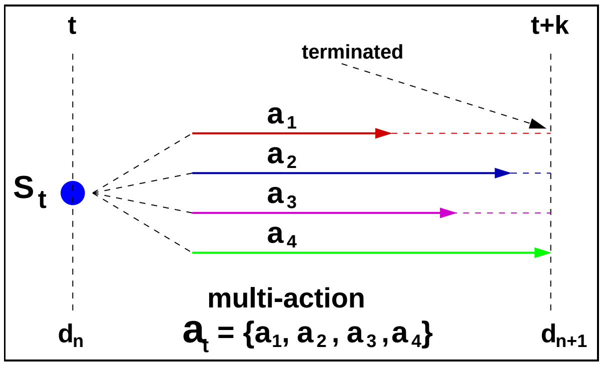


Figure 3.2. T_{all} termination scheme. The termination occurs when *all* of the concurrent activities terminate.

We can design other termination schemes by combining T_{any} and T_{all} : for example, another termination scheme called *continue* is one that always terminates based on the T_{any} termination scheme, but lets those primary actions that did not terminate naturally continue running, while initiating new primary actions if they are going to be useful (Figure 3.3).

A deterministic *Markovian* (memoryless) policy in CAMs is defined as the mapping $\pi : \mathcal{S} \rightarrow \wp(\mathcal{A})$. Note that even though the mapping is independent of the termination scheme, the behavior of a multi-action policy depends on the termination scheme that is used in the model. To illustrate this, let $\langle \pi, \tau \rangle$ (called a *policy-termination* construct) denote the process of executing the multi-action policy π using the termination scheme

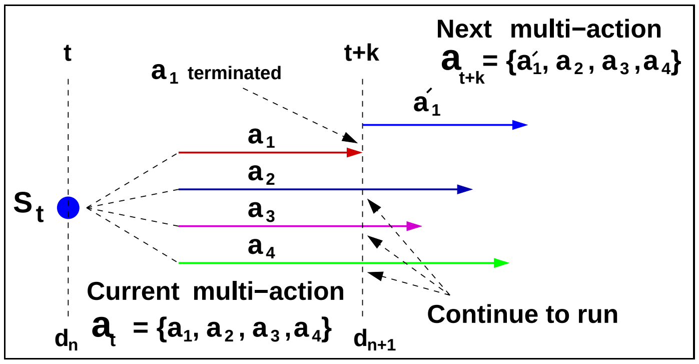


Figure 3.3. $T_{continue}$ termination scheme. The termination occurs when *any* of the concurrent activities terminate, however the next multi-action will continue the execution of those activities that did not terminate.

$\tau \in \{T_{any}, T_{all}\}$. To simplify notation, we only use this form whenever we want to explicitly point out what termination scheme is being used for executing the policy π. For a given Markovian policy, we can write the value of that policy in an arbitrary state given the termination mechanism used in the model. Let $\Theta(\pi, s_t, \tau)$ denote the event of initiating the multi-action $\pi(s_t)$ at time t and terminating it according to $\tau \in \{T_{any}, T_{all}\}$ termination scheme. We can write the value function equation for such a policy as:

$$\begin{aligned} \mathcal{V}^{\langle \pi,\tau \rangle}(s) &= E\{r_{t+1} + \gamma r_{t+2} + \ldots + \mid \Theta(\pi, s_t, \tau)\} \\ &= \sum_{\mathbf{a} \in \mathcal{A}_s} \pi(s, \mathbf{a}) \left[\mathcal{R}_s^{\langle \mathbf{a},\tau \rangle} + \sum_{s'} \sum_{k=1}^{\infty} \gamma^k \, \mathcal{P}^\tau(s, \mathbf{a}, s', k) \, \mathcal{V}^{\langle \pi,\tau \rangle}(s') \right] \end{aligned} \tag{3.1}$$

and:

$$\mathcal{R}_s^{\langle \mathbf{a},\tau \rangle} = E\{r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{k-1} r_{t+k} \mid \Theta(\mathbf{a}, s, \tau)\} \tag{3.2}$$

where $\mathcal{R}_s^{\langle \mathbf{a},\tau \rangle}$ represents the expected partial return of executing the multi-action $\mathbf{a}$ in state s until it terminates according to the τ termination scheme. In all of these equations, we augment the transition probability function with the termination scheme (i.e., $\mathcal{P}^\tau(s, \mathbf{a}, s')$) to denote the probability of transitioning from state s to state s' when the termination τ occurs.

Also let π^{*_τ} denote the optimal multi-action policy within the space of policies over multi-actions that terminate according to the $\tau \in \{T_{any}, T_{all}\}$ termination scheme. To simplify notation, we may alternatively use $*_\tau$ to denote optimality with respect to the τ termination scheme. Then the optimal value function can be written as:

$$\mathcal{V}^{*_\tau}(s) = \max_{\mathbf{a} \in \mathcal{A}_s} \left[\mathcal{R}_s^{\langle \mathbf{a}, \tau \rangle} + \sum_{s' \in \mathcal{S}} \sum_{k=1}^{\infty} \gamma^k \ \mathcal{P}^\tau(s, \mathbf{a}, s', k) \ \mathcal{V}^{*_\tau}(s') \right] \tag{3.3}$$

Similarly, we can write the Bellman equations for the value of a state-multi-action pair:

$$\begin{aligned} \mathcal{Q}^{\langle \pi, \tau \rangle}(s, \mathbf{a}) &= E\{r_{t+1} + \gamma r_{t+2} + \ldots + \mid \Theta(\pi, s_t, \tau), \mathbf{a}\} \\ &= \mathcal{R}_s^{\langle \mathbf{a}, \tau \rangle} + \sum_{s'} \sum_{k=1}^{\infty} \gamma^k \ \mathcal{P}^\tau(s, \mathbf{a}, s', k) \ \mathcal{V}^{\langle \pi, \tau \rangle}(s') \end{aligned} \tag{3.4}$$

where $Q^{\langle \pi, \tau \rangle}(s, \mathbf{a})$ denotes the multi-action value of executing $\mathbf{a}$ in state s (terminated using τ termination scheme) and following the policy π thereafter. For the optimal value of a state-multi-action pair we can write the Bellman equation as follows:

$$\mathcal{Q}^{*_\tau}(s, \mathbf{a}) = \mathcal{R}_s^{\langle \mathbf{a}, \tau \rangle} + \sum_{s'} \sum_{k=1}^{\infty} \gamma^k \ \mathcal{P}^\tau(s, \mathbf{a}, s', k) \ \max_{\mathbf{a}' \in \mathcal{A}_{s'}} \mathcal{Q}^{*_\tau}(s', \mathbf{a}') \tag{3.5}$$

Similarly, we can write *SMDP Q-learning* (Bradtke and Duff, 1995) update rules, for learning the state-multi-action value function through direct interaction with the environment:

$$\mathcal{Q}^{*_\tau}(s, \mathbf{a}) \leftarrow \mathcal{Q}^{*_\tau}(s, \mathbf{a}) + \alpha \left[\mathbf{r} + \gamma^k \max_{a' \in \mathcal{A}'_s} \mathcal{Q}^{*_\tau}(s', \mathbf{a}') - \mathcal{Q}^{*_\tau}(s, \mathbf{a}) \right] \tag{3.6}$$

where $\mathbf{r}$ is the reward received upon completion of the multi-action $\mathbf{a}$ (based on the τ termination scheme), and k is the duration of the multi-action $\mathbf{a}$.

The policy associated with the *continue* termination scheme is a *history dependent* policy, since for a given state s_t, the *continue* policy will select a multi-action such that it includes the set of all the primary actions of the multi-action executed in the previous decision epoch that did not terminate naturally in the current state s_t (we refer to this set as the *continue-set* represented by h_t). The *continue* policy is defined as the mapping $\pi_{cont} : \mathcal{S} \times \mathcal{H} \rightarrow \wp(\mathcal{A})$ in which $\mathcal{H}$ is a set of continue-sets h_t. Note that the value function definition for the *continue* policy should be defined over both state s_t and the continue-set h_t (represented by $\langle s_t, h_t \rangle$), i.e., $\mathcal{V}^{\pi_{cont}}(\langle s_t, h_t \rangle)$. Let the function $A(s_t, h_t)$ return the set of multi-actions that can be executed in state s_t that include the continuing primary actions in h_t. Then the *continue* policy is formally defined as:

$$\pi_{cont}(\langle s_t, h_t \rangle) = arg \max_{\mathbf{a} \in A(s_t, h_t)} Q^{\pi_{cont}}(\langle s_t, h_t \rangle, \mathbf{a})$$

To illustrate this, assume that the current state is s_t and the multi-action $\mathbf{a_t} = \{a_1, a_2, a_3, a_4\}$ is executed in state s_t. Also, assume that the primary action a_1 is the first action that terminates after k steps in state s_{t+k}. According to the definition of the *continue* termination scheme (that terminates based on T_{any}), the multi-action $\mathbf{a_t}$ is terminated at time $t + k$ and we need to select a new multi-action to execute in state s_{t+k} (with the continue-set $h_{t+k} = \{a_2, a_3, a_4\}$). The *continue* policy will select the best multi-action $\mathbf{a_{t+k}}$ that includes the primary actions $\{a_2, a_3, a_4\}$, since they did not terminate in state s_{t+k} (see Figure 3.3).

3.2 Theoretical Results

In this section, we present our theoretical results, establishing the correctness of the concurrent action model, and characterizing various coordination mechanisms that we presented in the previous section.

3.2.1 Model Correctness

First, we establish that CAMs are well defined. This asserts the correctness of the optimization problem expressed in terms of the value function equations (Equations 3.1 -

3.6). In (Rohanimanesh and Mahadevan, 2001) we showed that the model is well defined when the multi-actions are defined as options. Here, we prove it for the general case where the actions are modeled as the general SMDP actions, and hence the case for the options immediately follows from that.

According to the definition of CAM, the set of states and set of multi-actions are well defined. We only need to establish that the multi-action transition probabilities $\mathcal{P}^{\tau}(s, \mathbf{a}, s', k)$, and the multi-action reward function $\mathcal{R}_s^{\langle \mathbf{a}, \tau \rangle}$, for all $s, s' \in \mathcal{S}$, $\mathbf{a} \in \mathcal{S}_s$ based on various termination schemes $\tau \in \{T_{any}, T_{all}\}$ are well defined.

3.2.1.1 Correctness of the State-Prediction Model

We start by showing this for the multi-action transition probability model, for each termination scheme:

Case 1) T_{any} **termination scheme**: T_{any} termination scheme declares termination when any of the primary actions $a \in \mathbf{a}$ terminates. We can write this event as (assuming that $\tau = T_{any}$):

$$\mathcal{P}^{\tau}(s, \mathbf{a}, s', k) = \xi^{\tau}(s, \mathbf{a}, s', k)\,(1 - \prod_{a \in \mathbf{a}}(1 - \beta_a(s')) \tag{3.7}$$

where $\xi^{\tau}(s, \mathbf{a}, s', k)$ denotes the probability of taking the multi-action $\mathbf{a}$ in state s and after k steps *transitioning* (without termination) into state s'. The second term on the right hand side of the Equation 3.7 denotes the probability that at least one of the primary actions $a \in \mathbf{a}$ terminates in state s' (which is computed as one minus the probability that none of them terminates in state s'). Here we used $\beta_a(s')$ to represent the probability that the primary action $a \in \mathbf{a}$ terminates in state s' and it can be computed by marginalizing out the source state (i.e., s) and the number of steps to completion (i.e., k):

$$\beta_a(s') = \sum_{s \in \mathcal{S}} \sum_{k=1}^{\infty} \mathcal{P}(s, a, s', k) \tag{3.8}$$

note that $\mathcal{P}(s, a, s', k)$ is well defined and is independent of the termination scheme τ. Now we show that $\xi^{\tau}(s, \mathbf{a}, s', k)$ is also well defined. We can write a recursive expression for $\xi^{\tau}(s, \mathbf{a}, s', k)$ as follows:

$$\xi^{\tau}(s, \mathbf{a}, s', k) = \sum_{s_j \in \mathcal{S}} \xi^{\tau}(s, \mathbf{a}, s', k-1) \ (1 - \prod_{a \in \mathbf{a}} (1 - \beta_a(s_j)) \ \xi^{\tau}(s_j, \mathbf{a}, s', 1) \tag{3.9}$$

where the first term on the right hand side denotes the probability of initiating the multi-action $\mathbf{a}$ in state s and transitioning into an intermediate state s_j after $k-1$ steps (without termination) ; the second term denotes the probability that none of the actions $a \in \mathbf{a}$ terminates at state s_j; and the last term denotes the probability that the multi-action $\mathbf{a}$ is initiated in state s_j and executed for a single step, and ended up in state s'. Note that the last term (i.e., $\xi^{\tau}(s_j, \mathbf{a}, s', 1)$) is also independent of the termination scheme τ, since it terminates deterministically after one step, and thus it is well defined. The stopping criterion for the recursive expression in Equation 3.9 is when $k = 1$. Thus, we have shown that all the expressions are well defined and therefore $\mathcal{P}^{\tau}(s, \mathbf{a}, s', k)$ is well defined for $\tau = T_{any}$.

Case 2) T_{all} **termination scheme**: T_{all} termination scheme declares termination when all of the primary actions $a \in \mathbf{a}$ terminate. This case is a little more complex, since for any number of steps k, we have to consider every possible permutation of termination of primary actions $a \in \mathbf{a}$ within k steps. It can be computed based on the following recurrent equation (assuming that $\tau = T_{all}$):

$$\mathcal{P}^{\tau}(s, \mathbf{a}, s', k) = \sum_{i=1}^{k} \sum_{\mathbf{a}' \subseteq \mathbf{a}} \sum_{s_j \in \mathcal{S}} \xi^{\tau}(s, \mathbf{a}, s_j, i) \left[\prod_{a \in \mathbf{a}'} (1 - \beta_a(s_j) \right] \mathcal{P}^{\tau}(s, (\mathbf{a} - \mathbf{a}'), s', (k-i)) \tag{3.10}$$

The first summation runs over all intermediate steps $i \leq k$ at which none of the primary actions $a \in \mathbf{a}$ terminates. The second summation comprises every subset $\mathbf{a}'$ of the multi-action $\mathbf{a}$ that terminates at step i, and the third summation runs over every possible intermediate state in which the multi-action $\mathbf{a}$ terminates in $i \leq k$ steps. The first term inside the summation denotes the transition probability that the multi-action $\mathbf{a}$ initiated in state s transitions to the intermediate state s_j after i steps; the second term denotes the probability that some subset of the primary actions $a \in \mathbf{a}$ (i.e., $\mathbf{a}'$) terminates at the intermediate state s_j; and the third term denotes the $(k - i)$-steps transition probability of the rest of the primary actions (i.e., $(\mathbf{a} - \mathbf{a}')$) from state s to state s'.

In the proof of the $\tau = T_{any}$ case, we established that the first and second term of the right hand side of the Equation 3.10 are well defined. The stopping criterion for this recurrent expression is when the set $(\mathbf{a} - \mathbf{a}')$ contains only one element (i.e., one primary action). In that case $\mathcal{P}^\tau(s, \mathbf{a} = \{a\}, s', k - i)$ is the transition probability defined over a primary action, that is independent of the termination scheme $\tau = T_{all}$ and also well defined. In sum, we have shown that all the expressions are well defined and therefore $\mathcal{P}^\tau(s, \mathbf{a}, s', k)$ is well defined for $\tau = T_{all}$.

3.2.1.2 Correctness of the Reward Model

Correctness of the multi-step reward model 3.2 immediately follows the correctness of the transition model, since it establishes the fact that the distribution over next states given any termination scheme $\tau \in \{T_{any}, T_{all}\}$ is well defined, based on the Equation 3.2.

Theorem 3.1 (Concurrent Actions Model = SMDP): For any CAM, and any set of *concurrent Markov actions* defined on that CAM, the decision process that selects only among multi-actions, and executes each one until its termination according to the multi-option termination condition, forms a semi-Markov decision process.

Proof: For a decision process to be a SMDP, it is required to define (1) set of states, (2) set of actions, (3) an expected cumulative discounted reward defined for every pair of state and action and (4) a well defined joint distribution of the next state and next decision epoch.

In the concurrent action model, we have defined the set of states and, the set of actions are multi-actions. The expected cumulative discounted reward and joint distributions of the next state and next decision epoch have been defined in terms of the underlying MDP. The policy and termination condition for every option that belongs to a multi-action, and the termination condition for a multi-action have also been defined.

3.2.2 Optimality and the Coordination Mechanisms in CAMs

In this section we present our theoretical results (Rohanimanesh and Mahadevan, 2002) comparing the optimality of various policies under different termination schemes introduced in the previous section. In all of these theorems we use the partial ordering relation $\mathcal{V}^{\pi_1} \leq \mathcal{V}^{\pi_2} \leftrightarrow \pi_1 \leq \pi_2$ (Sutton and Barto, 1998), in order to compare different policies. Note that in Theorems 3.2 and 3.4, which compare the *continue* policy with $\pi^{*_{any}}$ and $\pi^{*_{all}}$ policies, the value function is written over the pair $\langle s_t, h_t \rangle$ to be consistent with the definition of the *continue* policy (i.e., π_{cont}). This does not influence the original definition of the value function for the optimal policies in T_{any} and T_{all} termination schemes, since they are independent of the continue-set h_t. First, we compare the optimal multi-action policies based on the T_{any} termination scheme and the π_{cont} policy.

Theorem 3.2: For every state $s_t \in S$, and all continue-set $h_t \in \mathcal{H}$,

$$\mathcal{V}^{\pi_{cont}}(\langle s_t, h_t \rangle) \leq \mathcal{V}^{*_{any}}(\langle s_t, h_t \rangle).$$

Proof: By writing the value function definition for each case we have:

$$\begin{aligned}
\mathcal{V}^{\pi_{cont}}(\langle s_t, h_t \rangle) &= \max_{\mathbf{a} \in A(s_t, h_t)} Q^{\pi_{cont}}(\langle s_t, h_t \rangle, \mathbf{a}) \\
&\leq \max_{\mathbf{a} \in A(s_t)} Q^{\pi_{cont}}(\langle s_t, h_t \rangle, \mathbf{a}) \\
&\leq \max_{\mathbf{a} \in A(s_t)} Q^{*_{any}}(\langle s_t, h_t \rangle, \mathbf{a}) \\
&= \mathcal{V}^{*_{any}}(\langle s_t, h_t \rangle)
\end{aligned}$$

The inequality holds since the maximization in π_{cont} is over a smaller set (i.e., $A(s_t, h_t)$) which is a subset of the larger set $A(s_t)$ that is maximized over, in the $\pi^{*_{any}}$ case.

Next, we show that the optimal policy with multi-actions that terminate according to the T_{any} termination scheme are better compared to the optimal policy with multi-actions that terminate according to the T_{all} termination scheme:

Theorem 3.3: For every state $s \in S$, $\mathcal{V}^{*_{all}}(s) \leq \mathcal{V}^{*_{any}}(s)$.

Proof: The proof is based on the following lemma which states that if we alter the execution of the optimal multi-action policy based on T_{all} (i.e., $\pi^{*_{all}}$) in such a way that at every decision epoch the next multi-action is still selected from $\pi^{*_{all}}$, but we terminate it based on T_{any} then the new policy-termination construct represented by $\langle *_{all}, any\rangle$ is better than the $\pi^{*_{all}}$ policy. Intuitively this makes sense, since if we interrupt $\pi^{*_{all}}(s)$ when the first primary action $a_i \in \mathbf{a} = \pi^{*_{all}}(s)$ terminates in some future state s', due to the optimality of $\pi^{*_{all}}$, executing $\pi^{*_{all}}(s')$ is always better than or equal to continuing some other policy such as the one in progress (i.e., $\pi^{*_{all}}(s)$). Note that the proof is not as simple as in the first theorem since the two different policies discussed in this theorem (i.e., $\pi^{*_{any}}$ and $\pi^{*_{all}}$) are not being executed using the same termination method.

Let $\mathcal{V}^{*_{all}}_{any}(s)$ is the value of executing the policy $\pi^{*_{all}}$ in state s using the termination mechanism T_{any} at every step:

Lemma 1: For every state $s \in S$, $\mathcal{V}^{*_{all}}(s) \leq \mathcal{V}^{*_{all}}_{any}(s)$.

Proof: Let $\mathcal{V}^{*_{all}}_{n,any}(s)$ denote the value of following the optimal $\pi^{*_{all}}$ policy in state s, where for the first n decision epochs we use the T_{any} termination scheme and for the rest we use the T_{all} termination scheme. By induction on n, we can show that $\mathcal{V}^{*_{all}}(s) \leq \mathcal{V}^{*_{all}}_{n,any}(s), \forall s \in S$ and for all n. We use induction to prove Lemma 1:

1. **Induction hypothesis:**

$$\mathcal{V}^{*_{all}}(s) \leq \mathcal{V}^{*_{all}}_{n,any}(s), \; \forall s \in S \tag{3.11}$$

2. **Induction base:** Let the primary action a_1 be the first primary action in the multi-action a^*_t that terminates at some time $t+k$, when the multi-action a^*_t is initiated in state s_t. We can rewrite $\mathcal{V}^{*_{all}}(s_t)$ based on the termination of a_1 as follows:

$$\begin{aligned}\mathcal{V}^{*_{all}}(s) &= E\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{k-1} r_{t+k} + \gamma^k \mathcal{Q}^{*_{all}}(s_{t+k}, \omega_{t+k})\} \\ &\leq E\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{k-1} r_{t+k} + \gamma^k Q^{*_{all}}(s, a^*_{t+k})\} \\ &= \mathcal{V}^{*_{all}}_{1,any}\end{aligned} \tag{3.12}$$

   where $\omega_{t+k} = a^*_t - a_1$. The inequality in equation 3.12 holds, because $a^*_{t+k} = \pi^{*_{all}}(s_{t+k})$ is the optimal multi-action in state s_{t+k} and thus $Q^{*_{all}}(s_{t+k}, \omega_{t+k}) \leq \mathcal{Q}^{*_{all}}(s_{t+k}, a^*_{t+k})$.

3. **Induction step:** We need to show that if:

$$\mathcal{V}^{*_{all}}(s) \leq \mathcal{V}^{*_{all}}_{n,any}(s), \; \forall s \in S \tag{3.13}$$

   then:

$$\mathcal{V}^{*_{all}}(s) \leq \mathcal{V}^{*_{all}}_{n+1,any}(s), \; \forall s \in S \tag{3.14}$$

   we first show that:

$$\mathcal{V}^{*_{all}}_{n,any}(s) \leq \mathcal{V}^{*_{all}}_{n+1,any}(s), \; \forall s \in S \tag{3.15}$$

and then the induction hypothesis will immediately follow from that. To show the Equation 3.15 let R^n denote the partial return of following the policy $\pi^{*_{all}}(s)$ and for the first

n times terminating it according to the T_{any} termination event until step n, and also let k_n denotes the total number of primitive steps until n_{th} termination event:

$$\begin{aligned}
\mathcal{V}^{*_{all}}_{n,any}(s) &= E\{R^n + \gamma^{k_n}\mathcal{V}^{*_{all}}(s_{t+k_n})\} \\
&\leq E\{R^n + \gamma^{k_n}\mathcal{V}^{*_{all}}_{1,any}(s_{t+k_n})\} \\
&= E\{R^{n+1} + \gamma^{k_{n+1}}\mathcal{V}^{*_{all}}(s_{t+k_n+1})\} \\
&= \mathcal{V}^{*_{all}}_{n+1,any}(s)
\end{aligned}$$

where the inequality is based on the induction base. Using the induction hypothesis and Equation 3.16 we obtain:

$$\begin{aligned}
\mathcal{V}^{*_{all}}(s) &\leq \mathcal{V}^{*_{all}}_{n,any}(s) \\
&\leq \mathcal{V}^{*_{all}}_{n+1,any}(s)
\end{aligned}$$

which completes the proof for the Lemma 1. Using Lemma 1 we obtain:

$$\mathcal{V}^{*_{all}}(s) \leq \lim_{n\to\infty} \mathcal{V}^{*_{all}}_{n,any}(s) = \mathcal{V}^{*_{all}}_{any}(s) \tag{3.16}$$

Based on the optimality of the policy π^*_{any} in the space of policies with the termination mechanism T_{any}, and the results of the Equation 3.16, it follows:

$$\mathcal{V}^{*_{all}}(s) \leq \mathcal{V}^{*_{all}}_{any}(s) \leq \mathcal{V}^{*_{any}}(s) \tag{3.17}$$

which completes the proof for Theorem 3.3.

In the next theorem, we show that if we execute the π_{cont} policy in which at any decision epoch we always execute the best set of primary actions along with those ones that were executed in the previous decision epoch and have not terminated yet, we achieve a better return compared to the case in which we execute the best set of primary actions, but always wait until all of the primary actions terminate before making a new decision:

Theorem 3.4: For every state $s_t \in S$:

$$\mathcal{V}^{*_{all}}(s_t) \leq \mathcal{V}^{*_{cont}}(s_t)$$

Proof: When executing $\pi^{*_{all}}$ policies, multi-actions are executed until all of the primary actions of that multi-action terminate. The $\pi^{*_{cont}}$ policy, however, may also initiate new primary actions whenever the current multi-action terminates according to the T_{any} termination mechanism. First we show that for every $\pi^{*_{all}}$ policy, there exist an equivalent π_{cont} policy. First we introduce a notation that will be used in the rest of the proof. Assume that we are executing the policy $\pi^{*_{all}}$ in some state s_t at time t. Executing the multi-action $a^*_t = \pi^{*_{all}}(s_t)$ continues until all of the primary actions in a^*_t terminate. Whenever a primary action terminates in some state s_{t+k}, the rest of the primary actions continue execution until all of them are terminated. Let $\omega_{t+k} = a^*_t - h_{t+k}$ represent the set of primary actions that did not terminate in state s_{t+k} at time $t + k$. The new multi-action ω_t will be executed in that state until all primary actions are terminated until execution (i.e., when $\omega_{t+k} = \emptyset$, for some k). This is equivalent to a policy π_{cont} that uses the T_{any} termination mechanism, and only let those primary actions that did not terminate to continue running (without initiating new primary actions). Thus we have $\mathcal{V}^{*_{all}}(s) = \mathcal{V}^{\pi_{cont}}(s)$. But we know that the policy $\pi^{*_{cont}}$ is the optimal policy among *continue* space of policies, thus:

$$\mathcal{V}^{*_{all}}(s) = \mathcal{V}^{\pi_{cont}}(s) \leq \mathcal{V}^{*_{cont}}(s)$$

which completes the proof.

Finally we show that the optimal multi-action policies based on T_{all} termination scheme are as good as the case where the agent always executes a single primary action at a time, as it is the case in standard SMDPs. Note that this theorem does not state that concurrent plans are always better than sequential ones; it simply says that if in a problem, the sequential execution of the primary actions is the best policy, CAM is able to represent and find that policy. Let $\pi^{*_{seq}}$ represent the optimal policy in the sequential case, where only one primary action can be executed at a time:

Theorem 3.5: For every state $s \in S$, $\mathcal{V}^{*_{seq}}(s) \leq \mathcal{V}^{*_{all}}(s)$, in which $\mathcal{V}^{*_{seq}}(s)$ is the value of the optimal policy when the primary actions are executed one at a time sequentially.

Proof: It suffices to show that sequential policies are within the space of concurrent policies. This holds since a single primary action can be considered as a multi-action containing only one primary action whose termination is consistent with either of the multi-action termination schemes (i.e., in the sequential case both T_{any} and T_{all} termination schemes are same).
Corollary 1 summarizes our theoretical results. It shows how different policies in a concurrent action model using different termination schemes compare to each other in terms of optimality.

Corollary 1: In a concurrent action model and a set of termination schemes

$$\{T_{any}, T_{all}, T_{cont}\}$$

the following partial ordering holds among the optimal policy based on T_{any}, the optimal policy based on T_{all}, the *continue* policy (i.e., π_{cont}), and the optimal sequential policy:

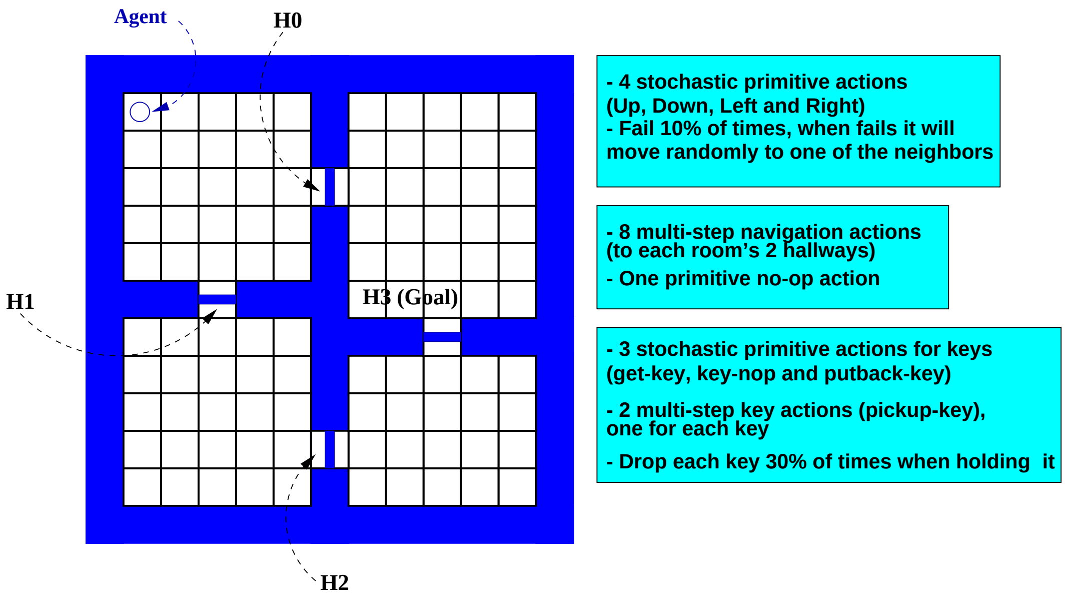
$$\pi^{*seq} \leq \pi^{*all} \leq \pi^{*cont} \leq \pi^{*any} \tag{3.18}$$

Proof: This follows immediately from the above theorems.

3.3 Experiments

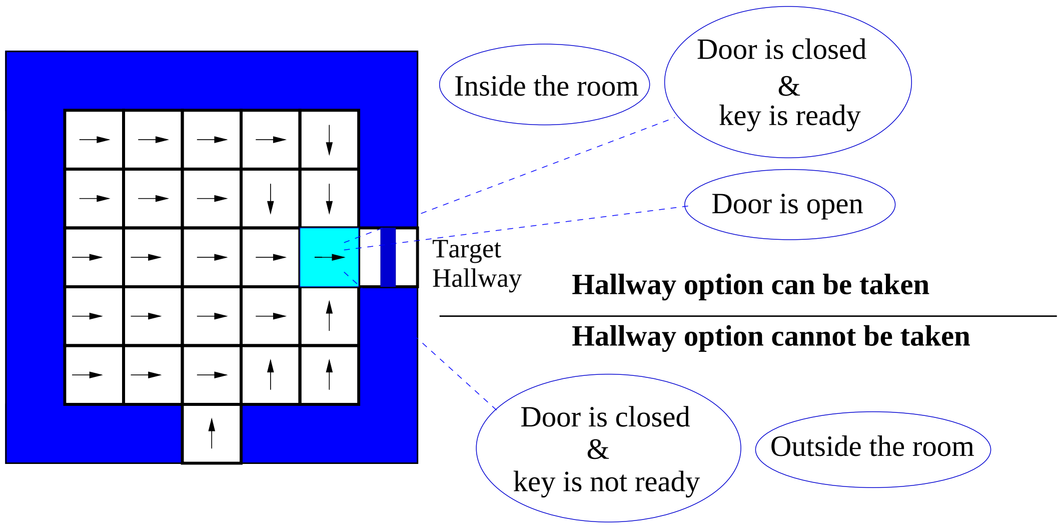
In this section, we present experimental results using a grid world task comparing various termination schemes (see Figure 3.4). Each hallway connects two rooms, and has a door with two locks. An agent has to retrieve two keys and hold both keys at the same time in order to open both locks. The process of picking up keys is modeled as a temporally extended action that takes different amount of times for each key. Moreover, keys cannot be held indefinitely, since the agent may drop a key occasionally. Therefore the agent needs to find an efficient solution for picking up the keys in parallel with navigation to act optimally. This is an episodic task, in which at the beginning of each episode the agent is placed in a fixed position (upper left corner) and the goal of the agent is to navigate to a fixed position goal (hallway H3).

The agent can execute two types of action concurrently: (1) *navigation* actions, and (2) *key* actions. Navigation actions include a set of one-step stochastic navigation actions (Up, Left, Down and Right) that move the agent in the corresponding direction with probability 0.9 and fail with probability 0.1. Upon failure the agent moves instead in one of the other three directions, each with probability $\frac{1}{30}$. There is also a set of temporally extended actions defined over the one step navigation actions that transport the agent from within the room to one of the two hallway cells leading out of the room (Figure 3.5). Key actions are defined to manipulate each key (get-key, putback-key, pickup-key, etc). Among them *pickup-key* is a temporally extended action (Figure 3.6). Note that each key has its own set of actions.



Agent
H0
H1
H2
H3 (Goal)
- 4 stochastic primitive actions (Up, Down, Left and Right)
- Fail 10% of times, when fails it will move randomly to one of the neighbors
- 8 multi-step navigation actions (to each room's 2 hallways)
- One primitive no-op action
- 3 stochastic primitive actions for keys (get-key, key-nop and putback-key)
- 2 multi-step key actions (pickup-key), one for each key
- Drop each key 30% of times when holding it

Figure 3.4. A navigation problem that requires concurrent plans. There are two locks on each door, which need to be opened simultaneously. Retrieving each key takes different amounts of time.



Inside the room
Door is closed & key is ready
Door is open
Target Hallway
Hallway option can be taken
Hallway option cannot be taken
Door is closed & key is not ready
Outside the room

Figure 3.5. The policy associated with one of the hallway temporally extended actions.

In this example, navigation actions can be executed concurrently with key actions. Actions that manipulate different keys can be also executed concurrently. However, the agent is not allowed to execute more than one navigation action, or more than one key action (from the same key action set) concurrently. In order to properly handle concurrent execution of actions, we have used a factored state space defined by state variables *position* (104 positions), *key1-state* (11 states) and *key2-state* (7 states). In our previous work we

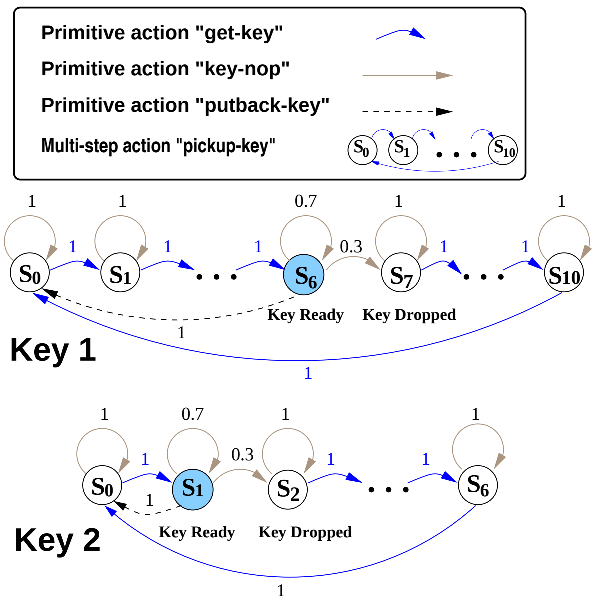

Primitive action "get-key"
Primitive action "key-nop"
Primitive action "putback-key"
Multi-step action "pickup-key"
Key Ready
Key Dropped
Key 1
Key Ready
Key Dropped
Key 2

Figure 3.6. Representation of the key pickup actions for each key process.

showed that concurrent actions formed an SMDP over primitive actions (Rohanimanesh and Mahadevan, 2001), which turns out to hold for all the termination schemes described above. Thus, we can use SMDP Q-learning (Bradtke and Duff, 1995) to compare concurrent policies over different termination schemes with the use of this method for purely sequential policy learning (Sutton et al., 1999). After each decision epoch where the multi-action $\mathbf{a}$ is taken in some state s and terminates in state s', the following update rule is used: $Q(s, \mathbf{a}) \leftarrow Q(s, \mathbf{a}) + \alpha \left[r + \gamma^k \max_{\mathbf{a}' \in A(s')} Q(s', \mathbf{a}') - Q(s, \mathbf{a}) \right]$, where k denotes the number of time steps since initiation of the multi-action $\mathbf{a}$ at state s and its termination at state s', and r denotes the cumulative discounted reward over this period. The agent is punished by -1 for each primitive action.

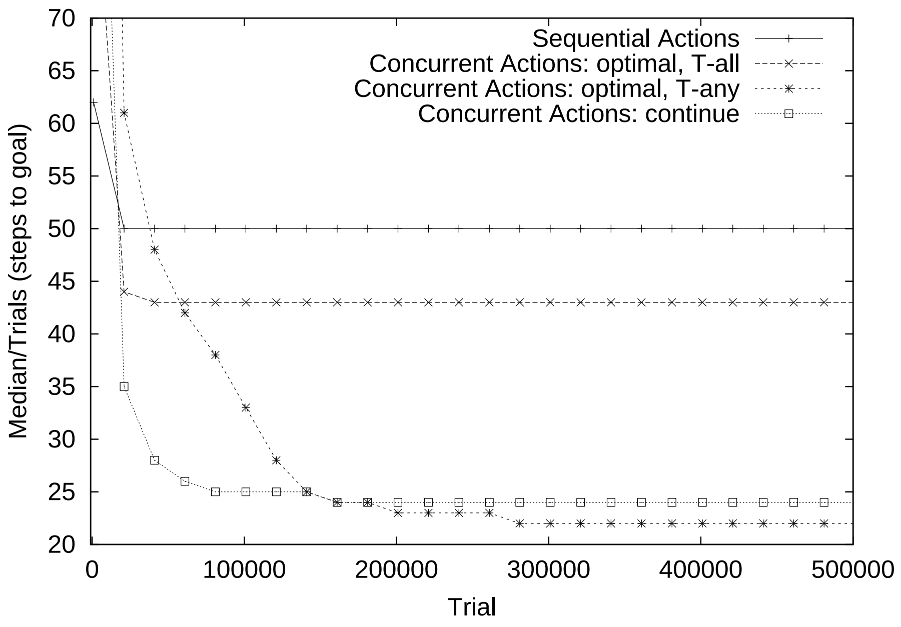


Figure 3.7. Moving median of number of steps to the goal.

Figure 3.7 compares the number of primitive actions taken until success, and Figure 3.8 shows the median number of decision epochs per trial, where for trial n, it is the median of all trials from 1 to n. These data are averaged over 10 episodes, each consisting of $500,000$ trials.

As shown in Figure 3.8, concurrent actions over any termination scheme yield a faster plan than sequential execution. Moreover, the policies learned based on T_{any} (i.e. both $\pi^{*_{any}}$ and π_{cont}) are also faster than T_{all} . Also, $\pi^{*_{any}}$ achieves higher optimality than π_{cont}, however the difference is small.

We conjecture that sequential execution and T_{all} converge faster compared to T_{any}, due to the frequency with which multi-actions are terminated. As shown in Figure 3.8, T_{all} makes fewer decisions, compared to T_{any}. This is intuitive since T_{all} terminates only when all of the primary actions in a multi-action are completed, and hence it involves less interruption compared to learning based on T_{any}. Note π_{cont} converges faster than $\pi^{*_{any}}$ and it is nearly as good as T_{any}. . We can think of π_{cont} as a blend of T_{all} and T_{any} . Even

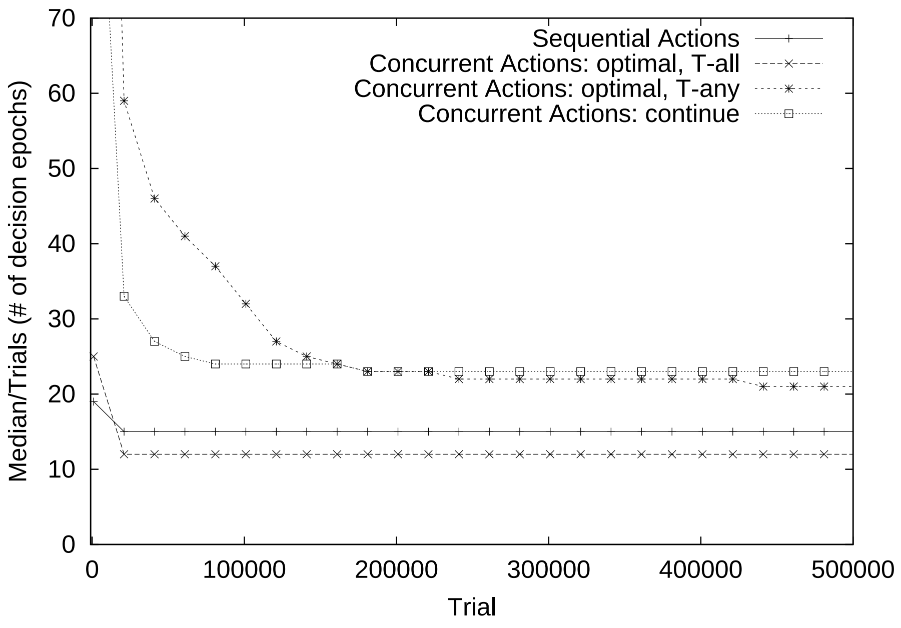


Figure 3.8. Moving median of number of multi-action level decision epochs taken to the goal.

though it uses the T_{any} termination scheme, it continues executing primary actions that did not terminate naturally when the first primary action terminates, making it similar to T_{all} .

3.4 Concluding Remarks

In this chapter we introduced a a general framework for modeling the concurrent decision making problem based on semi-Markov decision processes. The agent is given a set of primary actions that can be parallelized parallel (consisting of primitive actions, and also actions temporally extended actions). A multi-action is a subset of primary actions that the agent executes concurrently. We introduced different concurrent action coordination mechanisms to determine the termination of a multi-action.

We theoretically established that the model is well defined and is in fact a generalization of SMDPs for performing concurrent activities. We also presented theoretical results characterizing the optimality of the agent's behavior based on various coordination mechanisms. Our experiments back up the set of theoretical results that we presented throughout this chapter. However, as it can be observed from the experimental results, even in a simple grid world example that contains few hundred states and a few concurrent actions, the standard SMDP learning and planning methods take hundreds of thousands of step in order to learn the optimal behavior.

Although in our experiments we did not employ more powerful RL learning algorithms (such as TD methods (Sutton, 1988; Boyan, 1999) augmented with function approximation techniques (Bertsekas and Tsitsiklis, 1997; Sutton and Barto, 1998)), the computational complexity of such algorithms theoretically remain intractable as the system admits more degrees of concurrency. In the following chapters, we introduce a set of techniques in order to alleviate this problem.

CHAPTER 4

COARTICULATION: AN APPROACH FOR SOLVING THE CONCURRENT DECISION MAKING PROBLEM

In Chapter 3 we introduced a general framework for modeling the concurrent decision making problem. Benefiting from the abstractness of the model, we were able to characterize this class of problems in terms of a set of general theoretical results. In particular we studied various concurrent coordination mechanisms and theoretically demonstrated how such mechanisms partition the space of policies over concurrent actions into a set of equivalent classes of policies, each associated with a different degree of optimality. Thanks to Theorem 3.1 – where we proved CAMs are essentially SMDPs – we can now model any concurrent decision making problem theoretically as a CAM and use standard SMDP learning and planning methods to solve it.

Unfortunately such standard solutions do not scale to many concurrent decision making problems, when they require large numbers of activities to be executed concurrently. Even in a simple simulated domain with few hundred states and few concurrent actions, it can take hundreds of thousands of iterations for the standard SMDP learning and planning methods to converge to the optimal policy (see Figure 3.7 in Chapter 3). In order to alleviate this problem, we first need to take a step back and study the specific properties of concurrent decision making which make it theoretically difficult to scale to larger domains. We conjecture there are several reasons that CAMs cannot cope with large concurrent decision making problems:

• The *curse of dimensionality* incurred by the combinatorial space of concurrent activities. In general, the agent can execute any subset of its acquired skills concurrently, which renders the space of the concurrent actions exponential in the set of skills available to the agent. It is known that the complexity of planning in MDPs and the complexity of the near optimal reinforcement learning methods are polynomial in the set of states and actions (Papadimitriou and Tsitsiklis, 1987; Kearns and Singh, 1998) and thus such algorithms do not scale as the size of the problem exponentially grows in terms of both state and action spaces.

• CAMs ignore the exploitable reward structure that is present in many concurrent decision making problems. Many such problems can be viewed as a multi-objective optimization problem. In general it is intuitive to approximate the overall goal of such problems in terms of a function decomposable in the set of subgoals, each with a different level of priority. This view has also a long history in *multi-attribute utility theory* (Keeney and Raiffa, 1993), and also in multi-objective control (Nakamura, 1991; Grupen, 2006) in robotics. It has also been explored in the context of *multi-criteria reinforcement learning* (MRL) (Gabor et al., 1998) where the optimization criteria are expressed via a vector of reward signals.

Interestingly, this property can also be observed from the empirical results that we presented in Chapter 3. The experimental results show that by overlaying the decision epochs on activity completion events (for example, in T_{any} termination mechanism the decision epochs are points in time at which any activity terminates, or in T_{all} termination mechanism the decision epochs are points in time at which all activities in progress terminate), performance significantly improves. The experimental results with $T_{continue}$ also support the fact that activity completion is a prominent factor in the overall performance of the agent. This suggests that the system can be intuitively viewed as a multi-objective optimization problem, in which each activity optimizes a subgoal of some sort in the problem.

• In many problems (in particular multi-objective optimization problems) there is no simple way to express the optimization objective as a function of a single scalar reinforcement signal (Gabor et al., 1998). This can also be observed in many concurrent decision making problems. For example in a driving task, we perform various concurrent activities, such as safely navigating the car, drinking coffee, and so on (this example is further elaborated in Section 4.2). In general, it is not known how to design a single scalar reward function that captures various aspects of this type of optimization problem. Here, we do not argue against scalar-valued representation for the reward signal in this class of problems. In fact the agent may have an intrinsically motivated reward representation (Singh et al., 2004) in such problems. However, discovering the details of such a representation remains as a challenge in biological and artificial agents.

• From a task-independent agent-environment interaction perspective, in the life span of an agent the subgoals are not known a priori. As the agent continues to operate in its environment, new subgoals are introduced dynamically. In some cases, we may not be aware of the existence of some subgoals which may be introduced later in the problem. For example, an infant learns how to grasp an object in the absence of many other constraints that later on may actually influence the performance of that particular skill. Yet he is able to employ almost the same skill as he ages and gains more awareness about many different conditions in his surrounding environment. It is unrealistic and impractical to either form a general learning problem that takes into account every possible combination of the present/future subgoals, or to define a new learning problem for every combination of subgoals without caching out the previously acquired skills in some way. In general we tend to *reuse* our previously learned skills (Singh et al., 2004) and intelligently modify them when we are faced with a new task.

Based on these intuitions, in the rest of this chapter we introduce an approach (Rohanimanesh et al., 2004a,b) to cope with some of the issues that we discussed above. In our approach, we pose the concurrent decision making problem as an multi-objective optimization in which the overall objective is expressed in terms of a set of prioritized subgoals of the problem. We will demonstrate that concurrency is then naturally generated when the skills for achieving the individual subgoals interact and compete for limited/redundant set of resources in the system.

Abusing the terminology, by coarticulation we refer to a general class of the problems in which an agent simultaneously commits to multiple objectives, inspired by the phenomenon of coarticulation in speech and motor control research. The key idea in our approach is that in many goal-oriented activities – in addition to the optimal policy – often there exists a redundant set of policies that guarantee progressing toward the goal state, with a cost of a slight deviation from optimality. Such flexibility enables the agent to select a policy that simultaneously progresses toward multiple subgoals of the problem.

We argue that coarticulation is a natural way for generating concurrency for several reasons. First, the overall objective in many concurrent decision making problems can actually be viewed as concurrent optimization of a set of prioritized subgoals. Second, because of the multiplicity of DOFs in the system, learned activities/skills offer more flexibility in terms of the range of redundant goal-progressing policies associated with them. For example in driving, while the best policy for safely navigating the car would be to control the wheels using both arms, by exploiting the extra DOFs in our body we can perform the same task near-optimally by engaging one arm for controlling the wheels and releasing the other arm for committing to other subordinate subgoals such as drinking coffee. However, the key advantage of coarticulation in concurrent decision making lies in its efficient search within the exponential space of concurrent actions. The action selection mechanism in this approach (described later in this chapter) is restricted to those that progress toward the subgoals associated with each activity. This interactive search enables the agent to perform the search in a much smaller sub-space of concurrent actions with a controllable cost in optimality.

Figure 4.1 shows an abstract view of the coarticulation as a means for generating parallel execution plans. When the system admits a redundant set of goal-progressing policies for every subgoal of the problem, by exploiting the multiplicity of DOFs, coarticulation leads to concurrency in the system in general.

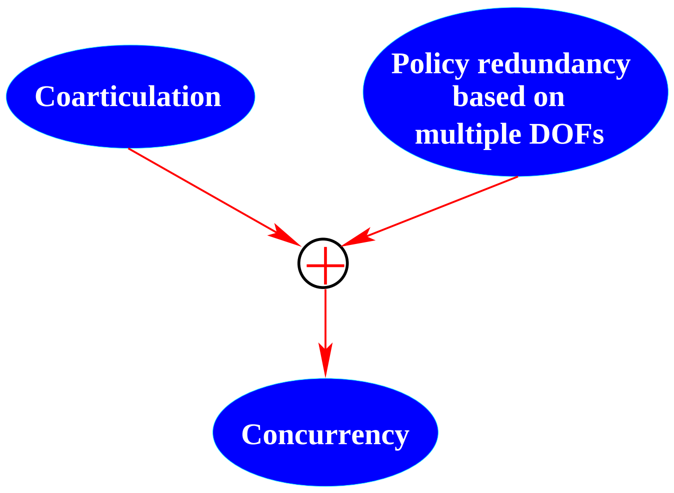


Figure 4.1. Coarticulation among a set of subgoals, for which there exists a redundant set of goal-progressing policies due to the multiple degrees of freedom in the system, leads to concurrency.

The rest of this chapter is organized as follows: In Section 4.1 we briefly overview the coarticulation framework as studied in speech and motor control communities. In Section 4.2 we formally describe the coarticulation problem in MDPs. Section 4.3 we describe redundant controllers as the building blocks in our coarticulation model. Sections 4.4 and 4.5 describe the sequential and coarticulation algorithms. In section 4.6 we present a set of theoretical results characterizing the performance of the coarticulation approach in comparison with the performance of an agent that achieves the subgoals with a strong sequential constraint. Finally we present our empirical results in a simulated domain in section 4.7.

4.1 Coarticulation: An Overview

Divide an conquer is one of the most effective strategies of solving problems in intelligent systems (Russell and Norvig, 2002). We often solve a problem by decomposing it into a sequence of subtasks and then we generate the overall solution by combining the local solutions to each subtask. In general how we solve each subtask is highly context sensitive and is influenced prominently by the set of past and future subtasks.

Coarticulation is best understood in the context of speech synthesis. A standard method of analyzing human speech is to divide it into its constituent linguistic elements, which are the basic blocks of speech called *phonemes*. Speech words and phrases are produced by concatenating phonemes. Synthesizing speech from individual phonemes, however, can result in discontinuous and sometimes unintelligible sounds. The transitions between phonemes that occur naturally during speech production are often missing in synthesized speech. The effect of these transitions is known as the coarticulation phenomenon, the overlap of an articulation with its neighbor (Kent and Minifie, 1977; Abbs et al., 1984).

Coarticulation phenomenon can be also observed in motor control. Complex movements are generally thought to consist of a series of simpler elements. It can be argued that a coarticulation mechanism enables the sensorimotor system to assemble the movement segments to produce a smooth motion. Examples of this form come from studies of the arm configuration for a three dimensional target tracking task (Breteler et al., 2003) where the human subjects are to perform arm movements to various targets placed in three-dimensional (3D) space. Each task consists of single, double, or triple segments in which the first segment was the same across the tasks but with different second targets. It was observed that the final arm posture consistently depended on which particular movement was to follow as the second segment. This provided evidence for coarticulation of the two segments at the level of arm posture.

Jordan (1990) also developed a model for motor learning with an emphasis on problems involving excess degrees of freedom. The model consists of an internal predictive model (referred to as a *forward model*) and a set of intrinsic constraints such as *smoothness, distinctiveness* and *rest configuration,* on motor learning. Experiments on a manipulator with six degrees of freedom showed that by using the smoothness constraint, the model uses the excess degrees of freedom to anticipate and manifest other actions.

Other studies of coarticulation in human motor control include recruitments of finger patters when playing piano (Engel et al., 1997), playing violin (Baader et al., 2005), or sequential typing (Soechting and Flanders, 1992). In all of these studies evidence of coarticulation can be observed where the patterns of finger recruitments consistently depend on the previous and subsequent notes, or letters in the sequence.

In all of the above examples we can observe a strict order of sequentiality among the basic blocks of the control systems (phonemes, or basic arm movements). For example in speech synthesis phonemes cannot completely overlap, or in piano playing, non-overlapping notes should be played according to the strict ordering presented in the music sheet. However the control mechanism for generating each basic elements can preshape or slightly overlap. This phenomenon can be more clearly observed in a class of problems that allow for complete overlap of control actions. In such problems the strict sequentiality constraint is relaxed to a *partial ordering* constraint, allowing two or more actions be performed concurrently. One clear example of this form of coarticulation is known as *pre-shaping* for *prehension* (Jeannerod, 1981; Hoff and Arbib, 1993). When learning to reach for grasping an object, the subject learns to open his hand while moving his arm toward the target. A similar phenomenon is observed in control systems with excess degrees of freedom (DOF), such as bi-manual coordination (Wiesendanger and Serrien, 2001). In such problems we use one hand to perform one task, and use the other to commit to the next subtask (e.g., reaching and opening a drawer with one hand, while taking out an object from the drawer with the other hand).

In reaching and grasping example, other forms of of coarticulation could also be incorporated, for example the way we position our hand for grasping a tool, may depend on the next subtask, or how we plan to use it. This form of coarticulation is known as *prospective coding*, or *anticipatory activity* studied in (Johnson and Grafton, 2003; Cohen and Rosenbaum, 2004).

4.2 Coarticulation and Concurrent Decision Making

In the previous section we briefly overviewed various forms of coarticulation. Of particular interest is the form of coarticulation with the relaxed assumptions on the strict order of sequentiality of the control actions, which would enable the agent to simultaneously commit to multiple subgoals of the problem. In addition if the system admits multiple DOFs, coarticulation can serve as a basis for generating concurrency in the system.

In this section, we develop a decision theoretic framework by extending CAMs for studying this form of coarticulation. To motivate our approach, consider the life span of an agent. The agent is constantly performing learning and planning based on the set of current and future tasks introduced in the system. The agent also constantly acquires new skills from each learning experience (Iba, 1988; McGovern and Barto, 2001; Pickett and Barto, 2002; Şimşek and Barto, 2004). Even when the agent is not faced with a specific task, it may continue acquiring useful task-independent skills by exploring the environment (Şimşek et al., 2005; Singh et al., 2004). Such skills are cached in and later are reused by the agent when it is faced with a new task.

We can think of such acquired skills as the basic blocks of the control system (same as phonemes in speech synthesis, or reaching to grasp skill in human) for performing coarticulation. Given an instance of a new problem – which can be expressed approximately in terms of the concurrent optimization of a set of subgoals – we coarticulate among the subgoals and efficiently modify the skills that we have learned for achieving them. As a result we produce a more natural course of action when solving a problem. This is illustrated further in the following examples:

Example 4.1 – Drinking Coffee and Running[1]: We have developed skills, such as drinking coffee that involves holding the coffee cup, and drinking without spilling the coffee on the ground or burning ourselves. We have also learned how to run, possibly in the

[1]This example was brought up during a conversation with Richard Sutton.

absence of many other subtasks such as drinking coffee. Yet we are able to perform both tasks simultaneously, and reasonably well [2].

Example 4.2 – Driving: Consider a driving task where we have acquired skills such as safely navigating the car, drinking coffee, talking on cell phone, etc. We can perform such tasks in parallel without impacting the overall performance too severely. We can observe a form of coarticulation in this example that exploits the many DOFs in our bodies to generate concurrency. Due to multiplicity of DOFs in our body, each subgoal can be achieved in many different ways. For example we can control the wheels optimally by both hands, or near optimally by one hand. In some cases we may totally disengage our hands from the wheels for a short amount of time in order to put on our overcoat if the road is clear and safe. Such redundancy in the space of goal progressing policies enables us to commit to other subgoals. For example, if suddenly we feel like having coffee, by coarticulating between the subgoal of navigating the car and the subgoal of drinking coffee, we may choose to release one hand from steering wheels and use it for holding a cup of coffee. Note that in this case the generated policy simultaneously makes progress toward both subgoals, although it may not behave optimally with regard to an individual subgoal (for example controlling the wheels by one hand is not as safe as using both hands).

Based on these intuitions, we introduce a framework based on CAMs for modeling coarticulation in concurrent decision making problem. Given a concurrent decision making problem modeled by a CAM, we introduce *COART* as a tuple[3]:

---

[2]Note that although in this dissertation we have not explicitly addressed learning based on coarticulation, our framework does not rule out the learning process. We believe that learning is an ongoing process and inseparable from the life-span of an agent. For example even though by coarticulation we can quickly find a reasonable initial policy when performing running and drinking coffee the performance initially may not be close to the optimal behavior. From there, we continue learning until we perform this task optimally. In Chapter 6, we present a set of ideas for exploring this problem for future investigation.

[3]This model also captures the form of coarticulation when the agent executes non-concurrent actions. This is based on the fact that a sequential model can be represented as a CAM with multi-actions of cardinality one.

$$\text{COART} \triangleq \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \Omega, \Delta, \bar{\mathcal{R}}, \mathcal{T}, \Upsilon, \triangleleft \rangle$$

where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{T}$ are the same as in the base CAM (refer to Chapter 3). The rest of the components of the model are as follows:

• $\Omega = \{\omega_1, \omega_2, \ldots, \omega_m\}$ are a set of subgoals, each representing a minimum *cost-to-goal* problem. Recall that we assumed that the overall objective in a CAM can be approximated in terms of concurrent optimization of a set of subgoals. For theoretical reasons, in this dissertation we focus on minimum cost-to-goal subgoals, as they also represent a large class of optimization problems.

• $\Delta = \{\delta_1, \delta_2, \ldots, \delta_m\}$ are a set of *stimuli* signals indicating what subgoals are presently active. Stimulus δ_i, for example, indicates whether or not the subgoal ω_i is currently introduced in the system (in driving task, δ_{coffee} is an indicator of whether the driver opts to drink coffee, and δ_{cell} is an indicator of a present call received on the phone, for example). Note that some stimuli signals are external to the agent and are controlled by the environment (such as a phone call).

• $\tilde{\mathcal{R}} = [r_1, r_2, \ldots, r_m]^T \in \mathbb{R}^m$ is a vectored-valued reward signal, where r_i is a reward function optimizing the subgoal ω_i of the problem. We deliberately use the notation $\tilde{\mathcal{R}}$ to emphasize that it expresses an approximation of the reward signal $\mathcal{R}$ in the original CAM. Note that since each subgoal ω_i models a minimum cost-to-goal problem, we have:

$$\forall i,\ 1 \leq i \leq m,\ r_i < 0$$

• $\Upsilon = [\epsilon_1, \epsilon_2, \ldots, \epsilon_m]^T$ is a vector of *flexibility* parameters. A flexibility parameter $0 < \epsilon_i \leq 1$ gives a measure of admissible near-optimality with respect to the subgoal ω_i (we will give a more precise description of these parameters when we introduce the value function).

• $\lhd$, is a binary relation that specifies the order of priority among subgoals. The expression:

$$\omega_j \ \lhd \ \omega_i$$

should read: subgoal ω_j *subject-to* the subgoal ω_i (taken from (Huber, 2000)) which expresses that the subgoal ω_i has a higher priority than the subgoal ω_j. A priority *ranking system* is then specified by a set of relations $\{\omega_j \ \lhd \ \omega_i\}$. The relation $\lhd$ is *reflexive*, *transitive*, and *anti-symmetric*. Without loss of generality, we assume that given a set of subgoals $\Omega = \{\omega_1, \omega_2, \ldots, \omega_m\}$, the following set of relations holds:

$$\omega_j \ \lhd \ \omega_i, \quad \text{iff } i \leq j$$

Given a policy π, the value of a state $\mathbf{s}$ can be defined as a vector given by:

$$\mathcal{V}^\pi(\mathbf{s}) = [\mathcal{V}_1^\pi(\mathbf{s}), \mathcal{V}_2^\pi(\mathbf{s}), \ldots, \mathcal{V}_m^\pi(\mathbf{s})]^T$$

where $\mathcal{V}_i^\pi(\mathbf{s})$ is the local value of state $\mathbf{s}$ with respect to the subgoal ω_i under the policy π.

In order to define the optimization objective (the optimal value function), we need to define a total ordering of the vectored value functions, taking into account the order of priority, and also the flexibility parameters. We introduce the total order (that is *reflexive*, *anti-symmetric*, *transitive*, and *trichotomous*) $\leq_{lex}^{\Upsilon}$ as follows:

Definition 4.1: Given two vectors $\mathcal{V} = [\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_m]^T$ and $\mathcal{W} = [\mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_m]^T$, the total order $\mathcal{V} \leq_{lex}^{\Upsilon} \mathcal{W}$ is defined as:

• if $\forall i, \ 1 \ \leq \mathbf{i} \leq \mathbf{m}, \ \ \mathcal{V}_i = \mathcal{W}_i,$ then $\mathcal{V} =_{lex}^{\Upsilon} \mathcal{W}$

or

• if $\forall i, \ 1 \leq \mathbf{i} \leq \mathbf{m}, \ \ \mathcal{V}_i < \mathcal{W}_i$, then $\mathcal{V} <_{lex}^{\Upsilon} \mathcal{W}$

or

• there exists $1 \leq \mathbf{j} < \mathbf{m}$, such that $\forall i, \ 1 \leq \mathbf{i} \leq \mathbf{j}$:

1. $\mathcal{W}_i \geq \mathcal{V}_i$ or $\frac{1}{\epsilon_i}\mathcal{V}_i < \mathcal{W}_i \leq \mathcal{V}_i$

2. $\mathcal{V}_{j+1} < \mathcal{W}_{j+1}$

then $\mathcal{V} <_{lex}^{\Upsilon} \mathcal{W}$.

A simple interpretation of the above relation would be as follows: given two vectored value functions, we prefer the one that achieves the most number of subgoals based on the order of priorities among the subgoals, and the flexibility parameters Υ. Note that since each subgoal ω_i models a minimum cost-to-goal problem, all values in a vectored value function are negative. Thus, the expression:

$$\epsilon_i \mathcal{V}_i(\mathbf{s}) > \mathcal{W}_i(\mathbf{s})$$

states that the value $\mathcal{V}_i(\mathbf{s})$ is within an acceptable optimality loss with respect to the value $\mathcal{W}_i(\mathbf{s})$ in state $\mathbf{s}$. In other words, $\mathcal{V}_i(\mathbf{s})$ is within an acceptable optimality loss with respect to the value $\mathcal{W}_i(\mathbf{s})$, if we have:

$$\frac{1}{\epsilon_i}\mathcal{W}_i(\mathbf{s}) < \mathcal{V}_i(\mathbf{s}) \leq \mathcal{W}_i(\mathbf{s})$$

Note that when $\Upsilon = [1, 1, \ldots, 1]$, the relation $\leq_{lex}^{\Upsilon}$ turns to the standard *lexicographic ordering* in a multi-criterion reinforcement learning (Gabor et al., 1998) setting.

The relation $\leq_{lex}^{\Upsilon}$ is an extension of the lexicographic ordering when we take into account flexibility measure expressed in terms of the parameters $\Upsilon = [\epsilon_1, \epsilon_2, \ldots, \epsilon_m]^T$. The optimal value function can then be defined as:

$$\mathcal{V}_{lex}^{*}(\mathbf{s}) \quad = \quad \sup_{\pi} \ \ \mathcal{V}^{\pi}(\mathbf{s})$$

and the optimal policy is represented by:

$$\pi^*_{lex}(\mathbf{s}) \quad = \arg \quad \sup_\pi \ \ \mathcal{V}^\pi(\mathbf{s})$$

Note that in general π^*_{lex} may not optimize all the subgoals simultaneously, although it optimizes the maximum possible number of subgoals based on their priority levels. To illustrate this further, let:

$$\tilde{\mathcal{V}}^*_{lex}(\mathbf{s}) = [\mathcal{V}^*_1(\mathbf{s}), \mathcal{V}^*_2(\mathbf{s}), \ldots, \mathcal{V}^*_m(\mathbf{s})] \tag{4.1}$$

where $\mathcal{V}^*_i(\mathbf{s})$ denotes the optimal value of state $\mathbf{s}$ with respect to the subgoal ω_i. Then $\mathcal{V}^*_{lex}$ is bounded by $\tilde{\mathcal{V}}^*_{lex}$:

$$\mathcal{V}^*_{lex}(\mathbf{s}) \quad \leq^{\Upsilon}_{lex} \quad \tilde{\mathcal{V}}^*_{lex}(\mathbf{s}) \tag{4.2}$$

We now describe an approach based on coarticulation for approximately solving the above optimization problem. We assume that the agent has access to a set of controllers $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$, where the controller $\mathcal{C}_i$ is modeled as a minimum cost-to-goal subgoal option (Precup, 2000) associated with the subgoal $\omega_i \in \Omega$ in the COART model. We interchangeably use the *subject-to* relation for specifying the order of priority among controllers:

$$\mathcal{C}_j \ \triangleleft \ \mathcal{C}_i, \quad \text{iff } \omega_j \ \triangleleft \ \omega_i$$

Returning to the examples 4.1 and 4.2, we can think that each controller models a general purpose skill that the agent has learned in its life span. At different points in time, a set of stimuli signals $\{\delta_1, \delta_2, \ldots, \delta_m\}$ are activated. Then the agent initiates the set of controllers $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ concurrently until they terminate according to the termination mechanism

$\mathcal{T}$. At this time the system checks for the new set of stimuli signals and repeats the above process. Each time a controller $\mathcal{C}_i$ arrives in its goal state, it resets the stimuli signal δ_i.

In general optimal policies of controllers do not offer flexibility required in order to commit to many subtasks. However, there exists a special class of admissible near-optimal policies that guarantee making progress toward the goal in every state of the problem on average. Given a controller, an admissible policy is either an optimal policy, or a policy that *ascends* the optimal state-value function associated with the controller (i.e., on average leads to states with higher values), and is not far from the optimal policy.

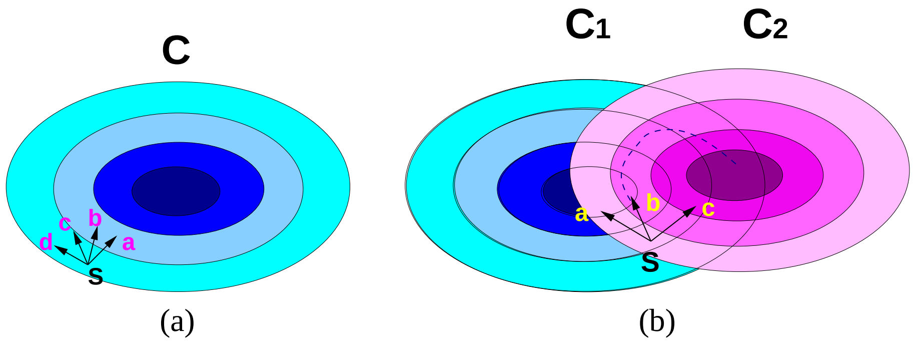


Figure 4.2. (a) actions **a**, **b**, and **c** are ascending on the state-value function associated with the controller $\mathcal{C}$, while action **d** is descending; (b) action **a** and **c** ascend the state-value function $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively, while they descend on the state-value function of the other controller. However action **b** ascends the state-value function of both controllers.

To illustrate this idea, consider Figure 4.2(a) showing a two dimensional state-value function. Regions with darker colors represents states with higher values. Assume that the agent is currently in state marked **s**. The arrows show the direction of state transition as a result of executing different actions, namely actions **a**, **b**, **c**, and **d**. The first three actions lead the agent to states with higher values, in other words they ascend the state-value function, while action **d** descends it. Figure 4.2(b) shows how introducing admissible policies enables simultaneous solution of multiple subgoals. In this figure, actions **a** and **c** are optimal in controllers $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively, but they both descend the state-value function of the other controller. However if we allow actions such as action **b**, we are guaranteed to ascend both value functions, with a slight degradation in optimality. In this example, by choosing action $\mathbf{b}$ we are coarticulating between both tasks while the first task takes precedence over the second task. The ascendancy property of policies in MDPs was first introduced by Perkins (2002) in the context of *Lyapunov functions* (Vincent and Grantham, 1997). We use a similar concept when we select the optimal value functions as the Lyapunov constraints themselves.

In the following sections we introduce a mathematical framework for modeling the above form of coarticulation. This involves developing a model for representing each coarticulatory controller and the flexibility they offer, together with a coarticulation algorithm that enables the agent to concurrently commit to multiple subgoals, each associated with a controller.

4.3 Redundant Controllers

In our approach for modeling coarticulation, we assumed that the agent has access to a set of acquired skills, each represented an SMDP controller. More formally, we use subgoal options (refer to Section 2.3.2) for modeling such skills. For generality, throughout this dissertation we refer to subgoal options simply as *controllers*. Also, for theoretical reasons, in this dissertation we assume that each controller optimizes a minimum cost-to-goal problem. An MDP $\mathcal{M}$ modeling a minimum cost-to-goal problem includes a set of goal states $\mathcal{S}_\Omega \subset \mathcal{S}$. We also represent the set of non-goal states by $\bar{\mathcal{S}}_\Omega = \mathcal{S} - \mathcal{S}_\Omega$. Every action in a non-goal state incurs some negative reward and the agent receives a reward of zero in goal states. A controller $\mathcal{C}$ is a minimum cost-to-goal controller, if $\mathcal{M}_\mathcal{C}$ optimizes a minimum cost-to-goal problem. The controller also terminates with probability one in every goal state.

In order to perform the form of coarticulation that we described in Section 4.2, each controller is required to offer some degree of flexibility, in terms of a class of policies that is admissible with respect to the optimization of the subgoal associated with that controller. For example, in driving, we may incorporate different policies such as controlling the wheels by both hands, or by only one hand, where the latter is not optimal. In many problems there might exist multiple optimal policies, but in general this set often offers limited flexibility required for concurrently committing to many subgoals. One way to increase the flexibility of a controller is to extend the set of admissible optimal policies from the optimal policies to those that ascend the value function and are not far from the optimal behavior (see Figure 4.2):

Definition 4.2: Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, a function $\mathcal{L} : \mathcal{S} \rightarrow \mathbb{R}$, and a deterministic policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, let:

$$\rho^{\pi}(\mathbf{s}) = E_{\mathbf{s}' \sim \mathcal{P}_s^{\pi(\mathbf{s})}}\{\mathcal{L}(\mathbf{s}')\} - \mathcal{L}(\mathbf{s}) \tag{4.3}$$

where $E_{\mathbf{s}' \sim \mathcal{P}_s^{\pi(\mathbf{s})}}\{.\}$ is the expectation with respect to the distribution over next states given the current state and the policy π. Then π is *ascending* on $\mathcal{L}$, if for every state s (except for the goal states if the MDP models a minimum cost-to-goal problem) we have $\rho^{\pi}(\mathbf{s}) > 0$. A policy π is called ascending on $\mathcal{M}$, if and only if π is ascending on $\mathcal{V}^*$ in $\mathcal{M}$.

For an ascending policy π on a function $\mathcal{L}$, function $\rho : \mathcal{S} \rightarrow \mathbb{R}^+$ gives a strictly positive value that measures how much the policy π ascends on $\mathcal{L}$ in state s. A deterministic policy π is descending on $\mathcal{L}$, if for some state s, $\rho^{\pi}(\mathbf{s}) < 0$. In general we would like to study how a given policy behaves with respect to the optimal value function in a problem. Thus we choose the function $\mathcal{L}$ to be the optimal state value function (i.e., $\mathcal{V}^*$). The above condition can be interpreted as follows: we are interested in policies that in average lead to states with higher values, or in other words *ascend* the state-value function surface.

Note that Definition 4.2 is closely related to the Lyapunov functions introduced in (Perkins, 2002; Perkins and Barto, 2001b,a). In fact, the ascendancy property of an ascending policy emerges naturally from the constraints imposed by a Lyapunov function $\mathcal{L}(\mathbf{s}) = \mathcal{V}^*(\mathbf{s})$.

The minimum and maximum rate at which an ascending policy in average ascends $\mathcal{V}^*$ are given by:

Definition 4.3: Assume that the policy π is ascending on the optimal state value function $\mathcal{V}^*$. Then π ascends on $\mathcal{V}^*$ with a factor at least α, if for all non-goal states $s \in \bar{\mathcal{S}}_\Omega$, $\rho^\pi(\mathbf{s}) \geq \alpha > 0$. We also define the guaranteed expected ascent rate of π as: $\kappa^\pi = \min_{s \in \bar{\mathcal{S}}_\Omega} \rho^\pi(\mathbf{s})$. The maximum possible achievable expected ascent rate of π is also given by $\eta^\pi = \max_{s \in \bar{\mathcal{S}}_\Omega} \rho^\pi(\mathbf{s})$.

One immediate problem with ascending policies is that Definition 4.2 ignores the immediate reward which the agent receives. For example it could be the case that as a result of executing an ascending policy, the agent transitions to some state with a higher value, but receives a huge negative reward. This can be counterbalanced by adding a second condition that keeps the ascending policies close to the optimal policy:

Definition 4.4: Given a minimum cost-to-goal problem modeled by an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, a deterministic policy π is ϵ-optimal on $\mathcal{M}$ if it satisfies the following condition:

$$\forall s \in \mathcal{S}, \mathcal{Q}^*(s, \pi(\mathbf{s})) \in [\frac{1}{\epsilon}\mathcal{V}^*(\mathbf{s}) \quad \mathcal{V}^*(\mathbf{s})] \tag{4.4}$$

Note that because $\mathcal{M}$ models a minimum cost-to-goal problem, all values are negative.

Definition 4.5: Given a minimum cost-to-goal problem modeled by an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, a deterministic policy π is ϵ-ascending on $\mathcal{M}$ if it satisfies the following conditions:

1. **Ascendancy**: π is ascending on $\mathcal{V}^*$

2. ϵ**-optimality**: π is ϵ-optimal on $\mathcal{V}^*$

Here, ϵ measures how close the ascending policy π is to the optimal policy. For any ϵ, the second condition assures that the ascending policy π is not far from the optimal policy. Naturally we often prefer policies that are ϵ-ascending for ϵ values close to 1. In section 4.6 we derive a lower bound on ϵ such that no policy for values smaller than this bound is ascending on $\mathcal{V}^*$ (in other words ϵ cannot be arbitrarily small). Similarly, a deterministic policy π is called ϵ-ascending on $\mathcal{C}$, if π is ϵ-ascending on $\mathcal{M}_{\mathcal{C}}$.

Remark 4.1: Given a minimum cost-to-goal problem modeled by an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, the optimal policy π^* is ϵ-ascending on $\mathcal{V}^*$ for $\gamma = 1$ and all $\epsilon \in (0, 1]$. Based on the Bellman optimality equation we have:

$$\begin{aligned}
\mathcal{V}^*(\mathbf{s}) &= E_{\mathbf{s'} \sim \mathcal{P}_s^{\pi^*(\mathbf{s})}} \{\mathcal{R}_{\mathbf{ss'}}^{\pi^*(\mathbf{s})} + \gamma \mathcal{V}^*(\mathbf{s'})\} \\
&= E_{\mathbf{s'} \sim \mathcal{P}_s^{\pi^*(\mathbf{s})}} \{\mathcal{R}_{\mathbf{ss'}}^{\pi^*(\mathbf{s})}\} + \gamma E_{\mathbf{s'} \sim \mathcal{P}_s^{\pi^*(\mathbf{s})}} \{\mathcal{V}^*(\mathbf{s'})\} \\
&< \gamma E_{\mathbf{s'} \sim \mathcal{P}_s^{\pi^*(\mathbf{s})}} \{\mathcal{V}^*(\mathbf{s'})\} \\
&= E_{\mathbf{s'} \sim \mathcal{P}_s^{\pi^*(\mathbf{s})}} \{\mathcal{V}^*(\mathbf{s'})\}
\end{aligned} \tag{4.5}$$

The inequality holds because $E_{\mathbf{s'} \sim \mathcal{P}_s^{\pi^*(\mathbf{s})}} \{\mathcal{R}_{\mathbf{ss'}}^{\pi^*(\mathbf{s})}\} < 0$ (since $\mathcal{M}$ is a minimum cost-to-goal problem, all rewards are negative). This satisfies the first condition. The second condition is also satisfied since for any optimal policy π^* we have $\mathcal{Q}^*(s, \pi^*(\mathbf{s})) = \mathcal{V}^*(\mathbf{s})$.

Remark 4.2: Given a minimum cost-to-goal problem modeled by an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ with a discount factor $\gamma \neq 1$ the optimal policy is not necessarily an ascending policy on $\mathcal{M}$. This immediately follows the fact that the equality in the last step of the Equation 4.5 may not hold for all values of γ.

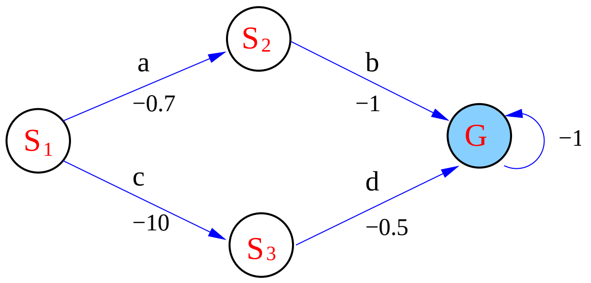


Figure 4.3. Example of an MDP in which the optimal policy is not ascending for some value of γ.

Figure 4.3 shows an example of a deterministic MDP in which the optimal policy is not ascending on the optimal state value function. All actions (**a**, **b**, **c**, and **d**) are deterministic. The arcs are labeled by the action and the immediate reward that the agent receives a a result of executing the action. The agent receives a reward of -1 in the goal state (marked by **G**) and does not change state afterward. Using a discount factor $\gamma = 0.5$, we can compute the value function for every state:

$$\mathcal{V}^*(G) = -2, \quad \mathcal{V}^*(S_2) = -2, \quad \mathcal{V}^*(S_3) = -1.5, \quad \mathcal{V}^*(S_1) = -1.7$$

The optimal policy in state $\mathbf{S}_1$ is to choose action **a**, but it causes a transition to state $\mathbf{S}_2$ that has a lower value than state $\mathbf{S}_1$. However, action **c** causes a transition to state $\mathbf{S}_3$ that has a higher value than state $\mathbf{S}_1$.

Definition 4.6: Given a minimum cost-to-goal controller $\mathcal{C}$, a deterministic policy π is ϵ-ascending on $\mathcal{C}$, if π is ϵ-ascending on the MDP $\mathcal{M}_\mathcal{C}$ associated with the controller $\mathcal{C}$.

Definition 4.7: A minimum cost-to-goal controller $\mathcal{C}$ is an ϵ-*redundant* controller if there exist multiple deterministic policies that are either optimal, or ϵ-ascending on $\mathcal{C}$. We represent the class of of such policies by $\chi^\epsilon_\mathcal{C}$.

In Definition 4.3 we introduced the minimum and maximum ascent rates for an individual ascending policy π. Given an ϵ-redundant controller we can define the minimum and maximum ascent rates of the controller by taking the minimum and maximum ascent rates across the class of ϵ-ascending policies the controller admits.

Definition 4.8 Let $\mathcal{C}$ be a minimum cost-to-goal controller. The minimum ascent rate of $\mathcal{C}$ is defined as $\bar{\kappa} = \min_{\pi \in \chi^{\epsilon}_{\mathcal{C}}} \kappa^{\pi}$. Similarly, the maximum ascent rate of $\mathcal{C}$ is defined as $\bar{\eta} = \max_{\pi \in \chi^{\epsilon}_{\mathcal{C}}} \eta^{\pi}$.

Similarly, we can extend the concept of ascendancy to the vectored value functions $\mathcal{V}^*_{lex}$ in the *COART* model:

Definition 4.9 Given a COART model $\mathcal{M}$, a set of controllers $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ defined over $\mathcal{M}$, and a vector $\epsilon_{lex} = [\epsilon_1, \epsilon_2, \ldots, \epsilon_m]^T$ where $\epsilon_i \in (0, 1]$, a deterministic policy π is ϵ_{lex}-ascending on $\tilde{\mathcal{V}}^*_{lex}(\mathbf{s})$, if for every state $\mathbf{s}$ either of the following relationship holds:

1. $\epsilon_i = 0$, or

2. $\epsilon_i \neq 0$, and the policy π is ϵ_i-ascending on $\mathcal{V}^*_i(\mathbf{s})$

4.3.1 Computing the ϵ-redundant Set

Given the optimal value function of a controller $\mathcal{C}$, we can compute the ϵ-redundant set of policies in every state $\mathbf{s}$. The main computations are verifying Ascendancy and ϵ-optimality conditions (Definition 4.3). Algorithm1 summarizes these steps.

4.4 Sequential Algorithm (No Coarticulation)

Before we present our algorithm for performing coarticulation among a set of prioritized subgoals, we outline a *sequential* algorithm that runs the controllers in a sequence back-to-back according to their order of priority. Whenever a controller of a higher priority terminates in some state $\mathbf{s}$ the next controller with the highest priority is invoked in that state until all the controllers are terminated. We refer to such policy as the *sequential policy* represented by π_{seq} which provides the trivial sequential solution to the problem.

Algorithm 1 Function ComputeRedundantSets($\mathbf{s}$, ϵ)

```
Inputs:
  $\mathbf{s}$                                   \\ Current state
  $\mathcal{Q}^*(\mathbf{s},\mathbf{a})$
  $\epsilon$          \\ The flexibility afforded by the controller $\mathcal{C}$ ($0 < \epsilon \leq 1$)
Outputs:
$\mathcal{A}^\epsilon(\mathbf{s})$                          \\ $\epsilon$-redundant set in state $\mathbf{s}$

1: $\mathcal{A}^\epsilon(\mathbf{s}) \leftarrow \emptyset$
2: for all $\mathbf{a}$ such that $\mathbf{a} \in \mathcal{A}(\mathbf{s})$ do
3:     Check  $E_{\mathbf{s}'\sim\mathcal{P}_{\mathbf{s}}^{\mathbf{s}}}\{\mathcal{V}^*(\mathbf{s}')\} - \mathcal{V}^*(\mathbf{s}) > 0$        \\ Ascendancy condition
4:     Check  $\mathcal{Q}^*(\mathbf{s},\mathbf{a}) \geq \frac{1}{\epsilon}\mathcal{V}^*(\mathbf{s})$                 \\ $\epsilon$-optimality condition
5:     If both conditions hold, add the pair $\mathbf{a}$ to $\mathcal{A}^\epsilon(\mathbf{s})$
6: end for
7: Return  $\mathcal{A}^\epsilon(\mathbf{s})$                 \\ Return the $\epsilon$-redundant set
```

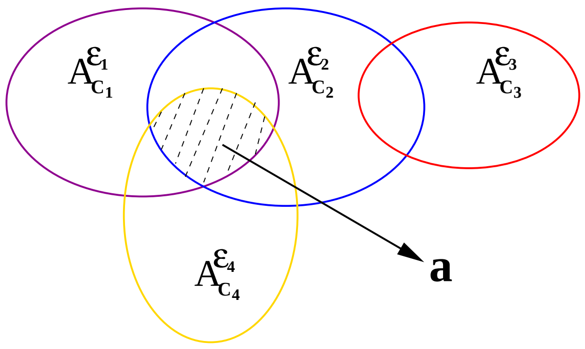
Algorithm 2 Sequence(s, $\mathcal{C}_1, \mathcal{C}_3, \ldots, \mathcal{C}_m$)

```
Input: current state $\mathbf{s}$; set of prioritized controllers $\mathcal{C}_i$.
Initialize: currentState = $\mathbf{s}$
2: for $i = 1, 2, \ldots, n$ do
     if $currentState \in \mathcal{I}_{\mathcal{C}_i}$ then
4:       Execute the optimal policy of the controller $\mathcal{C}_i$ until it terminates in a goal state
           $\omega_i$
         $currentState \leftarrow \omega_i$
6:   end if
   end for
```

We primarily use the sequential policy generated by the algorithm 2 as a basis for evaluating the performance of the coarticulation algorithm that we present in the next section.

4.5 Coarticulation Algorithm

In this section, we present an algorithm for performing coarticulation among a set of prioritized subtasks $\Omega = \{\omega_1, \omega_2, \ldots, \omega_m\}$ such that $\omega_j \triangleleft \omega_i$ if and only if $j \leq i$ (refer to the problem definition that we presented in Section 4.2). We assume that each subgoal ω_i is modeled by an ϵ-redundant controller $\mathcal{C}_i$. The objective is to devise a policy that maximizes the number of subgoals committed at every step. Assume that the current state of the agent is $\mathbf{s}$. Each controller $\mathcal{C}_i$ admits a class of policies represented by a redundant-set $\mathcal{A}^{\epsilon_i}_{\mathcal{C}_i}(\mathbf{s})$ in state $\mathbf{s}$. We select an action that is admissible to the subordinate controllers, constrained by the admissible policies with regard to the superior controllers.



$A^{\varepsilon_1}_{C_1}$
$A^{\varepsilon_2}_{C_2}$
$A^{\varepsilon_3}_{C_3}$
$A^{\varepsilon_4}_{C_4}$
a

Figure 4.4. A visualization of the action selection mechanism in *Coarticulate* algorithm.

The general idea of our algorithm is visualized in Figure 4.4. We first take the intersection of the redundant sets of the two top controllers (i.e., $\mathcal{A}^{\epsilon_1}_{\mathcal{C}_1}(\mathbf{s})$ and $\mathcal{A}^{\epsilon_2}_{\mathcal{C}_2}(\mathbf{s})$). Then we continue intersecting the resulting set with the third controller, fourth controller, and so on. At each step, if the intersection is empty, we skip over that controller and move to the next subordinate controller. In Figure 4.4 the intersection of the redundant sets of the superior controllers becomes empty with the controller $\mathcal{C}_3$. Thus the algorithm skips over the controller $\mathcal{C}_3$ and computes the intersection with the next subordinate controller, e.g. $\mathcal{C}_4$.

After scanning through all the controllers in a decreasing order of priority, the algorithm returns an action from the set of actions that maximizes the number of committed subgoals. Algorithm *Coarticulate* summarizes the above steps in more detail:

Algorithm 3 Coarticulate(s, $\mathcal{C}_1, \mathcal{C}_3, \ldots, \mathcal{C}_m$)

```
Input: current state $s$; set of controllers $\mathcal{C}_i$; redundant-sets $\mathcal{A}^{\epsilon_i}_{\mathcal{C}_i}(\mathbf{s})$ for every controller $\mathcal{C}_i$.
1: Initialize: $U_1 = \mathcal{A}^{\epsilon_1}_{\mathcal{C}_1}$
2: for $i = 2, 3, \ldots, n$ do
3:     $U_i = U_{i-1} \cap \mathcal{A}^{\epsilon_i}_{\mathcal{C}_i}$
4:     if $U_i = \emptyset$ then
5:         $U_i = U_{i-1}$
6:     end if
7: end for
8: return a $\in U_n$
```

In this algorithm, $U_i(\mathbf{s})$ represents the *ordered* intersection of the redundant-sets $\mathcal{A}^{\epsilon_j}_{\mathcal{C}_j}$ up to the controller $\mathcal{C}_i$ (i.e., $1 \leq j \leq i$) constrained by the order of priority. In other words, each set $U_i(\mathbf{s})$ contains a set of actions in state s that are all ϵ_i-ascending with respect to the superior controllers $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_i$. Due to the limited amount of redundancy in the system, it is possible that the system may not be able to commit to some of the subordinate controllers. This happens when none of the actions with respect to some controller $\mathcal{C}_j$ (i.e., $a \in \mathcal{A}^{\epsilon_j}_{\mathcal{C}_j}(\mathbf{s})$) are ϵ-ascending with respect to the superior controllers. In this case the algorithm skips the controller $\mathcal{C}_j$, and continues the search in the redundant-sets of the remaining subordinate controllers. The complexity of the above algorithm consists of the following costs:

1. Computational cost of computing the redundant-sets $\mathcal{A}^{\epsilon_i}_{\mathcal{C}_i}$ for controller $\mathcal{C}_i$ which is linear in the number of states and actions (assuming that we have access to the optimal value function):

$$\mathbf{O}(|S|\,|A|) \tag{4.6}$$

2. Computational cost of performing the coarticulation algorithm (Algorithm 3) in every state s, which is:

$$\mathbf{O}((m-1)\,(\max_{\mathcal{C}} |\mathcal{A}^{\epsilon}_{\mathcal{C}}(\mathbf{s})|)^2) \tag{4.7}$$

where $\mathbf{m}$ is the number of subgoals.

Note that while the computational complexity of computing the redundant-sets is polynomial in the set of states and actions (Equation 4.6), the computational complexity of the coarticulation approach is only polynomial in the size of the redundant-sets (Equation 4.7). In general the size of the redundant-sets are considerably smaller than the original size of the concurrent action space. Furthermore, we can always select a feasible subset of redundant-sets and perform the coarticulation algorithm tractably.

It is worth to note that the action selection mechanism based on the algorithm *Coarticulate* can be thought of the Lyapunov-constrained action selection approach described in (Perkins, 2002; Perkins and Barto, 2001b,a), when the optimal value functions of the controllers are selected as the Lyapunov functions themselves. In the next section, we theoretically analyze redundant controllers and the performance of the policy merging algorithm in various situations.

4.6 Theoretical Results

In this section, we present a set of theoretical results (Rohanimanesh et al., 2004a,b) characterizing ϵ-redundant controllers, in terms of the bounds on the number of time steps it takes for a controller to complete its task, and the performance of the coarticulation algorithm. To simplify our theoretical analysis, we assume that in all of the theorems the discount factor is set to one (i.e., $\gamma = 1$). This renders the set of all optimal policies to be ascending policies on the optimal state value function (refer to Remark 4.1). Note that since all the MDPs model minimum cost-to-goal problems, they can be considered as episodic tasks that terminate with probability one in a goal state and thus the convergence properties of the algorithms for learning the optimal state value function for the choice of $\gamma = 1$ is preserved.

In section 4.3 we stated that there is a lower bound on ϵ such that there exist no ϵ-ascending policy for values smaller than this bound. In the next theorem we compute this lower bound:

Theorem 4.1: Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ be a minimum cost-to-goal MDP and let π be an ϵ-ascending policy defined on $\mathcal{M}$. Then ϵ is bounded by $\epsilon > \frac{|\mathcal{V}^*_{max}|}{|\mathcal{V}^*_{min}|}$, where $\mathcal{V}^*_{min} = \min_{s \in \bar{\mathcal{S}}_\Omega} \mathcal{V}^*(\mathbf{s})$ and $\mathcal{V}^*_{max} = \max_{s \in \bar{\mathcal{S}}_\Omega} \mathcal{V}^*(\mathbf{s})$.

Proof: Since π is an ϵ-ascending policy, we have $\forall s \in \mathcal{S}$, $E_{\mathbf{s}' \sim \mathcal{P}_s^{\pi(\mathbf{s})}} \{\mathcal{V}^*(\mathbf{s}')\} - \mathcal{V}^*(\mathbf{s}) > 0$, and $\mathcal{Q}^*(s, \pi(\mathbf{s})) \geq \frac{1}{\epsilon} \mathcal{V}^*(\mathbf{s})$. By rearranging the terms in the latter, first we obtain $-\epsilon\, \mathcal{Q}^*(s, \pi(\mathbf{s})) \leq -\mathcal{V}^*(\mathbf{s})$. Then we substitute it for $\mathcal{V}^*(\mathbf{s})$ to obtain $E_{\mathbf{s}' \sim \mathcal{P}_s^{\pi(\mathbf{s})}} \{\mathcal{V}^*(\mathbf{s}')\} - \epsilon\, \mathcal{Q}^*(s, \pi(\mathbf{s})) > 0$. Thus:

$$\begin{aligned}
\epsilon &> \left| \frac{\sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}_{\mathbf{ss}'}^{\pi(\mathbf{s})} \mathcal{V}^*(\mathbf{s}')}{\mathcal{Q}^*(s, \pi(\mathbf{s}))} \right| \\
&\geq \frac{\mid \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}_{\mathbf{ss}'}^{\pi(\mathbf{s})} \mathcal{V}^*_{max} \mid}{\mid \mathcal{V}^*_{min} \mid} \\
&= \frac{\mid \mathcal{V}^*_{max} \mid \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}_{\mathbf{ss}'}^{\pi(\mathbf{s})}}{\mid \mathcal{V}^*_{min} \mid} \\
&= \frac{\mid \mathcal{V}^*_{max} \mid}{\mid \mathcal{V}^*_{min} \mid}
\end{aligned} \tag{4.8}$$

note that since $\mathcal{M}$ models a minimum cost-to-goal problem, we have $\mathcal{V}^*_{min} \leq \mathcal{V}^*_{max} < 0$, and $\mid \mathcal{V}^*_{min} \mid \geq \mid \mathcal{V}^*_{max} \mid$.

Such a lower bound characterizes the maximum flexibility we can afford in a redundant controller and gives us an insight on the range of ϵ values that we can choose for it. In the second theorem we derive an upper bound on the expected number of steps that a minimum cost-to-goal controller takes to complete when executing an ϵ-ascending policy:

Theorem 4.2: Let $\mathcal{C}$ be an ϵ-ascending minimum cost-to-goal controller and let s denote the current state of the controller. Then any ϵ-ascending policy π on $\mathcal{C}$ will terminate the controller in some goal state with probability one. Furthermore, termination occurs in average in at most $\lceil \frac{-\mathcal{V}^*(\mathbf{s})}{\kappa^\pi} \rceil$ steps and at least $\lceil \frac{-\mathcal{V}^*(\mathbf{s})}{\eta^\pi} \rceil$ steps, where κ^π and η^π are the minimum and maximum ascent rates of the policy π.

Proof: We first present the following lemma:

Lemma 4.1: Let $\mathcal{C}$ be a minimum cost-to-goal controller and let π be an ϵ-ascending policy on $\mathcal{C}$. Then for any non-goal state s, there exists a state $\mathbf{s}'$, such that $\mathcal{P}^{\pi}_{\mathbf{ss}'} > 0$ and $\mathcal{V}^*(\mathbf{s}') > \mathcal{V}^*(\mathbf{s})$.

We prove the lemma by contradiction. Let $\mathcal{H}$ be the set of states $\mathbf{s}' \in \mathcal{S}$ such that $\mathcal{P}^{\pi}_{\mathbf{ss}'} > 0$ (note that $\sum_{\mathbf{s}' \in \mathcal{H}} \mathcal{P}^{\pi(\mathbf{s})}_{\mathbf{ss}'} = 1$) and assume that $\forall \mathbf{s}' \in \mathcal{H}, \mathcal{V}^*(\mathbf{s}') \leq \mathcal{V}^*(\mathbf{s})$. Also let $\mathcal{W} = \max_{\mathbf{s}' \in \mathcal{H}} \mathcal{V}^*(\mathbf{s}')$. Then we have:

$$\begin{aligned}
\rho^\pi(\mathbf{s}) &= E_{\mathbf{s}' \sim \mathcal{P}^{\pi(\mathbf{s})}_{s}}\{\mathcal{V}^*(\mathbf{s}')\} - \mathcal{V}^*(\mathbf{s}) \\
&= \sum_{\mathbf{s}' \in \mathcal{H}} \mathcal{P}^{\pi(\mathbf{s})}_{\mathbf{ss}'} \mathcal{V}^*(\mathbf{s}') - \mathcal{V}^*(\mathbf{s}) \\
&\leq \sum_{\mathbf{s}' \in \mathcal{H}} \mathcal{P}^{\pi(\mathbf{s})}_{\mathbf{ss}'} \mathcal{W} - \mathcal{V}^*(\mathbf{s}) \\
&= \mathcal{W} \sum_{\mathbf{s}' \in \mathcal{H}} \mathcal{P}^{\pi(\mathbf{s})}_{\mathbf{ss}'} - \mathcal{V}^*(\mathbf{s}) \\
&= \mathcal{W} - \mathcal{V}^*(\mathbf{s}) \\
&\leq 0
\end{aligned} \tag{4.9}$$

this gives $\rho^\pi(\mathbf{s}) \leq 0$ which contradicts the fact that π is an ϵ-ascending policy on $\mathcal{C}$.

Now we present the proof for Theorem 4.2. Since $\mathcal{C}$ terminates only in goal states, we only need to show there exists a goal state $s_g \in \mathcal{S}_\Omega$ such that $\mathcal{P}^{\mathcal{C}}_{ss_g} > 0$, where $\mathcal{P}^{\mathcal{C}}_{\mathbf{ss}'}$ is the multi-step transition probability (Precup, 2000) that gives the probability of executing $\pi_{\mathcal{C}}$ in state s until it terminates in state $\mathbf{s}'$. Based on Lemma 4.1, there exists a path $\langle s, s_1, \ldots, s_n \rangle$ such that $\mathcal{P}^{\mathcal{C}}_{ss_n} > 0$, and the sequence $\langle \mathcal{V}^*(\mathbf{s}), \mathcal{V}^*(s_1), \ldots, \mathcal{V}^*(s_n) \rangle$ is monotonically increasing. Thus in the limit $\mathcal{V}^*(s_n) = \mathcal{V}^*(s_g)$ for some goal state $s_n = s_g$ and thus we have $\mathcal{P}^{\pi}_{ss_g} > 0$. Note that at every step the value of a state increases by κ^π in average, thus in average it will take at most $\lceil \frac{-\mathcal{V}^*(\mathbf{s})}{\kappa^\pi} \rceil$ steps for s_n in the sequence $\langle s, s_1, \ldots, s_n \rangle$ to converge to some goal state s_g. Similarly, in average it will take at least $\lceil \frac{-\mathcal{V}^*(\mathbf{s})}{\eta^\pi} \rceil$ steps for s_n in the sequence $\langle s, s_1, \ldots, s_n \rangle$ to converge to some goal state s_g.

Based on Theorem 4.2, we can define a measurement of time for completion of an ϵ-ascending policy π being initiated in some state $\mathbf{s}$. By completion we refer to the event of arriving in goal state:

Definition 4.10: Given a minimum cost-to-goal $\mathcal{M}$, and an ϵ-ascending policy π on $\mathcal{M}$, let $\mu_\pi(\mathbf{s})$ denote the expected time of arriving in a goal state in $\mathcal{M}$, when π is initiated in a state $\mathbf{s}$.

It is easy to verify that this time is bounded by:

$$\lceil \frac{-\mathcal{V}^*(\mathbf{s})}{\eta^\pi} \rceil \leq \mu_\pi(\mathbf{s}) \leq \lceil \frac{-\mathcal{V}^*(\mathbf{s})}{\kappa^\pi} \rceil$$

Similarly, we can define a measurement of time for achieving the subgoal associated with a controller in any state $\mathbf{s}$:

Definition 4.11 Let $\mathcal{C}$ be a minimum cost-to-goal ϵ-redundant controller. The worst expected time for completion of $\mathcal{C}$ in a state $\mathbf{s}$ is defined as:

$$\tau_{\mathcal{C}}(\mathbf{s}) = \lceil \frac{-\mathcal{V}^*(\mathbf{s})}{\bar{\kappa}} \rceil$$

and the best expected time for completion of $\mathcal{C}$ in a state $\mathbf{s}$ is defined as:

$$\sigma_{\mathcal{C}}(\mathbf{s}) = \lceil \frac{-\mathcal{V}^*(\mathbf{s})}{\bar{\eta}} \rceil$$

This result assures that the controller arrives in a goal state and will achieve its goal in a bounded number of steps. We use this result when studying performance of running multiple redundant controllers in parallel. Next, we study how concurrent execution of two controllers using Algorithm *Coarticulate* impacts each controller (this result can be trivially extended to the case when a set of $m > 2$ controllers are executed concurrently):

Theorem 4.3: Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, and any two minimum cost-to-goal redundant controllers $\{\mathcal{C}_1, \mathcal{C}_2\}$ defined over $\mathcal{M}$, the policy π obtained by Algorithm *Coarticulate* based on the ranking system $\{\mathcal{C}_2 \lhd \mathcal{C}_1\}$ is ϵ_1-ascending on $\mathcal{C}_1(\mathbf{s})$. Moreover, if $\forall s \in \mathcal{S}, \mathcal{A}^{\epsilon_1}_{\mathcal{C}_1}(\mathbf{s}) \cap \mathcal{A}^{\epsilon_2}_{\mathcal{C}_2}(\mathbf{s}) \neq \emptyset$, policy π will be ascending on both controllers with the ascent rate at least $\kappa^{\pi} = \min\{\kappa^{\pi_1}, \kappa^{\pi_2}\}$.
This theorem states that merging policies of two controllers using Algorithm *Coarticulate* would generate a policy that remains ϵ_1-ascending on the superior controller. In other words it does not negatively impact the superior controller.
Proof: Since $\mathcal{C}_2 \lhd \mathcal{C}_1$, Algorithm *Coarticulate* will always select an action from the set $\mathcal{A}^{\epsilon_1}_{\mathcal{C}_1}(\mathbf{s})$. Thus the resulting policy remains ϵ_1-ascending on $\mathcal{C}_1$. When we have $\forall s \in \mathcal{S}, \mathcal{A}^{\epsilon_1}_{\mathcal{C}_1}(\mathbf{s}) \cap \mathcal{A}^{\epsilon_2}_{\mathcal{C}_2}(\mathbf{s}) \neq \emptyset$, then the policy generated by Algorithm *Coarticulate* is ϵ_1-ascending with respect to the controller $\mathcal{C}_1$ and ϵ_2-ascending with respect to the controller $\mathcal{C}_2$, and thus it ascends both $\mathcal{V}^*_1(\mathbf{s})$ and $\mathcal{V}^*_2(\mathbf{s})$ with an ascent rate at least $\min\{\kappa^{\pi_1}, \kappa^{\pi_2}\}$.

In the next theorem we show that the coarticulated policy obtained by performing the Algorithm *Coarticulate* over a set of ϵ-redundant controller $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ is ascending on $\tilde{\mathcal{V}}^*(\mathbf{s})$. This intuitively implies that the policy generated by this algorithm will eventually achieve all the subgoals associated with the controllers according to their degree of significance. We also establish lower and upper bounds on ϵ_{lex}:

Theorem 4.4: Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, and a set of minimum cost-to-goal controllers $\{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_m\}$ defined over $\mathcal{M}$, let π denote the policy obtained by performing the Algorithm *Coarticulate* based on the ranking system $\{\mathcal{C}_j \lhd \mathcal{C}_i | i < j\}$. Then π is ϵ_{lex}-ascending on $\tilde{\mathcal{V}}^*_{lex}(\mathbf{s})$. Moreover, we have:

$$\epsilon^{min}_{lex} \leq \epsilon_{lex} \leq \epsilon^{max}_{lex}$$

where $\epsilon_{min} = [\epsilon_1,\ 0,\ \dots,\ 0]^T$ and $\epsilon_{max} = [\epsilon_1, \epsilon_2, \dots, \epsilon_m]^T$.

Proof: The worst case for the Algorithm *Coarticulate* takes place when it can only optimize the controller with the highest priority, i.e., $\mathcal{C}_1$. This happens when $\mathcal{A}^{\epsilon_1}_{\mathcal{C}_1} \cap \mathcal{A}^{\epsilon_i}_{\mathcal{C}_i} = \emptyset, \forall i \neq 1$. In this case π will be selected from the class of ϵ-ascending policies of $\mathcal{C}_1$, thus π is ϵ_1-ascending on $\mathcal{V}^*_1(\mathbf{s})$. Based on Definition 4.9, this implies that the policy π is ϵ^{min}_{lex} ascending on $\tilde{\mathcal{V}}^*_{lex}(\mathbf{s})$, where $\epsilon^{min}_{lex} = [\epsilon_1, 0, \dots, 0]^T$. The best case for the Algorithm *Coarticulate* takes place when it can optimize all of the controllers in state $\mathbf{s}$ (i.e., $\cap^m_{i=1} \mathcal{A}^{\epsilon_i}_{\mathcal{C}_i} \neq \emptyset$). In this case the policy π will be ϵ_i-ascending on $\mathcal{V}^*_i(\mathbf{s})$ for all $1 \leq i \leq m$. Based on Definition 4.9, this implies that the policy π is ϵ^{max}_{lex} ascending on $\tilde{\mathcal{V}}^*_{lex}(\mathbf{s})$, where $\epsilon^{max}_{lex} = [\epsilon_1, \epsilon_2, \dots, \epsilon_m]^T$.

In the next theorem, we establish bounds on the expected number of steps that it takes for the policy obtained by Algorithm *Coarticulate* to achieve a set of prioritized subgoals $\omega = \{\omega_1, \dots, \omega_m\}$ by concurrently executing the associated controllers $\{\mathcal{C}_1, \dots, \mathcal{C}_m\}$:

Theorem 4.5: Let $\zeta = \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_m\}$ be a set of minimum cost-to-goal ϵ_i-redundant ($i = 1, \dots, m$) controllers defined over MDP $\mathcal{M}$. Let the policy π_{coart} denote the policy obtained by Algorithm *Coarticulate* based on the ranking system $\{\mathcal{C}_j \lhd \mathcal{C}_i |\ \text{iff}\ \ i < j\}$. Let $\mu_{coart}(\mathbf{s})$ denote the expected number of steps for the policy π_{coart} for achieving all the subgoals $\{\omega_1, \omega_2, \dots, \omega_m\}$ associated with the set of controllers, when it is initiated in state $\mathbf{s}$. Then $\mu_{coart}(\mathbf{s})$ is bounded by:

$$\max_{\mathcal{C}_i}\lceil\frac{-\mathcal{V}_i^*(\mathbf{s})}{\bar{\eta}_{\mathcal{C}_i}}\rceil \;\leq\; \mu_{coart}(\mathbf{s}) \;\leq\; \sum_{h\in\mathcal{H}}\mathcal{P}_{\pi_{seq}}(h)\sum_{i=1}^{m}\lceil\frac{-\mathcal{V}_i^*(h(i))}{\bar{\kappa}_{\mathcal{C}_i}}\rceil \qquad (4.10)$$

where $\bar{\eta}_{\mathcal{C}_i}$ is the maximum expected ascent rate for the controller $\mathcal{C}_i$ (see Definition 4.3), $\mathcal{H}$ is the set of sequences $h = \langle s, \omega_1, \omega_2, \ldots, \omega_m\rangle$ in which ω_i is a goal state in controller $\mathcal{C}_i$ (i.e., $\omega_i \in \mathcal{S}_{\Omega_i}$). The probability distribution $\mathcal{P}_{\pi_{seq}}(h) = \mathcal{P}_{s\omega_1}^{\mathcal{C}_1}\prod_{i=2}^{m}\mathcal{P}_{\omega_{i-1}\omega_i}^{\mathcal{C}_i}$ over sequences $h \in \mathcal{H}$ gives the probability of executing the set of controllers in sequence based on the order of priority starting in state s (i.e., executing the policy π_{seq} obtained from *Sequence* algorithm), and observing the goal state sequence $\langle\omega_1, \ldots, \omega_m\rangle$.

Proof: Based on Theorem 4.3, when Algorithm *Coarticulate* always finds a policy π that optimizes all controllers (i.e., $\forall s \in \mathcal{S}, \cap_{i=1}^{m}\mathcal{A}_{\mathcal{C}_i}^{\epsilon_i}(\mathbf{s}) \neq \emptyset$), policy π_{coart} will ascend on all controllers. Thus in average the total time for all controllers to terminate equals the time required for a controller that takes the most time to complete which has the lower bound of $\max_{\mathcal{C}_i}\lceil\frac{-\mathcal{V}_i^*(\mathbf{s})}{\bar{\eta}_{\mathcal{C}_i}}\rceil$. The worst case happens when the policy generated by Algorithm *Coarticulate* can not optimize more than one controller at a time. In this case it always optimizes the controller with the highest priority until its termination, then optimizes the second highest priority controller and continues this process to the end in a sequential manner (which is equivalent to the π_{seq} policy). The right hand side of the inequality given by Equation 4.10 gives an upper bound for the expected time required for all controllers to complete when they are executed sequentially.

Note that the upper bound right in Equation 4.10 is not exactly equal to the expected time that would take for the policy π_{seq} obtained from the algorithm *Sequence* to achieve all the subgoals, although it still provides an upper bound for this time. Although Theorem 4.4 provides bounds for both π_{coart} and π_{seq}, it does not establish theoretically guaranteed better performance for the coarticulated policy over the sequential policy. In the next set of our theoretical results we express conditions under which the coarticulated policy performs better, or strictly better than the sequential policy. First we limit the scope of the class of admissible policies in a controller only to the redundant set of optimal policies it admits (i.e., ϵ-ascending policies with $\epsilon = 1$). Next theorem, demonstrates that in such controllers the coarticulated policy will be at worst performing as good as the sequential policy. For simplicity we only consider two prioritized subgoals. The results can be extended to more than subgoals similarly:

Theorem 4.6: Assume $\zeta = \{\mathcal{C}_1, \mathcal{C}_2\}$ is a set of prioritized minimum cost-to-goal 1-redundant (i.e., $\epsilon = 1$) controllers defined over MDP $\mathcal{M}$ with a ranking system $\{C_2 \triangleleft \mathcal{C}_1\}$. Let the policy π_{seq} denote the policy obtained by performing the algorithm *Sequence*, and let the policy π_{coart} denote the policy obtained by performing the algorithm *Coarticulate*. Assuming that the current state of the system is $\mathbf{s}$, the following conditions hold:

- **(a)** The policy π_{coart} achieves the subgoals in time no worse than the policy π_{seq}.
- **(b)** If the following conditions are satisfied:

  1. In every state $\mathbf{s}'$: $\quad \mathcal{A}^1_{\mathcal{C}_1}(\mathbf{s}') \cap \mathcal{A}^1_{\mathcal{C}_2}(\mathbf{s}') \neq \emptyset$
  2. In the current state $\mathbf{s}$: $\quad \omega_{\mathcal{C}_1}(\mathbf{s}) > \tau_{\mathcal{C}_2}(\mathbf{s})$

  then the policy π_{coart} achieves the subgoals in average in time strictly better than the policy π_{seq}.

Proof: To prove the first part, note that the worst case of the Algorithm *Coarticulate* takes place when $\mathcal{A}^1_{\mathcal{C}_1}(\mathbf{s}') \cap \mathcal{A}^1_{\mathcal{C}_2}(\mathbf{s}') = \emptyset$ for all $\mathbf{s}' \in \mathcal{S}$. In this case π_{coart} only optimizes $\mathcal{C}_1$ until it terminates in some subgoal of $\mathcal{C}_1$, and then subsequently optimizes $\mathcal{C}_2$. Thus $\pi_{coart} \equiv \pi_{seq}$ and therefore in the worst case, the policy π_{coart} achieves the subgoals in average in time no worst than the policy π_{seq}.

To prove the second part, let $\mu_{coart}(\mathbf{s})$ and $\mu_{seq}(\mathbf{s})$ respectively denote the expected number of steps for the policies π_{coart} and π_{seq} for achieving all the subgoals $\{\omega_1, \omega_2\}$ associated with the set of controllers, when it is initiated in state $\mathbf{s}$. The first condition implies that at any state both controllers will be optimized by π_{coart}. Based on the second condition (i.e., $\omega_{\mathcal{C}_1}(\mathbf{s}) > \tau_{\mathcal{C}_2}(\mathbf{s})$), this implies that π_{coart} will achieve $\mathcal{C}_2$ while committing to $\mathcal{C}_1$, thus the time required for completion of π_{coart} is bounded by the worst expected time to achieve $\mathcal{C}_1$:

$$\begin{aligned}\mu_{coart}(\mathbf{s}) &= \mu_{\pi_1^*}(\mathbf{s}) \\ &< \mu_{\pi_1^*}(\mathbf{s}) + \mu_{\pi_2^*}(\omega_1) \\ &= \mu_{seq}(\mathbf{s})\end{aligned}$$

note that the right hand side is the expected time of completion for the policy μ_{seq}. This is indeed true since we have limited the class of ascending policies only to the set of redundant policies, thus any policy selected by Algorithm *Coarticulate* is an optimal policy with respect to each controller.

Recall that we are interested in ϵ-redundant controllers that represent a class of policies larger than the set of redundant optimal policies in order to offer more flexibility. Note that when we execute an ϵ-ascending policy for $\epsilon \neq 1$, at every step we deviate from optimality. Thus we need a measurement of loss of optimality for such policies.

Proposition 4.1 The expected maximum loss of optimality incurred by executing any ϵ-ascending policy in an ϵ-redundant controller $\mathcal{C}$, in some state $\mathbf{s}$ is bounded by:

$$\mathbf{LOSS}_{\mathcal{C}}(\mathbf{s}) \triangleq -(\frac{1-\epsilon}{\epsilon})\mathcal{V}^*(\mathbf{s})$$

This can be verified directly from the ϵ-optimality (Definition 4.4) property of any ϵ-ascending policy. Based on the Proposition 4.1, we can compute bounds for the expected loss of optimality when an ϵ-ascending policy is executed until it terminates in some goal state.

Proposition 4.2 Let π be an ϵ-ascending policy in a minimum cost-to-goal ϵ-redundant controller $\mathcal{C}$. Let $\mathbf{loss}_{\pi}(\mathbf{s})$[4] represent the expected loss of optimality as result of execution

[4]We use lower case letters for representing the expected loss of optimality, and upper case letter for representing the maximum loss of optimality.

of π in some state $\mathbf{s}$ until termination in some goal state in $\mathcal{M}_{\mathcal{C}}$. Then $\mathbf{loss}_{\pi}(\mathbf{s})$ is bounded by:

$$\mathbf{LOSS}_{\mathcal{C}}(\mathbf{s}) \lceil \frac{-\mathcal{V}^{*}(\mathbf{s})}{\eta^{\pi}} \rceil \leq \mathbf{loss}_{\pi}(\mathbf{s}) \leq \mathbf{LOSS}_{\mathcal{C}}(\mathbf{s}) \lceil \frac{-\mathcal{V}^{*}(\mathbf{s})}{\kappa^{\pi}} \rceil$$

Recall that according to the Theorem 4.2, the policy π being initiated in state $\mathbf{s}$ will terminate in some goal state on average in at most $\lceil \frac{-\mathcal{V}^{*}(\mathbf{s})}{\kappa^{\pi}} \rceil$ steps and at least $\lceil \frac{-\mathcal{V}^{*}(\mathbf{s})}{\eta^{\pi}} \rceil$ steps, where κ^{π} and η^{π} are the minimum and maximum ascent rates of the policy π. According to the Proposition 4.1, at every step the loss of optimality is bounded by $\mathbf{LOSS}_{\mathcal{C}}(\mathbf{s})$. This is true for all the subsequent states, since π is an ascending policy and all the subsequent states visited as the result of executing π, in average have larger values than $\mathcal{V}^{*}(\mathbf{s})$, and since all the values are negative, $\mathcal{V}^{*}(\mathbf{s})$ has the largest absolute value among all. Therefore the expected loss is bounded by the time taken for completion multiplied by the maximum loss of optimality in every time step.

Recall that any ϵ-redundant controller $\mathcal{C}$ represents a class of ϵ-ascending policies. Based on Proposition 4.2, we can derive bounds for the expected loss of optimality that characterizes the optimality of the controller:

Proposition 4.3: Let $\mathcal{C}$ be an ϵ-redundant controller defined over a minimum cost-to-goal MDP $\mathcal{M}$. Let $\mathbf{loss}_{\mathcal{C}}(\mathbf{s})$ represent the expected loss of optimality as result of execution of any ϵ-ascending policy $\pi \in \chi^{\epsilon}_{\mathcal{C}}$ in some state $\mathbf{s}$ until termination in some goal state in $\mathcal{M}$. Then $\mathbf{loss}_{\mathcal{C}}(\mathbf{s})$ is bounded by:

$$l_{\mathcal{C}}(\mathbf{s}) \leq \mathbf{loss}_{\mathcal{C}}(\mathbf{s}) \leq L_{\mathcal{C}}(\mathbf{s})$$

where:

$$l_{\mathcal{C}}(\mathbf{s}) = \mathbf{LOSS}_{\mathcal{C}}(\mathbf{s})\ \sigma_{\mathcal{C}}(\mathbf{s})$$

$$L_{\mathcal{C}}(\mathbf{s}) = \mathbf{LOSS}_{\mathcal{C}}(\mathbf{s})\ \tau_{\mathcal{C}}(\mathbf{s})$$

This can be easily verified from the bounds provided by the Proposition 4.2, and the fact that $\tau_{\mathcal{C}}(\mathbf{s})$ and $\sigma_{\mathcal{C}}(\mathbf{s})$ represent the worst and best expected time of completion of the controller $\mathcal{C}$ being initiated in state $\mathbf{s}$ respectively (see Definition 4.9).

Theorem 4.7: Assume $\zeta = \{\mathcal{C}_1, \mathcal{C}_2\}$ is a set of prioritized minimum cost-to-goal ϵ-redundant controllers defined over MDP $\mathcal{M}$ with a ranking system $\{C_2 \triangleleft \mathcal{C}_1\}$. Let the policy π_{seq} denote the policy obtained by performing the algorithm *Sequence*, and let the policy π_{coart} denote the policy obtained by performing the algorithm *Coarticulate*. Assuming that the current state of the system is $\mathbf{s}$, the policy π_{coart} is strictly more optimal than the policy π_{seq} if the following conditions hold:

1. In every state $\mathbf{s}'$: $\mathcal{A}^{\epsilon}_{\mathcal{C}_1}(\mathbf{s}') \cap \mathcal{A}^{\epsilon}_{\mathcal{C}_2}(\mathbf{s}') \neq \emptyset$
2. In the current state $\mathbf{s}$: $\omega_{\mathcal{C}_1}(\mathbf{s}) > \tau_{\mathcal{C}_2}(\mathbf{s})$
3. In the current state $\mathbf{s}$: $L_{\mathcal{C}_1}(\mathbf{s}) < l_{\mathcal{C}_2}(\mathbf{s})$

Proof: Let $\mu_{coart}(\mathbf{s})$ and $\mu_{seq}(\mathbf{s})$ respectively denote the expected number of steps for the policies π_{coart} and π_{seq} for achieving all the subgoals $\{\omega_1, \omega_2\}$ associated with the set of controllers, when it is initiated in state $\mathbf{s}$. Following the proof of the Theorem 4.6, the first condition implies that at any state both controllers will be optimized by π_{coart}. Based on the second condition (i.e., $\omega_{\mathcal{C}_1}(\mathbf{s}) > \tau_{\mathcal{C}_2}(\mathbf{s})$), which implies that π_{coart} will achieve $\mathcal{C}_2$ while committing to $\mathcal{C}_1$, thus the time required for completion of π_{coart} is bounded by the worst expected time to achieve $\mathcal{C}_1$:

$$\mu_{coart}(\mathbf{s}) < \mu_{seq}(\mathbf{s})$$

where the right hand side is the expected time of completion for the policy μ_{seq}. If the third condition holds, the loss incurred by the execution of the policy π_{coart} will be compensated by optimizing both subgoals concurrently and hence π_{coart} will be strictly more optimal than the policy π_{seq}.

4.7 Experiments

In this section, we present experimental results analyzing redundant controllers and the performance of the *Coarticulate* algorithm described in section 4.5. Note that in this set of experiments, we do not explicitly address the concurrent decision making, and we focus more on evaluating the general form of coarticulation in MDPs. In the next chapter, we solve a concurrent decision making problem using a simulated platform.

Figure 4.5(a) shows a 10×10 grid world where an agent is to visit a set of prioritized locations marked by $G_1, \ldots, G_m$ (in this example $m = 4$). The agent's goal is to achieve all the subgoals according to their degree of significance in the shortest amount of time.

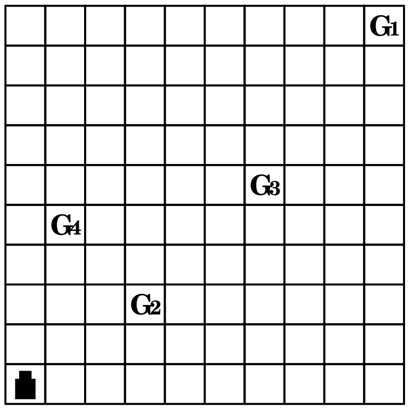


Figure 4.5. A 10×10 grid world where an agent is to visit a set of prioritized subgoal locations.

We model this problem by an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$, where $\mathcal{S}$ is the set of states consisting of 100 locations in the room, and $\mathcal{A}$ is the set of actions consisting of eight stochastic navigation actions (four actions in the compass direction, and for diagonal actions). Each action moves the agent in the corresponding direction with probability p and fails with probability $(1-p)$ (in all of the experiments we used success probability $p = 0.9$).

Upon failure the agent is randomly placed in one of the eight-neighboring locations with equal probability. If a movement would take the agent into a wall, then the agent will remain in the same location. The agent also receives a reward of -1 for every action executed. We assume that the agent has access to a set of controllers $\mathcal{C}_1, \ldots, \mathcal{C}_m$, associated with the set of subgoal locations $G_1, \ldots, G_m$. A controller $\mathcal{C}_i$ is a minimum cost-to-goal subgoal option $\mathcal{C}_i = \langle \mathcal{M}_{\mathcal{C}_i}, \mathcal{I}, \beta \rangle$, where $\mathcal{M}_{\mathcal{C}_i} = \mathcal{M}$, $\mathcal{I}$ includes any locations except for the subgoal location, and β forces the option to terminate only in the subgoal location.

4.7.1 ϵ-ascending policies

First, for every controller $\mathcal{C}_i$ we compute the set of admissible policies. Figure 4.6(a) shows the optimal policy of the controller $\mathcal{C}_1$ (navigating the agent to the location G_1). Figures 4.6(b) and 4.6(b) show the class of ϵ-redundant policies for $\epsilon = 0.95$ and $\epsilon = 0.90$ respectively. Note that by reducing ϵ, we obtain a larger set of admissible policies although less optimal.

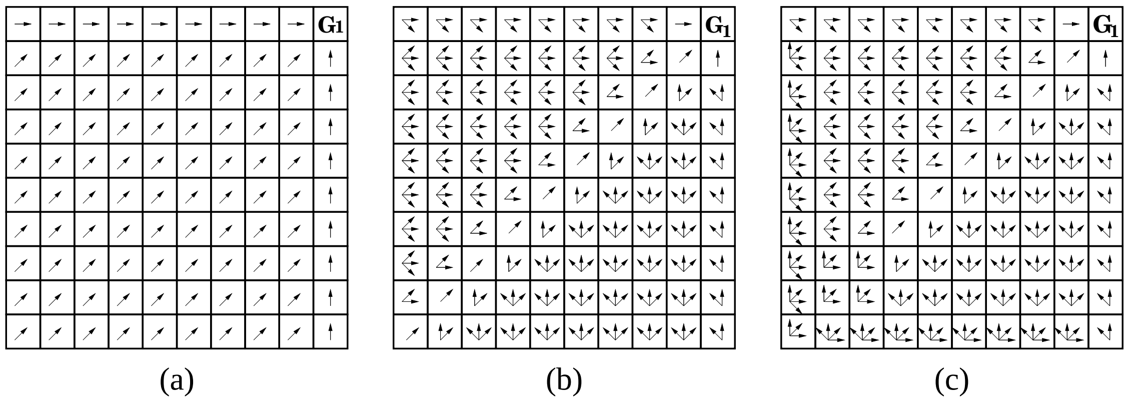


Figure 4.6. (a) The optimal policy associated with the subgoal G_1; (b) The class of ϵ-ascending policies for $\epsilon = 0.95$; (c) The ϵ-ascending policy for $\epsilon = 0.90$.

4.7.2 Performance

In this set of experiments, we study the performance of the coarticulation algorithm in the above grid-world problem. Our results compare the performance of the following approaches:

• **Sequential:** We use *Sequence* (Algorithm 2) where we achieve the subgoals of the problem by sequentially executing the controllers associated with them.

• **Coarticulate:** We use the coarticulation framework based on the algorithm *Coarticulate* (Algorithm 3) for solving the problem.

In the first set of experiments, we fixed the number of subgoals. At the beginning of each episode the agent is placed in a random location, and a fixed number of subgoals (in our experiments $m = 4$) are randomly selected. Next, the set of admissible policies (using $\epsilon = 0.9$) for every subgoal is computed. Figure 4.7(a) shows the performance of both planning methods, for every starting location in terms of number of steps for completing the overall task. The concurrent planning method consistently outperforms the sequential planning in all starting location.

Next, for the same task, we measure how the performance of both methods varies by varying ϵ, when computing the set of ϵ-ascending policies for every subgoal. Figure 4.8 shows the performance of both methods and Figure 4.9 shows the average number of subgoals committed by the agent – averaged over all states – for different values of ϵ. In this experiment, we varied ϵ from 0.6 to 1.0. All of these results are also averaged over 100 episodes, each consisting of 10 trials.

Note that for $\epsilon = 1$, the only admissible policy is the optimal policy and thus it does not offer much flexibility with respect to the other subgoals. This can be also seen in Figure 4.9 in which the policy generated by the merging algorithm for $\epsilon = 1.0$ has the minimum commitment to the other subgoals. As we reduce ϵ, we obtain a larger set of admissible policies, thus we observe improvement in the performance. However, the more we reduce ϵ, the lesser optimal admissible policies we obtain. Thus, the performance degrades (here

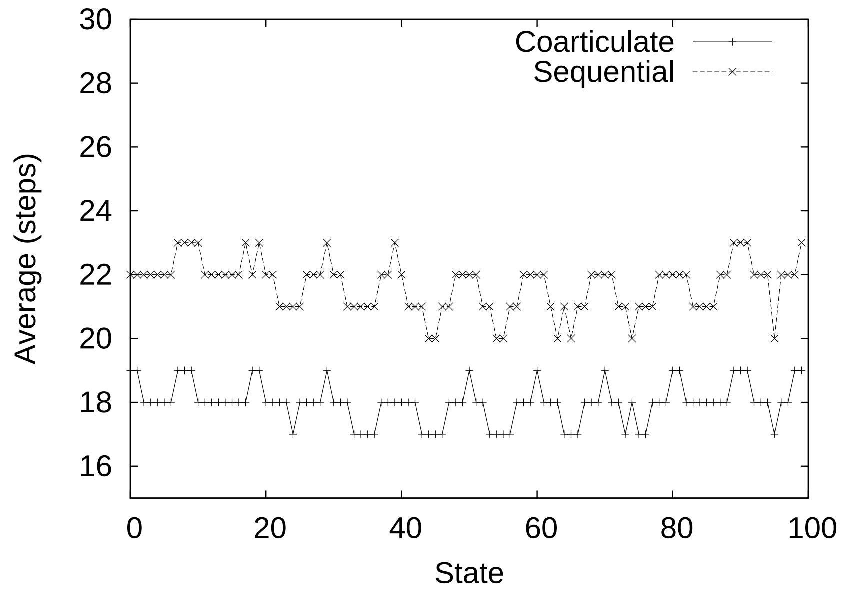


Figure 4.7. Performance of both planning methods in terms of the average number of steps in every starting state.

we can observe it for the values below $\epsilon = 0.85$). Figure 4.9 also shows by relaxing optimality (reducing ϵ), the policy generated by the merging algorithm commits to more subgoals simultaneously.

4.7.3 Performance: Varying the Number of Subtasks

In the final set of experiments, we set ϵ to 0.9 and varied the number of subgoals from $m = 2$ to $m = 50$ (all of these results are averaged over 100 episodes, each consisting of 10 trials). Figure 4.10 shows the performance of both planning methods. It can be observed that the concurrent method consistently outperforms the sequential method by increasing the number of subgoals.

Figure 4.11 shows the difference between the two plots in Figure 4.10 which demonstrates by increasing the number of subgoals, the performance of the concurrent method improves over the sequential method. This result is backed up by Figure 4.12 which shows

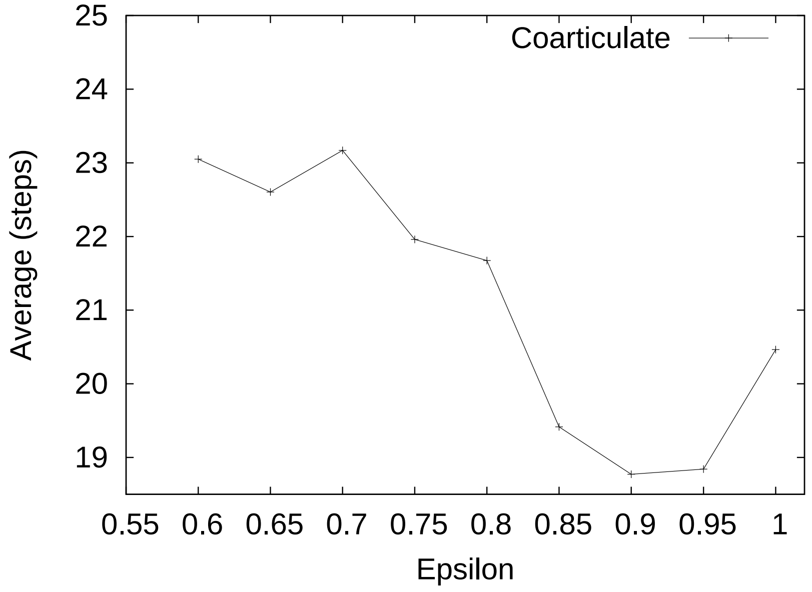


Figure 4.8. Performance for different values of ϵ.

by increasing the number of subgoals introduced in the problem, the average number of subgoals simultaneously committed by the concurrent method increases.

4.8 Concluding Remarks

In this chapter, we introduced an approach for alleviating the curse of dimensionality in concurrent decision making. The key idea in our approach is based on the fact that in many complex concurrent decision making problems, the overall objective is intuitively decomposable in terms of concurrent optimization of a set of simpler subgoals of the problem. We argue that concurrency naturally emerges from concurrent optimization of such subgoals when the system admits a redundant set of resources that could be simultaneously allocated for achieving them. Thus rather than posing the problem as a learning problem with an exponentially large set of concurrent actions – which is intractable for every possible combination of subgoals– we assume that the agent generates parallel execution plans

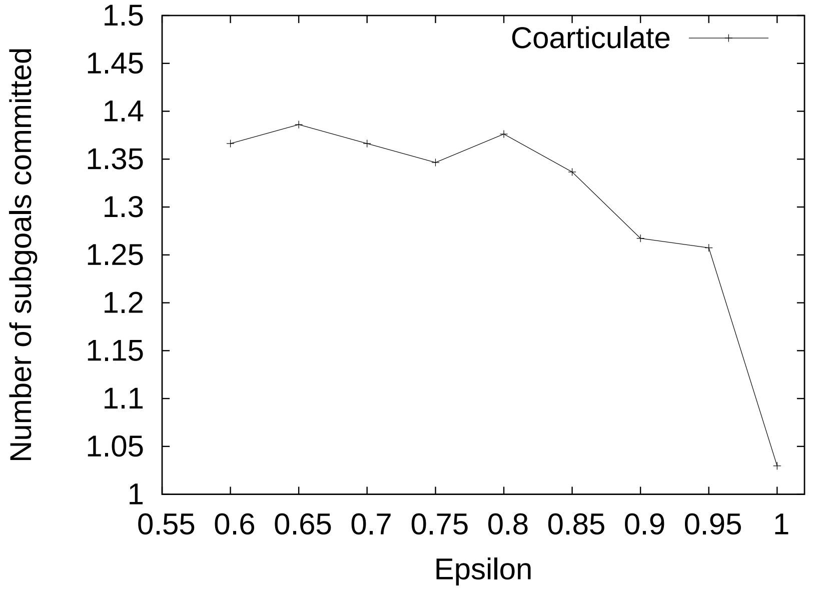


Figure 4.9. Average number of subgoals concurrently committed for different values of ϵ.

by dynamically combining a set of previously acquired skills, each designed for achieving a subgoal of some sort.

We used the term coarticulation to refer to our approach because of the similarities it bears with the coarticulation phenomenon in motor control research. We introduced the redundant controllers which serve as the building blocks of the coarticulation framework. Each controller represents a class of ascending policies which represent the degree of flexibility that such controllers afford. We presented algorithms for computing such policies and for performing coarticulation in a general class of problems that also encompass the class of concurrent decision making problems.

We also showed that while the computational complexity of computing the redundant-sets is polynomial in the set of states and actions (Equation 4.6), the computational complexity of the coarticulation approach is only polynomial in the size of the redundant-sets (Equation 4.7). In general the size of the redundant-sets are significantly smaller than the size of the set of all concurrent actions. Theorem 4.1 also states that the size of a redundant-set does not arbitrarily grow as we reduce the flexibility parameter ϵ. Thus by choosing a

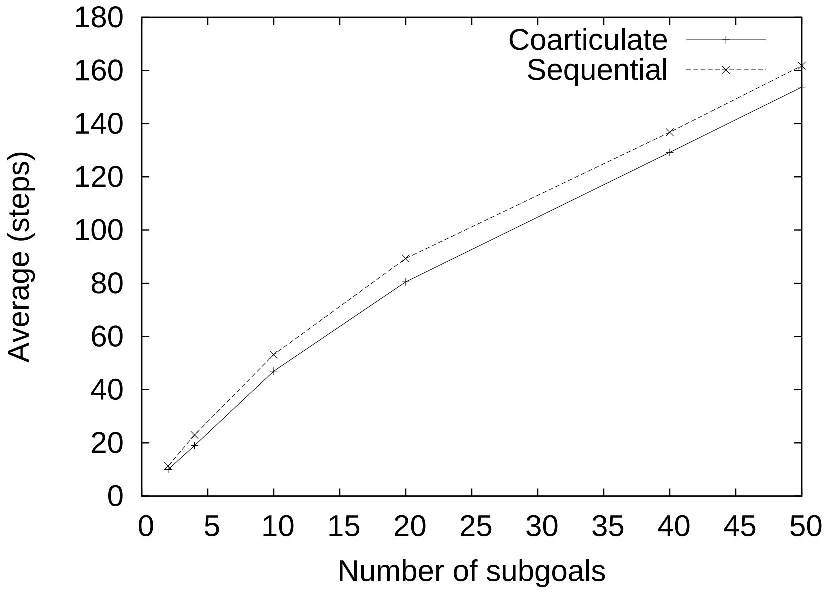


Figure 4.10. Performance of the planning methods in terms of the average number of steps in every starting state.

feasible size for the redundant-sets, we can tractably perform coarticulation in every state of the problem.

We also presented a set of theoretical results characterizing each controller, and also analyzing the performance of the coarticulation algorithm in terms of the established bounds on the time required for the algorithm to accomplish the goals of the problem. In particular in Theorem 4.4, we showed that our coarticulation algorithm (Algorithm 3) generates a policy that is ascending on $\mathcal{V}^*_{lex}(\mathbf{s})$ and thus it finds an approximate solution for the COART model. We also presented a theorem (Theorem 4.7) that derives conditions under which coarticulation approach is theoretically proven to outperform the sequential, or non-coarticulated solutions.

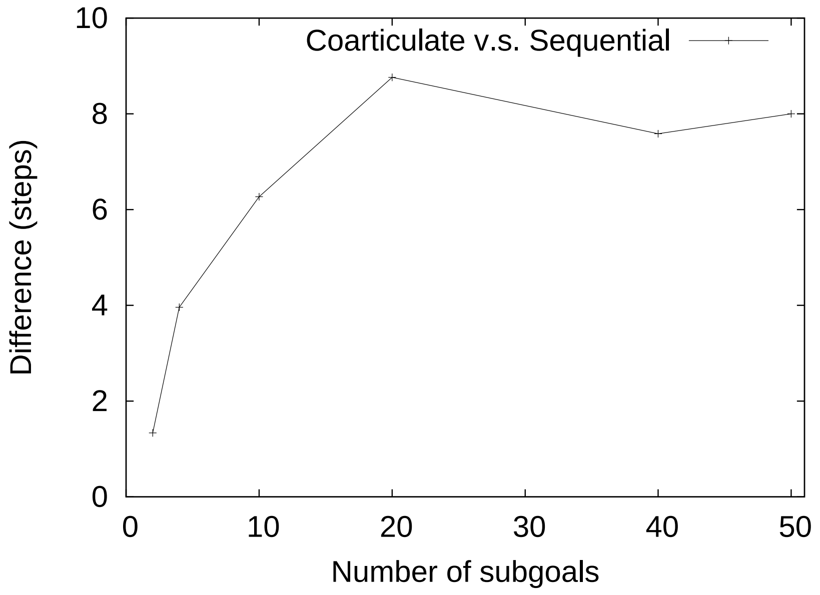


Figure 4.11. The difference of performance of the planning methods.

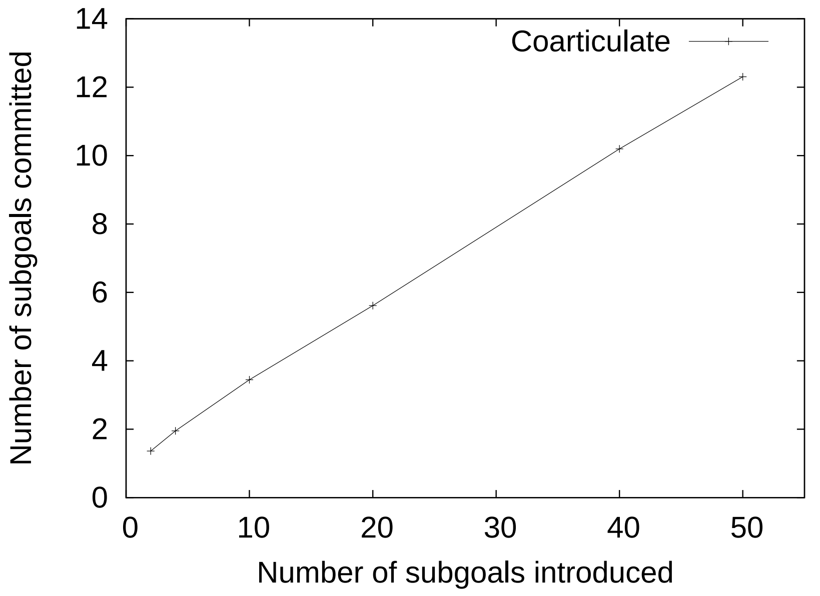


Figure 4.12. Average number of subgoals concurrently committed v.s. the total number of the subgoals.

CHAPTER 5
SCALING COARTICULATION TO LARGE DOMAINS

In the previous chapter we introduced a decision theoretic framework for modeling a form of coarticulation in a class of problems where the problem objective can be expressed in terms of concurrent optimization of a set of prioritized subgoals. We demonstrated that coarticulation can be viewed as one natural way for generating concurrency in the system. At the heart of the coarticulation framework lies the ϵ-redundant controllers which serve as the basic blocks for performing coarticulation. Each controller represents a class of redundant ϵ-ascending policies and offers an affordable degree of flexibility for achieving a subgoal of some sort. Such flexibility enables the agent to simultaneously commit to multiple subgoals, while committing more strongly to those of higher priority. As a result, when a system offers redundancy based on the multiplicity of DOFs, by coarticulating among the subgoals of a problem, it can generate parallel execution plans for achieving them.

In Chapter 4 we showed that the *Coarticulate* algorithm (Algorithm 4.3) alleviates the curse of dimensionality in concurrent decision making. The computational costs of the coarticulation framework are due to two major steps:

• The computational costs of computing the redundant-sets of ϵ-ascending policies for the set of coarticulatory controllers (Algorithm 1). The computational complexity of this step is polynomial in the set of states and the set of concurrent actions in the system (Equation 4.4.6).

• The computational costs of performing coarticulation (Algorithm 3). We showed that this complexity is quadratic in the size of the redundant-sets (Equation 4.4.7).

Although by applying coarticulation we reduced the intractable exponential complexity to a tractable quadratic complexity for finding an approximate solution, the complexity of computing the redundant-sets remains intractable (i.e., the computational complexity of this step is still exponential in the set of primitive skills that the agent possesses). In the rest of this chapter we present approximate algorithms for scaling the coarticulation framework to such problems.

5.1 Approximate Algorithms for Performing Coarticulation

In order to perform coarticulation for solving a concurrent decision making problem, we first need to design a set of redundant controllers. Each controller is parametrized by the parameter ϵ, which controls the flexibility offered by the controller and identifies a class of ϵ-ascending policies in the controller. Computation of the set of ϵ-ascending policies (or equivalently, the redundant-sets) is the key step of the coarticulation approach.

In Chapter 4 we described an algorithm for computing the redundant-sets in a controller (Algorithm 4.4.6) when the agent has access to the optimal state value function that optimizes the subgoal associated with a controller. Briefly, the computation of this step involves verifying the Ascendancy, and ϵ-optimality conditions (Equations 4.3 and 4.4 in Chapter 4) for every action, in every state of the controller. As we discussed in the introduction, this is impractical when there exists an exponentially large number of concurrent actions. Furthermore, assuming that we have access to the true optimal value function of a controller is also unrealistic. This is because the standard RL methods without any generalization techniques cannot also cope with the curse of dimensionality in the action space.

In order to address this challenge, we employ function approximation techniques (Bertsekas and Tsitsiklis, 1997; Sutton and Barto, 1998) for computing an approximate value function. By exploiting the generalization offered by such approximation techniques, we present a set of techniques for compactly computing the redundant-sets (Rohanimanesh and Mahadevan, 2005). The key idea in our approach is rather than verifying the Ascendancy and ϵ-optimality conditions for an exponential set of concurrent actions – which is intractable – we only verify them for the top $\mathbf{h}$ concurrent actions that have the top $\mathbf{h}$ best state-action values in every state $\mathbf{s}$. We demonstrate that for a certain class of value function approximation methods – namely the linear function approximation – our approximate algorithm computes the top $\mathbf{h}$ concurrent actions with computational complexity logarithmic in $\mathbf{h}$, and the computational complexity polynomial in $\mathbf{h}$ for performing the coarticulation algorithm. The parameter $\mathbf{h}$ is an input parameter which can be tuned to balance the tradeoffs between the computational complexity and the flexibility of the controller.

More formally, let $\mathcal{C}$ be an ϵ-redundant controller defined over an MDP $\mathcal{M}$ modeling a concurrent activity. To develop a representation of a concurrent action, we assume that set of concurrent actions $\mathcal{A}$ are described via a set of discrete action variables $\mathbf{a} = \{a_i\}_{i=1}^{n}$. For clarity, we use bold-face letters for referring to a concurrent action (e.g., $\mathbf{a}$), and a normal form when referring to one of the constituent action elements (e.g., $a_i \in \mathbf{a}$). Each action variable a_i takes on discrete values from some finite domain $Dom(a_i)$. Without loss of generality, we assume that:

$$\forall a_i \in \mathbf{a}, \quad |Dom(a_i)| = d$$

We also use the notation $\bar{\mathbf{a}}$ to refer to a particular assignment of the action variables $a_i \in \mathbf{a}$.

By exploiting the action structure in this model, we can use function approximation techniques (Bertsekas and Tsitsiklis, 1997; Sutton and Barto, 1998) in order to learn an approximation of the optimal state-action value function. In particular, we are interested in a class of function approximation methods known as the linear function approximation. As we will demonstrate later in this section, the linear additive nature of such approximation techniques could be efficiently exploited for compactly computing the redundant-sets.

Assume that the optimal state-action value function $\mathcal{Q}^*$ associated with the controller $\mathcal{C}$ is approximated using linear function approximation techniques and admits the following linear additive form:

$$\begin{aligned} \mathcal{Q}^*(\mathbf{s}, \{a_i\}_{i=1}^n) &\approx \bar{\mathcal{Q}}^*(\mathbf{s}, \mathbf{a}) \\ &= \sum_{i=1}^{m} \mathcal{Q}_i(\mathbf{s}, \mathbf{u}_i) \end{aligned} \tag{5.1}$$

where each $\mathcal{Q}_i(\mathbf{s}, \mathbf{u}_i)$ is a *local basis* function defined over the set of states $\mathbf{s}$ and a subset of action variables $\mathbf{u}_i \subset \mathbf{a}$.

This form of approximation is based on the intuition that although in general value functions might not be structured, there are many domains that admit exploitable additive structure (Koller and Parr, 1999). In general it is intuitive to linearly approximate the value of a state in terms of the set of subgoals achieved at that state. This view of value function has also a long history in *multi-attribute utility theory* (Keeney and Raiffa, 1993), and in particular seems very suitable for the concurrent decision making problem.

By exploiting such additive structure of the approximate state-action value function, we describe a tractable algorithm (Rohanimanesh and Mahadevan, 2005) for computing the redundant-sets, as the key component for performing coarticulation. We introduce an operator $\Gamma_{\mathbf{a}}^{\mathbf{h}}$ which returns the top $\mathbf{h}$ maximum values and the assignments to the set of the action variables $\{a_i \in \mathbf{a}\}$ for a function $\mathcal{Q}^*(\mathbf{s}, \mathbf{a})$:

Definition 5.1: Let $\mathcal{F}$ represent the space of all state-action value functions $\mathcal{Q}^*(\mathbf{s}, \mathbf{a})$ defined over the set of states $\mathbf{s} \in \mathcal{S}$ and a set of action variables represented by $\mathbf{a} = \{a_1, a_2, \ldots, a_n\}$. We introduce an operator:

$$\Gamma_{\mathbf{a}}^{\mathbf{h}} : \mathcal{F} \to (\mathbb{R} \times \mathbb{N}^n)^{\mathbf{h}}$$

where $\Gamma_{\mathbf{a}}^{\mathbf{h}} \left(\mathcal{Q}^*(\mathbf{s}, \mathbf{a})\right)$ returns the top $\mathbf{h}$ maximum values of $\mathcal{Q}^*(\mathbf{s}, \mathbf{a})$ and the top $\mathbf{h}$ assignments to the set of action variables $\mathbf{a}$, that achieves each value. Note that each action $\mathbf{a}$ can be expressed in terms of a set of assignments to its $\mathbf{n}$ constituent actions variables which takes on discrete values from a finite domain $\{0, 1, \ldots, d\}$. Thus the operator $\Gamma_{\mathbf{a}}^{\mathbf{h}}$ returns $\mathbf{h}$ pairs, where each pair consists of a vector $\mathbf{a} \in \mathbb{N}^{\mathbf{n}}$ and its value.

By exploiting the additive structure of the approximate state-action value function (Equation 5.1) we can use an algorithm in spirit similar to the variable elimination algorithm in Bayesian networks (Jordan and Bishop, 2002) and efficiently compute $\Gamma^{\mathbf{h}}_{\mathbf{a}}\left(\bar{\mathcal{Q}}^*(\mathbf{s},\mathbf{a})\right)$. Our approach is inspired by the action selection algorithm introduced in (Guestrin et al., 2002) that actually solves the special case for $\mathbf{h} = 1$ (i.e., $\Gamma^{1}_{\mathbf{a}}$, which is the standard $\max_{\mathbf{a}}$ operator). It is also closely related to the problem of finding the $\mathbf{h}$ most probable configurations in probabilistic expert systems (Nilsson, 1998).

The key idea is by exploiting the additive structure of the state-action value function, rather than summing all local functions and then performing the $\Gamma^{\mathbf{h}}_{\mathbf{a}}$ operator, we perform it over one variable one at a time. At every step we use only summands that involve the eliminated variable. For example, consider the following state-action value function defined over a set of action variables $\mathbf{a} = \{a_1, a_2, a_3\}$:

$$\begin{aligned}\bar{\mathcal{Q}}^*(\mathbf{s},\mathbf{a}) &= \bar{\mathcal{Q}}^*(\mathbf{s}, a_1, a_2, a_3)\\ &= \mathcal{Q}_1(\mathbf{s}, a_1) + \mathcal{Q}_2(\mathbf{s}, a_1, a_2) + \mathcal{Q}_3(\mathbf{s}, a_2, a_3)\end{aligned}$$

by propagating the $\Gamma^{\mathbf{h}}_{\mathbf{a}}$ operator through the additive structure of the function, and performing the variable elimination algorithm we obtain:

$$\begin{aligned}&\Gamma^{\mathbf{h}}{}_{\{a_1,a_2,a_3\}}\left(\bar{\mathcal{Q}}^*(\mathbf{s}, a_1, a_2, a_3)\right) =\\ &\Gamma^{\mathbf{h}}{}_{\{a_1,a_2,a_3\}}\left(\mathcal{Q}_1(\mathbf{s}, a_1) + \mathcal{Q}_2(\mathbf{s}, a_1, a_2) + \mathcal{Q}_3(\mathbf{s}, a_2, a_3)\right) =\\ &\Gamma^{\mathbf{h}}{}_{\{a_1\}}\left(\mathcal{Q}_1(\mathbf{s}, a_1) \oplus \Gamma^{\mathbf{h}}_{\{a_2\}}\left(\mathcal{Q}_2(\mathbf{s}, a_1, a_2) \oplus \Gamma^{\mathbf{h}}_{\{a_3\}}\left(\mathcal{Q}_3(\mathbf{s}, a_2, a_3)\right)\right)\right)\end{aligned}$$

At every step, when an action variable is eliminated, for every setting of the non-eliminated variables that are involved in the corresponding summands, a set of top $\mathbf{h}$ best values and the assignment to the eliminated variable is returned. In the above example, when the action variable a_3 is eliminated, the local operation returns the top $\mathbf{h}$ best values of $\mathcal{Q}_3(\mathbf{s}, a_2, a_3)$ and the corresponding assignment to a_3, for every value of the non-eliminated action variable a_2.

Note that in the above equation, we used the special sum operator $\oplus$, because at each elimination step the summation is performed over *many-to-one* functions that return the top $\mathbf{h}$ maximum values of the past elimination steps. We then need to perform a *cross-summation* across different sets of size $\mathbf{h}$ each pertaining to a past elimination step in order to obtain the updated top $\mathbf{h}$ maximum values, as a result of the elimination of the next variable. Note that in the special case of $\mathbf{h} = 1$, the $\oplus$ operator turns into the standard plus operator, and no cross-summation is required, because each elimination step returns only one maximum value.

Algorithm 4 Function $\Gamma^{\mathbf{h}}_{\mathbf{a}}$

```
Inputs:
  $\mathbf{s}$                                    \\ Current state
  $\sum_{i=1}^{m} \mathcal{Q}_i(\mathbf{s}, \mathbf{u}_i)$     \\ $\bar{\mathcal{Q}}^*$ function
  $\{a_i\}$                                      \\ Elimination order
  $\mathbf{h}$                                    \\ Number of top max elements
Outputs:
  $\langle \bar{\mathbf{a}}_i, v_i \rangle_{i=1}^{h}$          \\ Top $\mathbf{h}$ assignments and values

1: Let $\mathcal{F} = \{\mathcal{Q}_i(\mathbf{s}, \mathbf{u}_i)\}_{i=1}^{m}$       \\ set of summands
2: while not all variables eliminated do
3:     Pick the next variable $a_i$
4:     Extract all summands $\{\mathcal{H}_j\}$ from $\mathcal{F}$
           that involve $a_i$
5:     Perform: $\mathcal{H}_i \leftarrow \oplus(\{\mathcal{H}_j\})$
6:     Eliminate $a_i$ from $\mathcal{H}_i$ to obtain $\mathcal{H}_i^-$
7:     Add $\mathcal{H}_i^-$ to $\mathcal{F}$
8: end while
```

The above procedure is summarized in Algorithm 4. The key computational steps are the steps 5 and 6 of this algorithm. Before describing the details of these two steps, first we introduce some useful notation. Let $\mathcal{H}(\mathbf{w})$ denote a one-to-many mapping defined over a set of variables $\mathbf{w}$. Figure 5.1 shows a tabular view of this function, and demonstrates the details of the computations performed in the step 5 of the Algorithm 4. For every setting of variables $\bar{\mathbf{w}}$, it returns the sorted top $\mathbf{h}$ values and assignments to the subset of eliminated variables from the previous steps (tables $T(\bar{\mathbf{w}})$ in Figure 5.1).

5.1.1 The Cross-Summation Step (Function $\oplus$)

Before the elimination algorithm starts, we can represent each summand $\mathcal{Q}_i(\mathbf{s}, \mathbf{u}_i)$ as some function $\mathcal{H}_i(\mathbf{u}_i)$ (to simplify notations, we omit the state $\mathbf{s}$ from the notations), where every assignment of the variables $\bar{\mathbf{u}}_i$ is mapped to a single value $\mathcal{Q}_i(\mathbf{s}, \bar{\mathbf{u}}_i)$. Assume that the algorithm is at iteration i, where the variable a_i is selected for elimination. Let $\{\mathcal{H}_j\}_{j=1}^k$ be the set of summands that involve the variable a_i. Also let $\mathbf{y}_i$ denote the rest of the variables involved in $\{\mathcal{H}_j\}_{j=1}^k$ that are connected to a_i, and let $\mathbf{w}_i = \mathbf{y}_i \cup \{a_i\}$. As shown in Figure 5.1, for every setting of variables $\bar{\mathbf{w}}_i$ summand $\mathcal{H}_j$ returns a sorted top $\mathbf{h}$ values and also the assignments to a subset of past eliminated variables (represented as tables $T_j(\bar{\mathbf{w}}_i)$). There are k such tables and we need to compute the top $\mathbf{h}$ maximum values from the set of all cross summations of k elements, one from each table $T_j(\bar{\mathbf{w}}_i)$.

There are $\mathbf{h}^k$ such values and a naive approach would first compute the whole $\mathbf{h}^k$ summations, and then extract the top $\mathbf{h}$ maximum values, with the computational complexity of:

$$O(\mathbf{h}^k(k-1) + \mathbf{h}\, log(\mathbf{h}))$$

where the first term denotes the complexity of computing the summations, and the second term denotes the complexity of sorting these values. However considering that each table is sorted, we can perform the above computation more efficiently. Rather than summing all the values across all tables, we perform the summation over two tables at a time, and extract a new table with the top $\mathbf{h}$ maximum values of the pairwise table summation. We then repeat it for the rest of the tables. When performing the pairwise cross summation over two tables, we only need to perform the summation only over the top $\sqrt{\mathbf{h}}$ elements from each table, since the tables are sorted. The computational complexity of this approach is:

$$O((k-1)(\mathbf{h} + \mathbf{h}\ log(\mathbf{h}))) \tag{5.2}$$

where $(\mathbf{k}-1)$ denotes the total number of times the above procedure is performed for $\mathbf{k}$ tables. Each time, there are $(\sqrt{\mathbf{h}})^2 = \mathbf{h}$ summations to be performed to produce a local list of top $\mathbf{h}$ elements. However this list is not sorted, therefore applying a standard sort algorithm, it adds an additional complexity of $\mathbf{h}\ log(\mathbf{h})$ for every step. The final sorted list of top $\mathbf{h}$ elements are stored in a new function $\mathcal{H}_i(\mathbf{w}_i)$ for the setting $\bar{\mathbf{w}}_i$.

5.1.2 The Variable Elimination Step

The details of the computations of step 6 of the Algorithm 4 are demonstrated in Figure 5.2. Note that step 5 returns a newly introduced function $\mathcal{H}(\mathbf{w}_i)$ that involves the variable a_i. The elimination takes place in step 6. First, a new function $\mathcal{H}^-(\mathbf{y}_i)$ is introduced that involves only the variables connected to a_i (i.e., $\mathbf{y}_i$). Every setting $\bar{\mathbf{y}}_i$ is mapped to d tables (where $|Dom(a_i)| = d$), each for one assignment of the variable a_i. Each table contains the top $\mathbf{h}$ values and settings for a subset of eliminated variables in the previous steps. We need to extract the top $\mathbf{h}$ values across these tables. There are d sorted tables of size $\mathbf{h}$, and we can extract the top $\mathbf{h}$ values across them with the computational complexity of:

$$O(\mathbf{h}\ .\ d) \tag{5.3}$$

The new set of values are then stored in the function $\mathcal{H}_i^-(\mathbf{y}_i)$ for the setting $\bar{\mathbf{y}}_i$ (see Figure 5.2).

5.1.3 Computational Complexity

The main computational costs are due to the steps 5 and 6 of Algorithm 4. The computational complexity of the step 5 of the algorithm is $O(\mathbf{h}^k(k-1)+\mathbf{h}\,log(\mathbf{h}))$ (Equation 5.2), and the computational complexity of the step 5 of the algorithm is $O(\mathbf{h}\ .\ d)$ (Equation 5.3). This yields the overall computational complexity of:

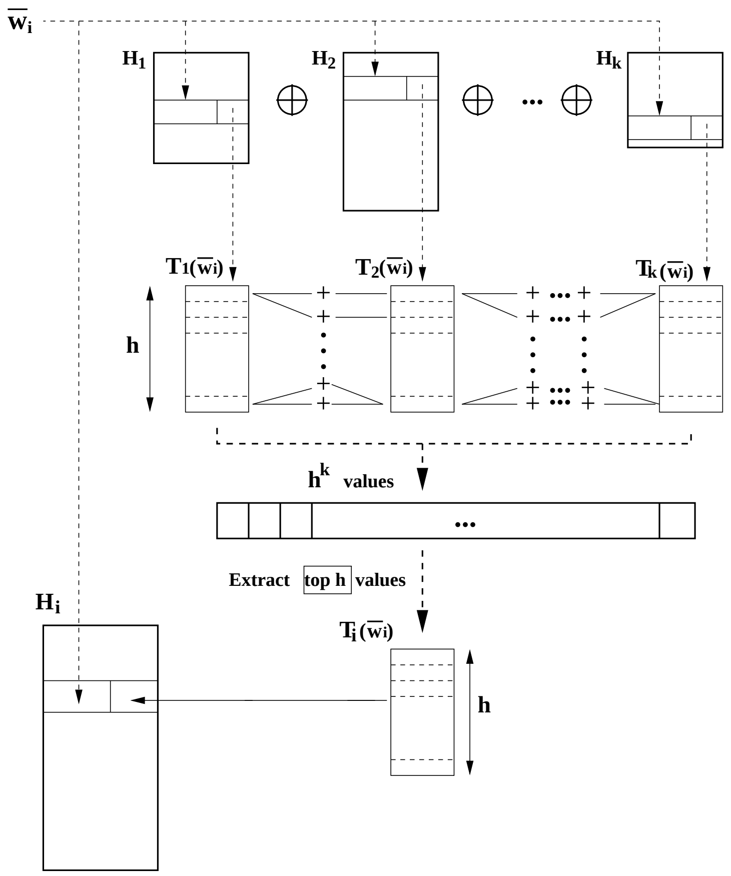


Figure 5.1. Visualization of step 5 of the Algorithm 4. Each function $\mathcal{H}_i$ returns a sorted table of size **h**. From cross summation of table values across $\mathcal{H}_i$ functions, a new table of the top **h** summations is produced.

$$O(n\ d^{\ |\mathbf{w}|}\ (k\ \mathbf{h}\ log(\mathbf{h}) + \mathbf{h}\ .\ d)) = O(n\ k\ d^{\ |\mathbf{w}|}\ \mathbf{h}\ log(\mathbf{h})) \tag{5.4}$$

for Algorithm 4. This complexity is logarithmic in **h**, and exponential in the *network width* (Dechter, 1999) induced by the structure of the approximate state-action value function.

5.1.4 Computing the Redundant-Sets

By performing Algorithm 4 in a state **s**, we obtain the top **h** concurrent actions and their values. We can then verify the ascendancy and ϵ-optimality conditions that we described in Definition 4.4, for each action. In either case, we need to compute $\mathcal{V}^*(\mathbf{s})$. From the Bellman optimality equation (Puterman, 1994; Sutton and Barto, 1998) we have:

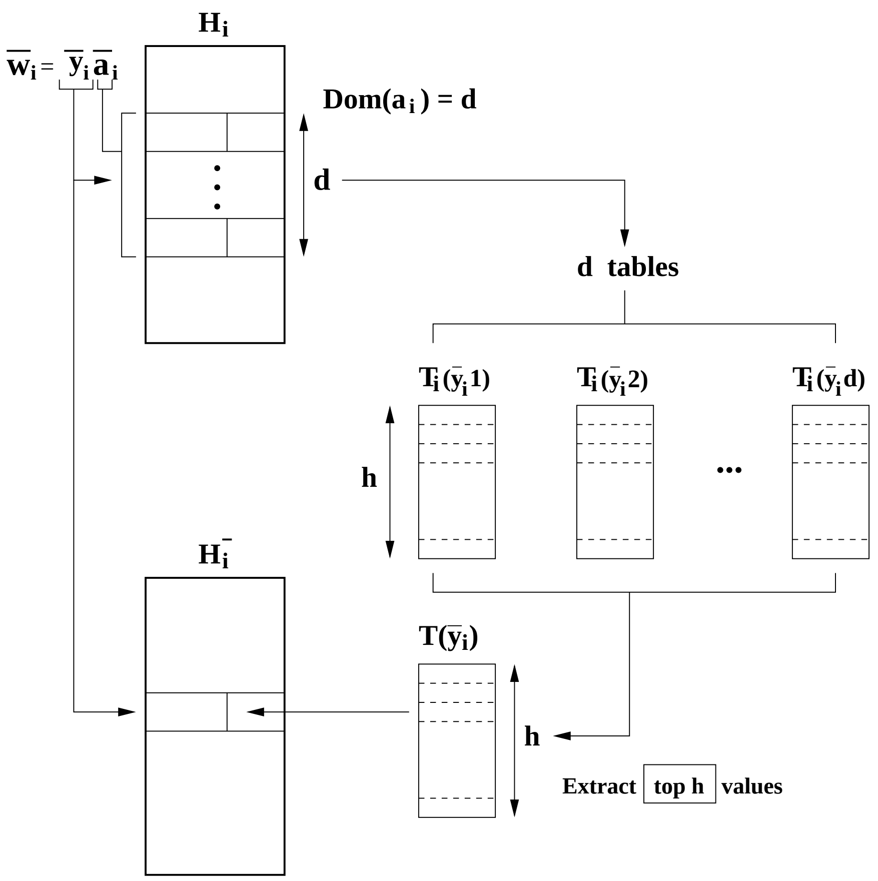


Figure 5.2. Visualization of step 6 of the Algorithm 4 where the variable a_i is eliminated.

$$
\begin{aligned}
\mathcal{V}^*(\mathbf{s}) &= \max_{\mathbf{a}} \mathcal{Q}^*(\mathbf{s}, \mathbf{a}) \\
&\approx \Gamma^1_{\mathbf{a}} \left(\bar{\mathcal{Q}}^*(\mathbf{s}, \mathbf{a})\right) \qquad (5.5)
\end{aligned}
$$

which can be computed using Algorithm 4. Thus verification of the ϵ-optimality condition can be efficiently done. For the ascendancy condition, we need to compute the expected optimal value of the next states given that the concurrent action $\bar{\mathbf{a}}$ is executed in state $\mathbf{s}$. Expanding the optimal state-action value function for the action $\bar{\mathbf{a}}$ yields:

$$
\mathcal{Q}^*(\mathbf{s}, \bar{\mathbf{a}}) = \mathcal{R}(\mathbf{s}, \bar{\mathbf{a}}) + \gamma E_{s' \sim \mathcal{P}_s^{\bar{\mathbf{a}}}}\{\mathcal{V}^*(\mathbf{s}')\}
$$

by subtracting $\gamma\mathcal{V}^*(\mathbf{s})$ from both sides and rearranging the terms, we obtain:

$$E_{s' \sim \mathcal{P}_s^{\bar{\mathbf{a}}}}\{\mathcal{V}^*(\mathbf{s}')\} - \mathcal{V}^*(\mathbf{s}) = \frac{1}{\gamma}(\mathcal{Q}^*(\mathbf{s},\bar{\mathbf{a}}) - \mathcal{R}(\mathbf{s},\bar{\mathbf{a}}) - \gamma\mathcal{V}^*(\mathbf{s})) \tag{5.6}$$

Algorithm 5 Function PruneActions($\mathbf{s}$, ϵ, $\langle \bar{\mathbf{a}}_i,\ v_i \rangle_{i=1}^{\mathbf{h}}$)

```
Inputs:
  $\mathbf{s}$                                   \\ Current state
  $\epsilon$                                   \\ The flexibility afforded by the controller $\mathcal{C}$ ($0 < \epsilon \leq 1$)
  $\sum_{i=1}^{m} \mathcal{Q}_i(\mathbf{s},\mathbf{u}_i)$             \\ $\bar{\mathcal{Q}}^*$ function
  $\{a_i\}$                               \\ Elimination order
  $\langle \bar{\mathbf{a}}_i,\ v_i \rangle_{i=1}^{h}$                   \\ Top $\mathbf{h}$ assignments and values
Outputs:
  $\mathcal{A}^\epsilon(\mathbf{s}) = \{\langle \bar{\mathbf{a}}_i,\ v_i \rangle\}$          \\ A set of $\epsilon$-ascending actions

1: Set $\bar{\mathcal{V}}^*(\mathbf{s}) = \Gamma^1_{\mathbf{a}}\ (\bar{\mathcal{Q}}^*(\mathbf{s},\mathbf{a}))$     \\ Compute the approximate value of this state
2: while (not all actions processed) do
3:     Pick the next top action $\bar{\mathbf{a}}$ and its value $\mathbf{v}$ from $\langle \bar{\mathbf{a}}_i,\ v_i \rangle_{i=1}^{\mathbf{h}}$
4:     Assert $\frac{1}{\gamma}(\mathbf{v} - \mathcal{R}(\mathbf{s},\bar{\mathbf{a}}) - \gamma\mathcal{V}^*(\mathbf{s})) > 0$     \\ Ascendancy condition
5:     Assert $\mathbf{v} \geq \frac{1}{\epsilon}\bar{\mathcal{V}}^*(\mathbf{s})$     \\ $\epsilon$-optimality condition
6:     If both conditions hold, add the pair $\bar{\mathbf{a}}$ to $\mathcal{A}^\epsilon(\mathbf{s})$
7: end while
```

Note that the right hand side of the Equation 5.6 can be efficiently computed for a concurrent action $\bar{\mathbf{a}}$ and can be used to verify the ascendancy condition. Function *PruneActions* (Algorithm 5) summarizes the above steps for extracting the set of ϵ-ascending actions from the set of top $\mathbf{h}$ best actions returned by the variable elimination step.

Algorithm 6 gives the complete algorithm for computing the redundant-sets for an ϵ-redundant controller $\mathcal{C}$ in some state $\mathbf{s}$.

5.1.5 Performing Coarticulation

Given a set of redundant controllers $\{\mathcal{C}_i\}_{i=1}^{k}$, we can perform Algorithm 6 and compute the redundant sets for each controller. For performing coarticulation, we can then use the algorithm *Coarticulate* (Algorithm 3 in Chapter 4). Note that the cardinality of the

Algorithm 6 Function ComputeApproximateRedundantSets($\mathbf{s}$, ϵ, $\mathbf{h}$)

```
Inputs:
  $\mathbf{s}$                                                    \\ Current state
  $\bar{\mathcal{Q}}^*(\mathbf{s},\mathbf{a}) = \sum_{i=1}^{m} \mathcal{Q}_i(\mathbf{s},\mathbf{u}_i)$      \\ $\bar{\mathcal{Q}}^*$ function
  $\{a_i\}$                                                \\ Elimination order
  $\epsilon$          \\ The flexibility afforded by the controller $\mathcal{C}$ ($0 < \epsilon \leq 1$)
  $\mathbf{h}$                                     \\ Number of top max elements
Outputs:
$\mathcal{A}^\epsilon(\mathbf{s})$                     \\ An $\epsilon$-redundant-set of cardinality at most $\mathbf{h}$

1: $\Theta \leftarrow \Gamma^{\mathbf{h}}_{\mathbf{a}}\left(\bar{\mathcal{Q}}^*(\mathbf{s},\mathbf{a})\right)$   \\ Perform the variable elimination algorithm for extracting the
      top $\mathbf{h}$ best values and concurrent actions
2: Return(PruneActions($\mathbf{s}$, $\epsilon$, $\Theta$))     \\ Prune the set of top $\mathbf{h}$ best actions to extract the
      $\epsilon$-ascending actions
```

redundant set for the controller $\mathcal{C}_i$ is at most $\mathbf{h}_i$. Thus the computational complexity of performing the algorithm *Coarticulate* in every state $\mathbf{s}$ can be expressed as:

$$O((k-1)\,(\max_i \mathbf{h}_i)^2) \tag{5.7}$$

which is quadratic in $\mathbf{h}$.

5.2 Experiments

In this section, we present a concurrent decision making task and apply the coarticulation approach based on the approximation techniques that we presented throughout this chapter. Figure 5.3[1] shows a simulated robot with three controllable resources, namely, the *eyes*, the *left arm*, and the *right arm*. The robot's task is to empty the dish-washer and stack the dishes into the dish-rack. Each arm of the robot at any time can be in three predefined positions: *washer*, *rack*, and *front* as shown in Figure 5.3. In order to make a successful arm movement from a source position to a target position, the robot needs to first fixate at the target position. The eyes of the robot can also fixate on any of these positions. The set of

[1]This example was suggested by Andrew G. Barto

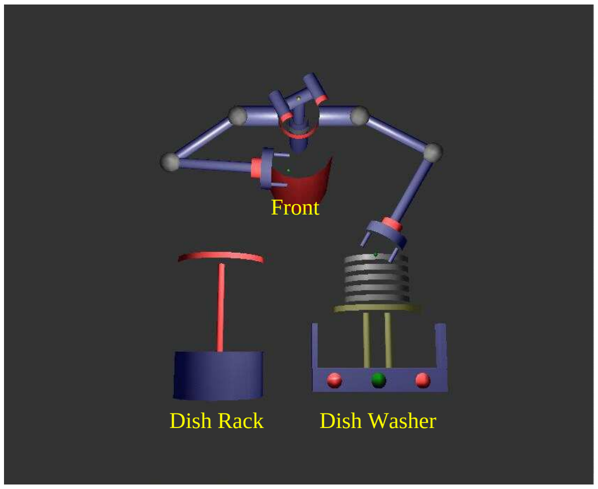


Figure 5.3. The robot's task is to empty the dish-washer and stack the dishes into the dish-rack.

Table 5.1. Action Variables

Left arm ($\mathbf{a}_l$)	Right arm ($\mathbf{a}_r$)	Eyes ($\mathbf{a}_e$)
pick	*pick*	*fixate-on-washer*
washer-to-front	*washer-to-front*	*fixate-on-front*
front-to-rack	*front-to-rack*	*fixate-on-rack*
rack-to-front	*rack-to-front*	*no-op*
front-to-washer	*front-to-washer*	
stack	*stack*	
no-op	*no-op*	

actions that the robot can perform is described via a set of action variables $\mathbf{a} = \{a_l, a_r, a_e\}$, each controlling one of the resources in the system. Action variable a_l controls the left arm, a_r controls the right arm, and a_e controls the eyes of the robot. Table 5.1 shows the set of values that each action variable can be assigned to. There are control actions that move one arm from a source position to a destination position. However, the arms cannot move directly from washer to rack, and vice versa. In order to perform such movements, the robot needs to first move the arm from the source position to the front position, and then from that position to the target position in two primitive steps. The control action *pick*, picks up a dish from the washer, if the arm is positioned at the dish-washer, and there is a dish to pick up. The control action *stack*, stacks a dish into the dish-rack if the arm is holding a dish and is positioned at the dish-rack. The robot can also transfer a dish from one arm to the other, if both arms are positioned in front of the robot, and the empty arm executes the *pick* control action. The control actions for the eye movements cause the robot to fixate on the specified position. There is also a *no-op* action that does not change the state of the arms and eyes.

Note that each control action for arms, controls a 2-DOF arm, i.e., the joint angles between the arm and the shoulder, and the joint angle between the arm and the forearm. In this experiment, we constructed three *proportional-derivative* (PD) controllers for controlling each of the arms, and the eyes:

$$PD_i: \quad \tau(\theta, \dot{\theta}) = \mathbf{W}_i[\mathbf{K}_p(\theta_i^* - \theta) - \mathbf{K}_d\dot{\theta}] \tag{5.8}$$

where $\theta \in \mathbb{R}^2$ and $\dot{\theta} \in \mathbb{R}^2$ are the joint positions and velocities of the arms, respectively, and $\tau \in \mathbb{R}^2$ is a vector of joint torques (for the eyes controller we have $\theta, \dot{\theta}, \tau \in \mathbb{R}$). In Equation 5.8 $\mathbf{W}_i$ is a 2x2 gain matrix (for the eye controller $\mathbf{W}_i \in \mathbb{R}$), θ_i^* is the target equilibrium point, and $\mathbf{K}_p$ and $\mathbf{K}_d$ are the nominal proportional and derivative gains, respectively. All $\mathbf{W}_i$ are initialized to the identity matrix. In these experiments for simplicity, we assume the eyes and arm movements take place in 2D $\mathbf{X}Y$-plane.

The states of the robot are also described via a set of state variables summarized in Table 5.2. State variable $\mathbf{s}_{washer}$ keeps track of the number of dishes in the dish-washer. State variable $\mathbf{l}_{pos}$ shows the current position of the left arm (i.e, *washer, front, rack*). State variable $\mathbf{l}_{stat}$ describes the current status of the left hand, i.e., whether it is holding a dish or it is empty. Similarly state variable $\mathbf{r}_{pos}$ and $\mathbf{r}_{stat}$ describe the position and status of the right arm. State variable $\mathbf{e}_{pos}$, describes the current gaze of the robot's eyes. Finally, state variable $\mathbf{s}_{rack}$ describes whether or not a dish has been stacked into the dish-rack. Any assignment to the set of action variables forms a concurrent action. However, not all concurrent actions are allowed for execution in every state. Actions are pruned to simplify learning and enforce safety constraints (Huber, 2000). For example the left arm can execute

Table 5.2. State Variables

$\mathbf{s}_{washer}$	$\mathbf{l}_{pos}$, $\mathbf{r}_{pos}$, $\mathbf{e}_{pos}$	$\mathbf{l}_{stat}$, $\mathbf{r}_{stat}$	$\mathbf{s}_{rack}$
$0, 1 \ldots, n$	*washer*	*has-dish*	*stacked*
	front	*empty*	*not-stacked*
	rack		

the action *pick* only when it is located at the dish-washer and is empty, and there is also a dish to pickup. Any concurrent action that violates the safety constraints is referred to as an invalid action. If the robot executes an invalid action, it receives a negative reward and the state of the robot do not change. The actions that control the gaze of the robot reflect the limitations of a real robot system. The robot is required to look at a target position before being able to move any of its arms to that position (except for any movement to the front of the robot, i.e., the position marked *front* in Figure 5.3).

5.2.1 Coarticulation Approach

Recall that the overall objective is to empty the dish-washer and stack the dishes into the dish-rack in the least number of steps. This objective can be approximated in terms of concurrent optimization of two competing subgoals: ω_{stack}, and ω_{pick}, with the priority ranking system:

$$\omega_{pick} \triangleleft \omega_{stack}$$

The objective of the subgoal ω_{stack} is to stack a dish that has already been picked up, into the dish-rack, and the objective of the subgoal ω_{pick} is to pick up a dish from the dish-washer.

More interestingly, the robot is also required to minimize its energy consumption. In other words the robot is required to perform as few moves as possible. The robot needs to balance its energy consumption and the speed at which it completes the overall task. This new constraint can be expressed through the following priority ranking system:

$$\omega_{pick} \lhd \omega_{stack} \lhd \omega_{energy}$$

All of these subgoals compete for the limited amount of resources in the robot (i.e., eyes and arms). We design three redundant controllers $\mathcal{C}_{pick}$, and $\mathcal{C}_{stack}$, and $\mathcal{C}_{energy}$ that achieve each subgoal. Note that $\mathcal{C}_{pick}$, and $\mathcal{C}_{stack}$ can be viewed as general purpose object manipulation controllers for picking up and stacking objects across different tasks (Singh et al., 2004). We use subgoal options (Precup, 2000) to model each controller:

- The controller $\mathcal{C}_{pick}$ is modeled as a minimum cost-to-goal subgoal option:

$$\mathcal{C}_{pick} = \langle \mathcal{I}_{pick}, \pi_{pick}, \beta_{pick} \rangle$$

where $\mathcal{I}_{pick}$ is the set of states from which the robot can pick up a dish (i.e., the set of states in which at least one of the robot's hands is empty and there is a dish in the dish-washer). The goal states consist of those states in which the robot successfully picks up a dish from the dish-washer. Policy π_{pick}, specifies a closed loop policy for picking up a dish. The termination condition occurs when the robot picks up a dish from the dish-washer (i.e., enters a goal state). When learning the optimal value function optimizing the subgoal of this controller, the robot receives a reward of -1 at every step, and a reward of zero in the goal states.

- The controller $\mathcal{C}_{stack}$ is modeled as a minimum cost-to-goal subgoal option:

$$\langle \mathcal{I}_{stack}, \pi_{stack}, \beta_{stack} \rangle$$

where $\mathcal{I}_{stack}$ is the set of states in which the robot can stack a dish (i.e., the set of states in which the robot is holding at least one dish). The goal states consist of those states in which the robot successfully stacks a dish into the dish-rack. Policy π_{stack} specifies a closed loop policy for stacking a dish. The termination condition for this option occurs when the robot stacks a dish in the dish-rack. When learning the optimal value function of this controller, the robot receives a reward of -1 at every step, and a reward of zero in the goal states.

• The controller $\mathcal{C}_{energy}$ is a *non-associative* controller (Sutton and Barto, 1998), where it terminates in every state with probability 1. The reward function for this controller favors actions which result in minimum physical movement in the robot. In other words it favors actions with more values of *no-op* for its constituent action variables. For example, an action $\mathbf{a} = \{$*washer-to-front*, *stack-plate*, *fixate-on-rack*$\}$, causes more movement than the action $\mathbf{a} = \{$*washer-to-front*, *no-op*, *no-op*$\}$. When learning the optimal value function of this controller, the robot receives a reward of:

$$\mathcal{R}(\mathbf{s}, \mathbf{a}) = -1 \times \text{number of } \textit{no-op} \text{ values in } \mathbf{a}$$

at every step.

Note that due to a redundant set of resources in the system, all of controllers are ϵ-redundant for some ϵ. For example the robot can pick up a dish either by its left arm, or by its right arm. Or, it can stack a dish either by moving the arm that is currently holding a dish to the dish-rack and stack the dish, or it can transfer it to the other hand and use the other hand to stack it. Controller $\mathcal{C}_{energy}$ is also ϵ-redundant for different values of ϵ. For example if the current choice of ϵ allows the robot to execute actions with at least two *no-op* values for its constituent action variables, then there are four choices of actions in every state (three actions with two *no-op* values, for its constituent action variables, and one all *no-op* action).

It can be verified that the sequential solution (no coarticulation) which involves executing $\mathcal{C}_{pick}$ and then $\mathcal{C}_{stack}$ in sequence, does not provide the most efficient solution. For example while the robot is stacking a dish held by its right hand, it can concurrently pick up a new dish with its left hand. By coarticulating between these two subgoals, the robot can select an action that achieves the objective of the superior controller (i.e., $\mathcal{C}_{stack}$), while committing to the objective of the subordinate controller (i.e., $\mathcal{C}_{pick}$).This is possible if the intersection of the redundant-sets of these two controllers is non-empty in the current state.

To further illustrate this, consider the following scenario: assume that the robot has picked up a dish with its right arm positioned at the front, and the controller $\mathcal{C}_{stack}$ is in progress. Also, assume that its left arm is positioned at front and is empty. In this state, the robot can execute at least two ϵ-ascending actions with respect to the $\mathcal{C}_{stack}$ controller: (1) move the right arm to the dish-rack and concurrently look at the front position; (2) move the right arm to the dish-rack and concurrently look at the dish-washer position. Note that the second action is also ϵ-ascending with respect to the $\mathcal{C}_{pick}$ controller, since by looking at the dish-washer position, the robot can then move its empty left arm to the dish-washer in order to pick a new dish. By coarticulating between these two controllers, the robot executes the second action that is ϵ-ascending with respect to both controllers. Note that while this action moves the robot's right arm to the dish-rack to stack the dish, concurrently it moves the empty left arm to the dish-washer in order to pick a new dish.

In our experiments, all controllers are defined over an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$, where the states and actions are described via the set of variables given in Tables 5.2, and 5.1. All actions are stochastic; they succeed with probability p and fail with probability $(1 - p)$ (we used the failure probability of $p = 0.1$ throughout our experiments). When actions succeed, they change the state of the robot to the next state as described above. Upon failure, or executing an invalid action, the robot does not change its state. All actions are also rewarded -1 upon termination.

5.2.2 Experimental Setup

Our empirical evaluation of the coarticulation framework for solving the above problem consists of the following experiments:

• A set of experiments evaluating the accuracy of the approximation techniques for computing the class of ϵ-ascending policies (i.e., the variable elimination method described in Algorithm 6). This involves measuring the accuracy of the algorithm for computing the redundant-sets for different values of $\mathbf{h}$ when computing the top $\mathbf{h}$ best actions, and for various levels of flexibility offered by the controllers in terms of the ϵ parameter.

• A set of experiments evaluating the performance of the coarticulation approach for solving the above problem.

• A set of experiments evaluating the influence of the energy minimizing controller (i.e., $\mathcal{C}_{energy}$) on the policy generated by the coarticulation algorithm.

As a basis for analysis of the coarticulation framework, we also use the following models, in addition to the approximate coarticulation techniques that we presented in this chapter:

Exact concurrent controller model: Given a controller, we compute the value function and subsequently the redundant-sets using the standard RL methods (no function approximation is involved).

Exact sequential controller model: Given a controller, we compute the optimal sequential policy using the standard RL methods (no function approximation is involved). In this case the robot is only allowed to execute non-concurrent actions. For example, in the simulated robot experiment, any concurrent action $\mathbf{a} = \{a_l, a_r, a_e\}$ with at least two *no-op* is a sequential action.

We use the term **Oracle** models to refer to the experiments that use the exact controller models, in both sequential and concurrent cases. Note that the oracle models are merely used for evaluating the accuracy of the approximate algorithm, and also the performance of the coarticulation framework.

5.2.3 Learning the Redundant Controllers

In order to apply our coarticulation algorithm to the above problem, we need to compute the redundant class of ϵ-ascending policies for each controller. Thus, we first need to learn the optimal value function associated with each controller, and then compute the redundant-sets using the Algorithm 1 for the oracle coarticulation model, and the Algorithm 6 for the approximate coarticulation approach.

Note that there are $|\mathcal{S}| = \mathbf{n} \times 3 \times 2 \times 3 \times 2 \times 3 \times 2 \times 2 = \mathbf{n} \times 432$ states – with $\mathbf{n}$ being the total number of dishes into the dish-washer – and $|\mathcal{A}| = 7 \times 7 \times 4 = 196$ concurrent actions. This produces $|\mathcal{S}||\mathcal{A}| = \mathbf{n} \times 432 \times 196 = \mathbf{n} \times 84672$ state-action values to be learned by an standard tabular RL algorithm. For even a small number of dishes, for example 10 dishes, the robot has to learn $|\mathcal{S}||\mathcal{A}| = 846720$ state-action values. Here, we can observe the curse of dimensionality in the action space: even in problems with few degrees of freedom, the total number of concurrent actions exponentially grows, as it is also evident in the above problem.

To overcome this problem, we use the approximation techniques for coarticulation that we introduced in Section 5.1. We assume that the optimal state-action value function can be approximated using linear function approximation techniques (Bertsekas and Tsitsiklis, 1997; Sutton and Barto, 1998). We used a sparse-coarse-coded function approximator (CMACs) (Albus, 1981) combined with Sarsa(λ) algorithm (Rummery and Niranjan, 1994; Sutton, 1996; Sutton and Barto, 1998). Each CMAC tiling represents a basis function defined over a subset of state and action variables. The approximate state-value function can be expressed as:

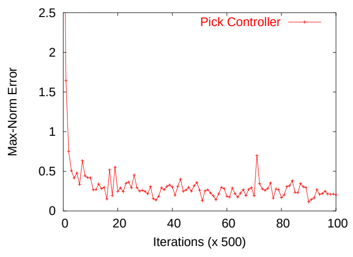
$$\bar{\mathcal{Q}}^{*}(\mathbf{s}, \mathbf{a}) = \sum_{i=1}^{m} \mathcal{I}_i(\mathbf{s}) \mathcal{Q}_i(\bar{\mathbf{s}}_i, \bar{\mathbf{a}}_i)$$

where $\mathbf{s}$ and $\mathbf{a}$ are the set of state and action variables spanning the state and action spaces respectively, $\mathbf{m}$ is the total number of tilings, and $\mathcal{Q}_i(\bar{\mathbf{s}}_i, \bar{\mathbf{a}}_i)$ is a tiling defined over a subset of state variables $\bar{\mathbf{s}}_i$ and a subset of action variables $\bar{\mathbf{a}}_i$. $\mathcal{I}_i(\mathbf{s})$ is an indicator function that returns 1 if the point $\{\mathbf{s}, \mathbf{a}\}$ falls within the $\mathbf{i}^{th}$ tiling. Table 5.3 shows the set of tilings that we used for the CMAC approximation of the state-value functions for the controller $\mathcal{C}_{pick}$. This set consists of 10 tilings, for the total of 5914 tiles. In all of the controllers, we allowed each tiling to be defined only over at most two action variables in order to reduce the

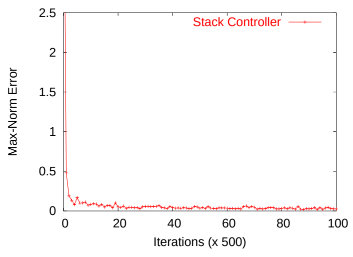
Table 5.3. CMACs tilings

Tiling	# of tiles
$\mathcal{Q}_1(\mathbf{s}_{washer}, l_{pos}, l_{stat})$	12
$\mathcal{Q}_2(\mathbf{s}_{washer}, r_{pos}, r_{stat})$	12
$\mathcal{Q}_3(\mathbf{s}_{washer}, s_{rack}, r_{pos}, r_{stat})$	24
$\mathcal{Q}_4(\mathbf{s}_{washer}, s_{rack}, \mathbf{a}_l, \mathbf{a}_r)$	196
$\mathcal{Q}_5(\mathbf{s}_{washer}, l_{pos}, l_{stat}, e_{pos}, \mathbf{a}_l)$	252
$\mathcal{Q}_6(\mathbf{s}_{washer}, r_{pos}, r_{stat}, e_{pos}, \mathbf{a}_r)$	252
$\mathcal{Q}_7(\mathbf{s}_{washer}, s_{rack}, l_{pos}, l_{stat}, e_{pos}, \mathbf{a}_l)$	504
$\mathcal{Q}_8(\mathbf{s}_{washer}, l_{pos}, l_{stat}, e_{pos}, \mathbf{a}_l, \mathbf{a}_e)$	1008
$\mathcal{Q}_9(\mathbf{s}_{washer}, r_{pos}, r_{stat}, e_{pos}, \mathbf{a}_r, \mathbf{a}_e)$	1008
$\mathcal{Q}_{10}(\mathbf{s}_{washer}, l_{pos}, r_{pos}, e_{pos}, \mathbf{a}_l, \mathbf{a}_r)$	2646

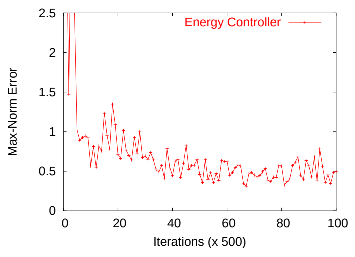
complexity of the variable elimination method that we described in Algorithm 4. In general



(a) $\mathcal{C}_{pick}$



(b) $\mathcal{C}_{stack}$



(c) $\mathcal{C}_{energy}$

Figure 5.4. Learning curves for controllers (a) $\mathcal{C}_{pick}$, (b) $\mathcal{C}_{stack}$, and (c) $\mathcal{C}_{energy}$. The horizontal axis shows the iterations of SARSA(λ) for $\lambda = 0.8$ and for every 500 iterations. The vertical axis shows the max-norm error at every iteration.

we can use the domain knowledge for selecting the best CMAC design for the problem. For example, the state variable $\mathbf{s}_{washer}$ has less relevance to the subgoal of stacking a dish in the dish-rack. Thus, when designing the CMAC tilings for the controller $\mathcal{C}_{pick}$, we can take into account such information. Note that in all three controllers, the exact number of dishes in the dish-washer has less relevance to the subgoals associated with them. A more relevant feature is whether or not there exists a dish in the dish-washer. Thus, when designing a CMAC for each controller, we set the state variable $\mathbf{s}_{washer}$ to take on only two values, namely, zero for when the dish-washer is empty, and one for when there exist at least one dish in the dish-washer. This is yet another advantage of the coarticulation approach in terms of reusability of skills, since for any number of dishes , the same learned skill can be incorporated when solving the problem.

In order to learn the approximate state-action value functions for each controller, we used Sarsa(λ) algorithm (Rummery and Niranjan, 1994; Sutton and Barto, 1998). Sarsa(λ) is an on-policy TD(λ) learning method that has been shown to have convergence properties with linear function approximation techniques (Dayan, 1992; Tsitsiklis and Roy, 1996)[2].

Figures 5.4(a), 5.4(b), and 5.4(c) show the max-norm error plots of the state-action value function approximation for the controllers $\mathcal{C}_{pick}$, $\mathcal{C}_{stack}$, and $\mathcal{C}_{energy}$ respectively. The horizontal axis shows the iterations of SARSA(λ) for $\lambda = 0.8$. The vertical axis shows the max-norm error at every iteration. As shown in these figures, all cases converge to the best possible approximation based on the CMAC design incorporated in the approximation.

5.2.4 Accuracy Analysis

In this set of experiments, we study the precision of the approximate technique for computing the redundant-sets in a controller. Note that the error in the approximate state-action value function due to the approximation based on CMACs propagates to the next level, where we use Algorithm 6 for computing the redundant-sets in the controller. Every choice of the parameters $\{\mathbf{h}, \epsilon\}$ specifies a class of ϵ-ascending policies in a controller $\mathcal{C}$. In order to measure the precision of the Algorithm 6 for computing the redundant-sets, we conduct an experiment where we choose different values for $\mathbf{h}$ (for selecting the top $\mathbf{h}$ best actions), and then apply the Algorithm 6 to compute the top $\mathbf{h}$ best actions in every state of the controller. For evaluating the number of correctly picked actions in a redundant-set, we compare the sets computed by both the oracle and approximate algorithms. Let

[2]The convergence of the TD methods with linear function approximation techniques is not to the minimum error approximation, but to a nearby approximation whose error is shown to be bounded (Tsitsiklis and Roy, 1996)

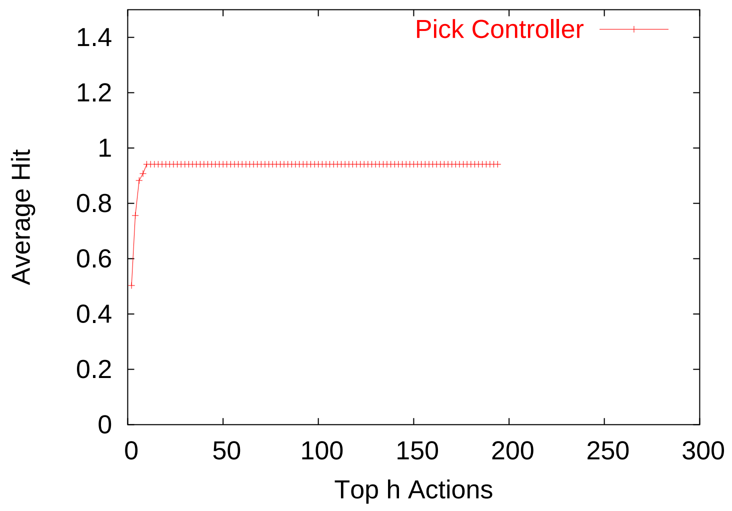


Figure 5.5. Average accuracy of the redundant-sets computed by Algorithm 6 versus different values of the parameter $\mathbf{h}$ for controller $\mathcal{C}_{pick}$. These results are averaged over all states.

$\bar{\mathcal{A}}_{\mathcal{C}}^{\epsilon,\mathbf{h}}(\mathbf{s})$ represent the top $\mathbf{h}$ best actions computed from the approximate model using the Algorithm 6, and $\mathcal{A}_{\mathcal{C}}^{\epsilon,\mathbf{h}}(\mathbf{s})$ represent the top $\mathbf{h}$ ϵ-ascending policies computed from the oracle model of the controller using the Algorithm 1 $\mathcal{C}$ in some state $\mathbf{s}$. We define the following measurement for studying the precision of the approximate approach:

- : $\mathbf{Hit}(\mathbf{s}) = \frac{|\bar{\mathcal{A}}_{\mathcal{C}}^{\epsilon,\mathbf{h}}(\mathbf{s}) \cap \mathcal{A}_{\mathcal{C}}^{\epsilon,\mathbf{h}}(\mathbf{s}))|}{|\bar{\mathcal{A}}_{\mathcal{C}}^{\epsilon,\mathbf{h}}(\mathbf{s})|}$

Figures 5.5, 5.6, and 5.7, show the average hit rate of the redundant-sets computed by Algorithm 6 – averaged over all states – versus different values of the parameter $\mathbf{h}$. All the controllers are 0.5-redundant (i.e., ϵ = 0.5 in all controllers). Note that for small values of $\mathbf{h}$ (i.e., $\mathbf{h}$ ¡ 4), the average hit rate is relatively low (0.75, 0.40, and 0.77 for $\mathcal{C}_{pick}$, $\mathcal{C}_{stack}$, and $\mathcal{C}_{energy}$ respectively). By increasing $\mathbf{h}$ the average hit rate in all the controllers increases. For controllers $\mathcal{C}_{pick}$ and $\mathcal{C}_{energy}$, and for larger values of $\mathbf{h}$, we attain very high average precision (0.94 and 1.0, for $\mathcal{C}_{pick}$ and $\mathcal{C}_{energy}$ respectively). For controller $\mathcal{C}_{stack}$ we attain the average precision of 0.76.

Note that by an intelligent design of CMACs (especially in case of the controller $\mathcal{C}_{stack}$ for example), we may attain better precision for the redundant-sets using our approximate

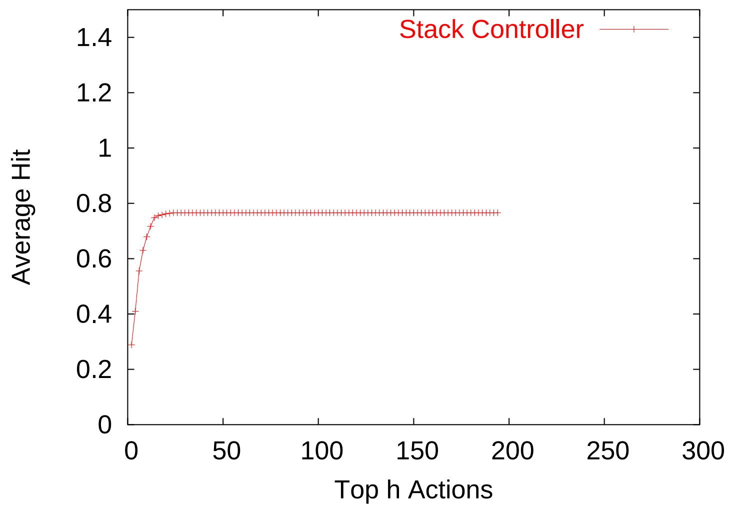


Figure 5.6. Average accuracy of the redundant-sets computed by Algorithm 6 versus different values of the parameter $\mathbf{h}$ for controller $\mathcal{C}_{stack}$. These results are averaged over all states.

algorithm. Based on these results we can observe that our algorithm attains relatively high precision for computing the redundant-sets.

5.2.5 Flexibility Analysis: The Influence of ϵ

In this set of experiments, we study the influence of the parameter ϵ on the amount of flexibility that ϵ-redundant controller offers. We define the flexibility in terms of the size of the redundant-sets obtained for a particular setting of the parameter ϵ. Any given ϵ is associated with a class of ϵ-ascending policies. Intuitively for larger values of ϵ, we enforce the policies to be near optimal and thus we expect to see less flexibility offered by a controller. For smaller values of ϵ, we allow the ϵ-ascending policies to deviate more from the optimal policy, and hence we obtain a larger class of ϵ-ascending policies, although less optimal. Figure 5.8 shows the plot of the average redundancy, averaged over all states for different values of ϵ, and for different controllers (i.e., $\mathcal{C}_{pick}$, $\mathcal{C}_{stack}$, and $\mathcal{C}_{omega}$). For controllers $\mathcal{C}_{pick}$ and $\mathcal{C}_{stack}$, we can distinctively identify three different regions where the average redundancy changes from one level to the other. Each region is associated with an

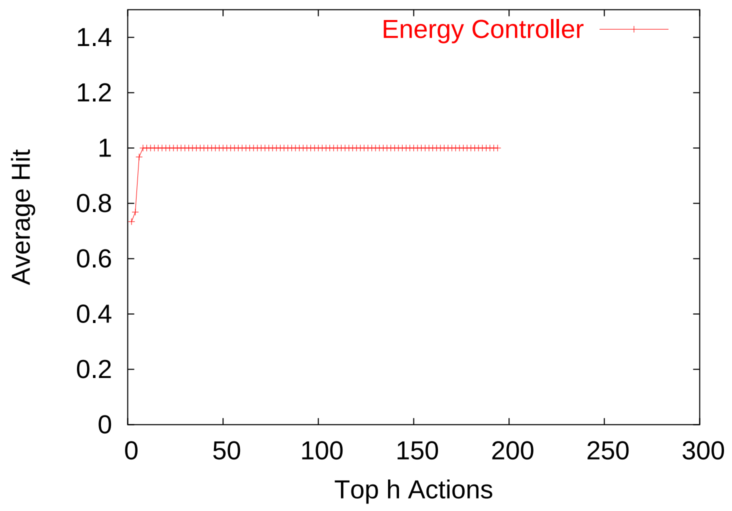


Figure 5.7. Average accuracy of the redundant-sets computed by Algorithm 6 versus different values of the parameter $\mathbf{h}$ for controller $\mathcal{C}_{energy}$. These results are averaged over all states.

interval defined over the values of ϵ, namely the intervals $[0.0\ 0.5]$, $[0.5\ 0.6]$, and $[0.6\ 1.0]$. The size of the redundant-sets remains constant in every region. We conjecture that this is due to the linearity of the CMAC approximation scheme, where the state-action values are generalized for the set of states that fall in the same tiling. We can also observe that how the flexibility of a controller is inversely proportional to the value of ϵ. For controller $\mathcal{C}_{energy}$ we can distinctively identify five different regions associated with the intervals $[0.0\ 0.2]$, $[0.2\ 0.4]$, $[0.4\ 0.5]$, $[0.5\ 0.6]$, and $[0.6\ 0.1]$. Similarly, each interval represents an equivalent class of ϵ-ascending policies for the values of ϵ in that interval. Taking a close look at the ϵ-ascending policies associated with each region, we identified three main regions, each representing a certain property of the $\mathcal{C}_{energy}$ controller. Figure 5.9 shows three intervals over ϵ. The interval $[0.6\ 1.0]$ is associated with the class of ϵ-ascending policies that strictly enforce the agent to execute *no-op* action for both arms and the eyes. We refer to this kind of controller as the *lazy* controller. The interval $[0.34\ 0.6]$ is associated with the class of ϵ-ascending policies that strictly enforce the agent to execute at most one none *no-op* action, for every settings of the action variables. As we demonstrate later in this section,

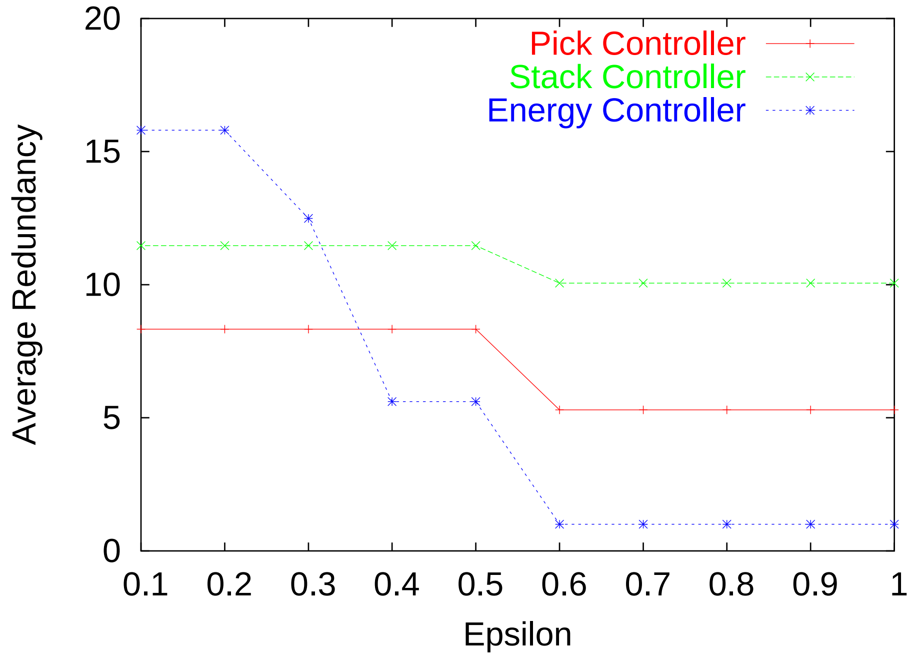


Figure 5.8. Average redundancy per state (average size of the redundant-sets) for different values of ϵ, and for different controllers.

coarticulating with this class of ϵ-ascending policies, would generate the sequential (non-concurrent) policy. The interval $[0.0\ 0.6]$ is associated with the class of ϵ-ascending policies that allows the agent to execute any action. this interval can be viewed as the region where the controller is an *active* controller (i.e., allowing none *no-op* actions to be performed).

5.2.6 Performance Analysis: Emptying the Dish-Washer

In this set of experiments, we study the performance of the coarticulation algorithm for emptying the dish-washer. Our results compare the performance of the following approaches:

• **Coarticulate-CMAC:** We use the approximation techniques based on the that we described in Section 5.2.3 for computing the redundant-sets. When performing the variable elimination algorithm (Algorithm 4), we used the following elimination order:

$$\langle a_l,\, a_r,\, a_e \rangle$$

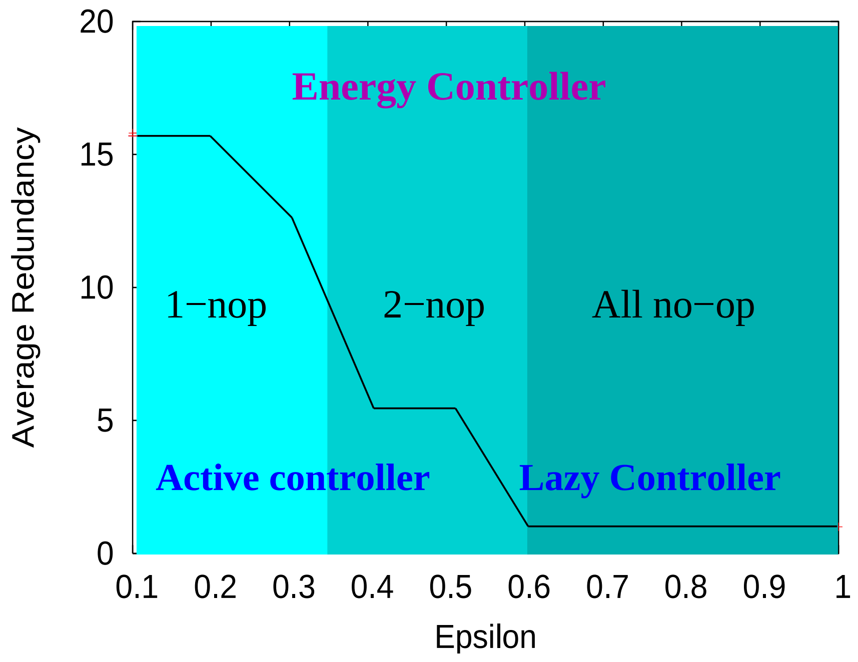


Figure 5.9. Average redundancy per state (average size of the redundant-sets) for different values of ϵ in controller $\mathcal{C}_{energy}$. Each region is associated with a certain class of policies and characterizes a different energy consumption minimization strategy.

We then use Algorithm 4.3 for performing coarticulation.

• **Coarticulate-Oracle:** We use the oracle models for computing the true redundant-sets. We then use Algorithm 4.3 for performing coarticulation.

• **Sequential-Oracle:** We use the oracle models of the controllers for performing the tasks sequentially. We use the algorithm *Sequence* (Algorithm 4.2) for solving the problem sequentially (no-coarticulation is involved).

In all of the experiments, we also used the $T_{continue}$ termination mechanism. In the first set of experiments, we used the priority ranking system:

$$\mathcal{C}_{pick} \lhd \mathcal{C}_{stack}$$

which should read $\mathcal{C}_{pick}$ *subject-to* $\mathcal{C}_{stack}$. Intuitively, by coarticulating between these two subgoals based on the above priority ranking relation, the robot will attempt to maximize stacking the dishes, while picking up new dishes from the dish-washer.

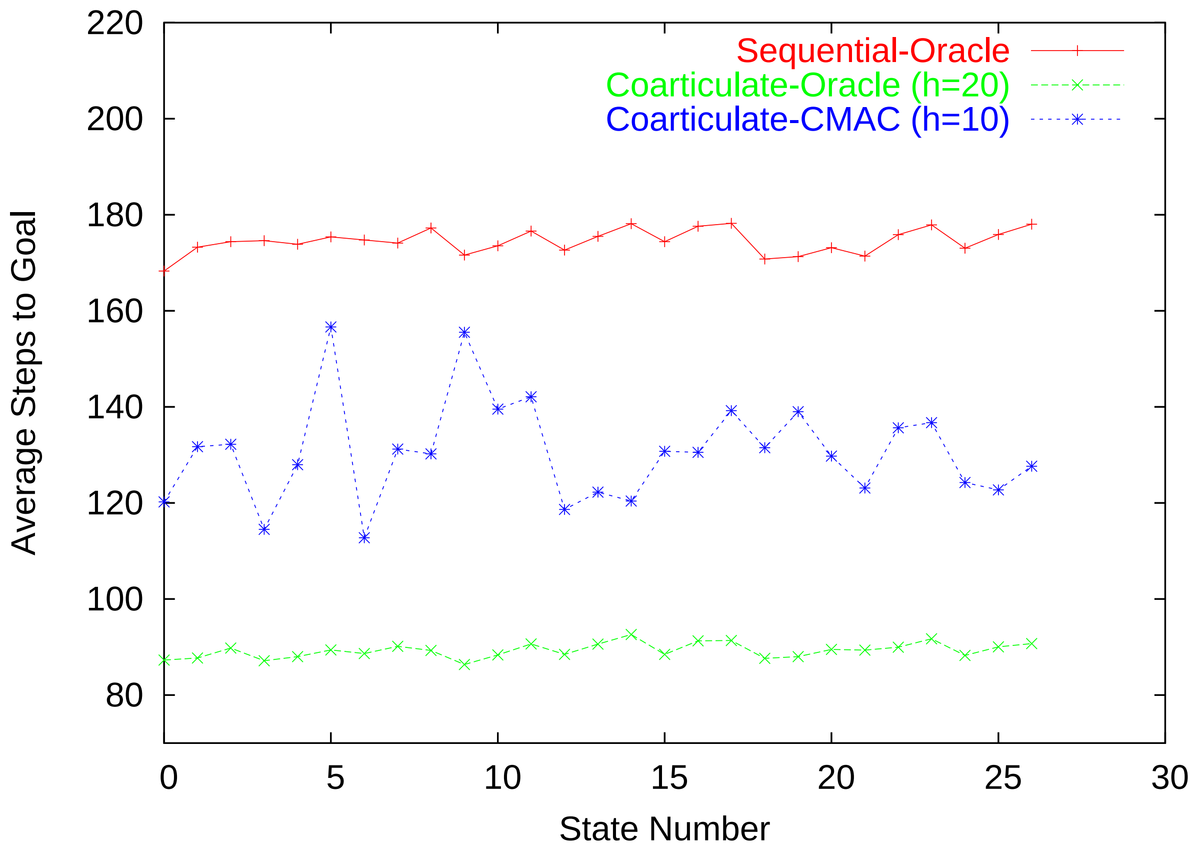


Figure 5.10. Performance of the coarticulate-oracle, coarticulate-CMAC, and sequential-oracle approaches for $\mathbf{h} = 10$ and $\epsilon = 0.5$. In this experiment, we only used two controllers: $\mathcal{C}_{pick}$, and $\mathcal{C}_{stack}$ with the priority ranking system $\mathcal{C}_{pick} \triangleleft \mathcal{C}_{stack}$. The horizontal axis shows a numbering of the starting states. The vertical axis show the average number of steps for completing the task. These results are averaged over 20 tasks, each consisting of 27 episodes, where each episode initializes the robot in one of 27 possible starting positions.

Figure 5.10 shows the performance of the coarticulate-oracle, sequential-oracle, and coarticulate-CMAC approaches for $\mathbf{h} = 10$. The performance is measured in terms of the total number of steps for completion of the task. These results are averaged over 20 tasks, each consisting of 27 episodes. Each episode is associated with a starting state, with 20 dishes in the dish-washer, and the robots arms are set to empty. The horizontal axis depicts the indices of the starting states. A starting state is defined in terms of the various configurations of the state variables (e.g., initial positions of the arms and the eyes, and their status, etc) that are relevant to the task (i.e., at least one controller can be initiated).

Not surprising, the best performance (bottom plot) is achieved by the coarticulate-oracle

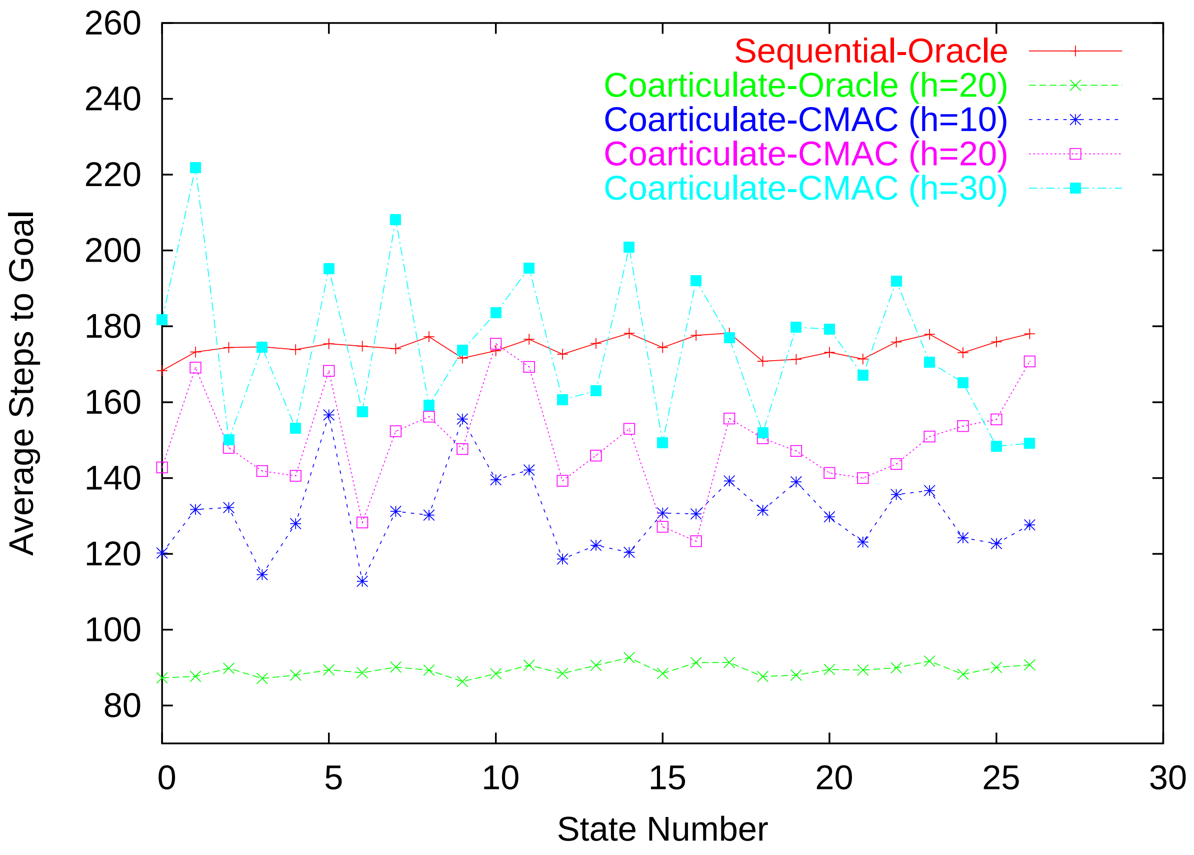
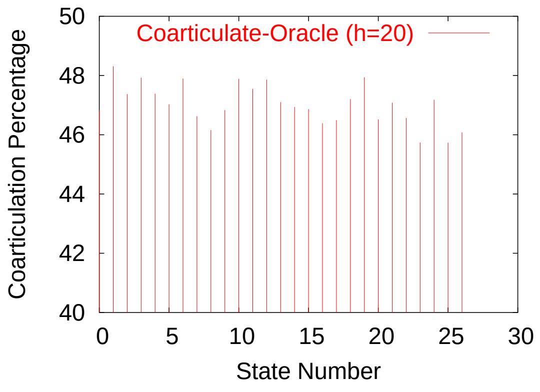


Figure 5.11. Performance of the coarticulate-oracle, coarticulate-CMAC, and sequential-oracle approaches using different values of **h**. In this experiment, we only used two controllers: $\mathcal{C}_{pick}$, and $\mathcal{C}_{stack}$ with the priority ranking system $\mathcal{C}_{pick} \triangleleft \mathcal{C}_{stack}$. The horizontal axis shows a numbering of the starting states. The vertical axis show the average number of steps for completing the task. These results are averaged over 20 tasks, each consisting of 27 episodes, where each episode initializes the robot in one of 27 possible starting positions.

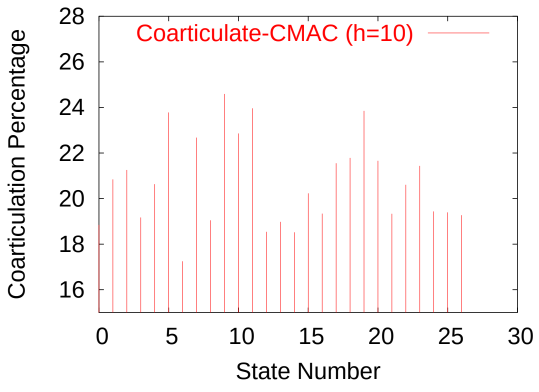
approach (roughly, 80 steps to complete the task). The worst performance (top plot) is achieved by the sequential-oracle approach (roughly, 170 steps to complete the task). The coarticulate-oracle approach significantly speeds up the task completion (almost over two times faster). The best performance of the coarticulate-CMAC approach is attained by choosing $\mathbf{h} = 10$, and in average completes the task between 120 and 140 steps, still outperforming the sequential-oracle approach by a large margin.

Figure 5.11 shows the same plot together with a set of coarticulate-CMAC experiments for different values of the parameter $\mathbf{h}$ (the top $\mathbf{h}$ best actions, for $\mathbf{h} = 10, 20, 30$). It can be observed that the performance of the coarticulate-CMAC approach degrades as we increase the size of the parameter $\mathbf{h}$. The main reason is that the error in the state-action value function approximation based on CMACs, propagates to the step (e.g., Algorithm 6) where we compute the redundant-sets. Intuitively, by using a better CMAC design for learning the approximate state-action value functions associated with the controllers, the coarticulate-CMAC approach would perform more optimally and closer to the performance of the coarticulate-oracle approach.

In all of the experiments that employ coarticulation, we also measured the total number of coarticulation performed within each episode. A coarticulation takes place in a state $\mathbf{s}$ whenever the algorithm *Coarticulate* finds a policy that achieves both subgoals. Note that in general coarticulation does not succeed in every state. Figures 5.12(a) and 5.12(b)



(a) coarticulate-oracle



(b) coarticulate-CMAC

Figure 5.12. Distribution of the coarticulation occurrence in every episode for (a) coarticulate-oracle approach, and (b) coarticulate-CMAC approach. These results are averaged over 20 tasks, each consisting of 27 episodes, where each episode initializes the robot in one of 27 possible starting positions.

show the distribution of the coarticulation occurrence for in every episode, for coarticulate-oracle and coarticulate-CMAC approaches, respectively. Note that in coarticulate-oracle experiment, the robot coarticulates almost more than 50% of the total number of steps before it completes the task. In coarticulate-CMAC experiment, the robot coarticulates almost 20% of the total number of steps task completion. These results are in accordance with the performance results that we presented in Figure 5.10. Note that in average, every coarticulating occurrence saves two steps compared to the sequential-oracle approach. In the coarticulate-oracle case, there is on average above 50% coarticulation per episode. Thus without performing coarticulation, it would take on average over a hundred steps, that is close to the performance of the sequential-oracle approach.

5.2.7 Incorporating Energy Constraints

In the last set of experiments, we study the effect of the controller $\mathcal{C}_{energy}$ when coarticulating with the $\mathcal{C}_{pick}$ and $\mathcal{C}_{stack}$ for emptying dish-washer problem. Note that among the best policies that complete the overall task, we prefer those that lead to less energy consumption in the robot. In order to add this constraint to the problem, we can simply perform coarticulation based on the following priority ranking system:

$$\mathcal{C}_{pick} \triangleleft \mathcal{C}_{stack} \triangleleft \mathcal{C}_{energy}$$

Implicitly, the above relation enforces the robot to select a concurrent action at every step, with less number of non *no-op* values for its constituent action elements. Figure 5.13 shows the average amount of energy consumed per steps of an episode, for all coarticulate-oracle ($\mathcal{C}_{pick} \triangleleft \mathcal{C}_{stack}$), coarticulate-oracle ($\mathcal{C}_{pick} \triangleleft \mathcal{C}_{stack} \triangleleft \mathcal{C}_{energy}$), and sequential-oracle approaches. These results are averaged over 20 tasks, each consisting of 27 episodes, where each episode initializes the robot in one of 27 possible starting positions. Note that in sequential-oracle approach, the robot consumes the least amount of energy because it only executes actions with at most one non *no-op* element. Both coarticulation approaches consume more energy, as they incorporate concurrent actions with no constraint. However

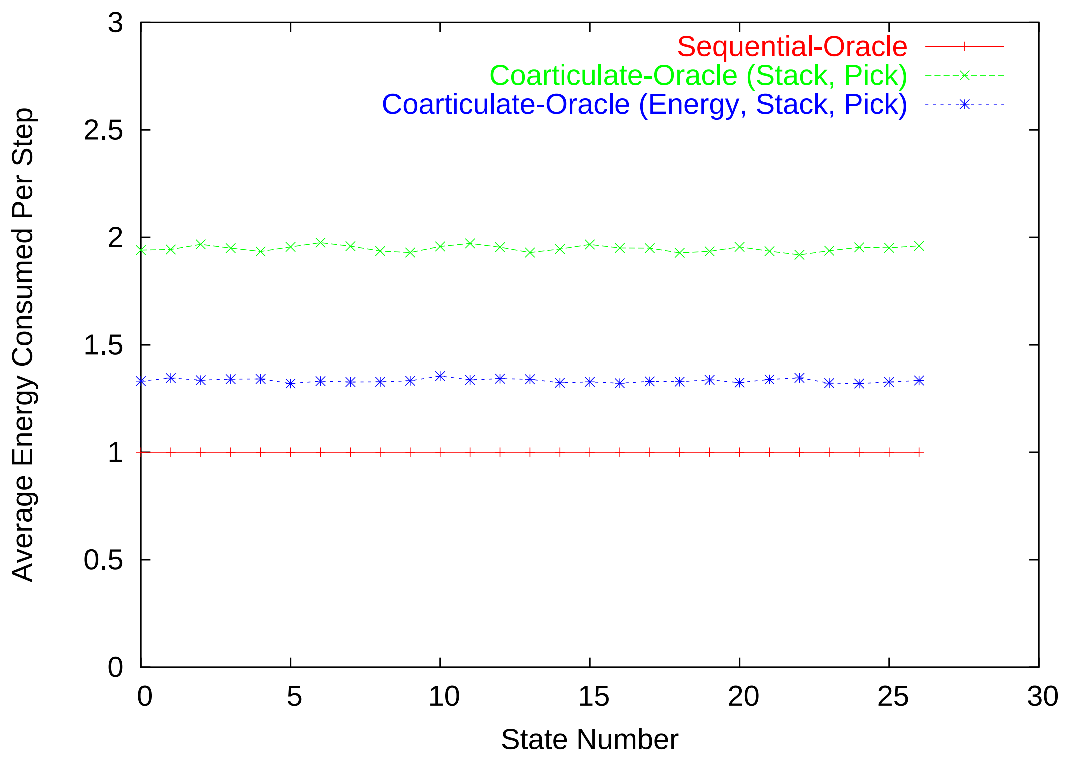


Figure 5.13. The average amount of energy consumed per steps of an episode, for coarticulate-oracle ($\mathcal{C}_{pick} \lhd \mathcal{C}_{stack}$), coarticulate-oracle ($\mathcal{C}_{pick} \lhd \mathcal{C}_{stack} \lhd \mathcal{C}_{energy}$), and sequential-oracle approaches. These results are averaged over 20 tasks, each consisting of 27 episodes, where each episode initializes the robot in one of 27 possible starting positions.

coarticulate-oracle ($\mathcal{C}_{pick} \lhd \mathcal{C}_{stack}$) on average consumes more energy than the coarticulate-oracle approach.

Figure 5.13 shows the performance of the coarticulate-oracle ($\mathcal{C}_{pick} \lhd \mathcal{C}_{stack}$), coarticulate-oracle ($\mathcal{C}_{pick} \lhd \mathcal{C}_{stack} \lhd \mathcal{C}_{energy}$), and also the sequential-oracle approaches using all three controllers. The performance is measured in terms of the total number of steps for completion of the task. Note that by incorporating the energy constraints, now the robot minimizes the energy consumption, while completing the overall task with slightly worse performance compared to the case with no energy constraints (i.e., $\mathcal{C}_{pick} \lhd \mathcal{C}_{stack}$). These results demonstrate the advantage of the coarticulation framework for reusability of various skills acquired by the robot, when facing a new task.

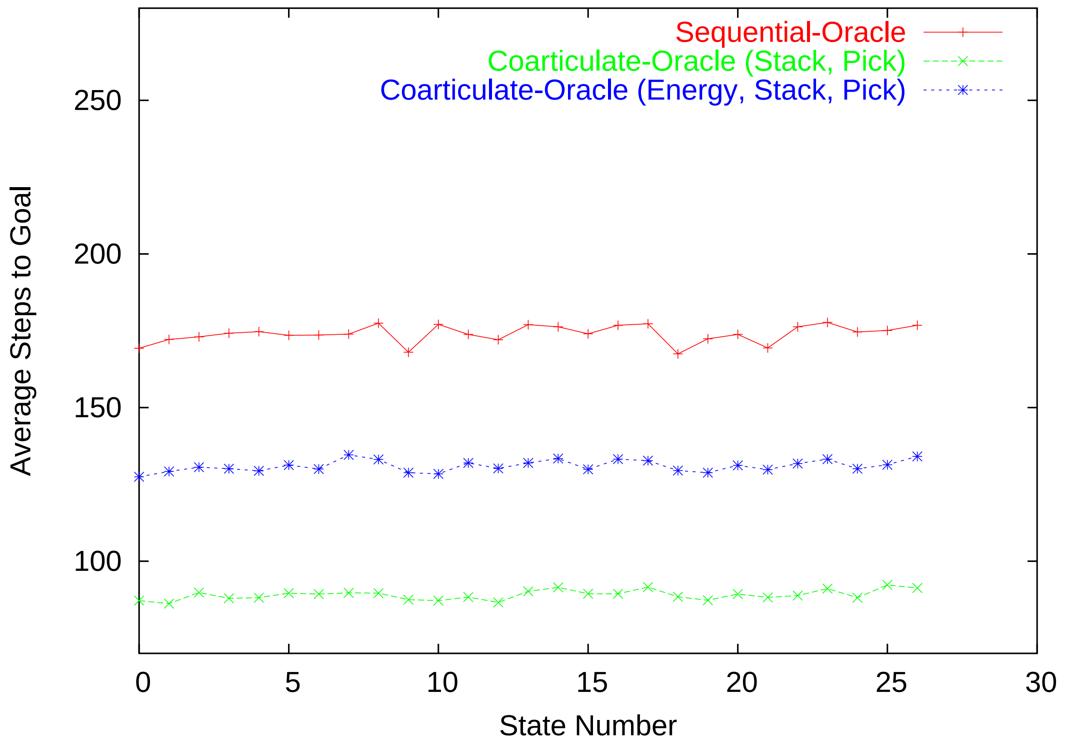


Figure 5.14. Performance of the coarticulate-oracle ($\mathcal{C}_{pick} \lhd \mathcal{C}_{stack}$), coarticulate-oracle ($\mathcal{C}_{pick} \lhd \mathcal{C}_{stack} \lhd \mathcal{C}_{energy}$), and sequential-oracle approaches for $\mathbf{h} = 10$ and $\epsilon = 0.5$. In this experiment, we only used all three controllers: $\mathcal{C}_{pick}$, $\mathcal{C}_{stack}$, and $\mathcal{C}_{energy}$ with the priority ranking system $\mathcal{C}_{pick} \lhd \mathcal{C}_{stack} \lhd \mathcal{C}_{energy}$. The horizontal axis shows a numbering of the starting states. The vertical axis show the average number of steps for completing the task. These results are averaged over 20 tasks, each consisting of 27 episodes, where each episode initializes the robot in one of 27 possible starting positions.

5.3 Concluding Remarks

In this chapter we presented a set of approximation techniques for scaling the coarticulation framework to large domains. Our approach is based on the key idea that in many concurrent decision making problems, the state-action value function can be approximated in terms of the linear combination of a set of local basis functions defined only over a subset of state and action variables. Such exploitable structure enabled us to design efficient approximate algorithms for compactly computing the redundant-sets with a logarithmic complexity in the pre-specified size of the redundant-sets, and also to perform coarticulation with a polynomial complexity in the size of the redundant-sets. By selecting a feasible subset of the redundant-sets in terms of the parameter **h**, we can balance the trade-offs between the computational costs, and the performance of coarticulation when solving a concurrent decision making problem.

We also empirically evaluated the coarticulation framework in a concurrent decision making problem. We demonstrated that how concurrency naturally emerges by performing coarticulation in such problems, and how it can be used to tackle a new problem *reusing* various skills that the agent has acquired through its life span.

CHAPTER 6

CONCLUSIONS AND FUTURE DIRECTIONS

Concurrent decision making is undoubtedly an inseparable component of decision-making in humans and many other biological systems. While in the long history of machine learning research – and in particular reinforcement learning – there has been a major interest on optimal control with less assumption on the structure of the actions, less effort has been placed in studying a large class of problems where the agent is capable of performing concurrent activities. As shown throughout this dissertation, one natural approach for generating concurrency in the system is a combination of techniques that take advantage of strategies from biology and machine learning.

6.1 Summary

In summary, we introduced a decision theoretic framework for modeling the concurrent decision making problem, and presented a set of techniques for addressing major challenges in this class of problems.

In Chapter 3 we introduced *concurrent action models* (CAMs) – a general abstract decision theoretic model – for decision making with a set of concurrent activities. The main challenge with the modeling effort stems from the fact that activities that run in parallel, do not terminate at the same time. To address this challenge, we introduced a set of concurrent coordination mechanisms that incorporate various natural activity completion mechanisms based on the individual termination of each activity. We presented a set of theoretical results that assert the correctness of the model semantics which then allows us to apply standard SMDP learning and planning for solving this problem. We also presented a set of theoretical results that characterize the optimality of the agent's concurrent behavior based on various coordination mechanisms.

Although CAMs provide a base-model for studying the concurrent decision making problem, they still suffer from the curse of dimensionality due to the combinatorial space of concurrent actions. Our experimental results with CAMs – where we used standard SMDP learning and planning techniques – showed that even in a problem with a small set of states and concurrent actions, it takes a long time for such methods to learn the optimal concurrent policy.

We addressed this challenge in Chapter 4 where we presented an alternative view of generating concurrency in the system. We demonstrated that one natural way for performing activities concurrently is through the interaction of previously acquired skills in the system when the agent is faced to solve a multi-objective problem. A related concept in biology is known as the *coarticulation* phenomenon with a long history in motor control research. We presented a model of coarticulation in Markov decision processes for generating concurrency in the system. The basic blocks of this model are a set of coarticulatory controllers – namely the ϵ-ascending controllers – where each controller represent a class of near-optimal policies that guarantee progress toward the goal state of the controller in every state of the problem. By exploiting such flexibility, we presented algorithms for performing coarticulation and demonstrated that how this approach reduces the curse of dimensionality by performing the search for the best concurrent policy in a much smaller space, constrained by the degree of flexibility that each redundant controller offers. Another major contribution of this chapter is a set of theoretical results that establish time bounds on time for task completion for coarticulation algorithm, and also the sequential algorithm (where the agent is allowed to perform only one activity at a time).

Although performing coarticulation would considerably alleviate the curse of dimensionality, the algorithms for computing the redundant sets of policies associated with a controller still do not scale to large problems. In Chapter 5 we presented a set of approximation techniques for addressing this challenge. The key idea in our approach is that many concurrent decision making problems are inherently multi-objective optimization problems, in which the overall objective of the problem can be approximately expressed in terms of a linear combination of a set of sub-utility functions, each associated with an objective of the problem. By exploiting such linear structure, we presented an approximate algorithm that performs the computation required for the coarticulation approach with a tractable complexity.

For evaluating the set of techniques and algorithms that we presented throughout this chapter, we conducted a set of experiments on a simulated robot capable of performing concurrent activities. We demonstrated how coarticulation can naturally generate parallel execution activities in the system. The results show the coarticulation approach can be tractably performed and it outperforms the sequential algorithms by a large margin.

In summary, coarticulation is a very natural way of generating concurrent activities. It enables the agents to reuse their previously acquired skills with a minimum effort when faced with a new task. In the life span of an artificial agent, the goals are not known a priori; while the agent is committing to the current set of goals, a new set of objectives may arrive in the system. Without coarticulation, it is not feasible to initiate a new learning experiment for every possible combination of goals.

6.2 Future Work

There are a number of questions and open issues that remain to be addressed and many interesting directions in which this work can be extended:

- In concurrent action models (CAMs) we introduced three concurrent termination mechanisms. In fact we conjecture any other form of co-termination that may take place among a set of parallel activities, can be considered as a valid termination condition. One interesting direction for future investigation is to study other forms of concurrent activity termination.

• Although the strict order of priority in the coarticulation framework facilitates theoretical analysis of the behavior of the agent, it is not a realistic assumption in general. In many problems, the strict order of priority of subgoals may be violated in order to improve the overall performance. Perkins (2002) characterized a set of conditions under which the agent can violate ascending on a Lyapunov surface. This idea can be extended to partially relax the strict order of priority in the coarticulation framework.

• The key advantage of coarticulation for alleviating the curse of dimensionality is that it performs the search for the best concurrent policy, in a tractable space of concurrent policies induced by the set of ascending policies associated with the subgoals of the problem. This is in particular plausible for a class of reinforcement learning problems – such as pure policy gradient methods (Williams, 1992; Sutton et al., 2000; Baxter and Bartlett, 2000) – that do not explicitly incorporate value functions. An interesting future direction is to investigate the concept of *ascendancy* in non-value based RL methods.

• We believe that learning is an ongoing process in any agent. Even when we use our previously learned skills to accomplish a new task, we continue learning and modifying our skills constantly. This process is not addressed in our coarticulation framework. One interesting direction for further investigation is to incorporate learning while performing coarticulation. For example when an agent is faced with a new task, it can balance between its exploratory actions, and coarticulatory actions in order to learn the optimal solution.

• As described in Section 4.1 of Chapter 4, one known form of coarticulation in biological systems is when there exist a partial order among subgoals. For example when typing, there exists a sequential constraint induced by the text when pressing the letters. One interesting direction for further investigation is to incorporate such sequential constraints in the model in order to perform coarticulation in a larger class of problems.

• Our approach for representing redundancy in a controller is based on direct search in the space of policies of the controller. There is an interesting connection between *proto value functions* (Mahadevan, 2005) and the *uncontrolled* manifold concept that provides a representation of redundancy in motor control research. One interesting direction for further investigation is to study how to derive a functional representation of redundancy in discrete MDPs.

CHAPTER 7

RELATED WORK

In this section we briefly overview various related work that touch upon different forms of concurrency in different fields of science and engineering.

7.1 Concurrent Actions in MDPs

In MDPs, one natural way to model concurrent actions would be to extend the actions to multi-dimensional actions, or *vector-actions*. Some of the related work that use multi-dimensional action representation do not explicitly study this problem from the concurrent decision making perspective, nevertheless the concurrent execution of actions are implied in them.

Markey (1993) and Tham and Prager (1994) maintain a set of local Q-functions, one for each element of the vector-action, and update them independently using Q-learning (Watkins, 1998). Cichosz (1995, 1997) combined the local Q-functions to define a utility function for a vector-action, simply by taking average of the local Q-functions (or average of maximum of each local Q-function) when computing the (optimal) Q-function over vector actions. The main problem with these methods is the naive way of combining Q-functions based on a total independence assumption among action components.

Rosenstein and Barto (2001, 2002); Rosenstein (2003) also addressed reinforcement learning with multi-dimensional actions in robot weight lifting task by direct policy search (Rosenstein and Barto, 2001), and also in a hybrid setting that combines supervised learning with an actor-critic architecture (Rosenstein and Barto, 2002; Rosenstein, 2003), where a separate policy structure (the *actor*) is used to compactly represent the policy over composite actions.

Peshkin et al. (1999) investigated the learning of policies with external memory (i.e., *stigmergic policies* (Peshkin et al., 1999)). In this model, actions were augmented by a set of *memory* actions that are used to modify the content of the internal memory bits. Thus, at every step, a set of actions are executed, some of them only pertaining to the internal memory.

Boutilier and Goldszmidt (1995) lay out the basic ideas for exploiting the structure of the problem for policy construction based on the stochastic version of goal regression in stochastic systems (Boutilier and Goldszmidt, 1995; Boutilier and Dearden, 1996; Boutilier et al., 2000, 1999). This is an important step that gives the basic framework for modeling a broad set of problems that have structure in states and action. Dean et al. (1998); Boutilier et al. (1999) develop representation for factored actions, and introduce variety of techniques for specifying the transition matrices in order to provide representations that are more compact than explicit transition matrices.

Gabor et al. (1998) study the *multi-criteria* sequential decision making problem in which the learning objective is to attain multiple goals. Traditionally, RL algorithms solve sequential decision making problems when the optimization criterion is expressed in a recursive form (i.e., Bellman equations) using a scalar valued reinforcement signal (reward function). However, there are problems where the optimization criteria may not be simply expressed as a function of a single scalar reinforcement (Gabor et al., 1998). One such class of problems is when the learning agent is required to attain multiple goals (generally competing with each other) simultaneously. Gabor et al. (1998) consider multi-criteria decision problems in the framework of *abstract dynamic programming* (Littman and Szepesvári, 1996), where the reinforcement signal is assumed to be vector valued (each element of the reinforcement vector is associated with one of the criterions to be achieved) and are compared by a total ordering defined over an appropriate vector space.

Another way of modeling multi-dimensional actions in MDPs is to model the problem as a *cross-product MDP*, where we assume that the original MDP (alternatively called the *composite MDP*) decomposes into a set of sub-MDPs, where the state space and action space of the composite MDP is the cross product of the state and action spaces of the sub-MDPs, respectively (Singh and Cohn, 1998). Using this model, Singh and Cohn (1998) introduced the *dynamic merging problem*, which is the problem of learning the optimal solution for the composite MDP, from the solutions of its sub-MDPs. More specifically, the proposed algorithm, efficiently learns the optimal composite policy given only bounds on the value functions of the component MDP. This work, assumes that the composite MDP completely factors into sub-MDPs and ignores joint action influence on each sub-MDP.

Meuleau et al. (1998) introduce a method for approximating optimal solutions in in MDPs with large state and action spaces. They assume that the overall objective of the problem can be expressed as a set of tasks whose utilities are independent, and the actions taken with respect to a task do not affect the other tasks. However, these sub-tasks (each modeled as a MDP) are weakly coupled by constraints imposed by the problem resources, i.e., actions selected for a task, constraints the set of actions available to other tasks. These two key properties are used to avoid explicitly enumerating the very large state and action spaces.

Sallans and Hinton (2000); Sallans (2002) explored the action selection problem in MDPs with structured states and actions (i.e, factored-MDPs), and used a family of undirected graphical models named *product of experts* (Hinton, 2000) to capture the dependencies among state and action variables. They showed that they can approximate the state-action values as the negative free-energy of the state-action pair up to an additive constant) under such models. This provides a compact representation of the state-action values and can be used for learning the parameters of such models.

Younes and Simmons (2004) introduce *generalized semi-Markov decision processes* where they explore stochastic decision making in a continuous time, asynchronous systems, capturing a large class of concurrent decision making problems, including those that also include temporal activities with a semi-Markov property.

Mausam and S.Weld (2004); Mausam and Weld (2005) introduce concurrent Markov decision processes and introduce various algorithms for fast concurrent policy construction. In their model they consider multiple parallel actions, each of a unit duration. Also, the model is augmented by a set of conditions under which actions can be executed concurrently. This model is plausible when there exists a conflicting set of actions in the system, something that we did not explicitly address in this dissertation.

7.2 Planning with Concurrent Actions

There has been an extensive study of action representation and reasoning with actions in situation calculus, temporal logic and reasoning, and planning. Parts of these lines of research deal with the specification and synthesis of concurrent actions. Here we give a summary of these works.

Baral and Gelfond (1997) provide a semantic account of concurrency based on the well known action description language $\mathcal{A}_c$ (Gelfond and Lifschitz, 1992) which bears many similarities to the situation calculus formulation of concurrent actions. However the action language they developed is based on the propositional logic. Bornscheuer and Thielscher (1994) build on the action description language $\mathcal{A}_c$ that addresses non-determinism and uncertainty. Reiter (1996a,b) provide generalization of the temporal situation calculus to include concurrency, continuous time and various types of actions. There has been also a massive literature on concurrent processes, dynamic logic, and temporal logic (see (Winskel, 2002) for an overview).

The idea of incorporating partial-order planning (Sacerdoti, 1975, 1977) to generate parallel execution plans has been studied since early days of planning. The problem of parallel plan execution has been also addressed based on temporal planning (Allen et al., 1991). Parallel execution problem can be handled by a temporal planner, but just the complexity associated with testing the satisfiability of a set of assertions is shown to be NP-hard (Vilain et al., 1989). Veloso et al. (1991); Regnier and Fade (1991) approach this problem by generating totally-ordered plans and converting them into a partially-ordered plan. However, the parallel execution plans are constraint by the particular choice of the totally-ordered plans. Backstrom (1993) studied the complexity of the general problem of finding an optimal parallel execution plan and showed that this problem is NP-hard.

Knoblock (1994) investigated the idea of using partial-order planning to generate parallel execution plans in terms of underlying assumptions and situations for correctness of such plans, and whether the planner can even find a plan if one exists. Boutilier and Brafman (2001b,a) extend the STRIPS action representation language to represent concurrent interacting actions. They augment the STRIPS representation for each action with a *concurrent action list* that constrains the set of actions that can be executed concurrently with this action and develop the semantics for defining concurrent actions. They also present the *partially ordered multi-agent planning* (POMP) algorithm that builds on standard partial ordered planning algorithms to allow planning with concurrent interacting actions.

7.3 Multi-Agent Systems

Naturally concurrency can be viewed as a sub-class of multi-agent systems with a cooperative-centralized coordination system, and hence all of the extensive studies and advances in multi-agent systems are related to the concurrent decision making problem. Due to a large literature on multi-agent systems, we only present the most relevant work to our approach in the context of MDPs (for a complete set of work on multi-agent systems please see the various proceedings of the International Conference on Multi-agent Systems).

Ephrati and Rosenschein (1994) propose a heuristic multi-agent planning framework that incorporates sub-goals and sub-plans to construct a global plan. The overall goal of the problem is decomposed into a set of sub-goals a priori, and each agent solves its local sub-goal. Then the sub-goals are merged into a global plan using a set of heuristics.

In MDPs, the multi-agent problem is often referred to as *multi-agent reinforcement learning* and most of the work can be divided into work on competitive models vs. cooperative models. Littman (1994), and Hu and Wellman (1999) investigated the problem of Markov games for competitive multi-agent systems. Tan (1997) studied the cooperative multi-agent systems where he extends the Q-learning to multi-agent Q-learning by using joint state-action values. However the agents have to communicate their states and actions.

Makar et al. (2001); Ghavamzadeh et al. (2006) studied the use of hierarchy in multi-agent reinforcement learning bas on MAXQ framework (Dietterich, 2000). In this approach, the agents learn to coordinate at higher levels of temporal abstraction (equivalently, higher levels in the MAXQ hierarchy) by using joint action representation for each agent at the highest level of the hierarchy.

Guestrin et al. (2002); Guestrin and Gordon (2002); Guestrin et al. (2001) developed efficient learning and planning algorithms for cooperative multi-agent dynamic systems based on factored MDPs (Boutilier et al., 1999). In this approach, each agent maintains a local Q-function that are combined linearly to provide an approximation of the overall Q-function. To cope with the action selection problem, at every iteration of the Q-learning, they use an algorithm similar to the variable elimination algorithm in graphical models to efficiently search for the best actions. The use of local Q-function is closely related to our approach for approximately computing the redundant-sets in controllers described in section 5.5.1.

Russell and Zimdars (2003) also use a very similar approach (to that of (Guestrin et al., 2002)) called *$\mathcal{Q}$-decomposition* in a centralized-control multi-agent setting that assumes that the agent's overall $\mathcal{Q}$ function can be additively decomposed into local $\mathcal{Q}$ functions for each agent (referred to as a *sub-agent*). Each agent then reports its local $\mathcal{Q}$-function to an arbitrator that would then choose an action that maximizes the overall $\mathcal{Q}$-function, that is the sum of the local $\mathcal{Q}$-functions. They showed that if each sub-agent runs SARSA algorithm (Rummery and Niranjan, 1994) to learn its local $\mathcal{Q}$ function, then a globally optimal policy can be achieved.

7.4 Robotics and Control

Redundancy has been extensively investigated in robotics and motor-coordination. The main body of work is based on utilization of redundancy in systems with excess degrees of freedom involving some form of pseudo-inverse, in robotics (Liegeois, 1977; Yoshikawa, 1984; Nakamura, 1991), and in motor-coordination (Pellionisz and Llinas, 1985; Saltzman and Kelso, 1987; Mussa-Ivaldi et al., 1988; Jordan and Rosenbaum, 1989; Jordan, 1990; Todorov and Jordan, 2002).

In robotics, concurrency is primarily introduced based on the redundancy in the system kinematics and the degrees of freedom (DOF) in the system (Nakamura, 1991). Such redundancy is then utilized in the context of concurrent decision making. The common trend is to approximate the overall task in terms of concurrent optimization of set of sub-tasks, based on their degrees of significance. Then redundancy in the system, then can be exploited in such a way that the subordinate sub-tasks are realized using redundancy not committed to satisfying the superior sub-tasks. This approach has been extensively studied in many robotics tasks such as obstacle avoidance (Khatib and Maitre, 1978; Nakamura, 1991), avoiding mechanical joint limits (Liegeois, 1977), and singularity avoidance (Yoshikawa, 1984).

Researchers from the *Laboratory for Perceptual Robotics* (LPR) at University of Massachusetts have extensively studied redundancy in various robotics problems (Huber et al., 1996; Sweeney et al., 2002; Platt et al., 2002; Huber and Grupen, 2002a; Huber, 2000; Huber and Grupen, 2002b). They employ the general concept of redundancy and null-space approach, based on the“subject-to” relation. This relation is represented as $\phi_i \triangleleft \phi_j$, where ϕ_i represents the i_{th} control law and $\triangleleft$ expresses the “subject to” relation. In this approach the control laws are combined by forming the actions of subordinate control laws subject to the constraints imposed by other control laws (for example *grasping* subject to the *obstacle-avoidance* constraint). Huber et al. (1996); Huber and Grupen (2002b) use a set of control laws (such as *configuration-space motion control, contact-configuration control* and *posture control*) and combine them by defining a set of constraints among them (in terms of the "subject-to" relation) for multi-legged walking and generating robust finger gaits in object manipulation. Sweeney et al. (2002) propose a framework for coordinating multiple communicating robots where the system is modeled as a highly redundant system with multiple objectives and concurrent decision making is performed using a generalization of null space control. Platt et al. (2002) proposed a method for combining multiple control laws for the *robot grasping*. They considered three control laws (*force-based contact position, moment-based contact position*, and *kinematic conditioning* control laws) that are simultaneously active and are combined by projecting the actions of the subordinate control laws into the null-space of the superior control laws, with the relation $\phi_{force} \triangleleft \phi_{moment} \triangleleft \phi_{kinematic}$.

Brooks (1986) developed a layered architecture, known as the *subsumption architecture*, for controlling mobile robots. Each layer implements a controller that interacts with the other controllers in the system. Higher level controllers can subsume lower level laws of control by suppressing their outputs. This is very similar to the approach that we described for exploiting redundancy in a sense that in subsumption architecture we have a set of concurrent controllers that are prioritized based on their degrees of significance. However in this architecture higher level control laws may completely suppress the subordinate control laws.

Jordan (1990) developed a model for motor learning with an emphasis on problems involving excess degrees of freedom. The model consists of an internal predictive model (referred to as a *forward model* and a set of intrinsic constraints such as *smoothness, distinctiveness* and *rest configuration,* on motor learning. Experiments on a manipulator with six degrees of freedom showed that by using the smoothness constraint, the model uses the excess degrees of freedom to anticipate and manifest other actions.

Todorov and Jordan (2002) developed a motor-coordination theory based on optimal feedback control that learns strategies that allows variability in redundant (task-irrelevant) dimensions. The redundancy in this model can be exploited to improve performance, attains task constrained variability, motor synergies, goal-directed corrections and noise buffering.

Perkins (2002); Perkins and Barto (2001b,a) investigated the use of Lyapunov function in RL systems to achieve stability in dynamical system. A Lyapunov function constraints the set of the actions available to the agent that would allow the agent to descent on the Lyapunov surface and guarantees achieving the goals in problems whose objective is to minimize cost to the goal. Some elements of our coarticulation approach is motivated by this work. In fact, the ascendancy property of a policy emerges naturally when the value functions associated with subgoals of the problem are used as the Lyapunov constraints themselves.

APPENDIX

GLOSSARY OF NOTATION

Table A.1. Glossary of Notation.

Notation	Definition
$\mathcal{T}, \tau$	Termination mechanism in CAMs
$\mathcal{T}_{any}$	Termination mechanism when any of the activities terminates
$\mathcal{T}_{all}$	Termination mechanism when all of the activities terminates
$\mathcal{V}^{\langle \pi, \tau \rangle}(s)$	Value of state $\mathbf{s}$ under policy π using the termination mechanism τ
$\mathcal{V}^{*_\tau}(s)$	The optimal value of state s using the termination mechanism τ
$\mathcal{V}^{*_{all}}_{any}(s)$	The value of state s under the policy $\pi^{*_{all}}$ when it is terminated using the termination mechanism any
$\mathcal{V}^{*_{all}}_{n,any}(s)$	The value of state s under the policy $\pi^{*_{all}}$ when it is terminated using the termination mechanism $\mathcal{T}_{any}$ for the first n steps, and $\mathcal{T}_{all}$ for the rest of the steps
$\mathcal{Q}^{\langle \pi, \tau \rangle}(s, a)$	State-action value of state $\mathbf{s}$ under policy π using the termination mechanism τ
$\mathcal{Q}^{*_\tau}(s, a)$	The optimal state-action value of state s using the termination mechanism τ
$\pi^{*_\tau}, *_\tau$	The optimal policy based on the termination mechanism τ
$\pi_{cont}(\langle s, h \rangle)$	The *continue* policy in state s and *continue-set* h
$\pi_{seq}(s)$	The *sequential* policy in state s
$\Theta(\pi, s_t, \tau)$	The event of initiating the multi-action $\pi(s_t)$ at time t in state s_t using the termination mechanism τ
$\mathcal{R}^{\langle \mathbf{a}, \tau \rangle}_s$	The expected partial return of executing the multi-action $\mathbf{a}$ in state s using the termination mechanism τ
$\mathbf{h}_t$	*continue-set*, or the set of activities that have not been terminated at time t
$\xi^\tau(s, \mathbf{a}, s', k)$	Multi-step transition probability of executing the multi-action $\mathbf{a}$ in state $\mathbf{s}$ and transitioning to state $\mathbf{s}'$ in $\mathbf{k}$ steps
$\mathcal{P}^\tau(s, \mathbf{a}, s', k)$	Multi-step transition probability of executing the multi-action $\mathbf{a}$ in state $\mathbf{s}$ and terminating in state $\mathbf{s}'$ in $\mathbf{k}$ steps
κ^π	The minimum ascend rate of an ascending policy π
η^π	The maximum ascend rate of an ascending policy π

Notation	Definition
$\bar{\kappa}_{\mathcal{C}}$	The minimum ascend rate of an ϵ-redundant controller $\mathcal{C}$
$\bar{\eta}_{\mathcal{C}}$	The maximum ascend rate of an ϵ-redundant controller $\mathcal{C}$
$\chi^{\epsilon}_{\mathcal{C}}$	The set of redundant policies for an ϵ-redundant controller $\mathcal{C}$
$\triangleleft$	*subject-to* relation
$\tau_{\mathcal{C}}(s)$	The worst expected time an ϵ-redundant controller $\mathcal{C}$
	to arrive in a goal state when initiated in state **s**
$\sigma_{\mathcal{C}}(s)$	The maximum ascend rate of an ϵ-redundant controller $\mathcal{C}$

BIBLIOGRAPHY

Abbs, J. H., Gracco, V. L., and Cole, K. J. (1984). Control of multimovement coordination: sensorimotor mechanisms in speech motor programming. *Journal of motor behavior*, 16:195–231.

Albus, J. (1981). *Brain, Behavior, and Robotics*. ByteBooks.

Allen, J., Kautz, H., and Pelavin, R. (1991). *Reasoning About Plans*. Morgan Kaufmann, San Mateo, CA.

Andrews, G. and Elliott, S. (1991). *Concurrent Programming: Principles and Practice*. Pearson Education POD.

Baader, A. P., Kasennikov, O., and Wiesendanger, M. (2005). Coordination of bowing and fingering in violin playing. *Cognitive Brain Research*, 23:436–443.

Backstrom, C. (1993). Finding least constrained plans and optimal parallel executions is harder than we thought. In *Current Trends in AI Planning: EWSP'93–2nd European Workshop on Planning*, pages 46–59, Vadstena, Sweden.

Baral, C. and Gelfond, M. (1997). Reasoning about effects of concurrent actions. *Journal of Logic Programming*, 31(1-3):85–117.

Baxter, J. and Bartlett, P. L. (2000). Reinforcement learning in POMDP's via direct gradient ascent. In *Proceedings of the 17th International Conference on Machine Learning*, pages 41–48. Morgan Kaufmann, San Francisco, CA.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, USA.

Bernstein, N. A. (1967). *The Co-ordination and Regulation of Movements*. Pergamon Press, Oxford.

Bertsekas, D. and Tsitsiklis, J. (1997). *Neuro-Dynamic Programming*. Athena Scientific.

Bornscheuer, S. and Thielscher, M. (1994). Representing concurrent actions and solving conflicts. In *KI - Kunstliche Intelligenz*, pages 16–27.

Boutilier, C. and Brafman, R. (2001a). Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136.

Boutilier, C. and Brafman, R. (2001b). Planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136.

Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94.

Boutilier, C. and Dearden, R. (1996). Approximatin value trees in structured dynamic programming. *In Proceedings of the thirteenth International Conference on Machince Learning*, pages 54–62.

Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1–2):49–107.

Boutilier, C. and Goldszmidt, M. (1995). Exploiting structure in policy construction. *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111.

Boyan, J. A. (1999). Least-squares temporal difference learning. In *Proc. 16th International Conf. on Machine Learning*, pages 49–56. Morgan Kaufmann, San Francisco, CA.

Bradtke, S. and Duff, M. (1995). Reinforcement learning methods for continuous-time markov decision problems. *Advances in Neural Information Processing Systems*, 7:393–400.

Breteler, M. D. K., Hondzinski, J. M., and Flanders, M. (2003). Drawing sequences of segments in 3d: kinetic influences on arm configuration. *Journal of Neurophysiology*, 89:3253–3263.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 14–23.

Cichosz, P. (1995). Learning multidimensional control actions from delayed reinforcements. In *Eighth International Symposium on System-Modelling-Control (SMC-8)*, Zakopane, Poland.

Cichosz, P. (1997). *Reinforcement Learning by Truncating Temporal Differences*. PhD thesis, Department of Electronics and Information Technology, Warsaw University of Technology.

Cohen, R. G. and Rosenbaum, D. A. (2004). Where grasps are made reveals how grasps are planned: generation and recall of motor plans. *Experimental Brain Research*, 157:486–495.

Craig, J. (1989). *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Pub Co.

Dayan, P. (1992). The convergence of $td(\lambda)$ for general λ. *Machine Learning*, 8:341–362.

Dean, T., Givan, R., and Kim, K. (1998). Solving planning problems with large state and action spaces. In *In Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (ICAPS-98)*, pages 102–110.

Dechter, R. (1999). *Bucket Elimination: A Unifying Framework for Probabilistic Inference*. Artificial Intelligence.

Dietterich, T. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.

Engel, K. C., Flanders, M., and Soechting, J. F. (1997). Anticipatory and sequenctial motor control in piano playing. *Experimental Brain Research*, 113:189–199.

Ephrati, E. and Rosenschein, J. S. (1994). Divide and conquer in multi-agent planning. In *National Conference on Artificial Intelligence*, pages 375–380.

Gabor, Z., Kalmar, Z., and Szepesvari, C. (1998). Multi-criteria reinforcement learning. In *Proceedings of International Conference of Machine Learning*.

Gelfond, M. and Lifschitz, V. (1992). Representing actions in extended logic programs. In *Proceedings of Joint International Conference and Symposium on Logic Programming*, pages 559–573. MIT Press.

Ghavamzadeh, M., Mahadevan, S., and Makar, R. (2006). Hierarchical multiagent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*.

Grupen, R. A. (pending 2006). *The Developmental Organization of Robot Behavior*. MIT Press.

Guestrin;, C. (2003). *Planning Under Uncertainty in Complex Structured Environments*. PhD thesis, Stanford University.

Guestrin, C. and Gordon, G. (August 2002). Distributed planning in hierarchical factored mdps. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 197–206, Edmonton, Canada.

Guestrin, C., Koller, D., and Parr, R. (2001). Multiagent planning with factored mdps. In *Proceedings of the 14th Neural Information Processing Systems*, Vancouver, British Columbia, Canada.

Guestrin, C., Lagoudakis, M., and Parr, R. (2002). Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 227–234, Sydney, Australia.

Hennessy, J. and Patterson, D. (1990). *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, San Mateo, CA.

Hinton, G. (2000). Training products of experts by minimizing contrastive divergence.

Hoff, B. and Arbib, M. (1993). Models of trajectory formation and temporal interaction of reach and grasp. *Journal of Motor Behvavior*, 25:175–192.

Howard, R. (1971). *Dynamic Probabilistic Systems:Semi-Markov and Decision Processes*. John Wiley and Sons Inc, New York.

Hu, J. and Wellman, M. (1999). Multiagent reinforcement learning in stochastic games.

Huber, M. (2000). *A Hybrid Architecture for Adaptive Robot Control*. PhD thesis, University of Massachusetts, Amherst.

Huber, M. and Grupen, R. (Sep. 2002a). Robust finger gaits from closed-loop controllers.

Huber, M. and Grupen, R. A. (2002b). Robust finger gaits from closed-loop controllers.

Huber, M., MacDonald, W., and Grupen, R. (1996). A control basis for multilegged walking.

Iba, G. A. (1988). A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3:285–317.

Jeannerod, M. (1981). *Intersegmental coordination during reaching at natural visual objects*, volume 9, pages 153–168. Erlbaum, Hillsdale, NJ.

Johnson, S. H. and Grafton, S. (2003). From 'acting on' to 'acting with': the functional anatomy of object-oriented action schemata. *Progress in Brain Research*, 142:127–139.

Jordan, M. (1990). Motor learning and the degrees of freedom problem. *Attention and Performance*, XIII:796–836.

Jordan, M. and Bishop, C. (2002). An introduction to graphical models.

Jordan, M. and Rosenbaum, D. (1989). Action. *M. I. Posner (Ed.), Foundations of Cognitive Science*.

Kearns, M. and Singh, S. (1998). Near-optimal reinforcement learning in polynomial time. In *Proc. 15th International Conf. on Machine Learning*, pages 260–268. Morgan Kaufmann, San Francisco, CA.

Keeney, R. and Raiffa, H. (1993). *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Cambridge University Press; New Ed edition, Cambridge, UK.

Kent, R. D. and Minifie, F. D. (1977). Coarticulation in recent speech production models. *Journal of Phonetics*, 5:115–117.

Khatib, O. and Maitre, L. (1978). Dynamic control of manipulators operating in a complex environment.

Knoblock, C. A. (1994). Generating parallel execution plans with a partial-order planner. In *Artificial Intelligence Planning Systems*, pages 98–103.

Koller, D. and Parr, R. (1999). Computing factored value functions for policies in structured mdps. *16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1332–1339.

Liegeois, A. (1977). Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Trans. Sys., Man, Cyber. SMC-7*, 12:868–871.

Littman, M. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, pages 157–163, New Brunswick, NJ. Morgan Kaufmann.

Littman, M. L. and Szepesvári, G. (1996). A generalized reinforcement-learning model: Convergence and applications. In Saitta, L., editor, *Proceedings of the 13th International Conference on Machine Learning (ICML-96)*, pages 310–318, Bari, Italy. Morgan Kaufmann.

Mahadevan, S. (2005). Proto-value functions: Developmental reinforcement learning. *Proceedings of the International Conference on Machine Learning*.

Makar, R., Mahadevan, S., and Ghavamzadeh, M. (2001). Hierarchical multi-agent reinforcement learning. In Müller, J. P., Andre, E., Sen, S., and Frasson, C., editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 246–253, Montreal, Canada. ACM Press.

Markey, K. (1993). Efficient learning of multiple degree-of-freedom control problems with quasi-independent q-agents. *Proceedings of the 1993 Connectionists Models Summer School*.

Marthi, B., Russell, S., Latham, D., and Guestrin, C. (2005). Concurrent hierarchical reinforcement learning. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*.

Mausam and S.Weld, D. (2004). Solving concurrent markov decision processes. *AAAI*.

Mausam and Weld, D. S. (2005). Concurrent probabilistic temporal planning. *ICAPS*.

McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proc. 18th International Conf. on Machine Learning*, pages 361–368. Morgan Kaufmann, San Francisco, CA.

Meuleau, N., Boutilier, C., Hauskrecht, M., Kaelbling, L. P., Kim, K., Peshkin, L., and Dean, T. (1998). Solving very large weakly coupled markov decision processes. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 165–172, Cambridge, Massachusetts, USA. MIT Press.

Mussa-Ivaldi, F., McIntyre, J., and Bizzi, E. (1988). Theoretical and experimental perspectives in arm trajectory formation: A distributed model of motor redundancy. *In. Biological and Artificial Intelligent Systems*, pages 563–577.

Nakamura, Y. (1991). *Advanced robotics: redundancy and optimization*. Addison-Wesley Pub. Co.

Nilsson, D. (1998). An efficient algorithm for finding the m most probable configurations in bayesian networks. *Statistics and Computing*, 8(2):159–173.

Papadimitriou, C. and Tsitsiklis, J. (1987). The complexity of markov chain decision processes.

Pellionisz, A. and Llinas, R. (1985). Tensor network theory of the metaorganization of functional geometries in the central nervous system. *Neuroscience*, 16, No. 2:245–273.

Perkins, T. (2002). *Lyapunov Methods for Safe Intelligent Agent Design*. PhD thesis, Department of Computer Science, University of Massachusetts Amherst.

Perkins, T. J. and Barto, A. G. (2001a). Heuristic search in infinite state spaces guided by lyapunov analysis. In *IJCAI*, pages 242–247.

Perkins, T. J. and Barto, A. G. (2001b). Lyapunov-constrained action sets for reinforcement learning. In *Proc. 18th International Conf. on Machine Learning*, pages 409–416. Morgan Kaufmann, San Francisco, CA.

Peshkin, L., Meuleau, N., and Kaelbling, L. P. (1999). Learning policies with external memory. In Bratko, I. and Dzeroski, S., editors, *The Sixteenth International Conference on Machine Learning*, pages 307–314, San Francisco, CA, USA. Morgan Kaufmann.

Pickett, M. and Barto, A. G. (2002). Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*.

Platt, R., Fagg, A., and Grupen, R. (2002). Nullspace composition of control laws for grasping.

Precup, D. (2000). *Temporal Abstraction in Reinforcement Learning*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst.

Puterman, M. (1994). *Markov decision processes*. Wiley Interscience, New York, USA.

Regnier, P. and Fade, B. (1991). Complete determination of parallel actions and temporal optimization in linear plans of actions. In *Hertzberg,J., ed., European Workshop on Planning*, pages 100–111. Springer-Verlag.

Reiter, R. (1996a). Natural actions, concurrency and continuous time in the situation calculus. *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96), Cambridge MA., November 5-8, 1996*.

Reiter, R. (1996b). Natural actions, concurrency and continuous time in the situation calculus. In Aiello, L. C., Doyle, J., and Shapiro, S., editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 2–13. Morgan Kaufmann, San Francisco, California.

Rohanimanesh, K. and Mahadevan, S. (2005). Coarticulation: An approach for generating concurrent plans in markov decision processes. *In Proceedings of the 22nd International Conference on Machine Learning (ICML-2005)*.

Rohanimanesh, K. and Mahadevan, S. (August 2001). Decision-theoretic planning with concurrent temporally extended actions. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 472–479, University of Washington, Seattle, Washington, USA.

Rohanimanesh, K. and Mahadevan, S. (December 2002). Learning to take concurrent actions. In *Proceedings of the Sixteenth Annual Conference on Neural Information Processing Systems*, Vancouver, Canada.

Rohanimanesh, K., Platt, R., Mahadevan, S., and Grupen, R. (2004b). A framework for coarticulation in markov decision processes. Technical Report 04-33, Department of Computer Science, University of Massachusetts, Amherst, Massachusetts, USA.

Rohanimanesh, K., Platt, R., Mahadevan, S., and Grupen, R. (December 2004a). Coarticulation in markov decision processes. In *Proceedings of the Eighteenth Annual Conference on Neural Information Processing Systems: Natural and Synthetic*, Vancouver, Canada.

Rosenstein, M. (2003). *Learning to Exploit Dynamics for Robot Motor Coordination*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, Massachusetts, USA.

Rosenstein, M. and Barto, A. (2001). Robot weightlifting by direct policy search. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, volume 2, pages 839–844, Seattle, Washington, USA.

Rosenstein, M. and Barto, A. (2002). Supervised learning combined with an actor-critic architecture. Technical Report 02–41, Department of Computer Science, University of Massachusetts, Amherst, Massachusetts, USA.

Rummery, G. and Niranjan, M. (1994). On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Engineering Department, Cambridge University.

Russell, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, New Jersey, USA.

Russell, S. and Zimdars, A. (2003). Q-decomposition for reinforcement learning agents.

Sacerdoti, E. (1975). The non-linear nature of plans. *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI75)*, pages 206–214.

Sacerdoti, E. (1977). *A structure for plans and ehaviors*. Elsevier/North-Holland, Amsterdam, London, New York.

Sallans, B. (2002). *Reinforcement Learning for Factored Markov Decision Processes*. PhD thesis, Department of Computer Science, University of Toronto.

Sallans, B. and Hinton, G. (2000). Using free energies to represent Q-values in a multiagent reinforcement learning task. In *Proceedings of the Neural Information Processing Systems*, pages 1075–1081.

Saltzman, E. and Kelso, J. (1987). Skilled actions: A task-dynamic approach. *Psychological Review*, 94:84–106.

Şimşek, Ö. and Barto, A. G. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 751–758. ACM Press.

Şimşek, Ö., Wolfe, A. P., and Barto, A. G. (2005). Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the Twenty-Second International Conference on Machine Learning*.

Singh, S., Barto, A., and Chentanez, N. (2004). Intrinsically motivated reinforcement learning. *Proceedings of the 17th Annual Conference on Neural Information Processing Systems*.

Singh, S. and Cohn, D. (1998). How to dynamically merge markov decision processes. *Proceedings of NIPS 11*.

Soechting, J. F. and Flanders, M. (1992). Organization of sequential typing movements. *Journal of Neurophysiology*, 67:1275–1290.

Sutton, R., McAllester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 1057–1063. MIT Press.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. The MIT Press.

Sutton, R. S. and Barto, A. G. (1998). *An introduction to reinforcement learning*. MIT Press, Cambridge, MA.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211.

Sweeney, J., Brunette, T., Yang, Y., and Grupen, R. (2002). Coordinated teams of reactive mobile platforms.

Tan, M. (1997). Multi-agent reinforcement learning: Independent vs. cooperative learning. In Huhns, M. N. and Singh, M. P., editors, *Readings in Agents*, pages 487–494. Morgan Kaufmann, San Francisco, CA, USA.

Tanenbaum, A. and Woodhull, A. (1997). *Operating Systems: Design and Implementation (Second Edition)*. Prentice Hall.

Tham, C. and Prager, E. (1994). A modular q-learning architecture for manipulator task decomposition. *Proceedings of the Eleventh International Conference on Machine Learning*, pages 309–317.

Todorov, E. and Jordan, M. (2002). Optimal feedback control as a theory of motor coordination. *Nature Neuroscience*, 5:1226–1235.

Tsitsiklis, J. N. and Roy, B. V. (1996). An analysis of temporal-difference learning with function approximation. Technical Report LIDS-P-2322, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA.

Veloso, M., Pérez, M., and Carbonell, J. (1991). Nonlinear planning with parallel resource allocation. In *Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 207–212.

Vilain, M., Kautz, H., , and van Beek, P. (1989). Constraint propagation algorithms for temporal reasoning: A revised report. In *In D. S. Weld and J. de Kleer, editors, Readings in Qualitative Reasoning about Physical Systems*, pages 373–381, San Mateo, CA. Morgan Kaufmann.

Vincent, T. L. and Grantham, W. J. (1997). *Nonlinear and Optimal Control Systems*. John Wiley and Sons Inc., New York, USA.

Watkins, C. J. C. H. (1998). *Learning from Delayed Reward*. PhD thesis, Cambridge University, Cambridge, England.

Weiss, G. (2000). *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, MA.

Wiesendanger, M. and Serrien, D. (2001). Toward a physiological understanding of human dexterity. *News in Physiological Science*, 15:228–233.

Williams, R. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256.

Winskel, G. (2002). Topics in concurrency: Part ii comp. sci. lecture notes. *Computer Science course at the University of Cambridge*.

Yoshikawa, T. (1984). Analysis and control of robot manipulators with redundancy. *Robotics research, eds. M. Brady and R. Paul*, pages 735–747.

Younes, H. and Simmons, R. (2004). A formalism for stochastic decision processes with asynchronous events. *AAAI Workshop on Learning and Planning in Markov Processes:Advances and Challenges*.