Exploiting Structure for Planning and Control

Shieu-Hong Lin
Ph.D. Dissertation

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-97-14
September 1997

Abstract of " Exploiting Structure for Planning and Control "
by Shieu-Hong Lin, Ph.D., Brown University, May 1997.
Thesis advisor Thomas L. Dean.

Discrete dynamical systems in the form of finite automata or Markov decision processes have been used as a representational and computational foundation for planning under uncertainty. Many AI planning problems can be conveniently viewed as control problems over the underlying discrete dynamical systems. Using AI-style representation, the features of application domains are represented as state variables, and planning problem instances compactly encode very large discrete dynamical systems. The standard algorithms to solve the corresponding control problems require explicit enumeration of the underlying state spaces. This is impractical since the sizes of the state spaces are exponential in the number of state variables.

In this thesis, we develop decomposition techniques to exploit structure for planning problems in different application domains. Given a problem instance, we first symbolically decompose the underlying dynamical system into subsystems. After analyzing the local dynamics within subsystems and the dependency across subsystems, we decompose the original planning problem into a sequence of subproblems governing the subsystems. Rather than explicitly constructing the entire system, we compactly construct each subsystem by abstracting away the state variables that are irrelevant to the subsystem and its interactions with the other subsystems. Solutions of the planning subproblems can be derived by applying standard algorithms for the corresponding control problems to these compactly constructed subsystems, while a solution to the original planning problem is derived by systematically compiling and propagating the solutions to the planning subproblems. These decomposition techniques shed light on how to exploit locality and dependency to cope with very large state spaces.

Exploiting Structure for Planning and Control

by
Shieu-Hong Lin
B. S., National Taiwan University, 1987
M. S., National Taiwan University, 1989

Thesis
Submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy in the Department of Computer Science
at Brown University

May 1997

This dissertation by Shieu-Hong Lin
is accepted in its present form
by the Department of Computer Science
as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____

Thomas L. Dean

Recommended to the Graduate Council

Date _____ _____

Leslie Pack Kaelbling

Date _____ _____

Craig Boutilier
University of British Columbia

Approved by the Graduate Council

Date _____ _____

Vita

I was born in 1966 in Daoliu, an agricultural town in central Taiwan. My grandparents were farmers, earning their living and raising their children through perseverance and perspiration. My parents inhere this family characteristics but my farther works as a mechanic engineer instead in supporting our family. Although one of my grandparents was a learned confucianist also, advanced education was a luxury for the generations of my parents and my grandparents. It is based on the foundation established in their generations that I was able to step onto this track of advanced education.

My interests in science started during my days in high school. At that time, I enjoyed studying physics and exploring theoretical models for natural phenomena. I received my bachelor degree and my master degree from the computer science department of National Taiwan University in 1987 and 1989 respectively, and came to Brown University in 1991 after serving two years in the Taiwanese Army.

My journey at Brown is accompanied by the providence of the Lord. With the Lord's grace, I was led to Him in this period of time. Through many things in this journey, I began to be aware of the righteousness, forgiveness, and faithfulness of the Lord.

Acknowledgements

My advisor, Professor Thomas Dean, has been a source of encouragement along the path of my research here, introducing to me a broad perspective of what research is and patiently allowing me time for exploration in the process. It has been my pleasure to have Tom as my advisor.

This dissertation in its current form well reflects my research at Brown and its results. I would like to thank Professor Leslie Kaebling here and Professor Craig Boutilier at the University of British Columbia for their feedbacks and patience in reading my dissertation. Due to time pressure, not every aspect of their feedbacks is perfectly addressed in the dissertation. For example, I should have provided more examples and more intuitive presentation of some of the hairy details. I will work on a technical-report version of this dissertation and interested readers may look at this improved version for a better understanding of the underlying research.

I would like to express my appreciation to all the members of the AI-group lunch meetings. These informal weekly lunch meetings have allowed me good exposure to different research areas within AI or related to AI, and have inspired my interest in research. My friends Yi-Jen Chiang, Hsueh-I Lu, together with their family encouraged and helped me a lot when they were PhD students in our department, and so did my friend Lloyd Greenwald.

Here, I am far away from my family in Taiwan, but I have brothers and sisters in the Chinese Christian Church of Rhode Island. They are my family in Christ. This dissertation is a fruit of the Lord's providence, including the early planting effort of my parents. My father, Kung-Chang Lin, and my mother, Fu-Lan Liu, did their best in providing me and my younger brother Chia-Huey Lin

with good education. I would like to share this fruit with my parents and my brother together with my family in Christ for they all have shared with me many precious things in life.

Contents

Vita iii

Acknowledgements iv

List of Figures x

1 Introduction 1

1.1 Planning under Uncertainty 1

1.2 Planning and Control 2

1.3 Exploiting Locality and Dependency 3

1.4 Outline of the Thesis 4

2 Planning and Control: Foundations 6

2.1 Discrete Dynamical Systems 6

2.2 Finite Automata and Markov Decision Processes 8

2.3 Compact Representations 10

2.3.1 Compactly Representing State-transition Relations 10

2.3.2 Compactly Representing Transition Probabilities and Costs 12

2.4 Planning and Control over Discrete Dynamical Systems 14

2.5 Algorithms for Optimal Control over Markov Decision Processes 17

3 Planning and Control: Applications and Computational Challenge 19

3.1 Multi-Level Task Networks and Temporal Projection 19

3.1.1 Multi-Level Task Networks 20

3.1.2 Multi-Level Task Networks as Finite Automata 21
3.1.3 Modeling a Process-Scheduling Domain 23
3.1.4 Application in Schedule Verification 26
3.1.5 Computational Complexity 26
3.2 Sequential Decision Networks and Optimal Decision Making . . . 27
3.2.1 Sequential Decision Networks 27
3.2.2 Sequential Decision Networks as Markov Decision Processes 30
3.2.3 Modeling a Production-Control Domain 31
3.2.4 Application in Factory Production Control 32
3.2.5 Application in Stochastic Transportation Planning 33
3.2.6 Computational Complexity 36
3.3 Coping with Very Large State Spaces 38

4 A Generic Overview of the Decomposition Techniques 41
4.1 Regional Decomposition and Inter-regional Interaction Patterns . 41
4.2 Structural Analysis of Inter-regional Interaction Patterns 44
4.3 Exploiting Structure for Computational Efficiency 45
4.3.1 Exploiting Structure in Multi-Level Permutation Patterns 47
4.3.2 Exploiting Structure in Acyclic Branching Patterns 49
4.3.3 Exploiting Structure in Strongly-connected Branching Patterns 51

5 Decomposition Techniques for the Temporal Projection Problem 54
5.1 More about Temporal Projection 54
5.2 Tradeoff in Computational Complexity 57
5.3 Locality in Multi-level Task Networks 61
5.4 Causal Dependency in Multi-level Task Networks 63
5.4.1 Subgoal Conditions and Problem Decomposition 64
5.4.2 Abstract Conditions and Abstract Events 67
5.4.3 Coupling Conditions and Localized Reasoning 70
5.5 Localized Temporal Projection Using Subgoals and Abstract Events 71

5.5.1 Temporal Projection as Global Search 71
5.5.2 Temporal Projection as a Sequence of Local Searches . 74
5.6 Quantitative Analysis of Computational Efficiency 78
5.7 Related Work 80
5.8 Summary 81

6 Decomposition Techniques for the Markov Decision Problem (I) 83
6.1 Coherent Fragment and Locality 84
6.2 Dependency among Coherent Fragments 88
6.3 A Decomposition Technique to Exploit Coherent Fragments . . . 91
6.4 Efficacy in Optimizing Sequential Decision Networks 94
6.4.1 Decomposing Sequential Decision Networks 95
6.4.2 Empirical Effectiveness in Dimensionality Reduction . . . 96
6.5 Related Work 103

7 Decomposition Techniques for the Markov Decision Problem (II) 104
7.1 Terminology and Notation 106
7.2 Abstraction and Hierarchical Policy Construction 109
7.2.1 Abstract Decision Processes 109
7.2.2 Hierarchical Policy Construction 110
7.3 The Iterative Approximation Approach 112
7.4 Example and Comparison 116
7.4.1 Employing Hierarchical Construction Approach 117
7.4.2 Employing the Iterative Approximation Approach 118
7.4.3 Comparing Computational Efficiency 119
7.5 Related work 122
7.6 Summary 123

8 Related Research 125
8.1 Real-time Planning in Stochastic Domains 125
8.2 Approximate Planning Using Abstract Policies 126
8.3 Aggregation and Abstraction 127

8.4 Probabilistic Inference in Bayesian Networks 127
8.5 Decomposition Theory in Relational Databases . 128

9 Conclusions **129**

A Proofs **131**

B Iterative Approximation and Linear Programming **142**
B.1 Linear Programming and the Dantzig-Wolfe Decomposition Principle 142
B.1.1 The Master Problem in the DW Decomposition 143
B.1.2 The Subproblems in the DW Decomposition 145
B.1.3 Conducting an Iteration of the DW Decomposition 146
B.2 The DW Decomposition and Markov Decision Processes 147
B.2.1 Formulating Markov Decision Processes as Linear Programs 148
B.2.2 The DW Approach in Solving Markov Decision Processes . 149

C More Details about the Iterative Approximation Method **151**
C.1 Relationship between the DW Approach and Iterative Approximation 151
C.2 Overview of the Iterative Method 153
C.3 Partitioning a State Space into Local Regions 154
C.4 The Abstract Action Space 154
C.5 Deriving Local Policies 155
C.5.1 Deriving the Local Policies over the Kernels 155
C.5.2 Deriving the Local Policies over the Peripheries 158
C.6 Further Information Associated with Local Policies 159
C.7 Updating the Policy Repository 161
C.8 Bounds, Stopping Criteria, and New Abstract Actions 162
C.9 The Initialization and Termination of the Iterative Method 164
C.9.1 Initializing the Policy Repository and Abstract Actions . . 164
C.9.2 Generating a Solution Policy from the Policy Repository . 165

List of Figures

2.1 Two actions and their associated causal rules 12

3.1 Process scheduling for a job shop 24
3.2 Preconditions and postconditions of the primitive tasks 25
3.3 A sequential decision network to improve product quality 31
3.4 A building of three floors and two elevators 34
3.5 Uncertainty in action outcomes 35
3.6 Three views of a three-dimensional state space with the corresponding local structure of space shown to the right. The top view represents the unstructured state space, the middle view represents an abstraction obtained by projection, and the bottom view represents a decomposition obtained by partitioning the states into aggregate states. 38
3.7 Region-by-region dimensionality reduction. A three-dimensional space is represented as the union of two-dimensional abstract subspaces shaded dark gray. 39

5.1 Temporal projection in a multi-level task network 56
5.2 The events and causal rules in the jobshop example 58
5.3 Complexity trade-off in temporal projection 59
5.4 The condition subsets characterizing the causal dependencies among regions 64
5.5 The regional subgoals and incremental subgoals of individual regions 66
5.6 The abstract events of individual regions 69

6.1 The fragment graph of the production-control decision network . . 86
6.2 Dependencies among coherent fragments 90
6.3 Statistics about decision networks with eighty state variables, sixteen decision stages, and a one-dimensional interconnection 99
6.4 Statistics about decision networks with eighty state variables, sixteen decision stages, and a two-dimensional interconnection 99
6.5 Statistics about decision networks with three hundred and twenty state variables, sixty four decision stages, and a one-dimensional interconnection 100
6.6 Statistics about decision networks with three hundred and twenty state variables, sixty four decision stages, and a two-dimensional interconnection 100
6.7 Effectivess of dimensionality reduction in decision networks with sixteen decision stages and a one-dimensional interconnection . . . 101
6.8 Effectivess of dimensionality reduction in decision networks with sixteen decision stages and a two-dimensional interconnection . . 101
6.9 Effectivess of dimensionality reduction in decision networks with sixty four decision stages and a one-dimensional interconnection . 102
6.10 Effectivess of dimensionality reduction in decision networks with sixty four decision stages and a two-dimensional interconnection . 102

7.1 Boundary (light gray) and periphery (darker gray) states of region R 107
7.2 R is adjacent to S ($R \rightsquigarrow S$) 107
7.3 Abstract Markov decision process 111
7.4 Abstract action for moving from R to S. Base-level action for moving from i to j. 111
7.5 Relationship between the partitions P and Q 114
7.6 An $N \times N$ area tessellated into N^2 squares and M^2 regions where $N = 8$ and $M = 4$ 120

C.1 The relationship between actual and biased action cost 157

Chapter 1

Introduction

1.1 Planning under Uncertainty

In classical planning, propositional STRIPS-like operators [FN71a] [AHT90] are employed to attain a goal where some state variables (propositions) have specified values. A classical plan is a multiset of totally ordered or partially ordered STRIPS-like operators [Cha87] [MR91]. The task of planning is to determine a plan such that by taking the actions (STRIPS-like operators) in an order consistent with the plan a specified goal is achieved at the end [FN71a] [Cha87] [AHT90] [PW92] [BW93] [Wel94].

Each planning instance involves a finite set of state variables. The values of the variables at a point in time determine a state. A STRIPS-like operator is only applicable in the states where the precondition associated with the operator is true. When applied in a state $\mathbf{s}$, an operator then changes the values of some state variables deterministically according to the state $\mathbf{s}$. In this setting, a STRIPS-like operator compactly represents a state-transition function over the state space determined by the set of state variables.

Recently, many researchers have devoted effort to generalize this classical planning framework to deal with planning under uncertainty [BD94] [BDG95] [DKKN93b] [DKKN93a] [DKKN95] [DHW94] [KHW94]. To cope with uncertainty, it is necessary to further extend the notions of STRIPS-like operators as

actions, plans as totally or partially ordered actions, and planning as a goal-achieving task under the classical framework. Rather than having a deterministic effect, taking an action in a specific state may end in one of several possible outcomes; the actual outcome is not known in advance. More general action dynamics such as state-transition relations are employed to model uncertainty and the classical notion of plans is extended to cope with the possibility of failure in attaining a goal when executing a plan. In addition, the information about action cost and the probability governing state transitions are often available also, and it is appropriate to consider more general performance criteria for planning rather than a simple dichotomy between attaining or not attaining a goal.

1.2 Planning and Control

In this thesis, we adopt discrete dynamical systems like finite automata [Paz71] [HU79] [LP81] and Markov decision processes [Paz71] [BS78] [Put94] as the representational and computational foundation for planning under uncertainty. A discrete dynamical system is composed of a finite state space, a finite action space, and an action dynamics governing how actions result in state transitions. A controller is an agent situated in a discrete dynamical system which regulates the system's state by taking actions. A discrete dynamical system evolves through a (possibly infinite) sequence of states as the controller takes actions. We define a plan to be a program executed by the controller in determining the actions to take. We investigate several performance criteria to evaluate the expected utility of executing a plan to control a discrete dynamical system. Under this framework, the task of planning is to search for a plan that provides a maximum expected utility regarding a specific performance criterion. This perspective conveniently casts planning under uncertainty as control over discrete dynamical systems [RW89a] [ZW90]. [Dea88] [Dea87] [BF88] [DW91] [LD95b] [LD96] and provides the computational leverage of employing existing algorithms developed by the operations research community.

1.3 Exploiting Locality and Dependency

As many planning problems can be formulated as control problems over compactly encoded discrete dynamical systems, a naive approach to solve these planning problems is to explicitly construct the underlying dynamical systems, and then directly apply standard search techniques or dynamic programming algorithms to derive an optimal solution. However, the sizes of the underlying state spaces usually grow exponentially in the sizes of the problem instances, which makes the naive approach infeasible. To cope with this computational barrier, researchers have explored different alternatives such as heuristic algorithms without quality guarantee [NK94] [DKKN93b] [DKKN93a] [DKKN95] approximation algorithms applicable under certain circumstances [BD94] [TVR96] and exact algorithms aggregating states on the fly in improving computational efficiency [BDG95] [BCn89].

In this thesis, we present exact decomposition techniques for planning and control. Computational efficiency is attained by exploiting locality and dependency recognized through static analysis of planning instances. In particular, we are interested in planning instances that employ AI-style compact representations [BD94] [BDG95] [BDH95] [LD95b] [LD96]. Compact representations often exhibit locality and dependency in action dynamics, which indicates regularity and redundancy in the underlying discrete dynamical systems. In many situations, explicit construction of the entire dynamical systems underlying such kinds of planning instances is not necessary. Rather than dealing with a large discrete dynamical system as a whole, we develop decomposition techniques that derive an optimal solution by systematically solving planning subproblems governing subsystems of reduced sizes. The utility of these decomposition techniques is orthogonal to the afore mentioned algorithms for planning and control. This is because we can always employ these existing algorithms in solving the planning subproblems and thus gaining additional computational efficiency.

We first symbolically decompose a discrete dynamical system into a set of regions, where a region is a subsystem concerning a state subspace together with the actions and dynamics centered around that subspace. Locality and dependency

is then recognized through static analysis of such a symbolic decomposition. Although every state variable is indispensable in describing global dynamics, often only a few actions in the action space are applicable in a specific system state. Similarly, only a few state variables affect or are affected by the outcome of a given action. After carefully analyzing the dependencies among regions, we can recognize a subset of state variables that can sufficiently represent the local dynamics in a given region R and capture the effects of R on iter-regional interactions; in other words, only the state variables in this subset are relevant in region R.

The original planning problem is then decomposed into a series of subproblems and each subproblem is concerned with a region instead of the entire state space. Furthermore, dimensionality reduction can be attained in a region R by aggregating states indistinguishable with respect to their values in the relevant variables in R. This is equivalent to abstracting away the irrelevant state variables in R. These decomposition techniques improve computational efficiency by exploiting locality and dependency embedded in problem instances. For each decomposition technique developed in thesis, we describe a structural parameter characterizing the locality and dependency exploitable in a problem instance and quantify the resulting computational efficiency in terms of this structural parameter [LD94] [LD95b] [DL95] [LD95a].

1.4 Outline of the Thesis

Chapter 2 investigates planning and control over discrete dynamical systems, introduces the definitions and notation used throughout this thesis, and describes AI representational schemas for compactly encoding action dynamics. Chapter 3 display the applications of planning problems such as the temporal projection problem and the Markov decision problem in different application domains. Chapter 4 provides a generic overview of the decomposition techniques depicted in Chapter 5, Chapter 6 and Chapter 7. In Chapter 5, we describe a decomposition technique for the temporal projection problem. In Chapter 6 and Chapter 7, we present two decomposition techniques for the Markov decision problem.

Chapter 8 describes the related research in planning, probabilistic reasoning in Bayesian networks, model minimization, and relational databases theory. Finally in Chapter 9, we describe future work and draw conclusions. Appendix A contains proofs for the theorems in this thesis. Appendix B provides the background knowledge about the Dantzig-Wolfe decomposition principle utilized in Chapter 7, while Appendix C describes the technical details about a decomposition technique depicted in Chapter 7.

Chapter 2

Planning and Control: Foundations

In this chapter, we describe the perspective of planning as control involving discrete dynamical systems in the form of finite automata [Paz71] [HU79] [LP81] and Markov decision processes [Paz71] [BS78] [DW91] [Put94], and introduce definitions and notation used throughout the thesis. We also present AI representational schemas for compactly representing action dynamics.

2.1 Discrete Dynamical Systems

A discrete dynamical system is a three-tuple $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Delta_{\mathcal{D}} \rangle$ where $\mathcal{S}$ is a finite state space, $\mathcal{A}$ is a finite action space, and $\Delta_{\mathcal{D}}$ is the underlying action dynamics concerning how the actions in $\mathcal{A}$ result in state transitions among the states in $\mathcal{S}$.

Factored State Spaces:

- The domain of a dynamical system $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Delta_{\mathcal{D}} \rangle$ is represented by a set of state variables $\text{Domain}(\mathcal{D}) = \{X_1, X_2, \ldots, X_m\}$. Each state variable X_i has a value at each point in time, and Ω_{X_i} denotes the finite set of the possible values for X_i. We refer to the size of the domain $\text{Domain}(\mathcal{D})$ as the *dimension* of the dynamical system $\mathcal{D}$.

- The state of the dynamical system $\mathcal{D}$ at a specific point in time is determined by the values of the state variables in Domain($\mathcal{D}$) $= \{X_1, X_2, \ldots, X_m\}$ at that time. $\mathcal{S} = \prod_{1 \leq i \leq m} \Omega_{X_i}$ is the factored state space of the dynamical system; a state $\mathbf{s}$ in $\mathcal{S}$ is represented as a vector $\langle v_1, v_2, \ldots, v_m \rangle$ where v_i is a specific value in Ω_{X_i} for the state variable X_i. The initial state of the system is determined by the initial values of the state variables.

- We use either $\mathbf{s}, \mathbf{t}, \mathbf{u}, \mathbf{v}, \mathbf{u}_0, \mathbf{u}_1, \ldots$ or simply $i, j, 0, 1, 2, \ldots$ to denote the individual states in a state space $\mathcal{S}$.

- We use v, w, or, $v_1, v_2, \ldots$ to denote the possible values for a variable in Domain($\mathcal{D}$).

Controllers:
A controller for a discrete dynamical system is a decision-making agent that regulates the system's state by taking actions. Throughout this research, we assume that there is only a single decision-making agent $\Psi_{\mathcal{D}}$ situated in a discrete dynamical system $\mathcal{D}$.

Representing Discrete Time:
We use T to denote a time variable and t to denote a specific point in time. Given a dynamical system, we are concerned with a set of discrete points in time abstractly represented as $\mathcal{T} = \{0, 1, 2, \ldots\}$, where (1) $T = 0$ is the point in time when the system is initialized to its initial state, and (2) $T = t$ is the point in time when the controller must take the tth action.

Enabled Actions and State Transitions:
Usually not every action can be taken in every state, and Enabled($\mathbf{u}$) (Enabled($\mathbf{u}$) $\subset \mathcal{A}$) denotes the set of actions that the controller $\Psi_{\mathcal{D}}$ can take in a state $\mathbf{u}$. At time t an action a in Enabled($\mathbf{u}$) is taken resulting in a change of state to $\mathbf{v}$ which may or may not be the same as $\mathbf{u}$; the system state $\mathbf{v}$ then persists until time $t + 1$ at which an action in Enabled($\mathbf{v}$) is taken.

Action Dynamics and the Markov property:
In this thesis, we focus on discrete dynamical systems whose action dynamics obey the following Markov property: when taking an action a at time t, the next state is independent of the system's state at any earlier time given the current state.

Goals and Target States:
A state $\mathbf{s}$ is an absorbing state in a discrete dynamical system if and only if as soon as the system enters $\mathbf{s}$ the system stays in $\mathbf{s}$ forever. Often we model the target states as the absorbing states of the system, implicitly specified by a boolean predicate $\mathcal{G}$ governing the values of the state variables. Target($\mathcal{G}$) denotes the set of absorbing states satisfying $\mathcal{G}$ while the goal $\mathcal{G}$ is achieved when the system ends in a target state in Target($\mathcal{G}$).

2.2 Finite Automata and Markov Decision Processes

In this thesis, we focus on three planning problems generically formulated as control problems over discrete dynamical systems such as finite automata [Paz71] [HU79] [LP81] and Markov decision processes [Paz71] [BS78] [DW91] [Put94].

Finite Automata:
A finite automaton is a discrete dynamical system $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Delta_{\mathcal{D}} \rangle$ whose action dynamics $\Delta_{\mathcal{D}}$ is simply a state transition relation, $\Delta_{\mathcal{D}} \subset \mathcal{S} \times \mathcal{A} \times \mathcal{S}$. The action space $\mathcal{A}$ forms an input alphabet, while the controller $\Psi_{\mathcal{D}}$ takes actions as inputs to the automaton. In each state $\mathbf{u}$, the controller can take any action a in Enabled($\mathbf{u}$) as an input to the automaton, and result in a state transition from state $\mathbf{u}$ to a state $\mathbf{v}$ where $(\mathbf{u}, a, \mathbf{v})$ is in $\Delta_{\mathcal{D}}$. Given an initial state $\mathbf{S}^0 = \mathbf{u}$ and a goal $\mathcal{G}$, the automaton starts from the initial state $\mathbf{u}$ and then ends in a target state in Target($\mathcal{G}$) as the final states of the automaton.

Markov Decision Processes:
A Markov decision process is a discrete dynamical system $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Delta_{\mathcal{D}} \rangle$ whose action dynamics $\Delta_{\mathcal{D}} = \langle \Pr, \text{Cost} \rangle$ is composed of a cost function Cost(.|.) and a probability function Pr(.|.) governing the state transitions over the state space $\mathcal{S}$:

- let A^t be a decision variable denoting the tth action ($t \geq 1$) taken by a controller in a dynamical system,
- let X_i^t be a random variable denoting the value of the state variable X_i immediately after taking action A^t,
- let $\mathbf{S}^t$ be a random variable denoting the system state immediately after taking action A^t,
- $\text{Cost}(\mathbf{S}^{t+1} = \mathbf{v} | A^{t+1} = a, \mathbf{S}^t = \mathbf{u}) = c_{\mathbf{uv}}(a)$ specifies the cost of taking action a in state $\mathbf{u}$ and ending up in state $\mathbf{v}$, and
- $\Pr(\mathbf{S}^{t+1} = \mathbf{v} | A^{t+1} = a, \mathbf{S}^t = \mathbf{u}) = p_{\mathbf{uv}}(a)$ specifies the conditional probability of taking action a in state $\mathbf{u}$ and ending up in state $\mathbf{v}$.

For simplicity, we denote this Markov decision process by $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \Pr, \text{Cost} \rangle$. A Markov decision process is simply a finite automaton augmented with the cost function Cost(.|.) and the probability function Pr(.|.). For a finite automaton $\langle \mathcal{S}, \mathcal{A}, \Delta_{\mathcal{D}} \rangle$, $\text{Next}(\mathbf{u}, a) = \{\mathbf{v} | \exists (\mathbf{u}, a, \mathbf{v}) \in \Delta_{\mathcal{D}}\}$ characterizes the set of possible next states when taking an action a in a given state $\mathbf{u}$. For a Markov decision process $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \Pr, \text{Cost} \rangle$, $\Pr(\mathbf{S}^{t+1} = \mathbf{v} | A^{t+1} = a, \mathbf{S}^t = \mathbf{u})$ characterizes a probability distribution over the possible next state$\mathbf{v}$, $\mathbf{v} \in \text{Next}(\mathbf{u}, a)$ when taking an action a in a given state $\mathbf{u}$.
The conditional probability Pr(.|.) fully embeds the information of a state-transition relation since a state transition from a state $\mathbf{u}$ to a state $\mathbf{v}$ immediately following an action a is possible if and only if the transition probability $p_{\mathbf{uv}}(a)$ is greater than zero.

2.3 Compact Representations

Large state spaces present a number of challenges. To begin with, the problem has to be efficiently encoded. A *factored* state-space representation uses state variables to represent different aspects of the overall state of the system. [1] Compact encodings for stochastic processes can be achieved for many applications using factored state-space representations, where the size of the model is usually logarithmic in the size of the state space [DK89]. Similarly, policies for large factored state spaces can often be efficiently encoded using decision trees that branch on state variables. [BDG95]. In the following, we describe a compact representation employed in this thesis for encoding state-transition relations, state-transition probabilities, and action costs. This is primarily for the ease of concretely explaining the algorithms and problem instances presented in this thesis. The algorithms and results in this thesis can be easily extended to other representation schemas, rather than strictly depending on this specific representation scheme.

2.3.1 Compactly Representing State-transition Relations

Instead of explicitly enumerating the state-transition relation $\Delta_{\mathcal{D}}$ of a discrete dynamical system $\mathcal{D}$, $\Delta_{\mathcal{D}}$ can be compactly represented as sets of causal rules [DB88a] [BD94] [LD94] [KHW94] [LD96]. Each action is associated with a set of causal rules, while each causal rule is basically a state-space operator that partially encodes how the associated action can result in state transitions. The following is a formal description of causal rules and their usage in representing state-transition relations.

Expressions:

An expression is represented as a set of variable/value pairs of the form $\langle X_i, v_i \rangle$ where v_i is a value in Ω_{X_i} for the state variable X_i.

[1] In artificial intelligence, such factored representations are often called *intentional* representations. Propositions representing fluents in STRIPS operators [FN71b] correspond to state variables in a factored state-space representation.

- An expression α is true in a state $\mathbf{u}$ if and only if for each variable/value pair $\langle X_i, v_i \rangle$ in α, v_i is the value of X_i in $\mathbf{u}$.
- Notationally, the variable/value pairs are enclosed by a pair of parentheses, and an empty expression () is true in every state. We usually use α and β to denote expressions.
- Two expressions α and β are mutually exclusive if α and β cannot both be true in the same state.

Causal Rules as State-space Operators:
A causal rule r is represented as $\alpha \rightarrow \beta$ where α and β are two expressions. α is the precondition and β is the postcondition of the causal rule $r = \alpha \rightarrow \beta$. A causal rule r is applicable in a state $\mathbf{u}$ if and only if the precondition of r is true in $\mathbf{u}$. A causal rule $r = \alpha \rightarrow \beta$ is a state-space operator that can map state $\mathbf{u}$ to state $\mathbf{v}$ if and only if

- r is applicable in state $\mathbf{u}$ (*i.e.,* α is true in state $\mathbf{u}$),
- β is true in state $\mathbf{v}$, and
- the values of the variables that are not involved in β remain the same in $\mathbf{v}$ as they are in $\mathbf{u}$.

Actions as Sets of State-space Operators:

- Each action in the action space $\mathcal{A}$ is associated with a set of state space operators. Taking an action a can result in a state transition from state $\mathbf{u}$ to state $\mathbf{v}$ if and only if one of the state-space operators associated with action a can map state $\mathbf{u}$ to state $\mathbf{v}$.

- An action a is enabled in a state $\mathbf{u}$ if and only if at least one of its state space operators is applicable in $\mathbf{u}$. Among the state-space operators associated with action a, several of them may be applicable in a state $\mathbf{u}$; in turn, there may be several possible next states after taking action a in state $\mathbf{u}$.

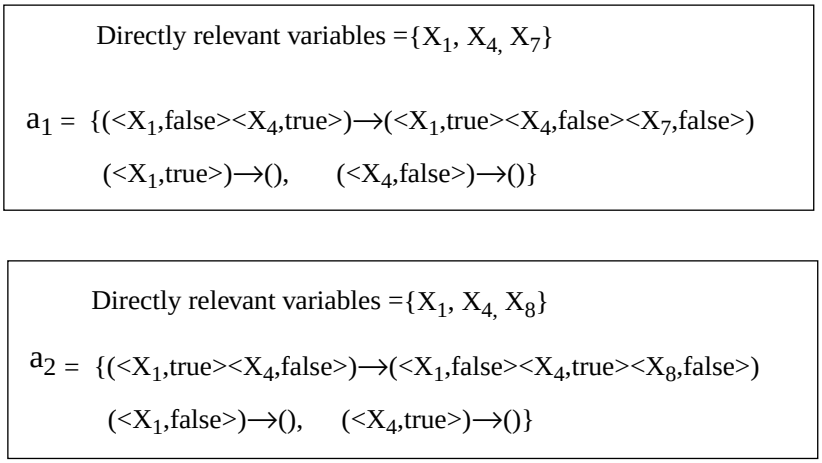

Figure 2.1: Two actions and their associated causal rules

Actions and Directly Relevant Variables:
For an action a compactly represented as a set of state-space operators, a variable X_i is directly relevant to a if and only if X_i appears in the precondition or postcondition of at least one of the state-space operator associated with a.

Figure 2.1 display two actions a_1 and a_2, their associated causal rules, and the variables directly relevant to the actions.

Encoding State-transition Relations:
Actions represented as sets of state-space operators compactly encode a state-transition relation $\Delta_{\mathcal{D}}$: For each action a, $(\mathbf{u}, a, \mathbf{v})$ is in the state transition relation $\Delta_{\mathcal{D}}$ if and only if one of the causal rules associated with a can map state $\mathbf{u}$ to state $\mathbf{v}$.

2.3.2 Compactly Representing Transition Probabilities and Costs

In the following, we augment state-space operators with the information of transition probability and action cost. We then illustrate how to compactly encode the information of transition probabilities and action costs using these probabilistic

state-space operators.

Probabilistic State-space Operators:

A probabilistic causal rule $r = \langle \Pr(r), \text{Cost}(r), \alpha \rightarrow \beta \rangle$ is a causal rule $r = \alpha \rightarrow \beta$ augmented with the information of an action cost $\text{Cost}(r)$ and a transition probability $\Pr(r)$. A probabilistic causal rule $r = \langle \Pr(r), \text{Cost}(r), \alpha \rightarrow \beta \rangle$ is a probabilistic state-space operator that can map state $\mathbf{u}$ to state $\mathbf{v}$ if and only if

- r is applicable in state $\mathbf{u}$ (*i.e.,* α is true in state $\mathbf{u}$),
- β is true in state $\mathbf{v}$, and
- the values of the variables that are not involved in β remain the same in $\mathbf{v}$ as they are in $\mathbf{u}$.

Encoding Transition Probability and Action Cost:

The information of a conditional probability $\Pr(.|.)$ about state transition and a function $\text{Cost}(.|.)$ for action cost can be compactly represented using probabilistic causal rules in the following way:

1. Each action a is associated with a set of probabilistic causal rules, where the preconditions (postconditions) of any two probabilistic causal rules are either identical or mutually exclusive.
2. For each state $\mathbf{u}$ and each action a, if there exists a probabilistic causal rule $r = \langle \Pr(r), \text{Cost}(r), \alpha \rightarrow \beta \rangle$ associated with a that maps state $\mathbf{u}$ to state $\mathbf{v}$, we have $\Pr(\mathbf{S}^{t+1} = \mathbf{v} | A^{t+1} = a, \mathbf{S}^t = \mathbf{u}) = p_{\mathbf{uv}}(a) = \Pr(r)$ and $\text{Cost}(\mathbf{S}^{t+1} = \mathbf{v} | A^{t+1} = a, \mathbf{S}^t = \mathbf{u}) = c_{\mathbf{uv}}(a) = \text{Cost}(r)$; otherwise, we have $\Pr(\mathbf{S}^{t+1} = \mathbf{v} | A^{t+1} = a, \mathbf{S}^t = \mathbf{u}) = p_{\mathbf{uv}}(a) = 0$.
3. The transition probabilities of the probabilistic causal rules that are associated with the same action and share the same precondition sum to one. This ensures that the transition probability is well defined over all states, and for every state $\mathbf{u}$ and every action a enabled in $\mathbf{u}$ we have $\Sigma_{\mathbf{v}} p_{\mathbf{uv}}(a) = 1$).

2.4 Planning and Control over Discrete Dynamical Systems

Plans and Control:
A controller takes actions according to a plan. There are different sorts of plans, which are categorized according to how they determine actions to take.

- A plan π is *conditional* if π employs the information of the current system state in determining an action to take; otherwise π is *unconditional*.
- A plan π is *time-variant* if π employs the information of the current time in determining an action to take; otherwise π is *stationary*.
- A plan π is *deterministic* if π determines a unique action given the information of current time and current system state; otherwise π is *nondeterministic*. In particular, we say that π is stochastic if π chooses one of the actions enabled in the current system state according to a probability distribution over actions.
- Stationary conditional plans are of special interest to us, and we refer to a stationary conditional plan as a *policy*. A deterministic (nondeterministic) policy maps a state to an action deterministically (nondeterministically); a stochastic policy maps a state to one of the actions enabled in that state according to a probability distribution over actions.
- A *linear plan* (*i.e.,* a finite action sequence) in classical planning is a deterministic time-variant unconditional plan in our classification, where the information of current time fully determines the action to take.
- Suppose that for each action instance we introduce a new state variable to keep track of whether the action instance has been taken. Then a *partially ordered plan* (*i.e.,* a multiset of actions together with a partial order on them) in classical planning can be interpreted as a nondeterministic stationary conditional plan (*i.e.,* a nondeterministic policy) in our classification.

This is because a partially ordered plan allows a controller to nondeterministically take any action instances which are enabled in the current system state and whose predecessors have all been taken.

Plan Executions and Trajectories:

We say that the controller $\Psi_{\mathcal{D}}$ executes a plan π when the controller takes actions according to π. Given an initial state $\mathbf{S}^0$, when the controller takes a sequence of actions $\langle A^1 = a_1, A^2 = a_2, \ldots \rangle$ according to the plan π, the dynamical system $\mathcal{D}$ evolves through a sequence of states $\langle \mathbf{S}^0 = \mathbf{u}_0, \mathbf{S}^1 = \mathbf{u}_1, \ldots \rangle$. We refer to $\langle \mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \ldots \rangle$ as a trajectory of the dynamical system, and refer to $\langle a_1, a_2, \ldots \rangle$ as an action trajectory of the dynamical system. Different executions of the same conditional plan π in a dynamical system can result in very different trajectories even if starting from the same initial state.

Performance Criteria:

A performance criterion indicates a measure of the quality of plans. In this thesis, we focus on minimizing the expected cumulative cost during plan execution, and/or maximizing the expected final reward associated with the system's target states at the end of plan execution.

- $\mathrm{E}_{\mathbf{s}}^{\pi}[\sum_{0 \leq t < \tau_G} \gamma^t \mathrm{Cost}(\mathbf{S}^{t+1} | A^{t+1} = \pi(\mathbf{S}^t), \mathbf{S}^t)]$, is the expected cumulative cost until we first reach a target state where
  - E_s^{π} is the expectation operator given that the initial state is $\mathbf{s}$ ($\mathbf{S}^0 = \mathbf{s}$),
  - τ_G is the indefinite time that we first reach a state satisfying a specific goal G, and
  - $0 < \gamma \leq 1$ is a discount factor.

  When $\gamma = 1$ ($0 < \gamma < 1$), it is an expected undiscounted (discounted) cumulative cost criterion.

- Reward($\mathbf{s}$) denotes the final reward associated with a target state $\mathbf{s}$ in $\mathcal{S}$.

  - To generate a plan to achieve a specific goal, we can adopt a reward function that uniformly assigns to each target state a large positive reward and assigns to the other states with zero reward.
  - If the state variables contribute independently to the final rewards associated with the target states, the reward function is additive and can be represented as a summation of the separate reward functions associated with the state variables, *i.e.,* $\text{Reward}(\mathbf{s} = \langle v_1, v_2, \ldots, v_n \rangle) = \Sigma_{1 \leq i \leq n} \text{Reward}_i(X_i = v_i)$.

- Note that final rewards associated with target states can be considered as extra negative cost involved when actions end in target states. Therefore, we can incorporate the final rewards associated with target states into action costs, and then form new cost functions of actions. In other words, we can consider the expected cumulative cost as the most basic performance criterion, and then model and optimize as its derived variants the performance criteria of maximizing the expected final reward, or minimizing the expected cumulative cost minus the expected final reward.

Planning Problems as Control Problems:

In this thesis, we formulate planning problems as control problems over discrete dynamical systems as follows:

- Each planning instance is composed of a discrete dynamical system $\mathcal{D}$, and a performance criterion $\mathcal{F}$. The controller $\Psi_{\mathcal{D}}$ takes actions according to a plan. The task of planning is to find a plan π for the controller $\Psi_{\mathcal{D}}$ such that π achieves the goal $\mathcal{G}$ optimally with respect to the performance criterion $\mathcal{F}$.

- For classical planning, the performance criterion is to maximize the final reward, given that each target state is uniformly assigned with a large positive reward while the other states are assigned with a zero reward. The task of classical planning is to derive a linear plan (*i.e.,* a deterministic

time-variant unconditional plan) or a partially ordered plan (*i.e.,* a nondeterministic stationary conditional plan) that returns an optimal reward at the end of plan execution (*i.e.,* ending in one of the target states).

The Markov Decision Problem:
The Markov decision problem is a control problem [Put94] [Der70] over Markov decision processes, and the task is to generate an optimal policy (*i.e.,* a stationary conditional plan) given a Markov decision process and a performance criterion.

2.5 Algorithms for Optimal Control over Markov Decision Processes

Markov decision processes have been well studied by the operations research community [Bel61] [How60] [Ber87] [Put94]. In this thesis, we extensively utilize the computational machinery developed by the operations research community in computing optimal policies for Markov decision processes.

Dynamic Programming:
The value iteration method [Bel61] and the policy iteration method [How60] are two well-known methods used to generate optimal policies for Markov decision processes, both employing the dynamic-programming principle. The value iteration method starts with an arbitrary value function that estimates for each state the expected cumulative costs of plan execution starting from the state. The value iteration method then repeatedly searches for a better value function until it ends in an optimal value function, which encodes the expected cumulative costs for individual states when adopting an optimal policy. An optimal policy can then be determined by examining the optimal value function.

The policy iteration method uses an arbitrary initial policy as an initial

seed, and then alternates between a value-determination phase and a policy-improvement phase until an optimal policy is determined at the end. The expected cumulative costs of the states with respect to the current policy are determined in a value-determination phase, this information is immediately employed in the following policy-improvement phase to derive an improved policy.

On each iteration, the value iteration method takes time quadratic in the size of the state space, while the policy iteration method takes time cubic in the size of the state space. Assuming finite precision and a constant discount rate γ strictly less than one, both methods converge to an optimal solution in a finite number of iterations, and are polynomial-time algorithms in terms of the sizes of the state spaces [Put94] [PT87a] [LDK95].

Linear Programming:

The instances of Markov decision problems can also be cast as linear programs [Der70] [D'E63] [KK71a] [Put94], and then solved by standard techniques for linear programming [MB90] [Las70] [DW60] [Chv80] like the well known simplex method. Although this linear programming approach is not as efficient as the dynamic programming approach [KK71a], it provides valuable insight [KC74a] into the relationship between the computational efficiency and structure in problem instances.

Chapter 3

Planning and Control: Applications and Computational Challenge

In this chapter, we describe the applications of planning problems generically formulated as control problems over discrete dynamical systems in the form of finite automata or Markov decision processes. Each planning instance compactly encodes a discrete dynamical system $\mathcal{D}$ and a performance criterion $\mathcal{F}$. The task of planning is to determine an optimal plan π of a specified category, which provides optimal utility with respect to the criterion $\mathcal{F}$ when the controller takes actions according to π. The exponential growth of the underlying state spaces composes a major computational challenge in solving these planning problems. At the end of this chapter, we depict the basic principles adopted in this thesis to cope with very large state spaces.

3.1 Multi-Level Task Networks and Temporal Projection

A multi-level task network specifies a hierarchy of tasks together with the temporal constraints regarding the order of performing these tasks. Given a multi-level

task network, there are many possible linear plans for performing the tasks in the network. To analyze such a task network, we would like to (1) determine the possibility of running into undesirable states, and (2) if possible report a linear plan that ends in an undesirable state as an evidence. If we consider undesirable states as target states and uniformly associate every undesirable states with a large positive reward, then the task of *temporal projection* [DB88b] [NB94] [LD94] [LD96] is to determine an optimal linear plan that returns a maximum final reward for a given multi-level task network.

3.1.1 Multi-Level Task Networks

A multi-level task network specifies a hierarchy of tasks together with the temporal constraints regarding the order of performing these tasks.

Primitive Tasks and Compound Tasks:
Tasks are classified as primitive tasks or compound tasks. A primitive task is an action that is represented as a set of state-space operators (*i.e.,* causal rules). A compound task is composed of a set of tasks, each of which can be either a primitive task or a compound task.

Multi-Level Task Networks:
A multi-level task network organizes a set of tasks $\{tk_1, tk_2, \ldots\}$ as a tree, and these tasks are performed through a depth-first exploration of the tree:

- Each leaf node represents a primitive task, and each internal node x represents a compound task whose component tasks are the child nodes of x. The component tasks of a compound task are siblings to one another, and the compound task is the parent of the component tasks.

- When a primitive task is performed, the associated action is taken to change the values of some state variables. When a compound task tk is performed, the component tasks of tk are performed in any order consistent with a given partial order over these component tasks. The component tasks of

the compound task tk are performed as an atomic group; the tasks associated with the siblings of tk are performed either before or after this atomic group is completed. Therefore the tasks are performed through a depth-first traversal through the multi-level task network, although there are potentially many possible ways of conducting such a traversal.

- Each task is performed exactly once; a primitive task is fully completed after the associated action is taken, and a compound task is fully completed when all of its component tasks are fully completed. A compound task is partially completed when some but not all of its component tasks are fully completed.

Linear Plans for Multi-Level Task Networks:
A linear plan for a multi-level task network is a total order on the primitive tasks satisfying all the ordering constraints specified by the network. An important reasoning task involving a multi-level task network is to search for linear plans of certain useful properties, such as linear plans that avoid undesirable states or end in a target state. This is one of the planning problems to be addressed in this thesis.

3.1.2 Multi-Level Task Networks as Finite Automata

A multi-level task network determines a finite automata $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Delta_{\mathcal{D}} \rangle$:

The State Space $\mathcal{S}$:
The state space $\mathcal{S}$ is determined by the state variables in
$\{X_1, X_2, \ldots\} \bigcup \{TK_1, TK_2, \ldots\}$ where

- X_i is a variable directly relevant to at least one of the primitive tasks (*i.e.,* actions), and

- TK_i is an auxiliary variable whose value can be either *not-performed-yet*, *fully-completed*, or *partially-completed* indicating respectively that the task tk_i is not yet performed, fully completed, or partially completed.

A state $\mathbf{u}$ in $\mathcal{S}$ is a vector $\langle x_1, x_2, \ldots, y_1, y_2, \ldots \rangle$ where x_i is the value of variable X_i and y_i is the value of variable TK_i in state $\mathbf{u}$. In the initial state, each variable TK_i has the value *not-performed-yet* since none of the tasks have been performed yet.

The Action Space $\mathcal{A}$:
The action space $\mathcal{A}$ is the union of $\{\hat{a}_1, \hat{a}_2, \ldots\}$ and $\{\bar{a}_1, \bar{a}_2, \ldots\}$ where

- for a compound task tk_i, action $\hat{a}_i$ signals the beginning of task tk_i and action $\bar{a}_i$ signals the completion of task tk_i, while
- for a primitive task tk_i, action $\hat{a}_i$ means applying one of the state-space operator associated with task tk_i, and action $\bar{a}_i$ signals the completion of task tk_i.

The Action Dynamics $\Delta_{\mathcal{D}}$:
The action dynamics $\Delta_{\mathcal{D}}$ is a state-transition relation characterizing different ways of performing the tasks in a task network. $(\mathbf{u}, \hat{a}_i, \mathbf{v})$ is in $\Delta_{\mathcal{D}}$ if and only if state $\mathbf{u} = \langle x_1, x_2, \ldots, y_1, y_2, \ldots \rangle$ satisfies the following requirements (*i.e.,* $\hat{a}_i$ is enabled in $\mathbf{u}$ to start task tk_i):

- if tk_j is the parent task of tk_i, tk_j must have been partially completed; *i.e.,* the state variable TK_j must have the value *partially-completed* ($y_j =$ *partially-completed*) in state $\mathbf{u}$;
- if tk_i has sibling tasks that must precede tk_i, then every such sibling task tk_j must have been fully completed; *i.e.,* the state variable TK_j must have the value *fully-completed* ($y_j =$ *fully-completed*) in state $\mathbf{u}$;
- the task tk_i is not yet performed; *i.e.,* the state variable TK_i must have the value *not-performed-yet* ($y_i =$ *not-performed-yet*) in state $\mathbf{u}$;
- if tk_i is a primitive task, then at least one of the state-space operators associated with tk_i must be applicable in state $\mathbf{u}$;

and state $\mathbf{v} = \langle x'_1, x'_2, \ldots, y'_1, y'_2, \ldots \rangle$ satisfies the following requirements (*i.e.,* $\mathbf{v}$ is a possible resulting state after starting task tk_i in state $\mathbf{u}$):

- the value of the variable TK_i has value *partially-completed* (y'_i = *partially-completed*), signaling that the task tk_i is partially completed now;
- if task tk_i is a primitive task, the values of the variables in $\{X_1, X_2, \ldots\}$ can be changed from $x_1, x_2, \ldots$ in $\mathbf{u}$ to $x'_1, x'_2, \ldots$ in $\mathbf{v}$ by applying a state-space operator associated with tk_i to state $\mathbf{u}$.

$(\mathbf{u}, \bar{a}_i, \mathbf{v})$ is in $\Delta_{\mathcal{D}}$ if and only if state $\mathbf{u} = \langle x_1, x_2, \ldots, y_1, y_2, \ldots \rangle$ satisfies the following requirements (*i.e.,* $\bar{a}_i$ is enabled in state $\mathbf{u}$ to finish task tk_i):

- The task tk_i is partially completed in state $\mathbf{u}$; *i.e.,* the state variable TK_i must have the value *partially-completed* (y_i = *partially-completed*) in state $\mathbf{u}$.
- If tk_i is a compound task, then every component task tk_j of tk_i must have been fully completed in state $\mathbf{u}$; *i.e.,* the state variable TK_j must have the value *fully-completed* (y_i = *fully-completed*) in state $\mathbf{u}$.

and state $\mathbf{v} = \langle x'_1, x'_2, \ldots, y'_1, y'_2, \ldots \rangle$ satisfies the following requirement (*i.e.,* $\mathbf{v}$ is the resulting state after finishing task tk_i in state $\mathbf{u}$):

- The value of the variable TK_i has value *fully-completed* (y'_i = *fully-completed*), signaling that the task tk_i is fully completed now.

3.1.3 Modeling a Process-Scheduling Domain

In Figure 3.1-a, six primitive tasks, $tk_1, tk_2, tk_3, tk_4, tk_5$ and tk_6, are going to be performed in a job shop. Three groups of primitive tasks $\{tk_1, tk_2\}$, $\{tk_3, tk_4\}$ and $\{tk_5, tk_6\}$ form three compound tasks tk_7, tk_8 and tk_9 respectively. The compound tasks $\{tk_7, tk_8, tk_9\}$ then compose another compound task tk_{10}. The environment of the job shop is modeled by eight binary variables $\{X_1, X_2, \ldots, X_8\}$, each of which can be either true or false at a point in time. For example, one state variable indicates whether the temperature is over a critical level or not, and

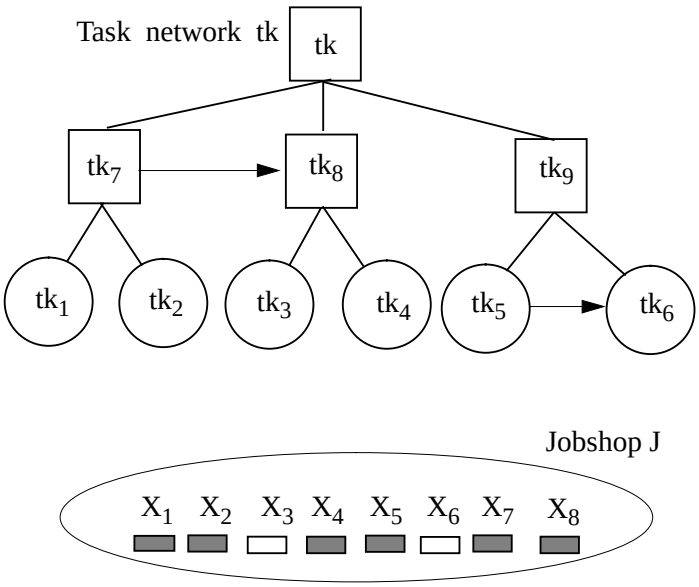

Figure 3.1: Process scheduling for a job shop

another indicates whether the concentration of a chemical in the air is over a threshold level or not.

Figure 3.1-c displays the constraints regarding the order of performing these tasks. The component tasks of a compound task must be perform as an atomic group, since they are closely related to one another. Compound task tk_7 can only be performed sometime after compound task tk_8 is performed. In compound task tk_9, primitive task tk_6 is always performed before primitive task tk_5.

While the primitive tasks are performed, the environment of the job shop is also changed accordingly. The effects of performing a primitive task depends on the conditions immediately before the primitive task is performed. For example, performing a task can increase the density of a chemical in the air over a threshold level if the temperature immediately prior to performing the task is over a critical level. On the other hand, a primitive task can be performed by setting up the job shop in multiple ways, each of which may have very different preconditions and postconditions. Figure 3.2-b describes the preconditions and postconditions of these primitive tasks using state-space operators.

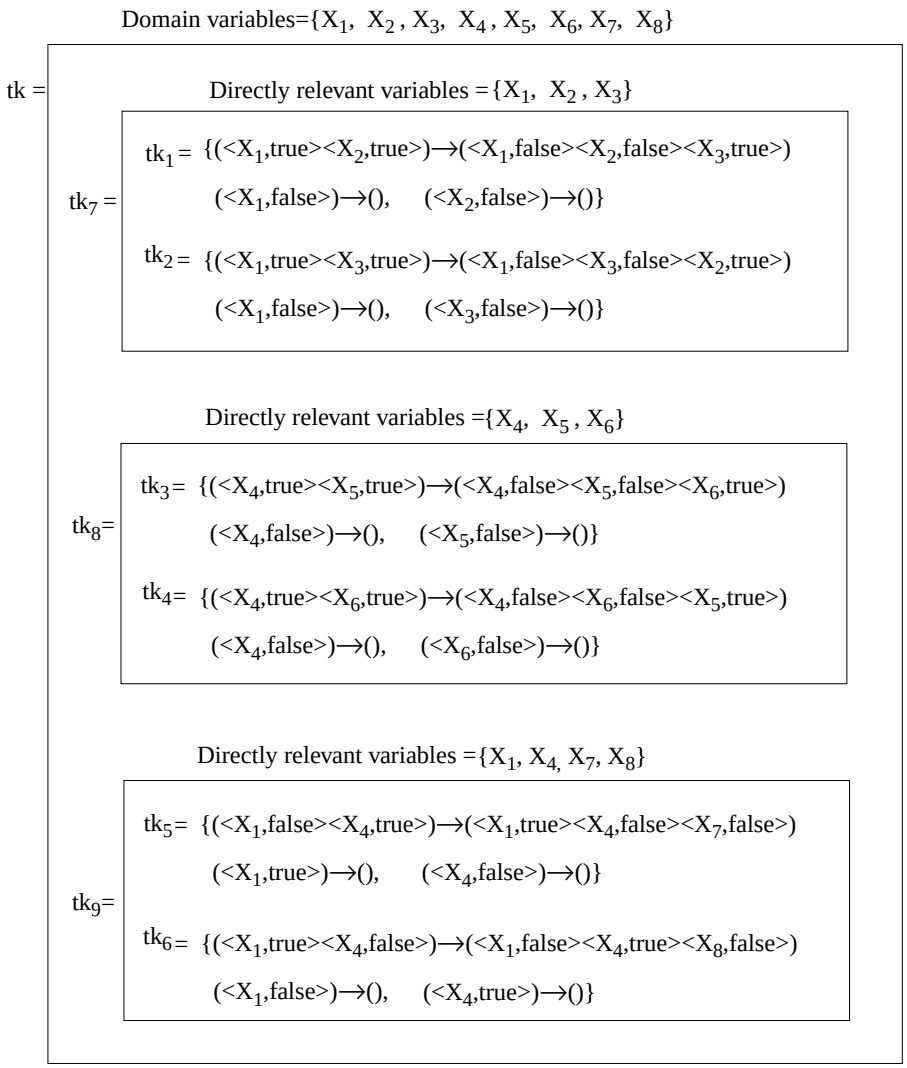

Figure 3.2: Preconditions and postconditions of the primitive tasks

3.1.4 Application in Schedule Verification

The following scenarios describe various temporal projection tasks concerning the consequences of performing the tasks in the task network depicted in Figure 3.1.

Scenario 1: If X_1, X_2, X_4, X_5, X_7 and X_8 are all true after performing the task network, the job shop needs to be cleaned up at extra cost before the job shop can continue its operation.

Scenario 2: If X_1, X_2, X_4, X_5, X_7 and X_8 are all true at the same time, a machine in the job shop will be inflicted with a permanent damage.

Scenario 3: If X_1, X_2, X_4, X_5, X_7 and X_8 are all true immediately after performing primitive task tk_5, it is very likely that the job shop will catch fire.

For these scenarios, the controller of the job shop needs to determine whether the variables X_1, X_2, X_4, X_5, X_7 and X_8 can all be true at a specific point in time. In scenario 1 , it is the point in time immediately after the task network is finished. In scenario 2, it is any point in time while performing the task network in scenario 2. In scenario 3, it is the point in time immediately after performing the primitive task tk_5. The controller also needs to generate such a linear plan as evidence to explain how the linear plan together with certain choices of the associated state-space operators of the primitive tasks can end in the desired or undesirable situation.

3.1.5 Computational Complexity

Classical Planning involving propositional STRIPS-like operators is PSPACE-hard [Byl94] [Byl96]. The Temporal Projection Problem investigated in this thesis is shown to be NP-Complete [DB88a] [NB94] [LD94] [LD96].

The temporal projection problem can be viewed as a special kind of planning problem where only linear plans of a fixed length (*i.e.,* the number of primitive

tasks in the given task network) are considered.

This makes the problem fall in the class NP, since a decision maker can non-deterministically determine a linear plan as a solution in polynomial time.

The temporal projection problem is NP-Complete even in very restricted cases [LD94] [LD96].

3.2 Sequential Decision Networks and Optimal Decision Making

3.2.1 Sequential Decision Networks

A decision stage for a decision maker contains a restricted set of possible actions together with the information about transition probabilities and action costs. Often complex decision processes can be broken down into elementary decision stages [HM84] [Sha86] [TS90] [SW95] [LD95b]. These decision stages can be combined to form a sequential decision network using a simple programming language specifying conditionals, loops, and sequences involving the decision tasks as primitive statements. A sequential decision network is composed of many decision-making stages interconnected as a directed graph using conditional branches and loops [SW95] [LD95b]. Sequential decision networks compactly represent Markov decision processes with rich structures in the underlying state spaces and action dynamics. For optimal decision making in a sequential decision network, the task of the decision maker is to determine a policy that optimizes a given performance criterion such as the expected cumulative cost in executing the policy.

Decision Stages:
A decision stage $d = (\alpha_d, \mathcal{A}_d, \mathcal{X}_d)$ is composed of (1) a guard expression α_d, (2) a set of actions $\mathcal{A}_d$, and (3) a set of the state variables $\mathcal{X}_d$. α_d is a boolean expression concerning the values of a subset of the variables in V, which must be true immediately prior to entering decision stage d. All activities within decision stage d is restricted to the actions in $\mathcal{A}_d$. Each action in $\mathcal{A}_d$ is represented as a set

of probabilistic state-space operators. $\mathcal{X}_d$ is the set of state variables *appearing* in decision stage d, which is the union of the state variables appearing in α_d and the state variables directly relevant to the actions in $\mathcal{A}_d$.

Control Nodes:
A control node $c = (\alpha_c)$ contains a guard expression α_c, which is a boolean expression concerning the values of some variables. A control node c is a filter in controlling the process of decision making, where the guard expression α_c must be true before the decision maker can bypass c to enter the next decision-making stage.

Sequential Decision Networks:
A sequential decision network H is represented as a directed graph $H = (N_H, E_H)$ where (1) each node d in N_H is either a decision stage or a control node, and (2) a directed edge (d, d') in graph E_H indicates that after taking an action enabled in a decision stage d (or after bypassing a control node d), the decision maker can enter a decision stage d' (or bypass a control node d') if and only if the guard expression $\alpha_{d'}$ is true.

Relationship among Decision Stages:
A decision stage d' is a *child* of a decision stage d (or d is a *parent* of d') if and only if d can reach d' bypassing zero or more control nodes in E_H. We then naturally extend this definition to define the *ancestors* and *descendants* of decision stages in an obvious way.

Looping and Conditional Branching in Sequential Decision Networks:
A decision maker can enter a decision stage d (or bypass a control node c) if and only if its guard expression α_d (α_c) is true immediately prior to entering d (bypassing c). After entering a decision stage d, the decision maker repeatedly takes actions in a decision stage d until he can successfully bypass zero or more control nodes and enter one of the child decision stages of d. The control nodes between

a decision stage d and d's child decision stages implicitly forms a decision tree for determining the next decision stage to enter.

The Starting Node and the Ending Node:
A starting node is a control node and the unique source in a sequential decision network, which according to the initial values of the variables determines the very first decision stage to enter. Similarly, an ending node is a control node and the unique sink in a sequential decision network, which signals the end of decision making in a sequential decision network. We assume that there is a unique source (sink) since we can always construct one by adding a dummy control node that leads to (follows) all the current sources (sinks).

Policies for Decision Making:
A policy for a sequential decision network maps a decision stage into an action enabled in that stage according to the values of the variables at the time of decision making.

Executing a Policy: For a sequential decision network, the execution of a policy π starts at the starting node, and then branches through and loops around decision stages until reaching the ending node; at each decision stage, the decision maker takes actions according to policy π [1]. Repetitive executions of the same policy for a sequential decision network may result in very different trajectories through the decision stages due to the uncertainty in the action outcomes.

Cost, Reward, and Optimal Decision Making:
Each time the decision maker takes an action, the action involves a cost. When entering the ending node, the decision maker gets a final reward $\sum_{1 \leq i \leq n} f_i(X_i)$, where f_i is a reward function that maps the values of variable X_i into a reward. For optimal decision making in a sequential decision network, the task of the decision maker is to determine an optimal policy that minimizes the expected

[1] We focus on sequential decision networks that with probability one, will end in the ending node in the long run.

cumulative action cost *minus* the expected final reward when executing the policy.

3.2.2 Sequential Decision Networks as Markov Decision Processes

A sequential decision network $H = (N_H, E_H)$ determines a Markov decision process $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \Pr, \text{Cost} \rangle$:

The State Space $\mathcal{S}$:
The state space $\mathcal{S}$ is determined by the state variables in $\{X_1, X_2, \ldots\} \bigcup \{D\}$ where

- X_i is a variable directly relevant to some of actions enabled in the decision stages, and
- D is an auxiliary variable whose value indicates the decision stage currently encountered.

A state $\mathbf{u}$ in $\mathcal{S}$ is a vector $\langle x_1, x_2, \ldots, d \rangle$ where x_i is the value of variable X_i and d is the value of variable D in state $\mathbf{u}$. In the initial state, variable D indicates the starting decision stage.

The Action Space $\mathcal{A}$:
The action space $\mathcal{A} = \bigcup_d \mathcal{A}_d$ is simply the union of the set of actions $\mathcal{A}_d$ enabled in d over every decision stage d.

Action Dynamics (Pr and Cost):
The transition probability Pr and action cost Cost of the Markov decision process $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \Pr, \text{Cost} \rangle$ capture the dynamics of decision making in the sequential decision network. For each action a, $p_{\mathbf{uv}}(a)$ equals $\sum_\gamma \Pr(\gamma)$ and $c_{\mathbf{uv}}(a)$ equals $\sum_\gamma \Pr(\gamma)\text{Cost}(\gamma)$ where a probabilistic state-space operator γ associated with a, if and only if state $\mathbf{u} = \langle x_1, x_2, \ldots, d \rangle$ satisfies the following requirements (*i.e.,* a is an action enabled in state $\mathbf{u}$):

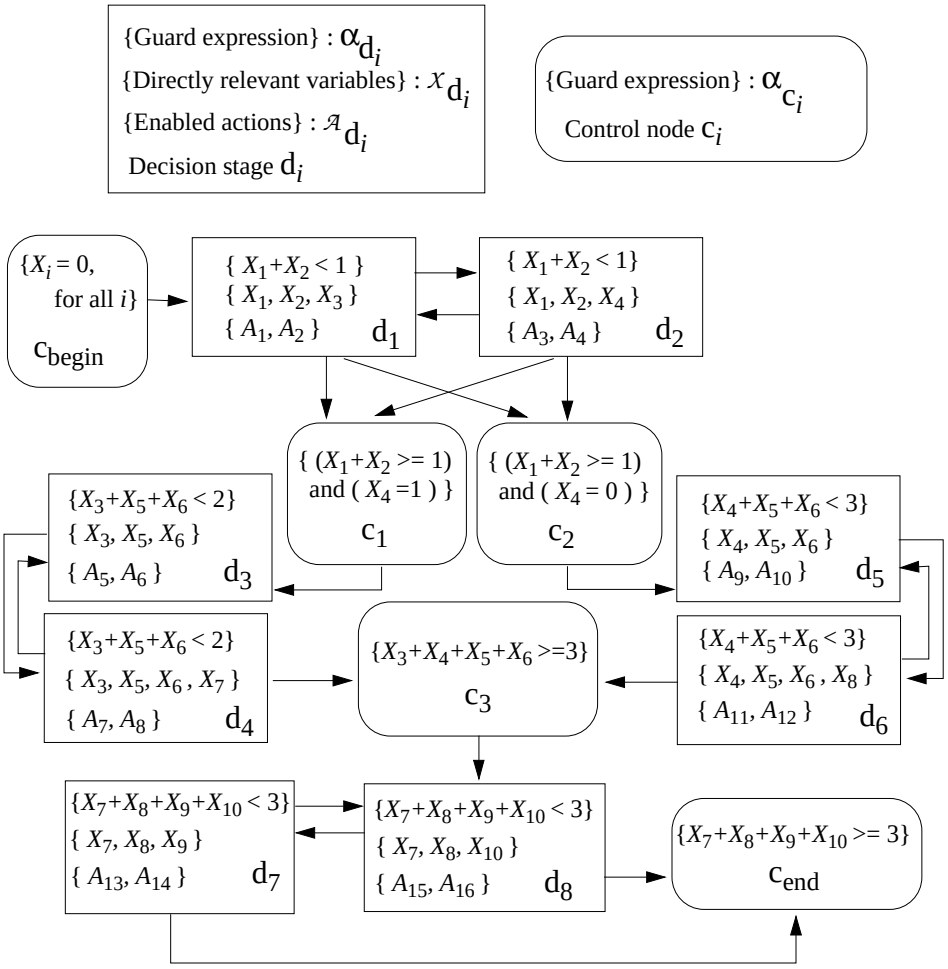

Figure 3.3: A sequential decision network to improve product quality

- $\mathbf{u} = \langle x_1, x_2, \ldots, d \rangle$ is a state, a is an action in the restricted action set $\mathcal{A}_d$ of decision stage d, and γ is a probabilistic state-space operator associated with action a and applicable in state $\mathbf{u}$;

- $\mathbf{v} = \langle x'_1, x'_2, \ldots, d' \rangle$ is a state, and by applying state-space operator γ to state $\mathbf{u}$, the values of the variables in $\{X_1, X_2, \ldots\}$ are changed from $x_1, x_2, \ldots$ in $\mathbf{u}$ to $x'_1, x'_2, \ldots$ in $\mathbf{v}$, ending in decision stage d'.

3.2.3 Modeling a Production-Control Domain

Figure 3.3 shows a sequential decision network to improve the overall quality of a product with ten components $X_1, \ldots, X_{10}$. The quality of each X_i is quantified as a value that is either zero or one. Initially the values of all of the components are rated as zero. At the end of the improvement, we receive a reward proportional to the number of the components X_i's whose values are rated as one. In addition, it is also required that the values of at least one of $\{X_1, X_2\}$, X_4 and two of $\{X_3, X_5, X_6\}$, and three of $\{X_7, X_8, X_9, X_{10}\}$ must be rated as one at the end of the improvement.

Figure 3.3 displays thirteen decision stages and control nodes interconnected as a directed graph, where (1) c_{start} and c_{end} are the starting node and the ending node respectively, (2) c_{start}, c_{end}, c_1, c_2, and c_3 are all control nodes to direct the execution of the plan, and (3) $d_1, d_2, \ldots, d_8$ are the decision stages that allow us to change the quality of the product by taking actions. Each of the decision stages in $\{d_1, d_2, \ldots, d_8\}$ is associated with two enabled actions. Taking an action enabled in a decision stage incurs a cost and changes the values of the components X_i's stochastically. Rather than providing the details of the transition probabilities and the costs associated with the actions $a_1, a_2, \ldots, a_{16}$, we only depict the variables relevant to the actions enabled in the decision stages in Figure 3.3.

3.2.4 Application in Factory Production Control

Taking an action enabled in a decision stage introduces a cost and changes the values of the components X_i's stochastically. The expected cumulative cost and the expected final reward are greatly affected by how we take actions in individual decision stages. For example, in decision stage d_1, action a_1 may improve the values of at least one of X_1 and X_2 to one with cost \$15 and probability 0.75 all the time, while action a_2 may improve the values of both X_1 and X_2 to one with cost \$20 and probability 0.99 when the value of X_4 is one immediately prior to taking action a_2. Instead of taking action a_1 in step d_1 all the time, it may be worthwhile to take action a_2 if the value X_4 is one. A policy of such a plan is a mapping that for each decision stage d_i in $\{d_1, d_2, \ldots, d_8\}$ determines the action to take according to the values of the components immediately prior to

taking an action enabled in d_i. Our task is to derive an policy that optimizes the the expected cumulative cost of actions during the production process minus the expected reward determined by the quality of the final product.

3.2.5 Application in Stochastic Transportation Planning

An agent plans to travel to a destination as efficiently as possible. At each stop, the agent can take actions (*e.g.,* using different combination of routes and vehicles) to reach adjacent locations. The cost and the probability of reaching a specific location depend on the current location, the conditions of the local environment (*e.g.,* the conditions of weather, traffic, road, ...), and the action taken. A policy maps the current location and the conditions of the environment into an action to take. For stochastic transportation planning, the agent would like to determine an optimal policy that minimizes the expected cumulative cost (*e.g.,* time and fuel) to reach the destination.

Such a stochastic transportation problem [Lat90a] [Jak94] [GD95] [CKL94] [MA95] [DKKN95], is a Markov decision problem:

- The state variables that model the physical locations and the conditions of the local environment determine the state space $\mathcal{S}$.
- The action space $\mathcal{A}$ is the union of the actions available at different locations.
- $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \Pr, \text{Cost} \rangle$ forms a Markov decision process, where Cost and Pr specify the cost and transition probability for state transition among $\mathcal{S}$ after taking actions in $\mathcal{A}$.
- The performance criterion $\mathcal{F}$ is the expected cumulative cost to reach the destination. To solve the Markov decision problem is to determine an optimal policy with respect to the performance criterion $\mathcal{F}$.

Figure 3.4 displays a building of three floors. Two elevators in the building transport people from one floor to another. Each floor is tessellated into one

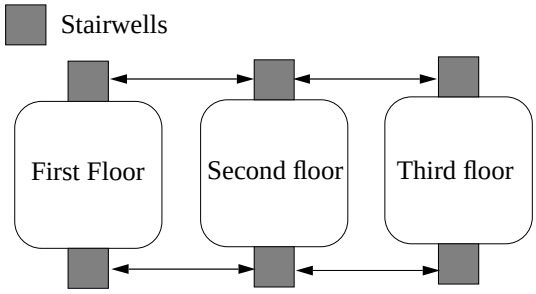

Figure 3.4: A building of three floors and two elevators

thousand squares with two of them corresponding to the positions for the two elevators. A robot moves around this building by taking actions to reach adjacent squares in the same floor or to go to a different floor using one of the elevators. All these actions take time and involve uncertainty in their outcomes:

Moving in different directions: As long as there is no obstacle blocking its way, the robot can move in four directions, namely east, west, north and south. To move in a specific direction, the robot first orients itself toward that direction, and then move one square forward. The expected amount of time spending in such an action depends on the orientation of the robot immediately prior to taking the action. When the robot take an action to move forward, it ends in the adjacent square in front of it most of the time with some chances that it may end in the square to the right or the left of the target square. The probability distribution over the possible outcomes depends on the local environment of the robot immediately prior to taking the action.

Entering an elevator: When standing in a square adjacent to an elevator, the robot can take an action to (1) turn itself toward the entrance to the elevator, (2) push the button to call for the elevator, (3) wait until the elevator comes up, and (4) enter the elevator and turn around. Both the expected amount of time and the probability of ending up on a specific floor depend on what the current floor is.

Heading toward another floor: When the robot enters an elevator, it can

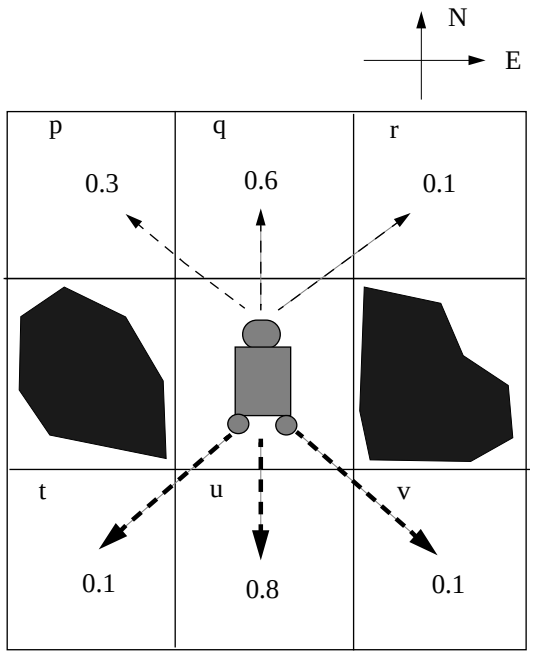

Figure 3.5: Uncertainty in action outcomes

take an action to (1) push a button to close the door, (2) choose a target floor, and (3) steps out of the elevator when the door is open again. The robot may end up on a floor different from the target floor it has in mind, since the elevator may be stopped before actually reaching the target floor. Both the expected amount of time and the probability of ending up on a specific floor depend on what the current floor is.

In Figure 3.5, the robot is in location l and is oriented toward the north. Both the east side and the west ride of l are blocked by obstacles. If the robot decides to go north (south), it takes five (ten) seconds on average to reach one of the locations p, q and r (t, u and v) with probability 0.3, 0.6 and 0.1 (0.1, 0.8 and 0.1) respectively. This is because the robot needs to spend five extra seconds to orient itself toward the south before it can actually move to the south.

At the first floor, there is a site to recharge the battery of the robot. As soon as the voltage of the battery is below certain level, the robot enters a panic mode to head for the site for recharge. We would like to determine a policy that maps each given pair of the physical location and orientation of the robot into an

action to take. In particular, we would like to determine an optimal policy that minimizes the expected amount of time for the robot to reach one of the charging sites. This can be modeled as a Markov decision problem over a Markov decision process $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \Pr, \text{Cost} \rangle$ and a performance criterion $\mathcal{F}$ as follows:

The state space: Each state $\langle X_1 = l, X_2 = d \rangle$ in $\mathcal{S}$ models a specific location l and a specific orientation d of the robot. This gives us a state space of 12,000 ($3 \times 1000 \times 4$) states, given the three floors, one thousand squares per floor, and four possible orientations.

The action space: $\mathcal{A}$ is the action space composed of the actions of (1) moving in four directions while staying in the same floor, (2) getting into an elevator, and (3) moving to another floor by elevator. The robot is enabled to move in a specific direction only if there are no obstacles blocking its way.

Action cost and transition probability: Cost and Pr specify the cost function and conditional probability distributions associated with actions and possible action outcomes. $c_{\mathbf{uv}}(a)$ ($p_{\mathbf{uv}}(a)$) is the expected amount of time (the probability) of taking action a in a state $\mathbf{u} = \langle X_1 = l, X_2 = d \rangle$ and ends in another state $\mathbf{v} = \langle X_1 = l', X_2 = d' \rangle$.

The performance criterion: $\{\mathbf{u} = (l, d) | l$ is the position of a charging site$\}$ is the set of target states. The performance criterion $\mathcal{F}$ is to minimize the expected cumulative cost to reach the target states.

3.2.6 Computational Complexity

Although an instance of the Markov decision problem can be solved in time polynomial in the size of the underlying state space [Put94] [PT87b], the Markov decision problem is PSPACE-hard in general.

This follows from the facts that classical planning involving propositional STRIPS-like operators is PSPACE-hard [Byl94] and that we can transform any instance of the classical STRIPS planning problem involving propositional STRIPS-like operators into an instance of the Markov decision problem in the following

way:

- propositions in STRIPSplanning are viewed as boolean state variables and implicitly specify a state space,
- the truth values of the propositions at a point in time determines the state of the underlying Markov decision process at that time,
- the set of STRIPS-like operators specifies a state transition function while each STRIPS-like operator is viewed as an action involving no cost,
- the states satisfying the goal are the only absorbing states and are uniformly associated with a very large positive reward, and
- the performance criterion is to maximize the final reward.

An optimal policy for the derived instance of the Markov decision problem maps a given state **u** to an action (*i.e.,* a STRIPS-like operator), and that action in turn deterministically maps state **u** to a next state. Consequently, by adopting an optimal policy for the derived Markov decision problem instance, a unique sequence of STRIPS-like operators (*i.e.,* a linear plan) is determined with a state trajectory leading from the initial state to a target state if and only if the goal is achievable.

Moreover, optimal decision making in sequential decision networks is PSPACE-hard even if only one decision stage is involved. This is because an instance of the classical planning problem can be reduced to a one-stage sequential decision network where

- the starting control node unconditionally leads to the only decision stage,
- the STRIPS-like operators in the planning instance represent the actions applicable in the decision stage, which involve no action cost,
- the goal in the original planning instance must be satisfied in the decision stage to enter the ending control node,

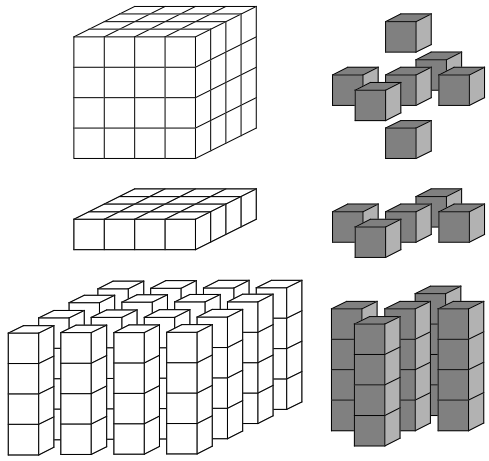

Figure 3.6: Three views of a three-dimensional state space with the corresponding local structure of space shown to the right. The top view represents the unstructured state space, the middle view represents an abstraction obtained by projection, and the bottom view represents a decomposition obtained by partitioning the states into aggregate states.

- the states satisfying the goal are the only absorbing states and are uniformly associated with a very large positive reward, and
- the performance criterion is to maximize the final reward.

An optimal policy for the derived one-stage sequential decision network then determines a linear plan (*i.e.,* a sequence of STRIPS-like operators) resulting in a state trajectory leading from the initial state to a target state if and only if the goal of the original planning instance is achievable.

3.3 Coping with Very Large State Spaces

A factored state-space representation with n boolean state variables represents an n-dimensional state space with $O(2^n)$ states. Large state spaces present a computational challenge. Assuming that both a problem instance and a solution (a plan) can be encoded in a compact form, we would like to generate solutions in time bounded by tractable factor of the problem and solution size. In this thesis, aggregation and abstraction play important roles toward this goal. Instead of

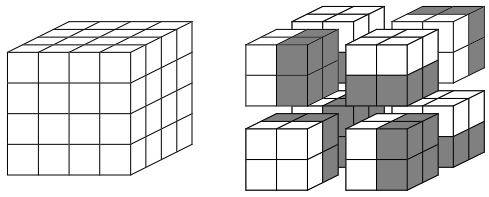

Figure 3.7: Region-by-region dimensionality reduction. A three-dimensional space is represented as the union of two-dimensional abstract subspaces shaded dark gray.

considering the entire state space $\mathcal{S}$ as a whole, the states in $\mathcal{S}$ can be decomposed into disjoint groups with each group considered as an aggregate state or a region. Figure 3.6 illustrates three views of a multi-dimensional state space. The bottom view displays sixteen regions, each of which is an aggregate state composed of four base-level states.

Abstraction is a special type of aggregation for projecting away some state variables. Given a subset $\mathcal{X}$ of Domain($\mathcal{D}$), the abstract state space determined by $\mathcal{X}$ is Space($\mathcal{X}$) $= \prod_{X_i \in \mathcal{X}} \Omega_{X_i}$. Each abstract state in Space($\mathcal{X}$) is composed of the states that share identical values over the variables in $\mathcal{X}$. Assuming a specific ordering on the variables in $\mathcal{X}$, an abstract state $\mathbf{s}$ in $\mathcal{S}$ is represented as a vector $\langle v_1, v_2, \ldots, v_h \rangle$ where (1) v_i is a specific value for the ith variable in $\mathcal{X}$ and (2) $h = \mid \mathcal{X} \mid$. We say that Space($\mathcal{X}$) is derived from the factored state space $\mathcal{S}$ by abstracting away the variables in Domain($\mathcal{D}$) $- \mathcal{X}$. The middle view in Figure 3.6 illustrates an abstract state space that projects away one dimension of the state space.

We assume that all of the state variables are relevant in at least some portion of the state space and so the dimensionality of the problem cannot be reduced by simply abstracting away some of the state variables without suffering some loss in performance. However, it is very likely that not all state variables are relevant in all portions of the state space (*e.g.,* when you are planning to take a walk in southern Florida you can neglect the possibility of snow). After decomposing a state space into m regions, at times only a small subset (of size no more than r, $r \ll n$) of the set of all state variables is relevant for decision making in each region. Consequently, we are concerned with an abstract subspace of size no more

than 2^r for each region, and the size of the union of these abstract subspaces is no more than $m2^r$. Figure 3.7 illustrates how a three-dimensional state space might be represented as the union of two-dimensional abstract subspaces.

Chapter 4

A Generic Overview of the Decomposition Techniques

In the next three chapters, we develop decomposition methods for the temporal projection problem, and the Markov decision problem. For these decomposition methods, we demonstrate the resulting computational efficiency in terms of the structural parameters in the underlying inter-regional interaction patterns. This chapter provides a generic overview of these decomposition methods.

4.1 Regional Decomposition and Inter-regional Interaction Patterns

Regions and Dynamical Subsystems:
Given a discrete dynamical system $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Delta_{\mathcal{D}} \rangle$, a region $R = \mathcal{S}'$ is simply a state subspace of $\mathcal{S}$, and region R determines a dynamical subsystem $\mathcal{D}' = \langle \mathcal{S}', \mathcal{A}', \Delta'_{\mathcal{D}} \rangle$ where (1) $\mathcal{A}'$ is an action subspace of $\mathcal{A}$ and is composed of actions that are enabled in one or more states in $\mathcal{S}'$, and (2) $\Delta'_{\mathcal{D}}$ is the fraction of $\Delta_{\mathcal{D}}$ which describes how the actions in $\mathcal{A}'$ result in state transitions in $\mathcal{S}'$.

Regional Decomposition:
A regional decomposition of a discrete dynamical system $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Delta_{\mathcal{D}} \rangle$ is a

collection of regions (*i.e.,* state subspaces) covering the state space $\mathcal{S}$.

The Directly Relevant Variables of a Region:
A variable X_i is directly relevant to a region R if and only if X_i is directly relevant to an action enabled in a state in region R. We use Relevant(R) to denote the set of variables directly relevant to a region R.

Inter-regional Interaction Patterns:
Given a regional decomposition of a discrete dynamical system, the system's trajectory through the state space is a sequence of local trajectories through local regions. After entering a region, the system evolves along the states in the region for a while, and then moves to another region. Such inter-regional interaction patterns reflect how the regional dynamical subsystems interact with one another. Decomposition techniques are then developed to exploit locality and dependency embedded in inter-regional interaction patterns.

Multi-Level Permutation Patterns in Multi-Level Task Networks:
In a multi-level task network, each task is associated with a region, the entire set of regions are organized as a region hierarchy.

- **The Parent-child Relationship in a Region Hierarchy:**
  In the region hierarchy, we can define a parent-child relationship as follows: a region R' is a child region of a region R if and only (1) R is associated with a task tk_i and (2) R' is associated with a child task of tk_i. Naturally, we can also extend this parent-child relationship to define the ancestor-descendant relationship in the region hierarchy.

- **Characteristics of Multi-Level Permutation Patterns:**
  In a multi-level task network, each region is visited (*i.e.,* entered and left) exactly once. After entering a region R, the child regions of R are visited one after another in any total order consistent with a given partial order (over the tasks associated with these child regions). Conceptually, a decision maker can arbitrarily permute the order in entering the child regions

of a region as long as the chosen total order is consistent with the given partial order. This is the reason why such patterns are called multi-level permutation patterns.

Branching Patterns in Markov Decision Processes:
When a given Markov decision process is decomposed into disjoint regions, the process evolves from region to region in a branching pattern.

- **The Parent-child Relationship in Branching Patterns:**
  We say that a region R is adjacent to a region R' if and only if the system can reach a state in region R' following a state transition from a state in region R. For convenience, we define a parent-child relationship in terms of adjacency among regions: a region R' is a child region of a region R if and only region R is adjacent to region R'. Naturally, we can then extend this parent-child relationship to define the ancestor-descendant relationship in the region hierarchy.

- **Characteristics of Branching Patterns:**
  Conceptually, disjoint regions in a Markov decision process are interconnected as a directed graph, where each region is a node and the directed edges represent the parent-child relationship among regions. After entering a region R, the underlying discrete dynamical system evolves along the states in R for a while, and then branches into one of the child regions of R. This is the reason why such patterns are called branching patterns.

- **Acyclic Branching Patterns:**
  An acyclic branching pattern is a special branching pattern where regions are interconnected as a directed acyclic graph. Each region is visited at most once in an acyclic branching pattern, since a region can never be visited again after leaving the region.

- **Strongly-connected Branching Patterns:**
  A strongly-connected branching pattern is a special branching pattern where regions are interconnected as a single strongly-connected component. Each

region can be visited for multiple times in a strongly-connected branching pattern, since after leaving a region R the system may branch through several other regions and then reenter region R.

- **Factoring a Branching Pattern:**
  In general, the regions in a branching pattern are interconnected as a directed graph with several strongly connected components. On the one hand, within a strongly connected component, the system evolves along the regions in that component in a strongly-connected branching pattern. On the other hand, when each connected component is considered as a macro region, the system evolves along these macro regions in an acyclic branching pattern. In this way, every branching pattern can be factored out as a product of an acyclic branching pattern and several strongly-connected branching patterns.

4.2 Structural Analysis of Inter-regional Interaction Patterns

Locality, Dependency, and Dimensionality Reduction:
Although generally all state variables are indispensable in modeling a discrete dynamical system, it is rare that every state variable is directly relevant in every region. At times, the action dynamics in a region R only directly involves a few state variables. For each region R, a decision maker is only concerned with the directly relevant state variables of R plus some additional state variables characterizing the causal dependencies between R and other regions; the other variables are irrelevant in region R. This allows a decision maker to attain dimensionality reduction in individual regions by properly abstracting away the irrelevant variables. Locality and causal dependency play critical roles in the decomposition methods developed in this thesis. In the following, we quantify the underlying locality and dependency using a structural parameter for each kind of inter-regional interaction patterns. The smaller a structural parameter is, the more efficiency

we can obtain by using the decomposition techniques developed in this thesis.

- **The Structural Parameter $\mathcal{L}_1$ in a Multi-level Permutation Pattern:** The structural parameter in a multi-level permutation pattern is

$$\mathcal{L}_1 = \max_R(b_R + c_R)$$

where (1) the branching factor b_R of a region R is the number of child regions of R, and (2) the coupling factor c_R of a region R is the number of variables directly relevant to at least two of R's child regions.

- **The Structural Parameter $\mathcal{L}_2$ in an Acyclic Branching Pattern :** The structural parameter in an acyclic branching pattern is

$$\mathcal{L}_2 = \max_R |A_R|$$

where A_R is the set of state variables that are either (1) directly relevant to R or (2) directly relevant to at least one ancestor region of R and also at least one descendant region of R.

- **The Structural Parameter $\mathcal{L}_3$ in a Strongly Connected Branching Pattern:** The structural parameter in a strongly connected branching pattern is

$$\mathcal{L}_3 = |\bigcup_R \text{Periphery}(R)|$$

where Periphery(R) is the set of states that are not in R but but reachable from a state in R.

4.3 Exploiting Structure for Computational Efficiency

To cope with very large state spaces, we symbolically decompose a state space into local regions, and exploit locality and dependency embedded to attain dimensionality reduction in individual regions.

Aggregation and Regional Decomposition:
Instead of considering the entire state space $\mathcal{S}$ as a whole, the states in $\mathcal{S}$ can be aggregated into a set of regions; each region corresponds to an aggregate state. Such a regional decomposition allows a planning problem to be decomposed into

- a master problem concerning the overall performance and the interactions among local regions, plus
- a sequence of planning subproblems regarding the local performance and local dynamics over individual regions.

The master problem iteratively determines, from a global perspective, the planning subproblem to solve in a region. The solution to this planning subproblem is utilized later by the master problem in determining other planning subproblems. The focus of this thesis is to exploit locality embedded in planning instances such that efficient local computation can be conducted in optimizing a global performance criterion. An obvious computational advantage of regional decomposition arises from the fact that (1) each subproblem involves only the states in a region plus some adjacent states in the neighboring regions and that (2) the master problem involves only the regions as aggregate states.

Abstraction and Dimensionality Reduction:
Given a regional decomposition, every variable is relevant to the action dynamics in one or more regions, assuming no redundancy in the description of the problem instance. However, it is rare that every variable is relevant to every region. At times, we can further aggregate states in a region R according to their values over the relevant variables in R. In this way, the irrelevant variables in a region can be abstracted away while solving the planning subproblems over that region. An appropriate region decomposition can introduce significant dimensionality reduction in individual regions. Dimensionality reduction is an important source of computational efficiency, since it can exponentially reduce the size of planning subproblems.

4.3.1 Exploiting Structure in Multi-Level Permutation Patterns

The following is an overview of the decomposition method for the temporal projection problem developed in Chapter 5. We refer the readers to Chapter 5 for the details of the algorithm and analysis.

An Overview of a Structure-exploiting Algorithm:

1. **Input and Output:**
   Given a multi-level task network and a goal, the algorithm exploits the multi-level permutation pattern underlying a regional decomposition on the network. If there exists a linear plan to achieve the goal, the algorithm returns such a plan; otherwise the algorithm reports that no solution exists.

2. **The Master Problem and Regional Subproblems:**
   In the master problem, the goal for a problem instance is decomposed into regional subgoals of individual regions. To solve the master problem, for each region R a subproblems over R is first devised to determine the possibility of attaining the regional subgoals of R. This is the first phase in solving the master problem. If it is possible to achieve every regional subgoal, for each region R a subproblem is devised to determine the ordering of R's child regions in order to achieve the regional subgoal; a linear plan to achieve the goal is then generated according to this information. This is the second phase in solving the master problem.

3. **Solving Subproblems and Propagating Regional Solutions:**

   - **The First Phase:** In this phase, a subproblem over a region R can only be devised and solved after all the subproblems over the child regions of R are solved. After solving a subproblem over a region R', an abstract information is compiled regarding all possible pairs of (1)

the values of the abstract variables of R' right before entering R' and (2) the values of the abstract variables of R' immediately after leaving R' in order to achieve the regional subgoal. This abstract information is then propagated to the parent region of R'. The subproblem over the parent region R of R' is then devised according to the abstract information propagated from R' and other child regions of R.

- **The Second Phase:** In this phase, a subproblem over a region R can only be devised and solved after the subproblem over the parent region of R is solved. After solving a subproblem over a region R, an abstract information is compiled for each child region R' of R regarding a specific pair of values of the abstract variables of R' right before entering R' and immediately after leaving R' in order to adopt a specific total order on the child regions of R. This abstract information is then propagated to region R'. The subproblem over the child region R' of R is then devised according to the abstract information propagated from R.

4. **Dimensionality Reduction:**
   For each region R, the subproblems over R are only concerned with information encoded in the abstract variables of R and R's child regions. It turns out that (1) the union of the sets of abstract variables of the child regions of R is simply the set of coupling variables of R and (2) an abstract variables of R is also a coupling variable of R. Consequently, a subproblem over R can be solved by considering the abstract state space determined by the coupling variables of R, rather than explicitly enumerating every state in R.

5. **Quantifying Computational Efficiency:**
   Note that the structural parameter $\mathcal{L}_1$ for the underlying multi-level permutation pattern is the maximal number of coupling variables in a region. The decomposition method described above runs in $O(n \times 8^{\mathcal{L}_1})$ time, where n is the number of primitive tasks in the task network. When $\mathcal{L}_1 = O(\log n)$,

the algorithm runs in time polynomial in the total number of primitive tasks n.

4.3.2 Exploiting Structure in Acyclic Branching Patterns

The following is an overview of the decomposition method for Markov decision problems with acyclic branching patterns. In particular, this algorithm can be applied to determine an optimal policy for a sequential decision network. We refer the readers to Chapter 6 for the details of the algorithms and analysis.

An Overview of a Structure-exploiting Algorithm:

1. **Input and Output:**
   Given a Markov decision process and a performance criterion, the algorithm exploits the acyclic branching pattern underlying a regional decomposition on the Markov decision process. At the end, the algorithm returns an optimal policy for the Markov decision process.

2. **The Master Problem and Regional Subproblems:**
   In the master problem, the given Markov decision process together with the given performance criterion is decomposed into subproblems concerning Markov decision processes over local regions with local performance criteria; each region is exactly associated with a subproblem. To solve the master problem, the subproblems over local regions are solved systematically; a subproblem over a region R is devised and solved only after all the subproblems over the child regions of R are solved. An optimal policy over the given Markov decision process is simply the union of the optimal policies over local regions, which are determined by solving the subproblems over local regions.

3. **Propagating Regional Solutions:**
   After solving a subproblem over a region R', an optimal policy π over R' with respect to a local performance criteria is determined. According to π,

for each parent region R of R', the information about the optimal expected cumulative cost received after reaching a state in R' from a state in R are compiled. This information is then propagated to the parent regions of R' for devising the subproblems over the parent regions of R'.

4. **Devising Regional Subproblems:**
   After receiving all the information propagated from the child regions of a region R, a new Markov decision process $\mathcal{M}_R$ over R is devised as follows:

   - The state space of $\mathcal{M}_R$ is the entire region R plus a special sink state representing all the child regions of R as a whole. The action space of $\mathcal{M}_R$ comprises all the actions enabled in some state in R.
   - Immediately following an action a that is enabled in a state $\mathbf{u}$ in R, the transition probability of entering the special sink state from $\mathbf{u}$ is the sum of the probabilities of entering the states outside R from $\mathbf{u}$.
   - Immediately following an action a enabled in a state $\mathbf{u}$ in R, the cost of entering the special sink state from $\mathbf{u}$ is the expectation (over the states outside R reachable from $\mathbf{u}$) of (1) the immediate action cost of entering a state outside R and (2) the expected cumulative cost after entering a state outside R.
   - The action cost and transition probability involving a state transition between two states in R remain the same as described in the original Markov decision process.

   The subproblem over region R is to determine a policy that minimizes the expected cumulative cost minus for the Markov decision process $\mathcal{M}_R$. Such a policy consequently minimizes the expected cumulative cost starting from any state in R.

5. **Dimensionality Reduction:**
   Any two states in a region R are indistinguishable with respect to the action dynamics in R if these two states share the same values for the active variables in R. Therefore, the states in a region R can be aggregated

according to the values of the abstract variables in R, and thus abstract away the passive variables in R. This allows a subproblem over a region R to be solved by simply considering the abstract state space determined by the active variables of R, rather than explicitly enumerating every state in R.

6. **Quantifying Computational Efficiency:**
   The decomposition algorithm described above runs in $O(m \times 8^{\mathcal{L}_2})$ time for a given Markov decision process where m is the number of regions in the regional decomposition. When $\mathcal{L}_2 = O(\log m)$, the algorithm runs in time polynomial in the total number of regions m.

4.3.3 Exploiting Structure in Strongly-connected Branching Patterns

The following is an overview of the decomposition method for Markov decision problems with strongly-connected branching patterns. We refer the readers to Chapter 7 for the details of this algorithm.

An Overview of a Structure-exploiting Algorithm:

1. **Input and Output:**
   Given a Markov decision process and a performance criterion, the algorithm exploits the strongly-connected branching pattern underlying a regional decomposition of the Markov decision process. At the end, the algorithm returns an optimal policy for the Markov decision process.

2. **The Master Problem and the Policy Repository:**

   - The master problem iteratively estimates the values of the states in the peripheries of region (*i.e.,* the expected cumulative cost starting from the states until reaching a target state). On each iteration, a

Markov decision problem and a regional decomposition with an acyclic branching pattern are devised according to the estimated values of the states in the peripheries. A global policy is determined at the end of an iteration by solving the Markov decision problem devised on that iteration.

- A policy repository maintains up to $u + 1$ of the previously generated global policies at a time, where u is the number of periphery states. When a global policy is determined at the end of an iteration, it replaces one of the existing policies in the repository. By examining the repository, we can determine whether an optimal policy or a policy within a specified range of the optimum can be generated as a solution; if so, we are able to generate such a policy by properly combining the policies in the repository. Otherwise, we are able to determine new estimates for the periphery states and go through another iteration of computation. The policy repository guarantees to produce an optimal policy in the long run.

3. **Devising and Solving Regional Subproblems:**
   On each iteration, a Markov decision problem and a regional decomposition with an acyclic branching pattern are devised according to the estimated values of the states in the peripheries.

   - First, we transform the given regional decomposition $P = \{R_1, \ldots, R_h\}$ into a regional decomposition $Q = \{U, K_1, \ldots, K_h\}$ where (1) U is $\bigcup_{R_i \in P} \text{Periphery}(R_i)$, which represents the peripheries of the regions R_i's, (2) for each region R_i, we have $K_i = \text{Kernel}(R_i) = R_i - U$, which represents the kernel of region R_i, and (3) the resulting regional decomposition Q has a star topology as depicted in Figure 7.5.
   - Second, we transform each state $\mathbf{u}$ in U into a target state, and associate $\mathbf{u}$ with the estimated cumulative cost of $\mathbf{u}$ in the original process as the final penalty when entering $\mathbf{u}$. Consequently, this provides a Markov decision process and a regional decomposition Q with an

acyclic branching pattern. We adopt the expected cumulative cost until reaching a target state as the performance criterion of the Markov decision problem.

- Using the regional decomposition Q and the decomposition method for acyclic branching patterns, the devised Markov decision problem can be further decomposed into regional subproblems over regions K_i's.

4. **Quantifying Computational Efficiency:**
   Note that the structural parameter $\mathcal{L}_3$ for the underlying strongly-connected branching pattern is the total number of states in the peripheries of regions.

   - The decomposition algorithm as described above improves the solution quality iteratively, and converges to an optimal policy in the long run.
   - The policy repository can be compactly maintained as $(\mathcal{L}_3+1)\times(\mathcal{L}_3+1)$ matrices and vectors of length $\mathcal{L}_3+1$.

   Instead of directly solving the original problem, the computation on each iteration comprises two parts: (1) deriving and solving regional subproblems over local regions and (2) maintaining a solution repository. The first part renders much computational efficiency by dividing a large problem into several subproblems of tractable sizes. The second part represents the additional cost to pay when compiling the solutions to subproblems. When a solution of good quality can be obtained within a few number of iterations, this decomposition technique can provide significant computational efficiency.

Chapter 5

Decomposition Techniques for the Temporal Projection Problem

5.1 More about Temporal Projection

In the following, we formally define temporal projection problems considered in this thesis, and introduce addiotional terminology and notation used in this chapter.

Conditions and Events:
Following the convention of literature, we refer to a binary state variable as a *condition* and say an *event* occur when an agent takes a specific action. An event can then be represented as a set of causal rules in the same way as an action is represented in Chapter 2.2. Multi-value state variables can be encoded by conditions without loss of generality. A set of conditions $\mathcal{P}$ determines a state space $\mathcal{S}_\mathcal{P}$ and the truth values of the conditions in $\mathcal{P}$ at a point in time determine a state in $\mathcal{S}_\mathcal{P}$.

Possible Event Sequences:

Given a set of events $\mathcal{E}$ and a set of ordering constraints $\mathcal{O}$ on $\mathcal{E}$, a possible event sequence $\mathbf{q}$ of length h over a set of events $\mathcal{E}$ ($h \leq n$) is a sequence $\langle e_1, e_2, \ldots, e_h \rangle$ where (1) $e_1, \ldots, e_h$ are distinct events in $\mathcal{E}$ and (2) the order in which the events appear in $\mathbf{q}$ is consistent with the ordering constraints in $\mathcal{O}$. We also define the following sets of event sequences:

$\mathbf{Q}_{\mathcal{O}}^{\mathcal{E}}$= { $\mathbf{q}$ | $\mathbf{q}$ is a possible event sequence of length n },

$\overline{\mathbf{Q}}_{\mathcal{O}}^{\mathcal{E}}$= { $\mathbf{q}$ | $\mathbf{q}$ is a possible event sequence of length equal to or less than n }, and

$\overline{\mathbf{Q}}_{\mathcal{O},e}^{\mathcal{E}}$= { $\mathbf{q}$ | $\mathbf{q} \in \overline{\mathbf{Q}}_{\mathcal{O}}^{\mathcal{E}}$ and event e is the tail of $\mathbf{q}$}.

Problem Instances for Temporal Projection:

A problem instance of temporal projection is a five tuple $\langle \mathcal{P}, \mathcal{E}, \mathcal{O}, s_0, \mathcal{G} \rangle$ where

- $\mathcal{P}$ is a finite set of conditions;
- $\mathcal{E}$ is a finite set of events;
- $\mathcal{O}$ is a finite set of ordering constraints on $\mathcal{E}$;
- s_0 is the initial state, $s_0 \in S_{\mathcal{P}}$;
- $\mathcal{G}$ is an expression and $\mathcal{G}$ specifies a set of *goal states* in $S_{\mathcal{P}}$ in which $\mathcal{G}$ is true.

The Tasks of Temporal Projection:

Given an instance of the temporal projection problem, the PRJ task, the PRJ1 task, and the PRJ2 task of temporal projection are to determine the existence of an event sequence $\mathbf{q}$ in $\mathbf{Q}_{\mathcal{O}}^{\mathcal{E}}$, $\overline{\mathbf{Q}}_{\mathcal{O}}^{\mathcal{E}}$, and $\overline{\mathbf{Q}}_{\mathcal{O},e}^{\mathcal{E}}$ respectively such that (1) the events can occur as the sequence $\mathbf{q}$ and (2) the dynamical system may enter a goal state immediately following $\mathbf{q}$. In addition, we generate such an event sequence $\mathbf{q}$ if one exists, and determine the rules applied by the individual events to enter a goal state.

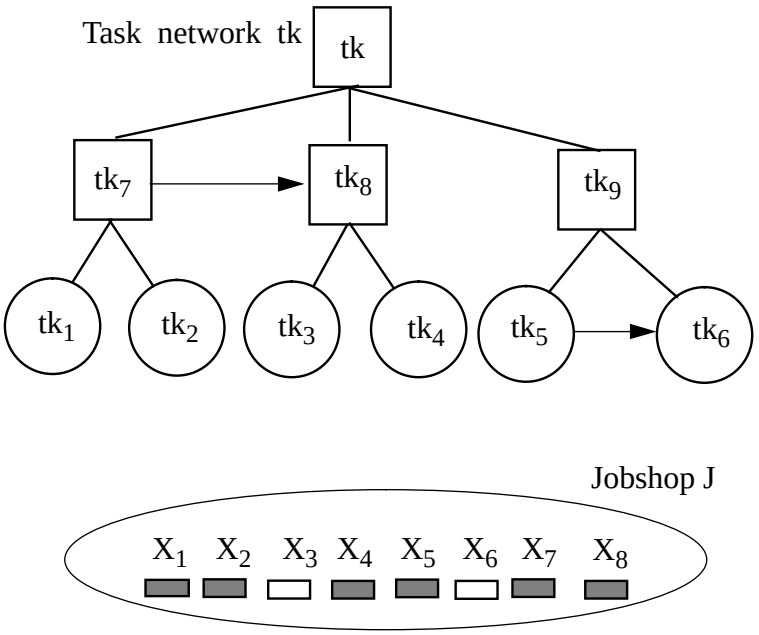

Figure 5.1: Temporal projection in a multi-level task network

Theorem 5.1 *Given an instance of the temporal projection problem, the PRJ task, the PRJ1 task, and the PRJ2 task can be reduced to one another by transforming the problem instance. All these transformations can be done in polynomial time by (1) adding at most two more conditions and two more events, and (2) appropriately modifying the ordering constraints and causal rules.* [1]

Throughout this paper, we consider the PRJ task [2] as the standard form of temporal projection. The decomposition algorithm and its analysis in this chapter focus on the PRJ task, which are also applicable to the PRJ1 task and the PRJ2 task after transforming the problem instances accordingly.

Consider the multi-level task network in the job-shop scheduling domain in Chapter 3.3. The three scenarios in Chapter 3.3 involve the PRJ task, the PRJ1 task, and the PRJ2 task respectively. In the remainder of this chapter, we use the first scenario in Chapter 3.3 as an example for illustrating algorithmic details,

[1] The proofs for all theorems stated in this thesis are provided in Appendix A at the end of this document.

[2] The PRJ1 task is considered by Lin and Dean in [LD93] and the PRJ2 task is considered by Dean and Boddy in [DB88b] and Nebel and Bäckström in [NB92].

which provides the following problem instance for temporal projection.

- $\mathcal{P} = \{X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8\}$. X_i, $1 \leq i \leq 8$, is a condition modeling the job shop environment.
- $\mathcal{E} = \{tk_1, tk_2, tk_3, tk_4, tk_5, tk_6\}$. For simplicity, we use tk_i to denote the event of performing the primitive task tk_i also. Figure 5.2 display the causal rules associated with the individual events.
- Figure 5.1-c depicts the ordering constraints $\mathcal{O}$ on $\mathcal{E}$. For convenience, we also refer to $\mathcal{E}$ as R. The events in the three event subsets R_7, R_8, and R_9 must occur as three atomic groups, where $R_7 = \{tk_1, tk_2\}$, $R_8 = \{tk_3, tk_4\}$, $R_9 = \{tk_5, tk_6\}$. The events in R_8 must occur before the events in R_7. Event tk_6 must occur before event tk_5.
- We assume that in the initial state s_0 all conditions are initially true.
- $\mathcal{G} = (\langle X_1, false\rangle\langle X_2, false\rangle\langle X_4, false\rangle\langle X_5, false\rangle\langle X_7, false\rangle\langle X_8, false\rangle)$.
- Our task is to (1) determine the existence of an event sequence $\mathbf{q}$ in $\mathbf{Q}_{\mathcal{O}}^{\mathcal{E}}$ that may end in a goal state, in which the goal $\mathcal{G}$ is true, and (2) if such a $\mathbf{q}$ exists, generate $\mathbf{q}$ and determine the rules applied by the individual events to enter a goal state.

5.2 Tradeoff in Computational Complexity

In the following, we show that (1) the kinds of events, (2) the ordering constraints on events, and (3) the size of the state space all contribute to the computational complexity. First we define the following terms.

Classifying Events:
A one-rule event is associated with a single rule that is applicable in exactly one state and maps that state to a distinct state. A two-rule event is associated with two rules, each of which is applicable in exactly one state and maps that state to

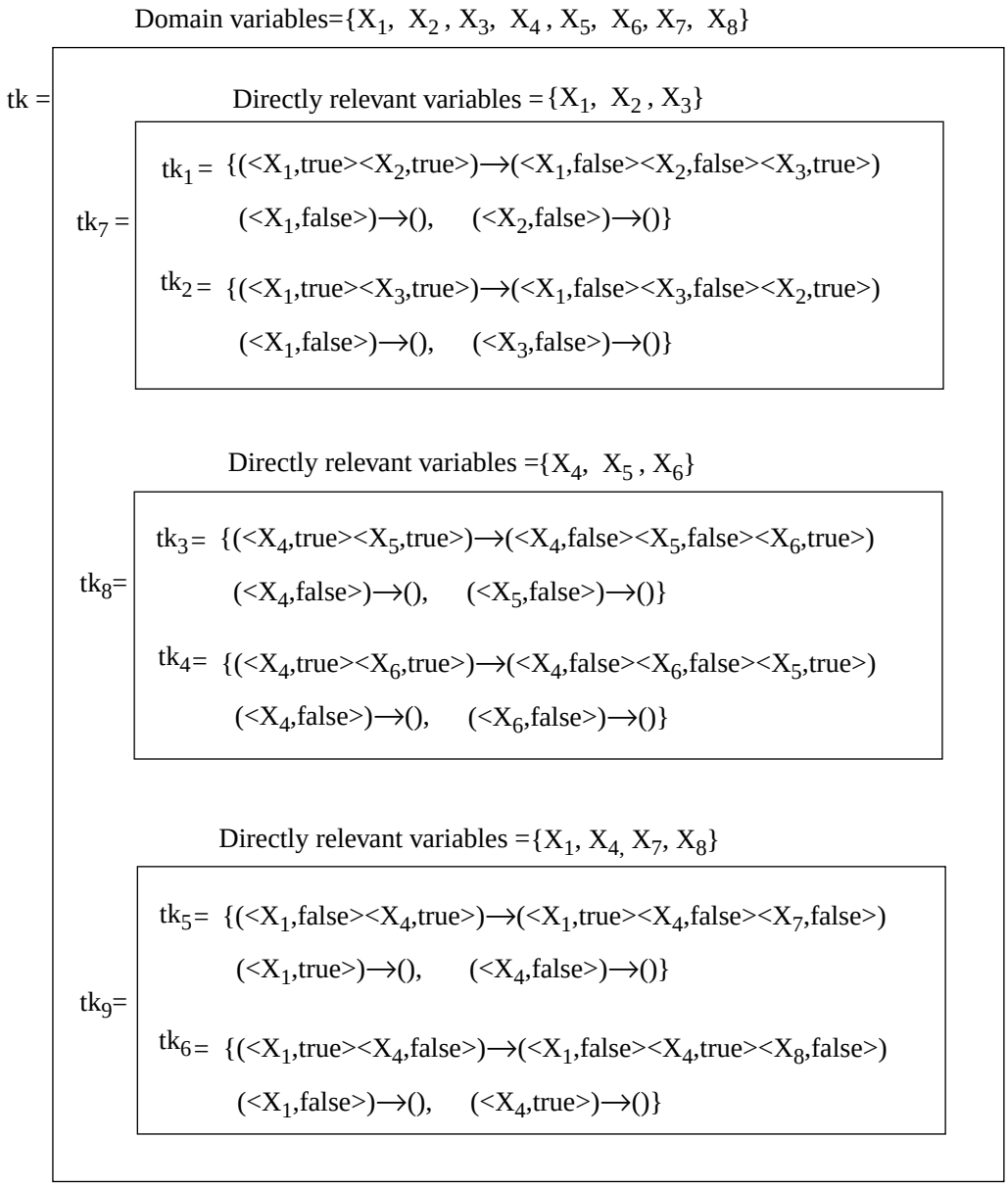

Figure 5.2: The events and causal rules in the jobshop example

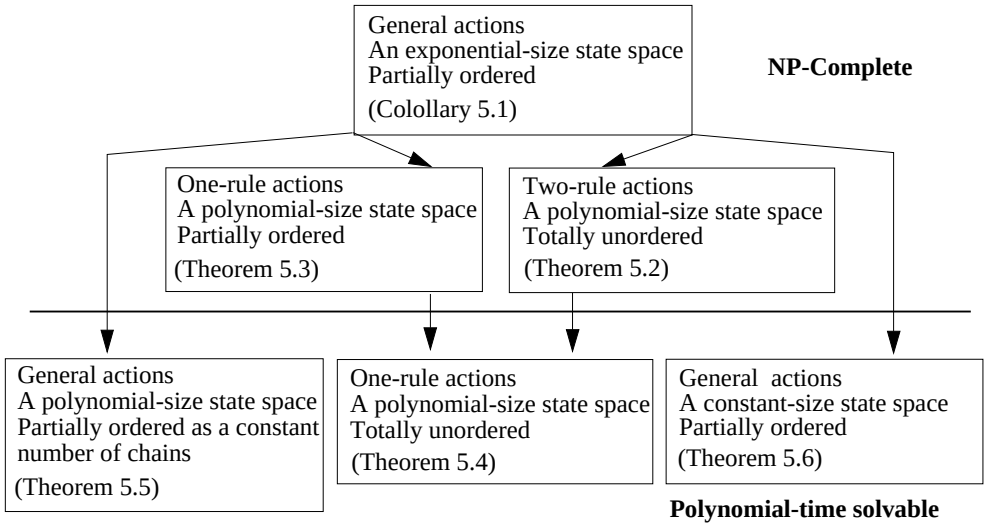

Figure 5.3: Complexity trade-off in temporal projection

a distinct state. In general, an event can have multiple rules, each of which can map many states to states other than themselves.

Classifying Ordering Constrints:

Given a set of events $\mathcal{E}$, we say that $\mathcal{E}$ is *partially ordered* if $\mathcal{E}$ is constrained by an arbitrary partial order $\prec$; we say that $\mathcal{E}$ is partially ordered as m chains if (1) $\mathcal{E}$ can be partitioned into m disjoint event subsets, (2) the events in each subset are totally ordered as a chain, and (3) the events in the entire set $\mathcal{E}$ can only be ordered by interleaving these m chains; we say that $\mathcal{E}$ is *totally unordered* if every total order on the set of events $\mathcal{E}$ is possible.

Classifying the Sizes of State Spaces:

Given a set of events $\mathcal{E}$ and a set of conditions $\mathcal{P}$, we say that the state space $S_{\mathcal{P}}$ is of constant size if $\mathcal{P}$ is composed of $O(1)$ number of conditions; we say that the state space is of polynomial size if $\mathcal{P}$ is composed of $O(\log |\mathcal{E}|)$ number of conditions; and we say that the state space is of exponential size if $\mathcal{P}$ is composed of $O(|\mathcal{E}|)$ number of condition.

Figure 5.3 displays the complexity trade-offs regarding events, ordering constraints, and state spaces, where

- events are categorized as either one-rule events, two-rule events or general events according to how they trigger state transitions;
- events can be either (1) partially ordered as a constant number of chains, (2) totally unordered, or (3) constrained by an arbitrary partial order;
- a state space can be either constant-size, polynomial-size, or exponential-size with respect to the number of events.

Given an instance of the temporal projection problem, we can first guess a solution event sequence $\mathbf{q}$ and the rules applied by the individual events. We can then verify in polynomial time whether a goal state is reached immediately following $\mathbf{q}$. Therefore we have the following lemma.

Lemma 5.1 *The temporal projection problem is in NP.*

When there is no structure in event ordering, the following two theorems indicate that we have very limited ability to deal with a polynomial-size state space determined by $O(\log |\mathcal{E}|)$ number of conditions.

Theorem 5.2 *Temporal projection regarding two-rule events, totally unordered, with a polynomial-size state space is NP-Complete.*

Theorem 5.3 *Temporal projection regarding one-rule events with an arbitrary partial order and a polynomial-size state space is NP-Complete* [3].

Corollary 5.1 *Temporal projection regarding general events with an arbitrary partial order and an exponential-size state space is NP-Complete.*

[3] Nebel and Bäckström [NB92] prove the NP-Completeness of a closely related case. Their proof technique is adapted here to prove Theorem 5.3

The following three theorems demonstrate that temporal projection is more tractable when we have (1) available structure in causal rules or event ordering or (2) a small state space. These complexity results motivate our effort to exploit inherent locality in event ordering and the dependencies among conditions and events.

Theorem 5.4 *Temporal projection regarding one-rule events, totally unordered, with a polynomial-size state space is solvable in polynomial time.*

Theorem 5.5 *Temporal projection regarding general events partially ordered as* $O(1)$ *number of chains of length* $O(|\mathcal{E}|)$ *or* $O(\log |\mathcal{E}|)$ *number of chains of* $O(1)$ *length with a polynomial-size state space can be solved in polynomial time.*

Theorem 5.6 *Temporal projection regarding general events with an arbitrary partial order and a constant-size state space is solvable in polynomial time.*

5.3 Locality in Multi-level Task Networks

When totally unordered, events can occur in an arbitrary order. However, there is locality in event ordering if the occurrences of events or groups of events closely relate to one another. Several closely related events may always occur as an atomic group; an event not in the group either occurs before or after the group. Similarly, several atomic groups may always occur as a macro atomic group. Consequently locality in event ordering can be introduced at many levels. This is the case in a multi-level task network. In the following, we model locality in event ordering when performing a mult-level task network using a *region hierarchy* and a set of *regional ordering constraints*.

Event Regions:

Given a set of events $\mathcal{E}$ and a set of ordering constraints $\mathcal{O}$ on $\mathcal{E}$, a subset X of $\mathcal{E}$ forms a region if and only if the events in X occur as an atomic group while an event not in X occurring either before or after the events in X. The entire set of events $\mathcal{E}$ and $\{e\}$ for every event e in $\mathcal{E}$ are regions by themselves.

In Figure 5.1, $R_i = \{tk_i\}$ for every primitive task $tk_1, tk_2, \ldots, tk_6$,
$R_7 = \{tk_1, tk_2\}$,
$R_8 = \{tk_3, tk_4\}$,
$R_9 = \{tk_5, tk_6\}$, and $R = R_7 \bigcup R_8 \bigcup R_9$ are all regions.

A Region Hierarchy:

A region X is a child region of a region Y and Y is the parent region of X if and only if Y is the smallest region that contains X as a proper subset. A region X is a descendant region of a region Y and Y is an ancestor region of X if and only if X is a proper subset of Y. A region hierarchy is a set of regions in which for any two regions either they are disjoint or one is a descendant region of the other.

In Figure 5.1, $R_7 = \{tk_1, tk_2\}$, $R_8 = \{tk_3, tk_4\}$, and $R_9 = \{tk_5, tk_6\}$, are the child regions of $R = R_7 \bigcup R_8 \bigcup R_9$. $\{tk_1\}$ and $\{tk_2\}$ are the child regions of R_7. $\{tk_3\}$ and $\{tk_4\}$ are the child regions of R_8. $\{tk_5\}$ and $\{tk_6\}$ are the child regions of R_9. Together, they form a region hierarchy, which is depicted in Figure 5.1-c as a tree.

Regional Ordering Constraints:

The set of regional ordering constraints $\mathcal{O}_R$ on a region R is a pair $\langle \sigma, \prec \rangle$, where $\sigma = \{R_1, R_2, \ldots\}$ is the set of child regions of R, and $\prec$ is a partial order on σ. The events in R_i must occur before the events in R_j if $R_i \prec R_j$.

In Figure 5.1, the possible event sequences are determined by the region hierarchy and the following regional ordering constraints:
$\mathcal{O}_{R_7} = \langle \{\{tk_1\}, \{tk_2\}\}, \emptyset \rangle$, $\mathcal{O}_{R_8} = \langle \{\{tk_3\}, \{tk_4\}\}, \emptyset \rangle$,
$\mathcal{O}_{R_9} = \langle \{\{tk_5\}, \{tk_6\}\}, \{\{tk_6\} \prec \{tk_5\}\} \rangle$, and $\mathcal{O}_R = \langle \{R_7, R_8, R_9\}, \{R_8 \prec R_7\} \rangle$.

A multi-level task network can be viewed as a region hierarchy with a set of regional ordering constraints. A primitive task in the task network is an event, which is a disjunction of several STRIPS-like operators and it changes the system state by using exactly one of the applicable operators. Tasks are organized as a hierarchy of event regions. The task associated with a region R

can be decomposed into the tasks associated with the child regions of R, and so on hierarchically. The tasks associated the child regions of a region R may be constrained by a partial order, but cannot be interleaved.

5.4 Causal Dependency in Multi-level Task Networks

Events affecting all conditions are rare. Instead, events tend to affect or be affected by a subset of the set of conditions $\mathcal{P}$. If a condition p is irrelevant to the events in a region R, we do not need to directly concern ourselves with p in reasoning about the changes caused by the events in region R. This can result in significant computational savings.

Causal Dependencies:

A condition p is dependent on a causal rule $\alpha \rightarrow \beta$ if p appears in α or in β. A condition p is dependent on an event e if p is dependent on a causal rule associated with e. A condition p is dependent on a region R if p is dependent on an event in R.

Problem Decomposition:

In Figure 5.1-b and Figure 5.1-c, the conditions X_1, X_2, and X_3 are dependent on region R_7, but independent of region R_8; the conditions X_4, X_5, and X_6 are dependent on region R_8, but independent of region R_7. If region R_9 were absent, we could decompose the problem instance into two independent parts: one with region R_7 and the conditions X_1, X_2, and X_3; the other one with region R_8 and the conditions X_4, X_5, and X_6. However, through the conditions X_1 and X_4 respectively, region R_9 interacts with the regions R_7 and R_8. This complicates the interactions among these three regions.

In general, two disjoint regions interact with one another through a few conditions that are dependent on both regions. In this situation, problem decomposition must be achieved by carefully exploiting structure in the causal dependencies

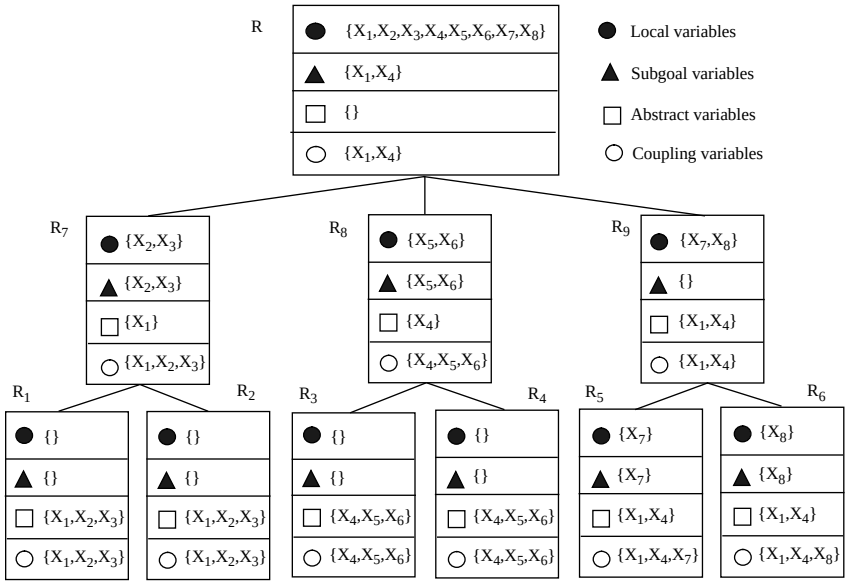

Figure 5.4: The condition subsets characterizing the causal dependencies among regions

among regions. In this section, we investigate the notions of the *subgoals* and *abstract events* of regions. These two notions reflect locality in causal dependencies. Using subgoals and abstract events, we are able to decompose the task of temporal projection into a sequence of local searches in individual regions.

5.4.1 Subgoal Conditions and Problem Decomposition

Given a region hierarchy, we can identify the *local conditions* and *subgoal conditions* of each individual region, which allows us to define the subgoals of regions.

Local Conditions and Subgoal Conditions:

A condition X_i is a local condition of region R if and only if X_i is dependent on R but not dependent on any events outside R. A condition X_i is a subgoal condition of a region R, if and only if (1) X_i is a local condition of region R, and (2) X_i is not a local condition of any child regions of R. Figure 5.4 displays the local conditions and subgoal conditions of the individual regions in the jobshop example.

The following lemmas directly come from the definitions of local conditions

and subgoal conditions.

Lemma 5.2 *If X_i is a local condition of a region R, the value of X_i can only be locally affected by the events in R. The value of X_i immediately before the events in R occur is the same as X_i's initial value. Immediately after all of the events in R have occurred, the value of X_i will not be further changed.*

Lemma 5.3 *If X_i is a subgoal condition of a region R, then R is the smallest region containing X_i as a local condition. The sets of subgoal conditions of different regions are mutually disjoint and together they form a partition of the set of conditions $\mathcal{P}$.*

Lemma 5.4 *The set of local conditions of a region R is the union of the local conditions of R's child regions and the subgoal conditions of R.*

Regional Subgoals:
Given a goal $\mathcal{G}$, the regional subgoal $\mathcal{G}_R$ of a region R is composed of the condition/value pairs in $\mathcal{G}$ whose condition components are local conditions of region R. The incremental subgoal $\tilde{\mathcal{G}}_R$ of region R is composed of the condition/value pairs in $\mathcal{G}$ whose condition components are subgoal conditions of region R.

The following theorem indicates the relationship between the goal and the regional subgoals.

Theorem 5.7 *The goal $\mathcal{G}$ is true immediately after all events have occurred if and only if for each region R, the regional subgoal $\mathcal{G}_R$ of R is true immediately after all of the events in R have occurred.*

By Lemma 5.4 and the definition of regional subgoals and incremental subgoals, we have the following lemma.

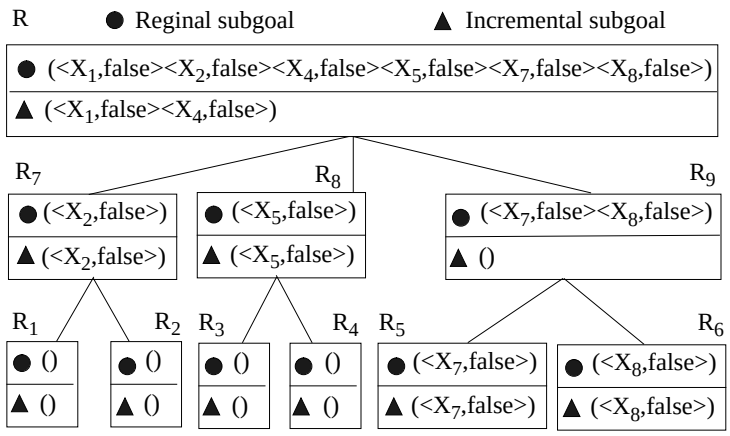

Figure 5.5: The regional subgoals and incremental subgoals of individual regions

Lemma 5.5 *The regional subgoal $\mathcal{G}_R$ of a region R is the union of the regional subgoals of R's child regions and the incremental subgoal $\tilde{\mathcal{G}}_R$ of region R.*

By Lemma 5.5, we have the following corollary of Theorem 5.7.

Corollary 5.2 *In order to achieve goal $\mathcal{G}$, for each region R the regional subgoals of R's child regions should be true respectively as soon as all of the events in the individual child regions have occurred; the incremental subgoal $\tilde{\mathcal{G}}_R$ should then be true immediate after all of the events in region R have occurred. Otherwise, the goal $\mathcal{G}$ cannot be true after all events have occurred.*

Figure 5.5 depicts the incremental subgoals and the regional subgoals in the problem instance corresponding to the jobshop example. For example, the regional subgoal of R_8 is $(\langle X_5, false\rangle)$, which indicates that condition X_5 must be false immediately after all of the events in R_8 have occurred. The goal $\mathcal{G}$ is equal to the regional subgoal $\mathcal{G}_R$ of the root region R, which contains all events in the problem instance. To achieve the goal $\mathcal{G}$ (i.e. $\mathcal{G}_R$), (1) the regional subgoals of regions R_7, R_8, and R_9 must be achieved respectively as soon as the individual regions are finished, and then (2) the incremental subgoal for $\tilde{\mathcal{G}}_R$ must be achieved immediately after all events have occurred.

5.4.2 Abstract Conditions and Abstract Events

Given a region hierarchy, we can identify the *abstract conditions* of each region, which allows us to define the abstract event of the corresponding region. Figure 5.4 displays the abstract conditions of individual regions in the jobshop example.

Abstract Conditions:
A condition X_i is an abstract condition of region R if X_i is (1) dependent on at least one event in R, and also (2) dependent on at least one event not in R.

Both the events in R and the events outside R can affect and be affected by the values of the abstract conditions of R. Therefore the set of abstract conditions of R is the media for inter-regional interactions between the events in R and the events outside R. For example, in Figure 5.1 the events tk_1 and tk_2 in region R_7 interact with the events outside R_7 through the abstract condition X_1 of R_7.

The number of the abstract conditions of a region R is a measure of the locality regarding the causal dependencies between the events in R and the events outside R. If region R does not have any abstract conditions, all of the conditions involved in the events of R are local conditions of R. In this case, the events in R together with the local conditions of R can be separated from the other events and conditions to form an independent problem instance.

The effects of the events in a region R can only affect and be affected by the local conditions and the abstract conditions of R. Since the values of the local conditions of R immediately before the events in R occur are the same as their initial values, the state transitions triggered by the events in a region R are determined by two factors: (1) the values of the abstract conditions of R immediately before the events in R occur, and (2) the ordering of the events in R. To achieve the regional subgoals $\mathcal{G}_R$, (1) the set of abstract conditions of R must have appropriate values immediately before the events in R occur, and (2) the events in R must be properly ordered. In turn, these two factors also affect the values of the abstract conditions of R after achieving the regional subgoals $\mathcal{G}_R$. This information about the possible preconditions and their effects on the values

of the abstract conditions can be compiled as an abstract event in achieving the regional subgoal. Before we define the abstract event of a region, we introduce the following terms.

Abstract States:
For an arbitrary subset $\mathcal{A} = \{p_1, \ldots, p_h\}$ of the set of conditions $\mathcal{P}$, $\mathcal{A}$ determines an abstract state space $S_{\mathcal{A}} = \{true, false\}^h$. The values of the conditions in $\mathcal{A}$ at a specific point in time determines an abstract state in $S_{\mathcal{A}}$. An abstract state u of $\mathcal{A}$ is represented as a vector $\langle v_1, \ldots, v_h \rangle$ where $v_i, 1 \leq i \leq h$, indicates the value of condition p_i.

Expressions and Abstract States:
Let (1) $\mathcal{A}$ and $\mathcal{B}$ be two subsets of the set of conditions $\mathcal{P}$, $\mathcal{A} \subseteq \mathcal{B}$, and (2) u be an abstract state in $S_{\mathcal{B}}$. $\alpha_u^{\mathcal{A}}$ denotes the expression $\{\langle p_i, v_i \rangle | p_i \in \mathcal{A}, v_i$ is the value of p_i in $u\}$.

$\alpha_u^{\mathcal{A}}$ is an expression that encodes the information about the values of the conditions in $\mathcal{A}$ in an abstract state u. In the following, we formally define the abstract event of a region.

Abstract Relation Δ_R:
Consider a region R where $\mathcal{A}$ is the set of abstract conditions of R, and $\mathcal{O}$ is the set ordering constraints. Δ_R is relation on $S_{\mathcal{A}} \times S_{\mathcal{A}}$ where $(u, v) \in \Delta_R$ if and only if there exists an event sequence $\mathbf{q}$ in $\mathbf{Q}_{\mathcal{O}}^R$ such that (1) u is the abstract state in $S_{\mathcal{A}}$ immediately before the events in R occur, (2) v is the abstract state in $S_{\mathcal{A}}$ immediately following $\mathbf{q}$, and (3) the regional subgoal $\mathcal{G}_R$ is achieved in v.

Abstract Events:
An abstract event e_R of a region R is associated with a set of causal rules where $\alpha_u^{\mathcal{A}} \rightarrow \alpha_v^{\mathcal{A}}$ is an associated rule if and only if (u, v) is in Δ_R.

The abstract relation Δ_R provides the information regarding the possible preconditions and their effects on the abstract conditions of R in order to achieve

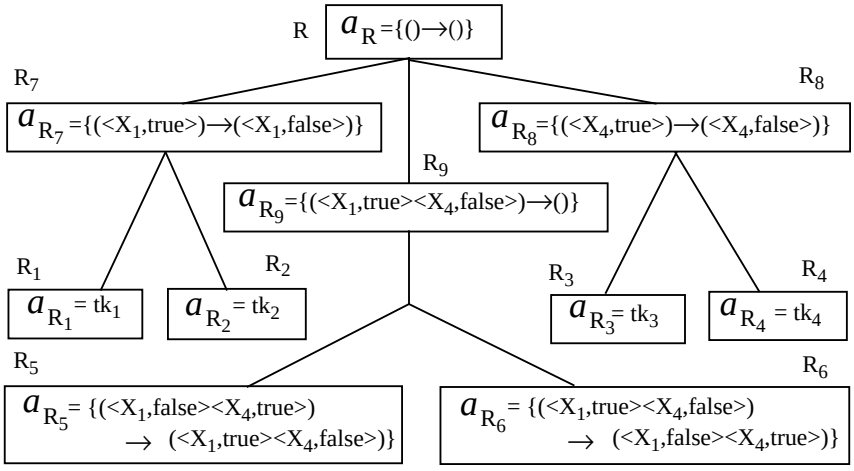

Figure 5.6: The abstract events of individual regions

the regional subgoals $\mathcal{G}_R$. The abstract event e_R compactly represents Δ_R by abstracting away the information in Δ_R about the local conditions of region R. The abstract event e_R is what we need to reason about the inter-regional interactions between the events in R and the events outside R. An algorithm to derive abstract events is provided in Section 7.

Figure 5.6 displays the abstract events of the regions regarding the jobshop example. At the top level, the global region R has no abstract conditions. Therefore $() \rightarrow ()$ is the only rule associated with the abstract event of the root region R. At the second level, it is not hard to see that X_1 and X_4 must be true immediately before the events in region R_7 and region R_8 respectively to achieve the regional subgoals that X_2 is false in R_7 and X_5 is false in R_8 respectively, and consequently X_1 and X_4 must be false immediately after region R_7 and region R_8 is finished respectively. This gives us the abstract events e_{R_7} and e_{R_8} described in Figure 5.6. In region R_9, (1) tk_6 must occur before tk_5, (2) X_8 must be false immediately after tk_6 occurs, and (2) X_7 must be false immediately after tk_5 occurs. Therefore, X_1 must be true and X_4 must be false immediately before tk_6 occur so that the first rule associated with tk_6 can be applied to make X_8 false. This also makes X_1 false and X_4 true immediately before tk_5 occurs. When tk_5 occurs following tk_6, it applies the first rule to make X_7 false and also restores X_1 to be true and X_4 to be false. This gives us the abstract

event $e_{R_9} = \{(\langle X_1, true\rangle\langle X_4, false\rangle) \rightarrow ()\}$. Similarly, we can derive the abstract events for the remaining regions in Figure 5.6.

5.4.3 Coupling Conditions and Localized Reasoning

In the following, we describe the *coupling conditions* of a region, which are the media for the inter-regional interactions among the child regions of R. Rather than considering the entire set of conditions $\mathcal{P}$, we only need to focus on the coupling conditions of a region R when reasoning about the events in R.

Coupling Conditions:
A condition X_i is a coupling condition of region R if and only if (1) region R is composed of two or more events and X_i is an abstract condition of a child region of R, *or* (2) region R is composed of a single event e and X_i is dependent on event e.

Figure 5.4 depicts the coupling conditions of the individual regions in the job-shop example. For example, the three child regions R_7, R_8, and R_9 of region R interact with one another through the coupling conditions X_1 and X_4 of region R_9.

The child regions of a region R are coupled together through the coupling conditions of R. This is because by definition the union of the conditions appearing in these abstract events is exactly the set of coupling conditions of R. On the other hand, to derive an abstract event of region R, we need to reason about the values of the abstract conditions and the subgoal conditions of R, and their union is again the set of coupling conditions according to the following theorem.

Theorem 5.8 *The set of coupling conditions of a region R is the union of two disjoint subsets: the set of abstract conditions of R and the set of subgoal conditions of R.*

5.5 Localized Temporal Projection Using Subgoals and Abstract Events

In this section, we reduce temporal projection into a sequence of local searches, in each of which we are concerned with the subgoal of a region R and the abstract events of the child regions of R.

5.5.1 Temporal Projection as Global Search

First of all, we describe how to formulate temporal projection as search. Given an instance of the temporal projection problem $I = \langle \mathcal{P}, \mathcal{E}, \mathcal{O}, s_0, \mathcal{G} \rangle$, the search graph is a directed graph $G_I = (V, A)$ that encodes the information of time, states, events in its vertices and arcs. Temporal projection is reduced to a graph reachability problem in the search graph.

Time and States:
The set of vertices V represents the possible system states at different phases of the system's evolution. Each vertex contains two components: (1) a system state, and (2) a record indicating whether the individual events in $\mathcal{E}$ have occurred or not.

Events:
The set of arcs A represents the occurrences of events at different phases of the system's evolution. We construct an arc (u, v) in A if (1) by applying a causal rule r, an event e can cause a state transition from the state encoded in vertex u to the state encoded in vertex v, (2) e can occur immediately after those events that are marked as having occurred in vertex u without violating the ordering constraints in $\mathcal{O}$, and (3) in vertex u, e is marked as having not occurred yet while in vertex v, e is marked as having occurred. In addition, we associate a *rule-tag* (e, r) with such an arc (u, v) to indicate that by applying rule r, event e can cause a state transition from the state encoded in vertex u to the state encoded in vertex v.

Root Vertex and Goal Vertices:
The root vertex u_0 is the vertex in which no events have occurred yet, and the state in u_0 is the initial state $\mathbf{s}_0$. A vertex t is a goal vertex if all events have occurred in vertex t and the state in t is a goal state.

Temporal Projection as Search: The task of temporal projection can be viewed as a search for a path from the root vertex u_0 to any goal vertex t. Since an arc in $G_I = (V, A)$ models the occurrence of an event, such a path corresponds to a possible event sequence immediately following which we reach a goal state. The rule-tags on the arcs tell us the rules applied by the individual events to enter the goal state.

The following procedure **Temporal-Search** provides the details of temporal projection as search. Our localized algorithm utilizes this procedure for local search.

Procedure Temporal-Search
input: an instance of temporal projection problem $I = \langle \mathcal{P}, \mathcal{E}, \mathcal{O}, \mathbf{s}_0, \mathcal{G} \rangle$.
output:
(1) a set F composed of the goal states that can be entered immediately following some event sequences in $\mathbf{Q}_{\mathcal{O}}^{\mathcal{E}}$;
(2) an event sequence $\mathbf{q}$ in $\mathbf{Q}_{\mathcal{O}}^{\mathcal{E}}$that ends in a goal state $\mathbf{f}$ in F;
(3) the rules applied by the individual events in $\mathbf{q}$ to enter the goal state $\mathbf{f}$;
(4) a trajectory $\mathbf{t_q}$ of $\mathbf{q}$ that ends in the goal state $\mathbf{f}$.

1. Partition the set of events $\mathcal{E}$ into m chains where the ith $(1 \leq i \leq m)$ chain is composed of l_i events that are totally ordered as a chain according to the ordering constraints in $\mathcal{O}$. Construct a directed acyclic graph $G_I = (V, E)$ in step 2 and step 3.

2. Construct the vertex set V of G_I:
   Each state $\mathbf{s}$ in $S_{\mathcal{P}}$ is mapped to $\prod_{1 \leq i \leq m}(1+l_i)$ nodes, each of which indicates a specific phase in the system's evolution. A vertex t in V is of the form

$(\mathbf{s}, x_1, x_2, \ldots, x_m)$. $\mathbf{s}$ ($\mathbf{s} \in S_{\mathcal{P}}$) indicates the corresponding state in vertex t. x_i ($0 \leq x_i \leq l_i$, $1 \leq i \leq m$) is a counter for the ith chain; x_i indicates that the first x_i events in the ith chain have occurred while the remaining events in the ith chain have not occurred yet. These vertices comprise the vertex set V of G_I.

3. Construct the arc set A of G_I:
   Add an arc from $(\mathbf{s}, x_1, x_2, \ldots, x_{k-1}, x_k - 1, x_{k+1}, \ldots, x_m)$ to $(\mathbf{t}, x_1, x_2, \ldots, x_{k-1}, x_k, x_{k+1}, \ldots, x_m)$ if (1) e is the x_kth event in the kth chain, (2) e can trigger a state transition from $\mathbf{s}$ to $\mathbf{t}$ by applying a causal rule r, and (3) e can occur immediately after the first $x_k - 1$ events in the kth chain and the first x_i events in the ith chain ($1 \leq i \leq m, i \neq k$) without violating the ordering constraints in $\mathcal{O}$. In addition, we associate a rule-tag (e, r) with this arc to indicate that event e can trigger such a state transition by applying causal rule r. These arcs comprise the arc set V of G_I, and model all possible situations regarding the occurrences of events.

4. In the following, we reduce the task of temporal projection to graph reachability. We can apply any standard graph-reachability algorithms to derive a solution.

   - Search the graph G_I. A goal state $\mathbf{t}$ can be reached immediate after an event sequence in $\mathbf{Q}^{\mathcal{E}}_{\mathcal{O}}$ if and only if the goal vertex $(\mathbf{t}, l_1, l_2, \ldots, l_m)$ is reachable from the root vertex $(\mathbf{s}, 0, 0, \ldots, 0)$ in graph G_I. Add such a state $\mathbf{t}$ to the set F.
   - Search for a directed path from the root vertex $(\mathbf{s}, 0, 0, \ldots, 0)$ to an arbitrary goal vertex $(\mathbf{f}, l_1, l_2, \ldots, l_m)$. The arcs in the directed path give us an event sequence $\mathbf{q}$ that ends in the goal state $\mathbf{f}$, and the rule-tags of the arcs tell us the rules applied by the individual events in $\mathbf{q}$ to enter f. The state components of the vertices in the path give us a trajectory of $\mathbf{q}$ which ends in $\mathbf{f}$.

Theorem 5.9 *The search graph $G_I = (V, E)$ constructed in procedure Temporal-Search is a directed acyclic graph. If we adopt a trivial partition that considers*

each individual event as a chain, the search graph G_I contains $2^{|\mathcal{P}|+|\mathcal{E}|}$ vertices. In this case, the procedure Temporal-Search runs in $O(2^{2(|\mathcal{P}|+|\mathcal{E}|)})$ time.

5.5.2 Temporal Projection as a Sequence of Local Searches

In the following, we present a localized temporal projection algorithm. In this algorithm, two procedures are systematically called region by region: (1) procedure Abstract-Event, which derives an abstract event of a region, and (2) procedure Generate-Sequence, which recursively generates an event sequence to achieve the regional subgoal of a region. Both procedures in turn call procedure Temporal-Search to conduct local search. For a region R, the following causal knowledge $\langle \mathcal{A}_R, \mathcal{U}_R, \mathcal{C}_R, \mathbf{s}_0, \tilde{\mathcal{G}}_R, \mathcal{X}_R, \mathcal{O}_R \rangle$ is provided as input to procedure Abstract-Event and procedure Generate-Sequence, where

- $\mathcal{A}_R$ is the set of abstract conditions of R;
- $\mathcal{U}_R$ is the set of subgoal conditions of R;
- $\mathcal{C}_R = \mathcal{A}_R \cup \mathcal{U}_R$ is the set of coupling conditions of R;
- $\mathbf{s}_0$ is the initial state of the problem instance;
- $\tilde{\mathcal{G}}_R$ is the incremental subgoal of R;
- if R contains more than one event, $\mathcal{X}_R$ is the set of abstract events $\mathcal{X}_R$ of R's child regions; otherwise $\mathcal{X}_R$ is composed of the single event in region R;
- $\mathcal{O}_R$ is the set of regional ordering constraints on the child regions of R.

Procedure Localized-Reasoning
input: a problem instance $\langle \mathcal{P}, \mathcal{E}, \mathcal{O}, \mathbf{s}_0, \mathcal{G} \rangle$.
output: Report an event sequence $\mathbf{q}$ in $\mathbf{Q}_{\mathcal{O}}^{\mathcal{E}}$ and the rules applied by the events in $\mathbf{q}$ such that we enter a goal state immediately following $\mathbf{q}$ if such an event sequence exists; otherwise, report that such a sequence does not exist.

1. Establish the following causal knowledge by scanning through the problem instance: (i) the regions, the region hierarchy, and the regional ordering constraints, (ii) the local conditions, the subgoal conditions, the abstract conditions and the coupling conditions of individual regions, (iii) the regional subgoals and incremental subgoals of individual regions.

2. Starting from the bottom level of the region hierarchy, call procedure Abstract-Event as described in Step 3 for each region at the same level; proceed in the same way level by level until the root region at the top level is done.

3. For each region R, call procedure Abstract-Event to determine the possibility of achieving the regional subgoal of R. If the regional subgoal cannot be achieved, abort the computation and report the infeasibility of achieving the goal $\mathcal{G}$; otherwise, derive and propagate the abstract event e_R to R's parent region.

4. Call procedure Generate-Sequence with a pair of empty abstract states $(\langle\rangle, \langle\rangle)$ and the causal knowledge $\langle \mathcal{A}_\mathcal{E}, \mathcal{U}_\mathcal{E}, \mathcal{C}_\mathcal{E}, \mathbf{s}_0, \tilde{\mathcal{G}}_\mathcal{E}, \mathcal{X}_\mathcal{E}, \mathcal{O}_\mathcal{E} \rangle$ associated with the root region $\mathcal{E}$ as input. Procedure Generate-Sequence recursively generates an event sequence $\mathbf{q}$ and the rules applied by the events in $\mathbf{q}$ to achieve the goal $\mathcal{G}$.

Procedure Abstract-Event

Input: the causal knowledge $\langle \mathcal{A}_R, \mathcal{U}_R, \mathcal{C}_R, \mathbf{s}_0, \tilde{\mathcal{G}}_R, \mathcal{X}_R, \mathcal{O}_R \rangle$ associated with region R.

Output: if the incremental subgoal of R can be achieved, report the abstract event e_R; otherwise, stop and report the infeasibility of achieving the regional subgoal.

1. Determine the set S of the possible abstract states in $S_{\mathcal{C}_R}$ immediately before the events in R occur, where the values of the subgoal conditions in $\mathcal{U}_R$ must be the same as their initial values in $\mathbf{s}_0$.

2. For each abstract state s in S, construct a new instance of the temporal projection problem $\langle \mathcal{C}_R, \mathcal{X}_R, \mathcal{O}_R, \mathbf{s}, \tilde{\mathcal{G}}_R \rangle$ regarding the set of coupling conditions $\mathcal{C}_R$, the set of abstract events $\mathcal{X}_R$, the set of regional ordering constraints $\mathcal{O}_R$, the possible abstract state $\mathbf{s}$ immediately before the events in R occur, and the incremental subgoal $\tilde{\mathcal{G}}_R$.

3. For each new instance $\langle \mathcal{C}_R, \mathcal{X}_R, \mathcal{O}_R, s, \tilde{\mathcal{G}}_R \rangle$, call procedure Temporal-Search to derive F, the set of reachable goal states in $S_{\mathcal{C}_R}$ regarding this new instance. Add the pairs $(\mathbf{s}, \mathbf{f})$ for each $\mathbf{f}$ in F into Δ_R.

4. If the set Δ_R is empty, we report the infeasibility of achieving the incremental subgoal. Otherwise, compile Δ_R into an abstract event e_R by associating a causal rule $\alpha_{\mathbf{s}}^{\mathcal{A}_R} \rightarrow \alpha_f^{\mathcal{A}_R}$ with e_R for each pair of abstract states $(\mathbf{s}, \mathbf{f})$ in Δ_R.

Procedure Generate-Sequence

Input:

(1) the causal knowledge $\langle \mathcal{A}_R, \mathcal{U}_R, \mathcal{C}_R, \mathbf{s}_0, \tilde{\mathcal{G}}_R, \mathcal{X}_R, \mathcal{O}_R \rangle$ associated with region R; (2) a pair of abstract states $(\mathbf{s}', \mathbf{f}')$ in $S_{\mathcal{A}_R}$ where $\mathbf{s}'$ and $\mathbf{f}'$ are the abstract states immediately before and after the events in R occur as a group respectively.

Output: an event sequence $\mathbf{q}$ in $\mathbf{Q}_{\mathcal{O}}^R$ and the rules applied by the events such that if the abstract state $\mathbf{s}'$ is true immediately before $\mathbf{q}$, immediately following $\mathbf{q}$ the abstract state is changed to $\mathbf{f}'$ and the regional subgoal $\mathcal{G}_R$ is true.

1. Determine the abstract states s in $S_{\mathcal{C}_R}$ where (i) the values of the conditions in $\mathcal{A}_R$ in $\mathbf{s}$ are the same as those in abstract state $\mathbf{s}'$, and (ii) the values of the conditions in $\mathcal{U}_R$ in $\mathbf{s}$ are the same as their initial values in $\mathbf{s}_0$.

2. Construct a new instance of the temporal projection problem $\langle \mathcal{C}_R, \mathcal{X}_R, \mathcal{O}_R, s, \tilde{\mathcal{G}}_R \rangle$ regarding the set of coupling conditions $\mathcal{C}_R$, the set of abstract events $\mathcal{X}_R$, the set of regional ordering constraints $\mathcal{O}_R$, the possible abstract state $\mathbf{s}$ immediately before the events in R occur, and the incremental subgoal $\tilde{\mathcal{G}}_R$.

3. For the new instance $\langle \mathcal{C}_R, \mathcal{X}_R, \mathcal{O}_R, \mathbf{s}, \tilde{\mathcal{G}}_R \rangle$, call procedure Temporal-Search to derive an event sequence $\mathbf{q}'$ in $\mathbf{Q}_{\mathcal{O}_R}^{X_R}$ that ends in a reachable goal state $\mathbf{f}$ where (i) the incremental subgoal $\tilde{\mathcal{G}}_R$ is true in $\mathbf{f}$, and (ii) the expression $\alpha_{\mathbf{f}'}^{\mathcal{A}_R}$ is true in $\mathbf{f}$.

4. Also use procedure Temporal-Search to determine (i) a trajectory $\mathbf{t}_{\mathbf{q}'}$ of $\mathbf{q}'$ that starts in $\mathbf{s}$ and ends in $\mathbf{f}$, and (ii) the rules applied by the events in $\mathbf{q}'$ that results in the trajectory $\mathbf{t}_{\mathbf{q}'}$. The ordering of the abstract events in $\mathbf{q}'$ determines a total order on the child regions of R.

5. For each child region R' of R, according to the trajectory $\mathbf{t}_{\mathbf{q}'}$ of $\mathbf{q}'$ derived in Step 3 , determine the abstract states $\tilde{\mathbf{s}}$ and $\tilde{\mathbf{f}}$ in $S_{\mathcal{A}_{R'}}$ immediately before and after the abstract event of R' in $\mathbf{q}'$ occurs respectively.

6. For each child region R' of R, (i) prepare the pair of abstract states $(\tilde{\mathbf{s}}, \tilde{\mathbf{f}})$ derived in Step 5 and the causal knowledge $\langle \mathcal{A}_{R'}, \mathcal{U}_{R'}, \mathcal{C}_{R'}, \mathbf{s}_0, \tilde{\mathcal{G}}_{R'}, \mathcal{X}_{R'}, \mathcal{O}_{R'} \rangle$ associated with R' as input, and (ii) call procedure Generate-Sequence to derive an event sequence over R'. Concatenate these event sequences over the child regions of R' into an event sequence $\mathbf{q}$ according to the total order on R's child regions derived in Step 4.

7. For each event e, determine the rule applied by event e according to the rule selected in Step 4 when procedure Generate-Sequence is recursively called to handle the region $\{e\}$.

Theorem 5.10 *Procedure Localized-Reasoning is sound and complete.*

We illustrate the use of the localized algorithm in solving the jobshop example. First, we derive the following knowledge by scanning through the problem instance: (1) the region hierarchy as depicted in Figure 5.4; (2) the local conditions, the subgoal conditions, the abstract conditions and the coupling conditions of the individual regions as depicted in Figure 5.4; and (3) the regional subgoals and incremental subgoals of individual regions as depicted Figure 5.5.

Second, starting from the bottom level of the region hierarchy, we call procedure Abstract-Event to derive the abstract events of regions as depicted in Figure 5.6. We proceed level by level until we reach the root region W at the top level. Since we can successfully derive the abstract event for every region, there exists an event sequence $\mathbf{q}$ in $\mathbf{Q}_{\mathcal{O}}^{\mathcal{E}}$to achieve the goal.

Finally, we call procedure Generate-Sequence to recursively derive an event sequence $\mathbf{q}$ in $\mathbf{Q}_{\mathcal{O}}^{\mathcal{E}}$to achieve the goal. The procedure Generate-Sequence recursively generates the following orderings $\langle R_8, R_9, R_7\rangle$, $\langle\{tk_3\},\{tk_4\}\rangle$, $\langle\{tk_6\},\{tk_5\}\rangle$, and $\langle\{tk_1\},\{tk_2\}\rangle$ for the child regions of the regions R, R_8, R_9, and R_7 respectively. This gives us an event sequence $\mathbf{q} = \langle tk_3, tk_4, tk_6, tk_5, tk_1, tk_2\rangle$ as an event sequence in $\mathbf{Q}_{\mathcal{O}}^{\mathcal{E}}$to achieve the goal. The causal rules applied by the events in $\mathbf{q}$ as a sequence are the first rule, the second rule, the first rule, the first rule, the first rule, and the second rule of the events respectively.

5.6 Quantitative Analysis of Computational Efficiency

We first define the *interaction measure* of a problem instance, which is used to quantify the computational efficiency of procedure Localized-Reasoning in the problem instance.

Measure of Causal Interactions:
Given the region hierarchy determined by an instance of the temporal projection problem, we define the branching factor b_R of a region R to be the number of child regions of R, the coupling factor c_R of a region R to be the number of coupling conditions of R, and the interaction measure $\mathcal{L}$ of the problem instance to be $\max_R (b_R + c_R)$.

The interaction measure $\mathcal{L}$ indicates the magnitude of the interactions among regions. A large interaction measure $\mathcal{L}$ indicates the existence of regions with many child regions or large sets of coupling conditions, which implies complicated

interactions among regions. Instead, a small interaction measure $\mathcal{L}$ indicates that the causal interactions are well structured and localized in individual regions. In other words, *small* interaction measure $\mathcal{L}$ implies *strong* locality in the problem instance.

Theorem 5.11 *Given an instance of the temporal projection problem, procedure Localized-Reasoning runs in $O(n \times 8^{\mathcal{L}})$ time, where n is the size of the event set $\mathcal{E}$, and $\mathcal{L}$ is the interaction measure of the problem instance.*

By Theorem 5.11, the worst-case performance degrades to be exponential in n when the branching factor or the coupling factor is of $O(n)$ size, where n is the size of the event set $\mathcal{E}$. This is as expected, because (1) a set of totally unordered events $\mathcal{E}$ corresponds to a single region containing the individual events as child regions, where the branching factor is of $O(n)$ size, and (2) temporal projection regarding totally unordered events is NP-complete even in very restricted cases as shown in Theorem 5.2.

On the other hand, our localized algorithm can exploit the locality inherent in problem instances to bring about computational efficiency. When there is strong locality in a problem instance, the interaction measure $\mathcal{L}$ is small. In this situation, our localized reasoning algorithm can substantially expedite temporal projection. The following two corollaries of Theorem 5.11 demonstrate the computational savings gained by exploiting locality.

Corollary 5.3 *When the interaction measure $\mathcal{L}$ is of $O(1)$ magnitude, the temporal projection problem can be solved by procedure Localized-Reasoning in $O(n)$ time, where n is the size of the event set $\mathcal{E}$.*

Corollary 5.4 Corollary 5.4. *When the interaction measure is of $O(\log n)$ magnitude, the temporal projection problem can be solved by procedure Localized-Reasoning in time polynomial in n, where n is the size of the event set $\mathcal{E}$.*

Consider a situation in which (1) every region except for the regions containing only a single event has $\tilde{b}$ child regions, and (2) for each region the child regions of the region are totally unordered.

- There are $\tilde{b}!^{(n-1)/(\tilde{b}-1)}$ possible total orders on events, where n is the size of the event set $\mathcal{E}$. It is infeasible to exhaustively examine this super-polynomial number of possible total orders one by one for a solution.

- However, procedure Localized-Reasoning can derive a solution in linear time if (1) $\tilde{b} = O(1)$ and (2) every region has only $O(1)$ number of conditions that are dependent on both the events in and the events not in the region. The interaction measure $\mathcal{L}$ in this case is of $O(1)$ magnitude. This is because for each region R (1) $b_R = \tilde{b} = O(1)$ and (2) the coupling factor c_R is no more than the sum over the $O(1)$ numbers of the abstract conditions of R's child regions according to Theorem 5.8.

- Similarly, procedure Localized-Reasoning can derive a solution in polynomial time if (1) $\tilde{b} = O(\log n)$ and (2) every region has only $O(\log n)$ number of conditions that are dependent on both the events in and the events not in the region. The interaction measure $\mathcal{L}$ in this case is of $O(\log n)$ magnitude. In both cases, procedure Localized-Reasoning provides substantial computational savings.

5.7 Related Work

The related work on the representation of time and temporal reasoning is extensive. There are several formalisms that can represent the temporal relationship among a set of points or intervals in time [All83] [Vil82] [MB83] [Dea84]. Reasoning about the consequence of events using the logic-based event calculus can be found in [KS86] [Chi94]. The idea of structuring ordering and causal knowledge to speed up temporal reasoning is also present in maintaining knowledge about temporal intervals [All83] [GS93] [KG77] and in the event calculus [Eva90] [Mea92].

The use of subgoals and abstract events in Chaprer 4 is inspired by the similar notions of subgoals and abstraction used in search [Kor87] and in abstract and hierarchical planning [Sac74a] [Chr90] [Kno91a] [YT90] in reducing the overall search effort. The idea of using localized reasoning to exploit locality is also studied by Lansky in GEM [Lan88] in event-based planning. In GEM, locality is similarly modeled by sets of interrelated events delimited by temporal logic constraints.

5.8 Summary

In this chapter, we formally describe a dynamical system modeled by a set of conditions, a set of events, and a set of ordering constraints on the events. The system state at a point in time is determined by the values of a finite set of conditions. We represent an event as a disjunction of STRIPS-like operators. The system evolves through a sequence of state transitions as the events occur and change the values of conditions. We are concerned with deriving an event sequence consistent with the ordering constraints to achieve a goal. We identify the factors affecting computational complexity and demonstrate the complexity trade-offs involving these factors. Our complexity results indicate that we have very limited ability to consider a large number of conditions at a time. To expedite temporal projection, we investigate locality in event ordering and causal dependency. Locality allows us to focus on many smaller numbers of conditions separately rather than considering the entire set of conditions at a time. The locality in event ordering in a particular problem instance is modeled by a hierarchy of regions. A region is a collection of closely related events which occur as an atomic group. For each region, we can identify four characteristic condition subsets, which reflect locality in causal dependencies. We develop a localized temporal reasoning algorithm that determines a solution through a sequence of local searches. The computational complexity of our algorithm is determined by the locality of a problem instance. Our theoretical results demonstrate the substantial performance improvement gained by exploiting locality. This work provides

solid evidence of the usefulness of localized reasoning in exploiting locality.

Chapter 6

Decomposition Techniques for the Markov Decision Problem (I)

Many goal-oriented planning problems can be modeled as Markov decision processes with absorbing states. Goals specify target states as absorbing states and the other states are transient states. Once reaching a target state, a process stays in that state forever without cost [Der70] [KK71b] [KC74b]. At times the underlying state space can be symbolically decomposed into regions that have an acyclic interconnection. We refer to these regions as *coherent fragments*. A Markov decision process will never end in a coherent fragment again after leaving the fragment.

In this chapter, we develop a decomposition algorithm to exploit acyclic interconnection among coherent fragments. Although every variable is relevant in describing the action dynamics, usually only a subset of the state variables is relevant to the action dynamics within a coherent fragment and the subsequent trajectories after leaving that coherent fragment. In a coherent fragment, two states are undistinguishable if they have the same configuration governing the values of the relevant variables. Significant dimensionality reduction can be attained by aggregating undistinguishable states in coherent fragments. This allows

us to construct an equivalent but smaller Markov decision process for computing an optimal policy.

For Markov decision processes compactly encoded as sequential decision networks, the strongly connected components of the networks are natural coherent fragments. We empirically investigate dimensionality reduction gained by this decomposition algorithm when solving Markov decision problems encoded as sequential decision networks. We employ three probabilistic models to generate problem instances with parameters reflecting different levels of locality and dependency. We then evaluate the effectiveness of this decomposition algorithm for problem instances generated under different settings of the parameters.

6.1 Coherent Fragment and Locality

In this section, we formally define the coherent fragments in a Markov decision process given a decomposition of the underlying state space into regions.

Interconnection among Regions:
We view a Markov decision process given a decomposition of the underlying state space into regions as a directed graph $H = (N_H, E_H)$ in the following way :

- each node in N_H uniquely represents a region;
- arcs in E_H represent the interconnection among regions where a region R is connected to a region R' iff there exist (1) a state $\mathbf{u}$ in R, (2) a state $\mathbf{v}$ in R', and (3) an action a such that taking action a in state $\mathbf{u}$ can ends in state $\mathbf{v}$ with nonzero probability.

Coherent Fragments:
Given a Markov decision process and a decomposition of the underlying state space into regions represented as a directed graph $H = (N_H, E_H)$, a *coherent fragment* F is a subset of the regions that compose a strongly connected component in H.

The Directly Relevent Variables in a Coherent Fragment:
A variable X_i is directly relevant in a coherent fragment F if and only if there exists an action a applicable in F such that X_i is directly relevant in encoding action a. We use V_F to denote the set of directly relevant variables in a coherent fragment F.

Interconnection among Coherent Fragments:
Given two coherent fragments F and G, we say that G is a child of F if and only if at least one of the regions in F is connected to a region in G. Naturally, we can also define descendant fragments, parent fragments, and ancestor fragments in a similar way. For a coherent fragment F, Children(F), Descendants(F), Parents(F), and Ancestors(F) denote the sets of the coherent fragments that are the children, the descendants, the parents, and the ancestors of F respectively.

Fragment Graphs:
A fragment graph displays the interconnection among a set of coherent fragments. Given a Markov decision process and a decomposition of the underlying state space into regions represented as a directed graph $H = (N_H, E_H)$, the fragment graph $(\mathcal{F}, E_\mathcal{F})$ of H is a directed graph where (1) $\mathcal{F}$ is composed of all coherent fragments in H, and (2) (F, F') is a directed edge in $E_\mathcal{F}$ if and only if F' is a child fragment of F.

Example:
For the production-control decision network in Figure 3.3, we can aggregate the states with the same decision step into a region. In other words, a decision step symbolically represents a region. Figure 6.1 depicts the coherent fragments, the directly relevant variables in coherent fragments, and the resulting fragment graph.

Property 6.1 *Given a Markov decision process and a decomposition of the underlying state space into regions, the resulting fragment graph is acyclic. Any*

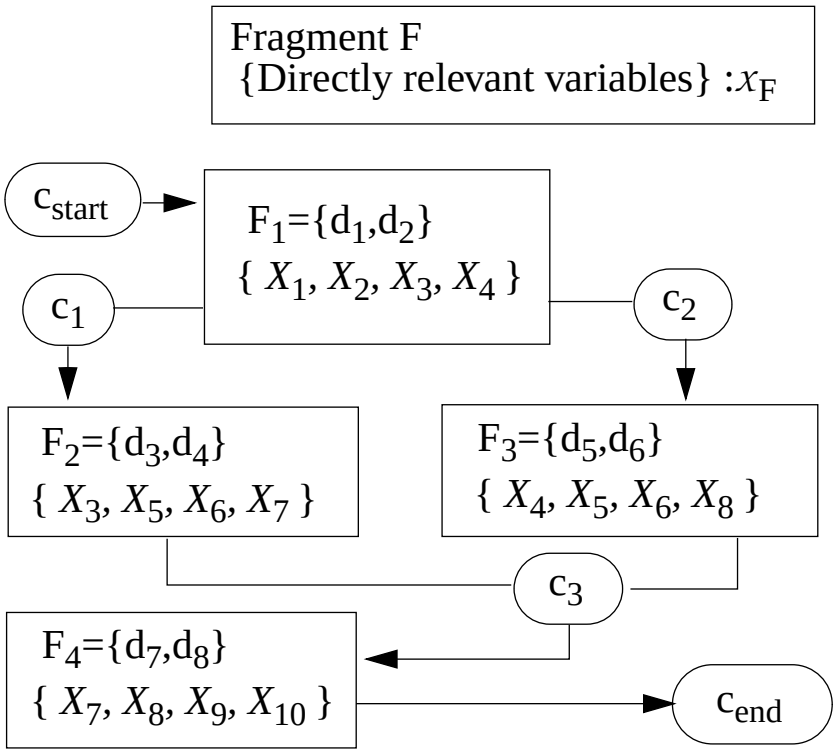

Figure 6.1: The fragment graph of the production-control decision network

trajectory of the process can be divided into blocks such that (1) each block is uniquely associated with a coherent fragment with no two blocks associated with the same coherent fragment, and (2) each block is composed of some states in the associated coherent fragment.

Property 6.1 directly follows from the definitions of coherent fragments and fragment graphs. This is because as soon as the process leaves a state in a coherent fragment F to enter a state in another coherent fragment, the process will never enter any states in F again. Often coherent fragments are loosely coupled with one another since only a few state variables are directly relevant in a coherent fragment, and a coherent fragment interacts with its child fragments only through a few state variables. This allows us to conveniently decouple a Markov decsion problem into subproblems governing coherent fragments.

Example:
Consider the production-control decision network in Figure 3.3. There are four coherent fragments, each of which fullfills a specific requirements on the final values of the components.

First, we (1) start with control node C_{start}, (2) enter coherent fragment F_1 through C_{start}, and (3) alternate between decision steps d_1 and d_2 to improve the values of at least one of X_1 and X_2 to one. As soon as this requirement is satisfied, we enter fragment F_2 or fragment F_3. Note that variables X_1 and X_2 become irrelevant after leaving fragment F_1.

If the value of X_4 has been improved to one when leaving fragment F_1, we (1) enter coherent fragment F_2 through control node C_1 and (2) alternate between steps d_3 and d_4 to improve the values of at least two of X_3, X_5 and X_6 to one. Otherwise, we (1) enter coherent fragment F_3 through control node C_2 and (2) alternate between steps d_5 and d_6 to improve the values of all of X_4, X_5 and X_6 to one. In either case, we leave fragment F_2 or fragment F_3 to enter fragment F_4 with at least three of the four variables X_3, X_4, X_5 and X_6 are improved to one. Note that X_4, X_5, and X_6 become irrelevant immediately after entering fragment F_4.

Finally, we (1) enter coherent fragment F_4 through control node C_3, (2) alternate between steps d_7 and d_8 to improve the values of at least three of X_7, X_8, X_9 and X_{10} to one, and (3) then finish the entire execution in control node C_{end}.

6.2 Dependency among Coherent Fragments

It is rare that every variable is directly relevant in every coherent fragment. More often, a variable is only directly relevant in a few coherent fragments interact with some other variables that are in the other fragments. In the following, we analyze dependency among coherent fragments by using several characteristic subsets of state variables.

The Active Variables in a Fragment:
A variable X_i is active in a coherent fragment F if and only if (1) X_i is directly relevant in fragment F, *or* (2) X_i is directly relevant in an ancestor fragment of F and also in a descendant fragment of F. A variable X_i is inactive in F if X_i is not active in F. We denote the set of active variables in a coherent fragment F as A_F.

The Latent Variables in a Fragment:
A variable X_j is *latent* in a coherent fragment F if and only if (1) X_j is directly relevant in at least one descendant fragment of F and (2) X_j is neither directly relevant in F nor in any ancestor fragment of F.

The Passive Variables in a Fragment:
A variable X_j is *passive* in a coherent fragment F if and only if X_j is neither directly relevant in F nor in F's descendant fragments.

Property 6.2 *If variable X_j is not active in a coherent fragment F, X_j is either latent or passive in F. If X_j is passive in F, the value of X_j will never change at any time after the process enters fragment F, If X_j is latent in F, the value of X_j remains unchanged after the process enters fragment F until the process*

enters a descendant fragment of F in which X_j is active.

The Terminal Variables between Two Fragments:
Given a coherent fragment F and a child coherent fragment G of F, a variable X_i is a terminal variable from F to G if and only if (1) X_i is active in F, and (2) X_i is passive in G. We denote the set of terminal variables from fragment F to fragment G as T_{FG}. Note that the variables in T_{FG} become irrelevant immediately after the process enters fragment G from fragment F.

The Starting Variables between Two Fragments:
Given a coherent fragment F and a child coherent fragment G of F, a variable X_i is a starting variable from F to G if and only if (1) X_i is latent in F, and (2) X_i is active in G. We denote the set of starting variables from fragment F to fragment G as S_{FG}. Note that the variables in S_{FG} become relevant immediately after the process enters fragment G from fragment F.

Property 6.3 *Given a coherent fragment F and a child coherent fragment G of F, we have the following equation :*

$$A_G = A_F - T_{FG} \cup S_{FG}.$$

The Coupling Variables between Two Fragments:
Given a coherent fragment F and a child coherent fragment G of F, a variable X_i is a coupling variable between F and G if and only if X_i is active in both F and G. We denote the set of coupling variables between fragment F and fragment G as C_{FG}. The set of coupling variables C_{FG} is the media for the interaction between two adjacent coherent fragments.

Property 6.4 *Given a coherent fragment F and a child coherent fragment G of F, we have the following equation :*

$$A_F - T_{FG} = C_{FG} = A_G - S_{FG}.$$

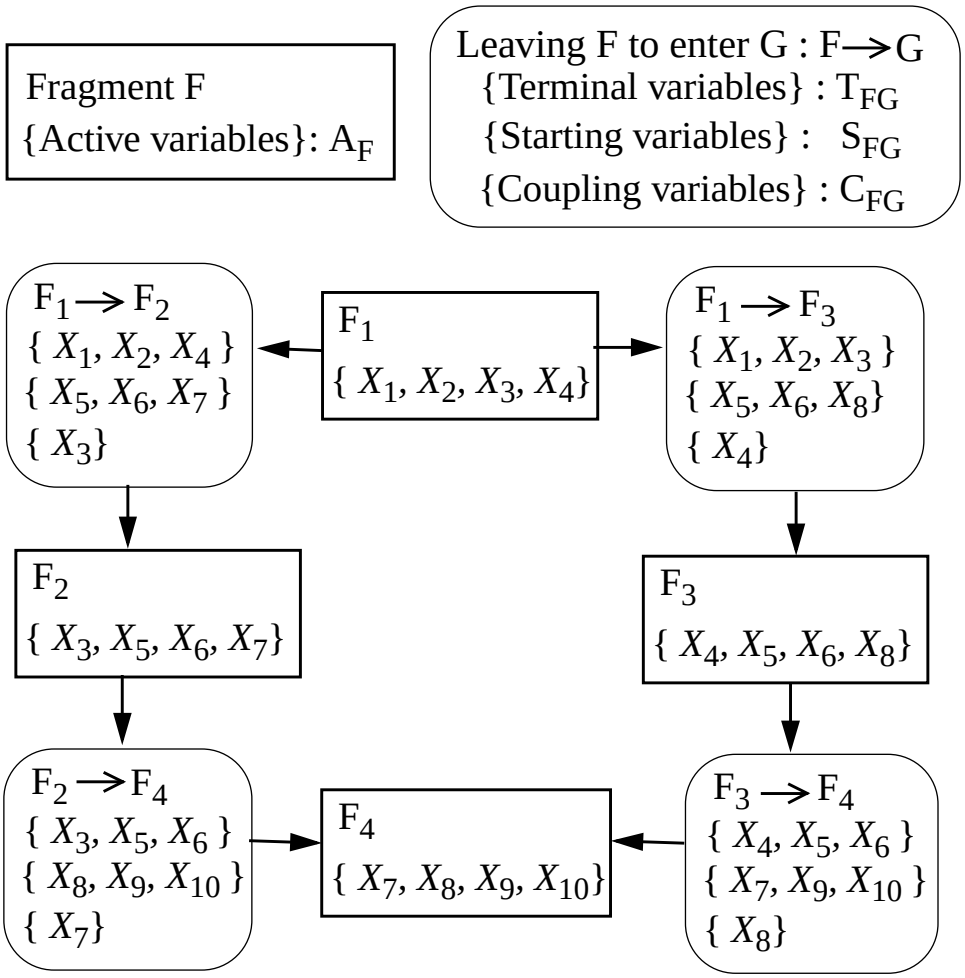

Figure 6.2: Dependencies among coherent fragments

Figure 6.2 displays characteristic subsets of state variables for the production-control decision network in Figure 3.3.

Consider the production-control decision network in Figure 3.3. Variables X_1 and X_2 are directly relevant only in fragment F_1, but indirectly affect the dynamics in fragments F_2 and F_3 through variables X_3 and X_4. This is because X_1 and X_2 directly interact with X_3 and X_4 in fragment F_1 while X_3 and X_4 are directly relevant in fragments F_2 and F_3.

6.3 A Decomposition Technique to Exploit Coherent Fragments

Given a Markov decision process $\mathcal{M}$, an initial state, and a decomposition of the underlying state space into regions, the following procedure analyzes the dependency structure among coherent fragments, constructs a reduced process $\mathcal{M}'$, and derives an optimal policy for $\mathcal{M}'$:

Procedure Reduced-MDP

input:

(1) a Markov decision process $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \Pr, \text{Cost} \rangle$;

(2) an initial state $\mathbf{u}_0, \mathbf{u}_0 \in \mathcal{S}$;

(3) a partition of the state space $\mathcal{S}$ into disjoint regions defined by a set of boolean predicates governing the values of the state variables.

output:

(1) a reduced Markov decision process $\mathcal{M}' = \langle \mathcal{S}', \mathcal{A}, \tilde{\Pr}, \tilde{\text{Cost}} \rangle$;

(2) an optimal policy π' for the reduced process $\mathcal{M}'$.

- Determine the interconnection among regions and derive the strongly connected components of regions as coherent fragments. Determine the interconnection among the coherent fragments. For each coherent fragment F,

1. determine the actions applicable in F and the directly relevant variables in F;
2. determine the set of active variables A_F as the intersection of (i) the union of the directly relevant variables of F and F's ancestor fragments and also (ii) the union of the directly relevant variables of F and F's descendant fragments;
3. determine the other characteristic subsets of variables using the information of the fragment graph and the subsets of active variables in individual coherent fragments according to the definitions in the last section.

- For each coherent fragment F, aggregate the states in F according to the values of the active variables of F to form a set of aggregate states SubSpace(F), SubSpace(F) $= \prod_{X_i \in A_F} \Omega_{X_i}$. An aggregate state $\mathbf{u}_F = \langle x_F^1, x_F^2, \ldots \rangle$ in SubSpace(F) represents as a whole the states in fragment F in which (i) x_F^i is the value of the active variable X_F^i in coherent fragment F and (ii) the values of the latent variables in fragment F are consistent with the corresponding values in the initial state $\mathbf{u}_0$ [1].

- Return the following reduced Markov decision process $\mathcal{M}' = \langle \mathcal{S}', \mathcal{A}, \tilde{\Pr}, \tilde{\text{Cost}} \rangle$ that respects the state-aggregation step above:
  - The state space $\mathcal{S}'$, $\mathcal{S}' = \bigcup_F \text{SubSpace}(F)$, is the union of the aggregate states in individual coherent fragments while the action space remains unchanged.
  - Within each fragment F, $\tilde{\Pr}$ and $\tilde{\text{Cost}}$ compactly replicate the original probability Pr and cost function Cost : for every pair of aggregate states $\mathbf{u}_F$ and $\mathbf{v}_F$ in SubSpace(F) $\tilde{p}_{\mathbf{u}_F \mathbf{v}_F}(a)$ equals $p_{\mathbf{uv}}(a)$ and $\tilde{c}_{\mathbf{u}_F \mathbf{v}_F}(a)$ equals $c_{\mathbf{uv}}(a)$ where (i) $\mathbf{u}$ is any state in F aggregated into $\mathbf{u}_F$, (ii) a is any action application in $\mathbf{u}$, and (iii) $\mathbf{v}$ in F is aggregated into $\mathbf{v}_F$ and

[1] By Property 6.2, those states in fragment F violating this requirement are not reachable from the initial state at all and are thus ignored.

also has the same configuration governing the values of the inactive variables of F as $\mathbf{u}$ does [2].

- Between a coherent fragment F and a descendant fragment G of F, $\tilde{\text{Pr}}$ and $\tilde{\text{Cost}}$ are direct extensions from Pr and Cost respecting the state-aggregation step : for every pair of aggregate states $\mathbf{u}_F$ and $\mathbf{v}_G$ ($\mathbf{u}_F \in \text{SubSpace}(F)$, $\mathbf{v}_G \in \text{SubSpace}(G)$) $\tilde{p}_{\mathbf{u}_F\mathbf{v}_G}$ equals $\sum_{\mathbf{v}} p_{\mathbf{uv}}\ \tilde{c}_{\mathbf{u}_F\mathbf{v}_G}$ equals $\sum_{\mathbf{v}} p_{\mathbf{uv}} c_{\mathbf{uv}}$ where (i) $\mathbf{u}$ is any state in F aggregated into $\mathbf{u}_F$, (ii) a is any action application in $\mathbf{u}$, and (iii) the two summations are over every state $\mathbf{v}$ in G that is aggregated into $\mathbf{v}_G$ and has the same configuration governing the values of the inactive variables of F as $\mathbf{u}$ does [3].
- As defined above, the new transition probability function $\tilde{\text{Pr}}$ and the new action cost function $\tilde{\text{Cost}}$ can be directly determined by applying the probabilistic operator representing the actions in $\mathcal{A}$ to the aggregate states in $\mathcal{S}'$.

- Determine an optimal policy π' for the reduced process $\mathcal{M}'$ fragment by fragment until all coherent fragments are processed. For each coherent fragment F without any descendant coherent fragments or all of whose descendant coherent fragments have been processed,
  - construct a local Markov decision process MDP(F) whose state space is the union of SubSpace(F) and Target(F), where Target(F) denotes the set of aggregate states outside F but reachable from SubSpace(F) in one state transition;
  - the transition probability fuction and action cost function of MDP(F) are basically the same as $\tilde{\text{Pr}}$ and $\tilde{\text{Cost}}$, except that each aggregate

[2] $\tilde{\text{Pr}}$ and $\tilde{\text{Cost}}$ are well defined for any pair of states $\mathbf{u}$ and $\mathbf{v}$ satisfying the requirements specified above. This is because the values of the inactive variables in state $\mathbf{u}$ remain unchanged after taking action a and are irrelevant to the transition probabilty and cost, which is a fact directly follows from the definition of active variables and Property 6.2.

[3] Again $\tilde{\text{Pr}}$ and $\tilde{\text{Cost}}$ are well defined for any state $\mathbf{u}$ aggregated into $\mathbf{u}_F$ since the values of the inactive variables in state $\mathbf{u}$ remain unchanged after taking action a and are irrelevant to the transition probabilty and cost.

state in Target(F) is considered as an absorbing state and is associated with an additional cost modeling the expected cumulative cost from entering that aggregate state for the first time until reaching a target state in the original process.

- determine an optimal policy for MDP(F) and also determine the expected cumulative cost for each aggregate state in SubSpace(F).

The union of the optimal policies over all local Markov decision processes then provides an optimal policy for the reduced process $\mathcal{M}'$.

Property 6.5 *The size of the reduced Markov decision process $\mathcal{M}'$ is bounded by $f \times c^r$, where (1) f is the total number of coherent fragments (2) r is the maximum of the numbers of active variables in coherent fragments, and (3) c is the maximum of the numbers of possible values for variables.*

Theorem 6.1 *Given an optimal policy π' for the reduced Markov decision process $\mathcal{M}'$ derived from a Markov decision process $\mathcal{M}$ by procedure* **Reduced-MDP**, *π' compactly encodes an optimal policy π for $\mathcal{M}$ with respect to a given initial state, where $\pi(\mathbf{u})$ equals $\pi'(\mathbf{u}_F)$ for every state $\mathbf{u}$ that is aggregated into an aggregate state $\mathbf{u}_F$ in a coherent fragment F.*

6.4 Efficacy in Optimizing Sequential Decision Networks

In this section, we investigate the application of Procedure **Reduced-MDP** in optimizing sequential decision networks, *i.e.*,, generating optimal policies for the underlying Markov decision processes. We employ three probabilistic models to generate sequential decision networks and then evaluate the effectiveness of Procedure **Reduced-MDP**.

6.4.1 Decomposing Sequential Decision Networks

For Markov decision processes encoded as sequential decision networks, one of the state variables X_{stage} encodes which decision stage is the current decision stage. Naturally, we can decompose such a state space into regions with each region corresponding to the states associated a specific decision stage. In other words, each region is symbolically specified by a predicate $P(X_{stage}) : X_{stage} = d_i$ for a decision stage d_i. This is the partitioning strategy implicitly assumed through out this section. Assuming the existence of actions that allow transition from every decision stage to its child decision stages, the resulting coherent fragments are simply the strongly connected components of the sequential decision network. Directly following from Property 6.5, we have the following property:

Property 6.6 *With regions defined by decision stages, Procedure* **Reduced-MDP** *can determine an optimal policy for a sequential decision network with m decision stages in $O(m^c)$ time for some constant c if every strongly connected component of the network involves only $O(\log m)$ number of active variables.*

Example:

Figure 3.3 depicts the active variables in each coherent fragment F of the production-control decision network. Since each coherent fragment is composed of two decision stages and four active variables other than the variable encoding the current decision stage, an optimal policy can be determined by solving four local Markov decision processes. Consequently the reduced Markov decision process has $4 \times 2 \times 2^4 = 128$ aggregate states. This is much smaller than the entire state space, which is of size $2^{10} \times 8 = 8192$ since we have ten binary state variables plus the variable encoding the eight possible decision stages.

6.4.2 Empirical Effectiveness in Dimensionality Reduction

A Probabilistic Model for Sequential Decision Networks:
In the following, we empirically evaluate the effectiveness of Procedure **Reduced-MDP** in optimizing sequential decision networks generated from the following probabilistic model:

- Some of the state variables are *global variables*, which are directly relevant to some applicable actions in every decision stage; the other state variables are *local variables*, which are not directly relevant to any applicable actions in some decision stages. Each local state variable is uniquely attached to a decision stage. We assume that there is only small deviation in the numbers of local state variables attach to decision stages. We use p_{local} to denote the percentage of local state variables within the entire set of state variables.

- Besides the global state variables, actions applicable in a decision stage d_i can only directly involve the local state variables attached to d_i or the local state variables attached to the adjacent decision stages of d_i. Let d_j be an adjacent decision stage of d_i. For each local state variable X_k attached to the decision stage d_j, p_a denotes the probability that X_k is directly relevant to some actions applicable in the decision stage d_i.

- We classify the interconnection of decision stages into two categories.

  One-Dimensional Interconnection: The first category simulates a one-dimensional assembly line in a factory process-control domain. In this category, m decision stages, d_1, d_2, ..., d_m, are interconnected as a one-dimensional array with d_1 as the starting stage; d_i $(i < m)$ is always connected to d_{i+1} while d_{i+1} $(i \geq 1)$ is connected to d_i with probability p_b. We refer to the arcs connecting some decision stage d_{i+1} to a decision stage d_i as the backward arcs in the network. The average out degree of a decision stages d_i (*i.e.,* the average number of adjacent decision stages from d_i) is thus $1 + p_b$.

Two-Dimensional Interconnection: The second category simulates a planar communication network in a stochastic transportation planning domain. In this category, m^2 decision stages, $d_{1,1}$, $d_{1,2}$, ..., $d_{1,m}, d_{2,1}$, $d_{2,2}$, ..., $d_{m,m}$, are interconnected as a two-dimensional array with $d_{1,1}$ as the starting decision stage; $d_{i,j}$ is always connected to $d_{i+1,j}$ when $i < m$ and $d_{i,j+1}$ when $j < m$ while $d_{i+1,j}$ and $d_{i,j+1}$ are connected to $d_{i,j}$ with probability p_b respectively. We refer to the arcs connecting some decision stage $d_{i+1,j}$ or $d_{i,j+1}$ to a decision stage $d_{i,j}$ as the backward arcs in the network. The average out degree of a decision stages d_i (*i.e.,* the average number of adjacent decision stages from d_i) is thus $2 + 2 * p_b$.

Parameter Configurations:

Since global state variables are active in every coherent fragment, we focus on evaluating how many local state variables are active in the coherent fragments of a sequential decision network. We generate random sequential decision networks with the following parameter settings:

- There is no global state variables and thus the precentage of local state variables p_{local} equals 1.0.
- There are either sixteen decision stages with eighty state variables or sixty four decision stages with three hundred and twenty state variables. In other words, there are five local state variables attached to each decision stages.
- p_a, the probability that the actions in a specific decision stage involve a specific local state variable attached to an adjecnt decision stage is either 0.2, 0.4, or 0.6.
- The decision stages either have a one-dimensional interconnection or a two-dimensional interconnection while p_b, the probability of having a specific backward arc in such networks, is either 0.05, 0.1, 0.15, ..., 0.95, or 1.0.

Evaluating the Effectiveness of Dimensionality Reduction:
Procedure **Reduced-MDP** takes time exponential in the maximal number of active state variables in a coherent fragment of a given sequential decision network. Given a sequential decision network with n state variables, we would like to evaluate the average percentage of local state variables that can be abstracted away in individual coherent fragments. More precisely, we define

$$1 - \mathcal{L}_2/n$$

as the effectiveness indicator of the dimensionality reduction attained by Procedure **Reduced-MDP** where A_F is the set of active variables in a coherent fragment F and

$$\mathcal{L}_2 = \max_F |A_F|$$

is the structural parameter for the sampled decision network. Note that we can at least abstract away $100(1 - \mathcal{L}_2/n)$ percents of the entire set of local state variables in any coherent fragment F.

Experiments and Data:
For a specific parameter configuration, we sample one hundred random sequential decision networks. For each sampled network, we analyze the maximal number of active variables in the coherent fragments of the network. The average size of the maximal number of active variables in the coherent fragments of a sequential decision network generated under different parameter settings is displayed in Figure 6.3 to Figure 6.6. Similarly the effectiveness of Procedure **Reduced-MDP** in dimensionality reduction is displayed in Figure 6.7 to Figure 6.10.

Observation and Comments:

- The effectiveness of Procedure **Reduced-MDP** in dimesionality reduction is mainly determined by parameter p_b, while parameter p_a plays a very minor role in the effectiveness of Procedure **Reduced-MDP**.

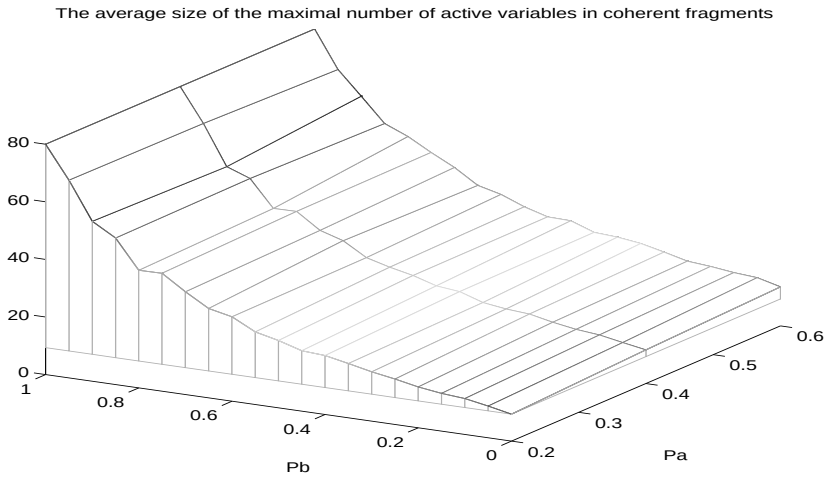

Figure 6.3: Statistics about decision networks with eighty state variables, sixteen decision stages, and a one-dimensional interconnection

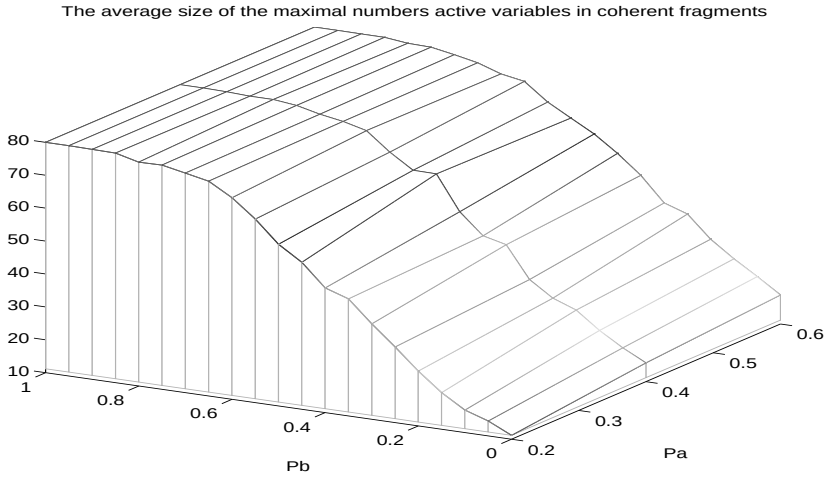

Figure 6.4: Statistics about decision networks with eighty state variables, sixteen decision stages, and a two-dimensional interconnection

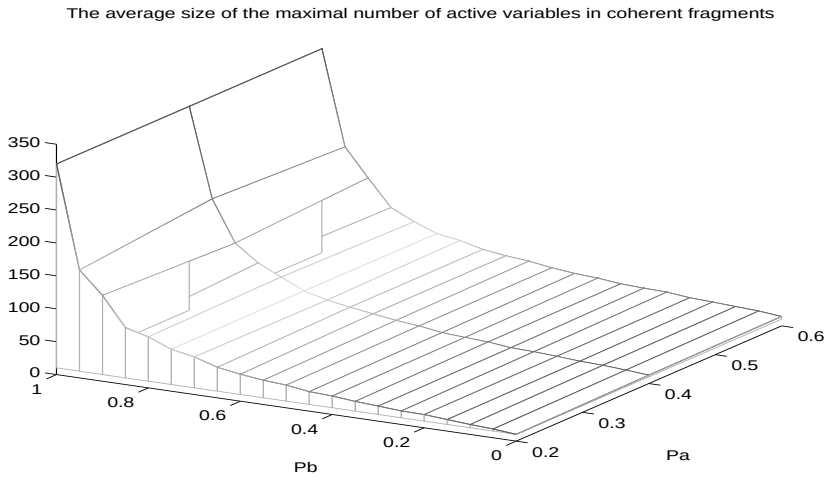

Figure 6.5: Statistics about decision networks with three hundred and twenty state variables, sixty four decision stages, and a one-dimensional interconnection

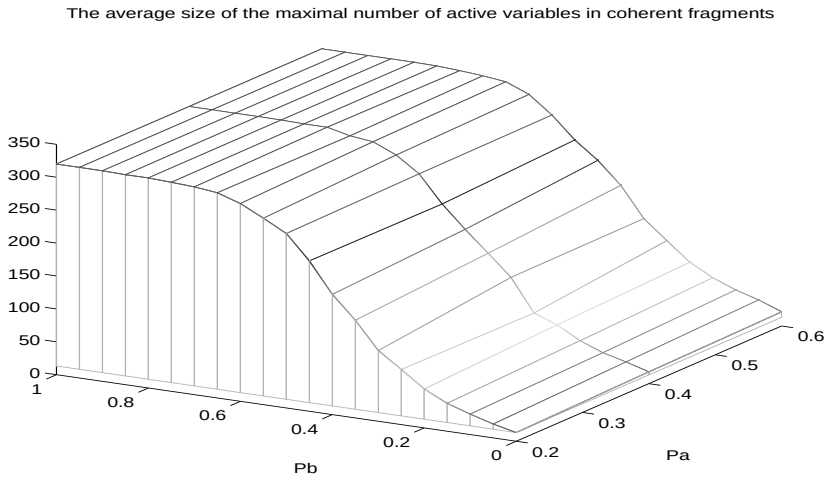

Figure 6.6: Statistics about decision networks with three hundred and twenty state variables, sixty four decision stages, and a two-dimensional interconnection

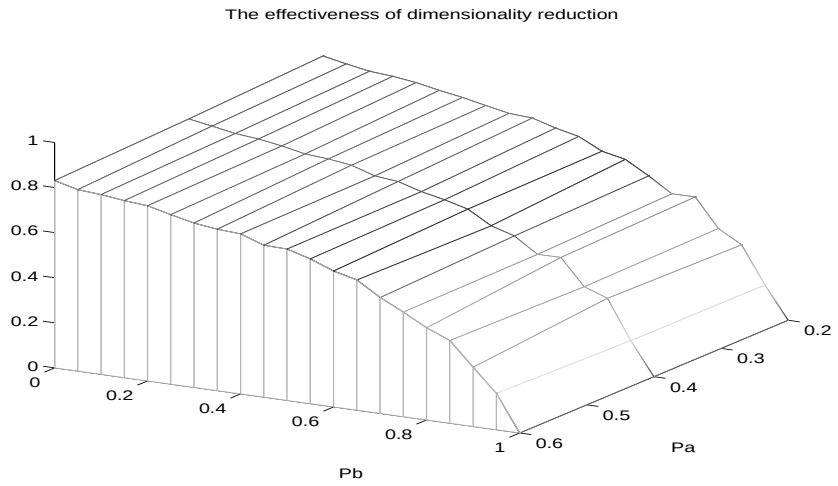

Figure 6.7: Effectivess of dimensionality reduction in decision networks with sixteen decision stages and a one-dimensional interconnection

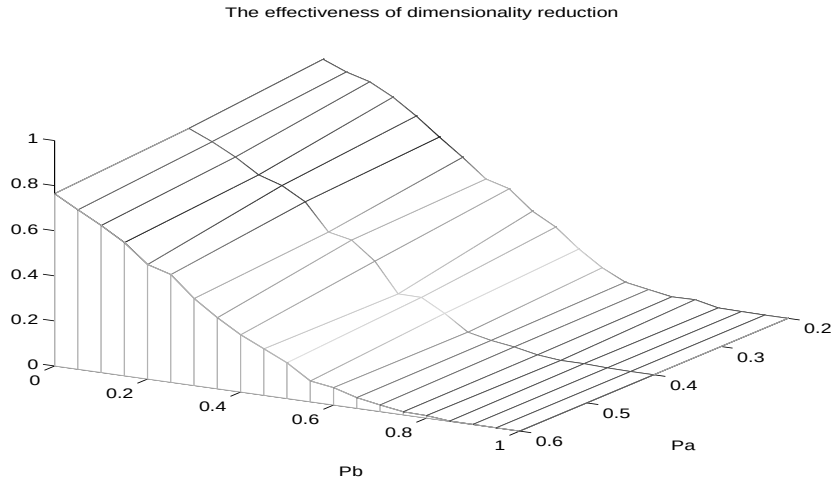

Figure 6.8: Effectivess of dimensionality reduction in decision networks with sixteen decision stages and a two-dimensional interconnection

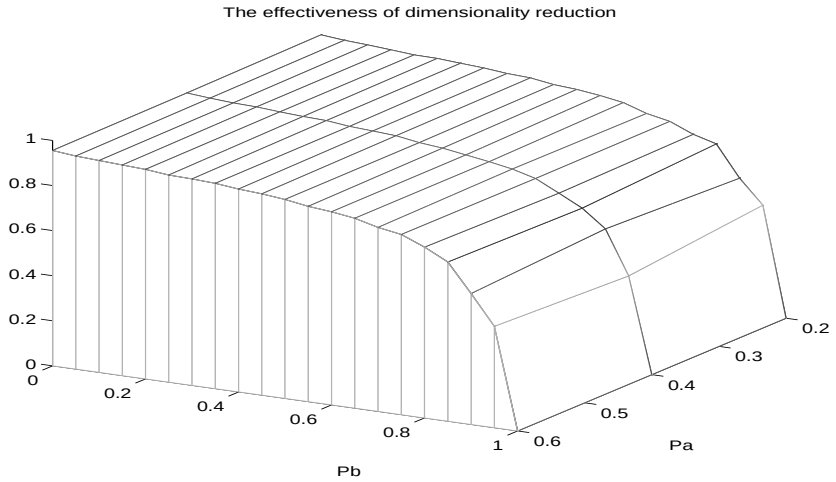

Figure 6.9: Effectivess of dimensionality reduction in decision networks with sixty four decision stages and a one-dimensional interconnection

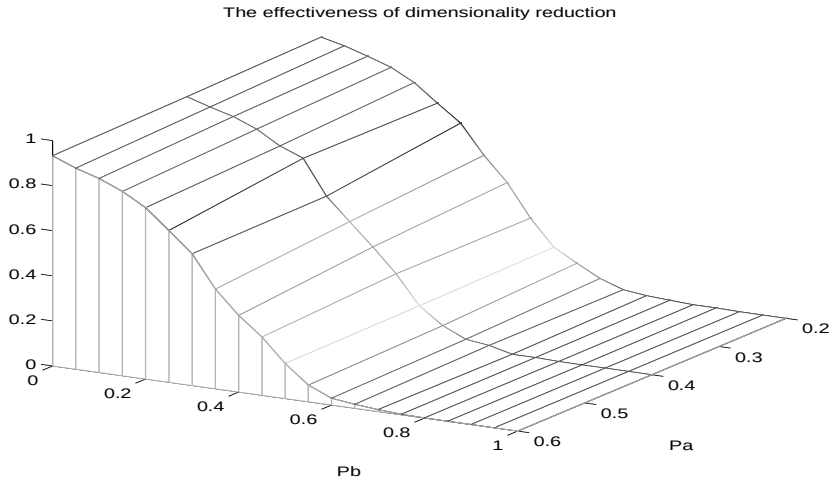

Figure 6.10: Effectivess of dimensionality reduction in decision networks with sixty four decision stages and a two-dimensional interconnection

- p_b is the probability of having a backward arc connecting two adjacent decision stages. When p_b is small, typically there are many coherent fragments and only a few active variables in each coherent fragments. Procedure **Reduced-MDP** ends in significant dimensionality reduction in this situation. When p_b is close to 1.0, typically there exists a giant coherent fragments with almost all state variables being active in this coherent fragments. In this situation, Procedure **Reduced-MDP** does not help much in dimensionality reduction.

- Although the experiments are conducted with $p_{local} = 1.0$ assuming all variables are local variables. It is easy to project what will happen to the experiments if p_{local} is less than 1.0 by counting the global variables as additional active variables in coherent fragments. This simply introduces a p_{local} discount over data governing the effectiveness of dimensionality.

6.5 Related Work

The particular representation of Markov processes in terms of sequential decision networks with conditional branches and loops that serves as inspiration for this paper is due to Smith and Williamson [SW95]. The particular methods for compiling fragment graphs are adapted from the work of Lin and Dean [LD94] on expediting temporal inference, which in turn builds on the work of Lansky [Lan88], Tenenberg [Ten91] and others in exploiting locality planning.

The general idea of restricting reachability in state space by enabling some actions and disabling others is common in control theory. See Ramadge and Wonhan [RW89b] for an survey of work on discrete event systems that concerns the synthesis of supervisory systems. Dean and Lin [DL95] describe a family of techniques for decomposing the computations required to solve large Markov decision processes. This family of techniques complements the representations and algorithms investigated in this paper.

Chapter 7

Decomposition Techniques for the Markov Decision Problem (II)

In Chapter 5, we focus on solving Markov decision problems given a partition of the state space into regions that are interconnected as many strongly connected components. In this chapter, we present appropriate decomposition techniques for solving Markov decision problems given a partition of the state space into regions that are interconnected as a single strongly connected component. For each region R in the partition of state space, we associate a set of topologically motivated parameters with R. These parameters summarize the interactions between R and the other regions in the partition. A specific estimate of the parameter values associated with region R allows us to construct a Markov decision process over the subspace associated with R. By solving such a Markov decision process, we determine a local policy on the region R, which is a solution to the subproblem associated with R. In this chapter, we describe two basic methods for combining solutions to subproblems in order to generate a solution to the global problem.

The first combination method is illustrated in an algorithm called *hierarchical policy construction*, which considers particular sets of parameter values that have an intuitive topological interpretation. These sets of parameter values give us a

set of candidate local policies associated with each region. We then construct an abstract Markov decision process by considering individual regions as abstract states and their candidate local policies as abstract actions. The solution to this abstract Markov decision process assigns a particular candidate policy to each region, thus yielding a policy on the entire state space. Hierarchical policy construction produces an optimal policy only in special cases; however, it does so relatively efficiently and has an intuitive interpretation that makes it particularly suitable for robot navigation domains.

The second method of combining solutions involves iterative approximation of the optimal parameter values, *i.e.,* those values associated with optimal solutions to the global problem. On each iteration of the iterative approximation method, we consider for each region R, a specific estimate of the parameter values of region R, and solve the resulting Markov decision process to obtain a local policy. By examining the resulting local policies, we obtain information to generate a new estimate of the parameter values that is guaranteed to improve the global solution. This information about local policies also tells us when the current solution is optimal or within some specified tolerance, and it is therefore appropriate to terminate the iterative procedure.

Section 1 describes the parameters modeling the inter-regional interactions, and the construction of a Markov decision process on a region R given a specific estimate of the parameter values of R. Section 2 presents the hierarchical construction algorithm. We describe the construction of abstract decision processes from a base-level process, and the use of such abstract decision processes to construct policies for the base-level process. Section 3 sketches the method of the iterative approximation and briefly addresses issues concerning convergence, optimality, and complexity. We refer readers to Appendix B for the connection between the iterative approximation method and the Dantzig-Wolfe decomposition for solving linear programs. We refer readers to Appendix C for more details of the iterative approximation method sketched in Section 3. In Section 4, we illustrate the use of decomposition techniques in a robot navigation domain, and also compare the computational efficiency of the two approaches presented in this

chapter.

7.1 Terminology and Notation

In this section, we describe a general method of decomposing a Markov decision process defined on a large state space into smaller Markov decision processes defined on local regions. Based on this regional decomposition framework, we develop two approaches in the following two sections that combine the solutions to the smaller Markov decision processes into a solution to the original Markov decision process.

Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, c)$ be a Markov decision process with finite state space $\mathcal{S}$, actions $\mathcal{A}$, state transition matrix p, and cost matrix c. For convenience, let $\mathcal{S} = \{1, 2, \ldots, N\}$. X_t (A_t) is a variable indicating the state (action) at time t.

$$p_{ij}(a) = \Pr(X_t = j | X_{t-1} = i, A_{t-1} = a) \quad i, j \in \mathcal{S} \quad a \in \mathcal{A}$$
$$c_{ij}(a) = \mathrm{Cost}(X_t = j | X_{t-1} = i, A_{t-1} = a) \quad i, j \in \mathcal{S} \quad a \in \mathcal{A}$$

where $\Pr(.|.)$ is a conditional probability distribution and $\mathrm{Cost}(.|.)$ is a real-valued cost function. A *policy* π is a function mapping states to actions $\pi : \mathcal{S} \rightarrow \mathcal{A}$.

To completely define a Markov decision process we also need a performance criterion. Two criteria that we consider are *expected discounted cumulative cost* and *expected cost to reach a specified goal*. In the former, the task is to find a policy minimizing the expected cumulative cost function,

$$\mathrm{E}_\pi(\Sigma_\gamma | i) \quad = \sum_{j \in \mathcal{S}} p_{ij}(\pi(i)) \left[c_{ij}(\pi(i)) + \gamma \mathrm{E}_\pi(\Sigma_\gamma | j) \right]$$

for all $i \in \mathcal{S}$ where $0 < \gamma < 1$ is the *discount rate*, Σ_γ represents the discounted cumulative cost, and $\mathrm{E}_\pi(.|.)$ denotes an expectation with respect to the policy π.

Let P be any partition of $\mathcal{S}$, $P = \{R_1, \ldots, R_m\}$ such that $\mathcal{S} = \bigcup_{i=1}^{m} R_i$ and $R_i \cap R_j = \emptyset$ for all $i \neq j$. We refer to a region $R \in P$ as an *aggregate* state. We refer to a state in $\mathcal{S}$ as a *base-level* state.

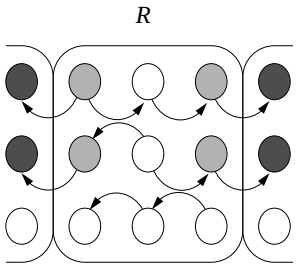

Figure 7.1: Boundary (light gray) and periphery (darker gray) states of region R

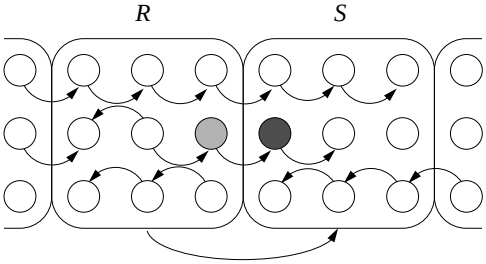

Figure 7.2: R is adjacent to S ($R \rightsquigarrow S$)

The *periphery* of an aggregate state R (denoted Periphery(R)) is the set of all base-level states not in R but reachable in a single transition from a base-level state in R.

$$\text{Periphery}(R) = \{j \,|\, j \notin R \wedge \exists i \in R, a \in \mathcal{A}, p_{ij}(a) > 0\}$$

The *boundary* of an aggregate state R (denoted Boundary(R)) is the set of all base-level states in R from which you can reach a base-level state not in R in a single transition.

$$\text{Boundary}(R) = \{i \,|\, i \in R \wedge \exists j \notin R, a \in \mathcal{A}, p_{ij}(a) > 0\}$$

In Figure 7.1, the boundary states are shaded light gray and the periphery states are shaded darker gray.

We say that aggregate state R in P is *adjacent* to aggregate state S in P (denoted $R \rightsquigarrow S$) just in case $\text{Periphery}(R) \cap S \neq \emptyset$ (see Figure 7.2).

Next, we introduce a set of parameters that we will use to model interactions among regions. Let $U = \bigcup_{R \in P} \text{Periphery}(R)$, and β_i for each $i \in U$ denote a real-valued parameter. Let $\bar{\beta} \in \mathbf{R}^{|U|}$ denote a vector of all such β_i parameters, and

$\bar{\beta}|_R$ denote a subvector of $\bar{\beta}$ composed of β_i where i is in Periphery(R). U is the medium for inter-regional interactions; a region R can only communicate with the other regions through the states in U. Parameter β_i serves as a measure of the expected cumulative cost of starting from a periphery state, and $\bar{\beta}|_R$ provides an abstract summary of how the other regions affect R.

Given a particular $\bar{\beta}$, the original Markov decision process can be decomposed into smaller Markov decision processes, each of which determines a local policy on a local region. For a region R and the subvector $\bar{\beta}|_R$ of $\bar{\beta}$, we define a Markov decision process $\mathcal{M}_{\bar{\beta}|_R} = (R \cup \text{Periphery}(R), \mathcal{A}, q, k)$ and the corresponding local policy $\pi_{\bar{\beta}|_R}$ as follows.

1. $R \cup \text{Periphery}(R)$ is the (local) state space for $\mathcal{M}_{\bar{\beta}|_R}$.

2. q_{ij} is the (local) state transition matrix for $\mathcal{M}_{\bar{\beta}|_R}$, where

   - $q_{ij} = p_{ij}$ for $i \in R$
   - $q_{ii} = 1$ for $i \in \text{Periphery}(R)$

3. k_{ij} is the (local) cost matrix for $\mathcal{M}_{\bar{\beta}|_R}$, where

   - $k_{ij} = c_{ij}$ for $i, j \in R$
   - $k_{ij} = \beta_j + c_{ij}$ for $i \in R$ and $j \in \text{Periphery}(R)$
   - $k_{ii} = 0$ for $i \in \text{Periphery}(R)$

4. $\pi_{\bar{\beta}|_R}$ corresponds to the local policy that is optimal for $\mathcal{M}_{\bar{\beta}|_R}$ with performance criterion expected cost to goal and target set Periphery(R).

$\mathcal{M}_{\bar{\beta}|_R}$ is the subproblem we associate with region R given $\bar{\beta}|_R$ as an abstract summary of R's interaction with the other regions. $\pi_{\bar{\beta}|_R}$ is the solution to the subproblem $\mathcal{M}_{\bar{\beta}|_R}$. A particular $\bar{\beta}$ determines a set of local policies (abstract actions) which in turn determines a policy on the entire state space. Let π^* denote an optimal policy for $\mathcal{M}$. If $\beta_i = \text{E}_{\pi^*}(\Sigma_\gamma | i)$, then the resulting local policies as defined above define an optimal policy on the entire state space. The algorithms considered in the following two sections offer various methods for either guessing or successively approximating $\text{E}_{\pi^*}(\Sigma_\gamma | i)$ for all $i \in U$.

7.2 Abstraction and Hierarchical Policy Construction

In this section, we first describe a general method for constructing an abstract decision process from a base-level process given a fixed partition of the state space. We then present a hierarchical policy construction method for using an abstract decision process to construct policies for the base-level process.

7.2.1 Abstract Decision Processes

Let $P = \{R_1, \ldots, R_m\}$ be any partition of $\mathcal{S}$. Each region $R \in P$ can be considered as an *abstract state*. A particular local policy $\pi_{\bar{\beta}|_R}$ on region R can be considered as an *abstract action* on R, which reflects our bias toward different periphery states described by $\bar{\beta}|_R$. Abstract actions for stochastic domains are the rough equivalent of macro operators for deterministic domains. The abstract action $\pi_{\bar{\beta}|_R}$ indicates how to act optimally in region R if the interactions with the other regions are captured in $\bar{\beta}|_R$. For example, a large value for β_j naturally discourages us from entering the periphery state j since it induces a large cost in k_{ij}. The family of all abstract actions is denoted $\mathcal{F} = \{\pi_{\bar{\beta}|_R} | R \in P, \bar{\beta} \in \mathbf{R}^{|U|}\}$.

The probability of ending up in S starting in R and following an abstraction $\pi_{\bar{\beta}|_R}$ is defined by

$$\begin{aligned} p'_{RS}(\pi_{\bar{\beta}|_R}) &= \frac{1}{|\text{Boundary}(R)|} \left[\sum_{i \in \text{Boundary}(R)} \varphi_i \right] \\ \varphi_i &= \left[\sum_{j \in S \cap \text{Periphery}(R)} p_{ij} \right] + \sum_{j \in R} p_{ij} \varphi_j \end{aligned}$$

where $p_{ij} = p_{ij}(\pi_{\bar{\beta}|_R}(i))$. Note that we assume here that there is an equal probability of starting in any state in the boundary of R.

The cost of ending up in S starting in R and following $\pi_{\bar{\beta}|_R}$ is defined by

$$c'_{RS}(\pi_{\bar{\beta}|_R}) = \frac{1}{|\text{Boundary}(R)|} \left[\sum_{i \in \text{Boundary}(R)} \vartheta_i \right]$$

$$\vartheta_i = \left[\sum_{j \in S \cap \text{Periphery}(R)} p_{ij} c_{ij} \right] + \sum_{j \in R} p_{ij} [c_{ij} + \vartheta_j]$$

where $c_{ij} = c_{ij}(\pi_{\bar{\beta}|_R}(i))$.

The resulting abstract decision process is then defined by $\mathcal{M}_{\bar{\beta}} = (P, \mathcal{F}, p', c')$. It is important to note that $\mathcal{M}_{\bar{\beta}}$ need not be Markov; in some cases, we may simply accept this as one of the inevitable consequences of abstraction and proceed as if the process is Markov; in other cases, we may attempt to ameliorate this condition (at some increase in computational cost) by using one of several standard techniques.

7.2.2 Hierarchical Policy Construction

In the following algorithm, called *hierarchical policy construction*, we restrict our attention to a finite subset of $\mathcal{F}$. For each R and each S adjacent to R, we construct a local policy $\pi_{R \to S}$ by setting $\beta_i = 0$ for $i \in S \cap \text{Periphery}(R)$ and setting $\beta_i = \kappa$ for $i \in \text{Periphery}(R) - S$, where κ is some fixed constant. If the performance criterion for the base-level process is expected cost to goal, then for each R containing one or more goal states, we add an additional action in which all the peripheral states get $\beta_i = \kappa$ and all of the goal states are made into sinks with $k_{ii} = 0$. The abstract decision process is $(P, \mathcal{F}_{\rightsquigarrow}, p', c')$ where $\mathcal{F}_{\rightsquigarrow} = \{\pi_{R \to S} | R, S \in P \wedge R \rightsquigarrow S\}$ and the performance criterion is expected discounted cumulative reward for a discount rate γ.

The local policies have the interpretation that $\pi_{R \to S}$ is the policy to take starting in R if you want to get to S. The larger κ is, the more incentive there is to get to S and avoid the rest of the periphery of R. Figure 7.3 illustrates an example in which $p'_{RS}(\pi_{\bar{\beta}|_R}) = 0.7$ and $p'_{RT}(\pi_{\bar{\beta}|_R}) = 0.3$. The abstract policy has the interpretation of providing a global perspective and indicating for each region the best local policy to use. Generally, it is best to set γ very close to one or use an alternative performance criterion such as average expected cost per step [Der70].

The following is an algorithm to construct a global policy using the abstract decision process $(P, \mathcal{F}_{\rightsquigarrow}, p', c')$.

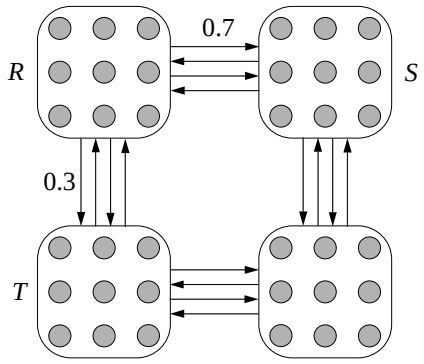

Figure 7.3: Abstract Markov decision process

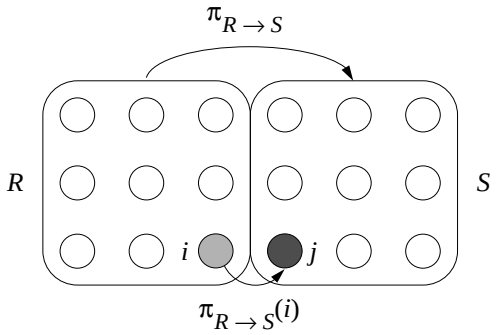

Figure 7.4: Abstract action for moving from R to S. Base-level action for moving from i to j.

1. Set κ and compute $\pi_{R \to S}$ for $R \rightsquigarrow S \in P$.

2. Calculate the abstract transition probabilities p' and abstract costs c'.

3. Set γ and solve the abstract decision process to obtain an abstract policy $\Pi : P \to \mathcal{F}_{\rightsquigarrow}$.

4. To determine the action to take in base-level state i, determine $R \in P$ such that $i \in R$ and take action $\Pi(R)(i)$.

Figure 7.4 illustrates the difference between abstract actions defined as local policies ($\pi_{\bar{\beta}|_R} \in \mathcal{F}$) and base-level actions ($\pi_{\bar{\beta}|_R}(i) \in A$). The above algorithm for constructing and solving abstract processes can be applied recursively and hence applies to hierarchical partitions.

Hierarchical policy construction does not guarantee to produce an optimal policy; however, it does so relatively efficiently and has an intuitive interpretation that makes it suitable for robot navigation domains. For a simple partition of the state space with no aggregation within regions, standard algorithms [Put94] on the base-level state space would be dominated by a factor quadratic in the size of the state space ($|\mathcal{S}|$), while hierarchical policy construction would be dominated by the number of regions in the partition ($|P|$) times the maximum number of neighbors for any region ($\max_{R \in P} |\{S | S \in P \wedge R \rightsquigarrow S\}|$) times the square of the size of the largest region ($\max_{R \in P} |R|$).

7.3 The Iterative Approximation Approach

Given a particular $\bar{\beta}$, the base-level process is decomposed into local processes, $\{\mathcal{M}_{\bar{\beta}|_R}\}$. By solving these local processes, we derive the corresponding local policies (abstract actions) $\{\pi_{\bar{\beta}|_R}\}$. A global policy, Π, to the base-level process is then derived by directly gluing these local policies together. The quality of Π critically depends on the choice of $\bar{\beta}$. Instead of relying on a particular $\bar{\beta}$, we can successively modify $\bar{\beta}$ and proceed through multiple iterations of decomposition and localized computation. Let $\Pi^{(i)}$ denote the global policy determined by a particular $\bar{\beta}$ adopted on the ith iteration of computation. Rather than simply

using one of the global policy $\Pi^{(i)}$ as the final solution policy, we may also cleverly combine these global polcies, $\{\Pi^{(i)}\}$, to generate an even better solution policy. There are several issues in realizing this iterative approximation framework:

- how to efficiently maintain useful information about the global policies, $\{\Pi^{(i)}\}$, derived on previous iterations,
- how to combine $\{\Pi^{(i)}\}$ to generate a solution policy if we need to terminate the computation at the end of a specific iteration,
- how to modify $\bar{\beta}$ iteratively so that the solution quality can be improved on each iteration and is guaranteed to converge to an optimal solution in a finite number of iterations, and
- how to determine it is time to terminate the computation when the solution is optimal or within some specified tolerance of the optimum.

In this paper, we focus on a particular iterative method that resolves these issues. This iterative method is based on a reduction to the methods of Kushner and Chen [KC74a] that demonstrate how to solve Markov decision processes as linear programs using the Dantzig-Wolfe decomposition principle [DW60]. The details of the material presented in this section depend on some understanding of linear programming [Chv80] and methods for decomposing and solving large systems [Las70]. We sketch the iterative method in the following, and refer the readers to appendix B and Appendix C for more details.

1. Given an arbitrary partition $P = \{R_1, \ldots, R_h\}$ of size h, we first transform P into a new partition $Q = \{U, K_1, \ldots, K_h\}$ of size $h + 1$ as follows.
   - U is $\bigcup_{R_i \in P} \text{Periphery}(R_i)$, which represents the peripheries of the regions R_i's.
   - For each region R_i, we have $K_i = \text{Kernel}(R_i) = R_i - U$, which represents the kernel of region R_i.

   This resulting partition Q induces a star topology that is critical in applying the techniques in [KC74a]. U is the *coupling region* in the new partition

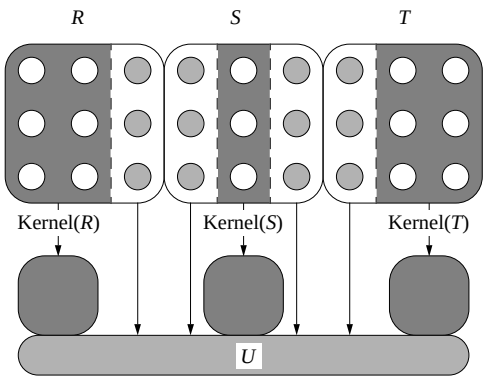

Figure 7.5: Relationship between the partitions P and Q

Q; removing the states in U separates the state space into isolated regions, each of which corresponds to a K_i, $1 \leq i \leq h$. In other words, the kernels of the regions R_i's in P are encapsulated and separated from one another by their peripheries, whose union is U. Figure 7.5 illustrates the relationship between the partitions P and Q.

2. On the ith iteration, we consider a particular $\bar{\beta}$ determined at the end of last iteration. We decompose the original base-level process into local processes $\{\mathcal{M}_{\bar{\beta}|_R} | R \in Q\}$, and derive the corresponding local policies $\{\pi_{\bar{\beta}|_R} | R \in Q\}$. Let $\Pi^{(i)}$ denote the global policy formed by gluing together these local policies.

3. We need to maintain a policy repository of at most $|U| + 1$ previously generated global policies, $\{\Pi^{(i)}\}$, at a time. As soon as policy $\Pi^{(i)}$ is generated, it replaces some policy $\Pi^{(j)}$, $j < i$, in the repository. The information associated with the current policy repository also allows us to determine an upper bound on the gap between the value of the current solution and the optimum. This bound allows us to decide whether we have reached the optimum or within a specified range of the optimum, or we should continue for another iteration of computation.

4. If we decide to terminate the computation, we are able to generate a final solution policy by properly combining the policies, $\{\Pi^{(i)}\}$, in the current

policy repository using its associated information; otherwise, we are able to determine a new $\bar{\beta}$ to go through another iteration of computation.

Theorem 7.1 *The iterative method described above improves the solution quality on each iteration, and converges to an optimal solution in a finite number of iterations.*

In the following, we briefly discuss the convergence rate and the time and space complexity of this iterative method.

- Empirical experience suggests that the Dantzig-Wolfe decomposition principle, which the analysis of the iterative method is based on, allows convergence to within 1–5% of the optimum fairly quickly, although the tail convergence rate can be very slow (see page 325 in [MB90]). In other words, it is likely that after a few number of iterations in the iterative method we are able to produce a solution of good quality, but it may not be worthwhile to continue after reaching 1–5% of the optimum.

- The computational task in an iteration is decomposed into two subtasks: (i) deriving and solving local Markov decision processes over local regions as subproblems, and (ii) maintaining a policy repository of up to $|U| + 1$ previously generated policies, where U is the union of Periphery(R) for the regions R in the original partition P.

  The computation efficiency of each iteration is critically affected by the structure of the given partition P: (i) $|\text{Kernel}(R_i)|$, the number of base-level states in the kernel of each region R_i in the partition P, and (ii) $|U|$, the total number of base-level states in the peripheries of the regions in the partition P.

- Compared with solving the original base-level process as a whole, the first subtask can be achieved more efficiently by applying the standard techniques for Markov decision processes over individual regions. This is a natural advantage of decomposition techniques, which divide large problems into subproblems of tractable sizes.

The second subtask can be achieved by maintaining a $(|U|+1) \times (|U|+1)$ matrix, whose computational efficiency critically depends on the topology of the given partition P. The second subtask can be performed efficiently if the size of U is relatively small. This is the additional cost to pay for decomposition techniques since we need to combine the solutions to sub-problems.

- In other words, this iterative method is promising if the given partition P divides the whole state space into many small regions, and the number of the states in the peripheries of the regions in P is also small.

7.4 Example and Comparison

In the following, we use the robot navigation example in Chapter 3.2 to illustrate the basic principle of the decomposition techniques described in this chapter.

Local regions: Since the elevators are accessible only in a small fraction of the entire state space $\mathcal{S}$, the robot tends to walk around the same floor for quiet a while before it either (i) enters the elevator to leave for another floor or (ii) reaches one of the charging sites. Therefore, the activities of the robot are highly localized in the individual floors. In the remainder of this section, we partition the state space into three regions R, S and T, which are composed of the states in the three floors respectively.

Loose coupling among regions: The connections among the three floors are also highly localized. Only if the robot is in a stairwell position on a floor, it can (i) transfer by elevator to another floor and (ii) only end in a correspond stairwell position on that floor. In other words, the peripheries of the three regions are very *thin*. The union of the peripheries, U, is composed of only twenty four states that correspond to the combination of six stairwell positions and four possible orientations.

Local interactions: Standing in a specific position beyond the stairwells, the robot can only move to the adjacent positions on the same floor. Therefore,

each state in the kernels of the regions R, S, and T can only transition to no more than thirty two other states that correspond to the combinations of eight adjacent positions and four possible orientations. We exploit this property in the theoretical analysis of the feasibility of the decomposition techniques.

7.4.1 Employing Hierarchical Construction Approach

We first construct the abstract decision process $(\{R, S, T\}, \mathcal{F}_{\rightsquigarrow}, p', c')$.

1. Determine the nine abstract actions in $\mathcal{F}_{\rightsquigarrow}$: $\pi_{S\to S}$, $\pi_{S\to T}$, $\pi_{S\to R}$, $\pi_{T\to S}$, $\pi_{T\to T}$, $\pi_{T\to R}$, $\pi_{R\to S}$, $\pi_{R\to T}$, and $\pi_{R\to R}$. Each abstract action corresponds to a local policy to be used on a floor that intends to increase the possibility of ending up in a specific floor in the long run. In other words, the abstract actions specify the most preferred next floors to go for individual floors.

2. To derive these abstract actions, we first choose an appropriate value κ to parameterize the pace the robot should take to transfer from one floor to the most preferred next floor to go. Second, for each floor F, we (i) uniformly assign zero to β_i's for those periphery states i's of F that are in the preferred floor, and (ii) uniformly assign κ to β_i for those periphery states i's of F that are not in the preferred floor. Third, we construct and solve these local Markov decision processes, each of which is concerned with 4,000 states instead of 12,000 states. The optimal policies for these local Markov decision processes are the abstract actions to be considered in individual floors.

3. Calculate the abstract transition probabilities p' and abstract costs c', which represent the estimates of the probabilities and costs of ending in another region when we adopt an abstract action (i.e. a local policy) in a specific region. For example, $p'_{ST}(\pi_{S\to R})$ is an estimate of the probability of entering region T from region S when we adopt the abstract action (the local policy) $\pi_{S\to R}$. Similarly, $c'_{ST}(\pi_{S\to R})$ is an estimate of the cost of entering region T from region S when we adopt the abstract action (the local policy) $\pi_{S\to R}$.

4. Solve the abstract decision process $(\{R, S, T\}, \mathcal{F}_{\rightsquigarrow}, p', c')$ to obtain an abstract policy $\Pi : P \rightarrow \mathcal{F}_{\rightsquigarrow}$. Π indicates which abstract action in $\mathcal{F}_{\rightsquigarrow}$ to take (i.e. which floor the robot should head for) when the robot is on a specific floor.

5. The robot takes the action $\Pi(F)(i)$, when (i) the robot is in a specific region F and (ii) i is its current state, which is determined by its current position and orientation.

Remark: In the hierarchical construction approach, our underlying intuition is to, for each floor, (i) first choose a specific floor (maybe the same floor) as the most preferred next floor to go, and (ii) second minimize the expected cumulative cost in achieving (i). We accomplish the first task by solving the abstract decision process that heuristically choose the most preferred next floor to go among all floors. We accomplish the second task by solving the local Markov decision processes associated with the state subspaces concerning individual floors.

7.4.2 Employing the Iterative Approximation Approach

1. We first initialize the policy repository and the $\bar{\beta}$ vector as described in Appendix C, and then iterate the following steps.

2. Since there are only twenty four (m=24) states in the peripheries, we only need to store twenty five (m+1=25) global policies at a time, each of which is compactly represented as a 25×1 column vector. Compared with the state space of size 12,000, the space taken by the policy repository is negligible.

3. $\bar{\beta}$ vector represents our current estimate of how valuable it is to enter a stairwell position. On each iteration, we solve the local Markov decision processes according to the current $\bar{\beta}$ vector. These solutions are the local policies on each floor, which are then glued together to yield a global policy. We then store this global policy in the policy repository to replace an old one there.

4. If the optimum or ϵ-optimum is not yet achieved, we (i) update the policy repository, (ii) determine a new $\bar{\beta}$ vector, and (iii) go back to Step 2. Otherwise, we combine the twenty five polices stored in the policy repository to generate the final policy.

Remark: Two major simplifications or assumptions in the hierarchical approach are (i) that each floor is restricted to have a unique most preferred next floor to go, and (ii) that each state in the peripheries is assigned either the value zero or the value κ. In the iterative approximation approach, we relax these restrictions. We allow each state in the peripheries to have arbitrary bias value. In other words, we numerically specify our preference over the states in the peripheries, rather than choosing the most preferred floor to go. In addition, we do not rely on a specific $\bar{\beta}$ vector and its resulting global policy. Instead, we (i) repeatedly modify $\bar{\beta}$ to generate a sequence of global policies, (ii) keep a policy repository of 25 such global policies at a time, and (iii) combine the policies stored in the policy repository to generate a solution policy when we decide to terminate the computation.

7.4.3 Comparing Computational Efficiency

In the following, we compare the computational efficiency of the decomposition techniques in deriving an optimal policy or a near optimal policy. We focus on the kind of robot navigation problem described in this section, where (i) a state is determined by the position and orientation of a robot and (ii) only a small set of actions are enabled in a state. We assume and exploit the general property that a robot can only move from one position to geometrically neighboring positions. Rather than assuming additional structure like floors loosely coupled by elevators, we consider a robot moving around an $N \times N$ area that is tessellated into N^2 square units as depicted in Figure 7.6.

Suppose we adopt a partition that divides the area into M^2 regions, each of which is a $\frac{N}{M} \times \frac{N}{M}$ local area as displayed in Figure 7.6. Therefore each region has no more than $\frac{4N}{M}$ neighboring squares. In summary, we have the following properties regarding the partitioning of state space:

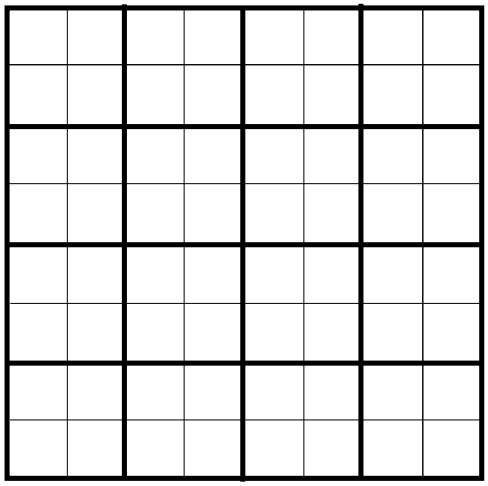

Figure 7.6: An $N \times N$ area tessellated into N^2 squares and M^2 regions where $N = 8$ and $M = 4$

- We assume a robot can be oriented in four different ways, namely east, west, south, and north. This yields a state space of size $4N^2$, given the N^2 squares there. Each region R is composed of $\frac{4N^2}{M^2}$ states and is adjacent to no more than $\frac{16N}{M}$ states in other regions.

- For each region R, Kernel(R) is of size $\Theta(\frac{N^2}{M^2})$. The size of U, the union of the peripheries of regions, is $M^2\Theta(\frac{N}{M}) = \Theta(MN)$.

In the following, we (i) compare the computational efficiency of different approaches, and (ii) indicate when we should apply the decomposition techniques.

Modeling the Computation Complexity of the Standard Techniques

Note that

- Standard techniques like the policy iteration method and the value iteration method converge to optimal policies in linear and quadratic convergence rate respectively [Put94].

- Each iteration of the policy iteration method and the value iteration method takes time cubic and quadratic respectively in the size of the state space [Der70] [Put94].

- We model the computational complexity of directly applying the standard techniques to the state space as cn^k, where (i) both c and k are constants and (ii) n is the size of the state space. k is 3 and 2 respectively when applying the policy iteration method and the value iteration method respectively, while in applying the value iteration method c is much greater than that in applying the policy iteration method.

- Therefore, the overall computation complexity of directly applying the standard techniques in this robot navigation domain is $\Theta(N^{2k})$, since we have a state space of size $4N^2$.

Computation Efficiency of the Hierarchical Construction Approach

- The overall computation complexity is

$$\Theta(M^2(\frac{N}{M})^{2k} + M^{2k}).$$

  The first term concerns with solving the M^2 local Markov decision processes over the local regions, each of whose size is $\Theta(\frac{N^2}{M^2})$. The second term concerns with solving the abstract decision process of size M^2.

- When $M = \Theta(N^{1/2})$, the first term and the second term of the overall computation complexity are roughly balanced, which yields a $\Theta(N^{k+1})$ time complexity.

- Compared with the $\Theta(N^{2k})$ complexity of directly applying standard techniques, the hierarchical construction method is always much more efficient. However, there is no guarantee about the quality of the derived policy.

Computation Efficiency of the Iterative Approximation Approach

- The computation complexity of each iteration of the iterative approximation method is

$$\Theta(M^2(\frac{N}{M})^{2k}) + (MN)^3).$$

The first term concerns with solving the M^2 local Markov decision processes over the kernels of the local regions, each of whose size is $\Theta(\frac{N^2}{M^2})$. The second term concerns with deriving the local policy over U, the union of the peripheries, by solving $\Theta(MN)$ linear equations that correspond to the $\Theta(MN)$ states in U.

- When $M = \Theta(N^{(2k-3)/(2k+1)})$, the first term and the second term of the overall computation complexity are roughly balanced, which yields a $\Theta(N^{(12k-6)/(2k+1)})$ time complexity on each iteration. More precisely, we have $\Theta(N^{18/5})$ time complexity when $k = 2$, and $\Theta(N^{30/7})$ time complexity when $k = 3$.

- Compared with the $\Theta(N^{2k})$ overall complexity in directly applying standard techniques, the iterative approximation approach is more efficient if the computation converges to an optimal (or near optimal) solution within (i) $O(N^{2/5})$ number of iterations when $k = 2$ and (ii) $O(N^{12/7})$ number of iterations when $k = 3$.

7.5 Related work

The related work on abstraction and decomposition is extensive. In the area planning and search assuming deterministic action models, there is the work on macro operators [Kor85] and hierarchies of state-space operators [Sac74a] [Kno91a]. Closely related is the work on decomposing discrete event systems modeled as (deterministic) finite state machines [ZW90] [CW90].

In the area of reinforcement learning, there is work on deterministic action models and continuous state spaces [MA95] and stochastic models and discrete state spaces [Kae93]. The hierarchical policy construction method described in Section 7.2 provides an alternative formulation of Kaelbling's hierarchical learning algorithm [Kae93] and suggests how Moore and Atkeson's parti-game algorithm might be extended to handle discrete state spaces.

The analysis in this paper of the paper borrows heavily from the work in operations research and combinatorial optimization for representing Markov decision

processes as linear programs [D'E63] [Der70] and decomposing large systems generally [DW60] [Las70] and Markov decision processes specifically [KC74a]. The approach described in [DKKN93b] represents a special case of the framework presented here, in which the partition consists of singleton sets for all of the states in the envelope and a set for all the states in the complement of the envelope.

7.6 Summary

The benefit of decomposition techniques is that we are able to deal with subproblems of smaller size, the tradeoff is that often extra effort is required to combine the solutions to these subproblems into a solution to the original problem. The leverage of decomposition techniques is orthogonal to that of standard techniques used to solve the original problem. In the case of a problem instance of very large size, decomposition techniques are valuable even if standard polynomial-time algorithms exist. It is always more efficient to apply a standard technique to solve smaller subproblems than to solve the original problem directly (unless the the standard technique provides a linear-time algorithm). The only question is whether we can efficiently combine the solutions to the subproblems into a solution to the original problem.

There exist standard techniques that can solve Markov decision processes in time polynomial in the size of the given state space. However, given a factored state-space representation for a Markov decision process, the underlying state space is exponential in the number of state variables mentioned in a problem instance. This exponential state space makes it impractical to apply the standard techniques to many AI planning problems that have factored state-space representations.

Frequently, a domain expert may exploit the factored state-space representation and partition the state space into regions, each of which has a abstract description of reduced dimensions. In this paper, we develop decomposition techniques for Markov decision processes, given an arbitrary partition of the state space into regions. Subproblems correspond to local Markov decision processes

over regions associated with a $\bar{\beta}$ parameter that provides an abstract summary of the interactions among regions.

In this Chapter, we develop two approaches to combining the solutions to subproblems: a hierarchical construction approach and an iterative approximation approach. The hierarchical construction approach provides a quick solution with an intuitive interpretation. The iterative approximation approach, iteratively decomposes a problem instance into different subproblems. On each iteration, we combine the solutions to the subproblems to obtain an improved solution and determine whether the current solution is optimal or within some specified tolerance. In particular, we relate the iterative approximation framework to the research on the use of Dantzig-Wolfe decomposition in solving Markov decision processes as linear programs, which can be interpreted as an iterative method. This iterative method is guaranteed to converge to the optimum in a finite number of iterations. For practical purposes, a few number of iterations may be sufficient for a solution of good quality.

Chapter 8

Related Research

Besides exploiting locality and dependency, there are techniques to deal with time-critical constraints on planning and to approximate acceptable solution instead of optimal solutions. Other than the directly related work mentioned in the previous chapters, the idea of exploiting locality and dependency to attain computational efficiency are also present in the research in many other related areas. In this chapter, we provide an overview of these research areas and briefly describe the underlying work.

8.1 Real-time Planning in Stochastic Domains

For time-critical planning applications involving large Markov decision process [DKKN93b] [DKKN93a] [DKKN95], often only a small Markov decision process $\mathcal{M}'$ is considered at a time as a local envelope.

- The Markov decision process $\mathcal{M}'$ composed of (1) a small portion of the original state space together with the associated transition probabilities and action costs, (2) a super sink state corresponding to the contraction of the remainder of the state space, and (3) the estimated costs and the estimated transition probabilities of entering the super sink state.

- A local policy π' that is optimal with respect to $\mathcal{M}'$ is determined, and the controller takes actions according to π' as long as the system state falls in

the portion of the original state space covered by $\mathcal{M}'$.

- If the system state falls out of the portion of the original state space covered by $\mathcal{M}'$, $\mathcal{M}'$ is properly expanded and shrank to cover the newly explored state and then a new local policy is determined.

This approach can be viewed as a heuristic decomposition method that synamically partition a given state space into two regions and focus on computing a local policy for the region containing the current states. It is simple and efficienct, but usually provides no formal quality guarantee of the solutions.

8.2 Approximate Planning Using Abstract Policies

Instead of determining an optimal policy for a huge Markov decision process $\mathcal{M}$, the approximation approach [BD94] pursues policies of suboptimal quality by approximating $\mathcal{M}$ with an abstraction of $\mathcal{M}$.

- In the approximation approach, a procedure is devised to determine a set of relevant variables by inspecting $\mathcal{M}$. The states in $\mathcal{M}$ are then aggregated into abstract states according to the set of relevant variables; an abstract state is a collection of the states in $\mathcal{M}$ that have the same values for the relevant variables.

- Given a set of relevant variables, an abstraction of $\mathcal{M}$ is a Markov decision process $\mathcal{M}'$ determined by (1) the abstract state space composed of the abstract states with respect to the set of relevant variables, (2) the transition probabilities and the estimated action costs in the abstract state space.

- The approximation approach determines an optimal policy π' with respect to $\mathcal{M}'$, which maps each abstract state into an action. This abstract policy π' is directly augmented into a policy in $\mathcal{M}$ by mapping all the states aggregated into the same abstract state into the same action adopted in the abstract state.

Since the number of relevant variables can be much smaller than the total number of variables, the approximate approach can potentially allow great computational efficiency. The solution quality of the policy π' compared with an optimal policy of $\mathcal{M}$ can be bounded by inspecting $\mathcal{M}$ and $\mathcal{M}'$. A problem with the approximation approach is that often all or most of the state variables turn out to be relevant; in this situation, the approximation approach does not help.

8.3 Aggregation and Abstraction

When computing an optimal policy for a large Markov decision process, it is useful to aggregate indistinguishable states dynamically [BDG95] [BCn89] [Sch84] [DG97] or statically using different notions of indistinguishability to reduce redundancy in computation. The decomposition technique in Chapter 6 employs structural analysis to identify a static aggregation of states before solving the underlying Markov decision process.

For model checking og complicated communication protocols, aggregation techniques have been investigated to derive equivalent finite-state machines on the fly to minimize computation time of validation or verification [LY92] [LY94] [BCL$^+$94] [DG97].

Abstraction is a special way of aggregation, in which two states are indistinguishable if they share the same configuration governing the values of a selected subset of variables. Aggregation and abstraction are employed as tools for constructing compact and equivialent Markov decision processes or finite automata to improve computational efficiency. Different abstraction techniques have been investigated in classic STRIPS planning [Kno91b] [Ten91] [Sac74b] [HS94] to cope with very large search spaces.

8.4 Probabilistic Inference in Bayesian Networks

Localized probabilistic inference in Bayesian networks [LS88] [JLO90] provides an analogy to the decomposition approach for planning under uncertainty in this

thesis. In a Bayesian network, a joint probability over many random variabless is factored into the product of many conditional probabilities over subsets of random variables, with the dependecies among state variables compactly encoded in the network [LS88][Pea88]. Dependencies among random variables provide computational leverage in computing marginal probabilities and conditional probabilities. Rather than globally enumeratng possible combination of variable values, local computation and propagation exploiting dependency and result in much better computational efficiency [JLO90] [LS88] [SDDF90] [AWFA87].

8.5 Decomposition Theory in Relational Databases

The decomposition theory for relational databases provides an analogy to the decomposition approach for planning under uncertainty in this thesis. Dependencies among domain features play an important role in decomposition theory for relational databases, and provide computational leverage in time and space [ADA93] [Ull88] [KS91]. Instead of keeping a large global relational schema covering all attributes, often small local relational schemas are employed, each covering only a subset of the attributes. This allows database managers to keep small databases over the local relational schemas, rather than a large database over the global schema. To answer a query, small relational databases over the local schemas are joined and processed to provide an answer, and this is usually more efficient than maintaining and probing a large database over the global schema.

Chapter 9

Conclusions

Discrete dynamical systems in the form of finite automata or Markov decision processes have been used as a representational and computational foundation for planning under uncertainty. Many AI planning problems can be conveniently viewed as control problems over the underlying discrete dynamical systems. Using AI-style representation, the features of application domains are represented as state variables, and planning problem instances compactly encode very large discrete dynamical systems. The standard algorithms to solve the corresponding control problems require explicit enumeration of the underlying state spaces. This is impractical since the sizes of the state spaces are exponential in the number of state variables.

In this thesis, we develop decomposition techniques to exploit structure for planning problems in different application domains. Given a problem instance, we first symbolically decompose the underlying dynamical system into subsystems. Often only a subset of the state variables are directly relevant to the dynamics within a subsystem, and the interactions among subsystems are characterized by the dependency across these subsets of the state variables. After analyzing the local dynamics within subsystems and the dependency across subsystems, we decompose the original planning problem into a sequence of subproblems governing the subsystems. Rather than explicitly constructing the entire system, we compactly construct each subsystem by abstracting away the state variables that are irrelevant to the subsystem and its interactions with the other subsystems.

Solutions of the planning subproblems can be derived by applying standard algorithms for the corresponding control problems to these compactly constructed subsystems, while a solution to the original planning problem is derived by systematically compiling and propagating the solutions to the planning subproblems. These decomposition techniques shed light on how to exploit locality and dependency to cope with very large state spaces.

Appendix A

Proofs

Theorem 5.1. *Given an instance of the temporal projection problem, the PRJ task, the PRJ1 task, and the PRJ2 task can be reduced to one another by transforming the problem instance. All these transformations can be done in polynomial time by (1) adding at most two more conditions and two more events, and (2) appropriately modifying the ordering constraints and causal rules.*

Proof. Given an instance I for the PRJ task, we can construct the following instance I' for the PRJ2 task. We add a new condition X_i and a new event e into I such that the only effect of event e is to make X_i true unconditionally. Condition X_i is constrained to be false initially. Event e is constrained to happen after the other events. We add $\langle X_i, true \rangle$ into the goal. It is obvious that the goal in instance I can be true immediately after all events have occurred if and only if the new goal in instance I' can be true immediately after event e has occurred.

Given an instance I for the PRJ2 task, we can construct the following instance I' for the PRJ1 task. Suppose e is the specified tail event in the PRJ2 task. We add a new condition X_i and add a new event e'. Event e' makes X_i true if the goal in instance I is true immediate before e' occurs. Condition X_i is constrained to be false initially. Event e' is constrained to happen immediately following e. $(\langle X_i, true \rangle)$ is the new goal. It is obvious that the goal of instance I can be true immediately after the specified event e has occurred if and only if the new goal can be true immediately following a possible event sequence in the new instance

I'.

Given an instance I for the PRJ1 task, we can construct the following instance I' for the PRJ task. We add a new condition X_i. All events remain the same except that in addition every event makes X_i true if the goal in instance I is true immediate before the event occurs. Condition X_i is constrained to be false initially. ($\langle X_i, true \rangle$) is the new goal. It is obvious that the goal of instance I can be true immediately following a possible event sequence if and only if the new goal in instance I' can be true after all events have occurred.

Each of the transformations above adds at most one more condition and one more event. Since the tasks can be reduced to one another by composing at most two of the transformations above, at most two more conditions and two more events will be added into the final transformed problem instance. □

Theorem 5.2. *Temporal projection regarding two-rule events, totally unordered, with a polynomial-size state space is NP-Complete.*

Proof. Given a directed graph with a set of forbidden pairs of arcs, the forbidden pairs of arcs problem [GJ79] determines whether there is a path between two nodes without using both arcs in any forbidden pair of arcs. It is NP-Complete even if all forbidden pairs are disjoint.

In the following, we reduce an instance of the disjoint forbidden pairs of arcs problem to an instance of the temporal projection problem regarding two-rule events, totally unordered, with a polynomial-size state space. (1) Each non-isolated vertex in the directed graph is mapped to a distinct state in a state space. In particular, the starting vertex s and the target vertex t are mapped to the initial state and a unique goal state respectively. (2) Each directed arc (u, v) in the graph is mapped to a distinct causal rule that is only applicable at state u and causes a state transition from u to v. (3) Each forbidden pair of arcs is mapped to a distinct two-rule event which is associated with the two causal rules that correspond to the two arcs in the forbidden pair. Events are allowed to occur in arbitrary order.

Since an event corresponds to a forbidden pair of arcs, the event can trigger a state transition along exactly one of the two arcs. Therefore an event sequence with a trajectory starting from s and ending in t corresponds to a solution directed path in the forbidden pairs of arc problem instance. On the other hand, each arc in the directed graph is uniquely associated with a two-rule event. Therefore a directed path from s to t in the directed graph corresponds to an event sequence starting from s and ending in t. This reduction together with Lemma 5.1 and Theorem 5.1 prove the NP-Completeness of this special case of temporal projection.□

Theorem 5.3. *Temporal projection regarding one-rule events with an arbitrary partial order and a polynomial-size state space is NP-Complete.*

Proof. Given a directed graph with a set of forbidden pairs of arcs, the forbidden pairs of arcs problem [GJ79] determines whether there is a path between two nodes without using both arcs in any forbidden pair of arcs. This problem is NP-Complete even for directed acyclic graphs.

In the following, we reduce an instance of the forbidden pairs of arcs problem on a directed acyclic graph to an instance of the temporal projection problem regarding one-rule events with an arbitrary partial order and a polynomial-size state space. (1) Each non-isolated vertex in the directed graph is mapped to a distinct state in a state space. In particular, the starting vertex s and the target vertex t are mapped to the initial state and a unique goal state respectively. (2) Each directed arc (u, v) in the graph is mapped to a distinct causal rule that is only applicable at state u and causes a state transition from u to v. Each causal rule is then uniquely associated with a one-rule event. In other words, each arc is uniquely mapped to a one-rule event. (3) For each forbidden pair of arcs (e, e') where e and e' correspond to the one-rule events tk_1 and tk_2 respectively, we impose an ordering constraint $tk_1 \prec tk_2$ ($tk_2 \prec tk_1$) if there is a directed path with arc e' preceding arc e (e preceding arc e'). Since the graph is acyclic, these ordering constraints define a partial order on the events.

For a forbidden pair of arcs (e, e') where e and e' correspond to the one-rule

events tk_1 and tk_2 respectively, the imposed ordering constraint on the events tk_1 and tk_2 prevents the arcs e and e' from both appearing in the same trajectory of any possible event sequence. Note that each arc is uniquely mapped to a one-rule event. Given the ordering constraints, event tk_1 must precede event tk_2 if there is a directed path in the graph with arc e' preceding arc e. However, event tk_1 precedes event tk_2 implies that a transition along the arc e must precede a transition along arc e' if both transitions are contained in the trajectory of an event sequence. However, this is impossible, since in a directed acyclic graph the existence of a directed path with arc e' preceding arc e implies the nonexistence of a path with arc e preceding arc e'. This proves that arcs e and e' can not both appear in the trajectory of any possible event sequence if (e, e') is a forbidden pair of arcs. Note that each arc is uniquely mapped to a one-rule event. Therefore a possible event sequence with a trajectory starting from s and ending in t gives us a solution directed path in the forbidden pairs of arc problem instance and vice versa. This reduction together with Lemma 5.1 and Theorem 5.1 prove the NP-Completeness of this special case of temporal projection. □

Theorem 5.4. *Temporal projection regarding one-rule events, totally unordered, with a polynomial-size state space is solvable in polynomial time.*

Proof. For this special case, the PRJ1 task of temporal projection can be reduced to a graph reachability problem, which is solvable in polynomial time [CLR91]. We first construct a directed graph G by mapping (1) each state to a distinct vertex in G and (2) each one-rule event that triggers a state transition from a state u to a state v to an arc (u, v) in G. Finding an event sequence that ends in a goal state is equivalent to finding a path from s to t in graph G where s and t correspond to the initial state and a goal state respectively. This reduction together with Theorem 5.1 prove that this special case of temporal projection is solvable in polynomial time. □

Theorem 5.5. *Temporal projection regarding general events partially ordered as*

$O(1)$ number of chains of length $O(|\mathcal{E}|)$ or $O(\log|\mathcal{E}|)$ number of chains of $O(1)$ length with a polynomial-size state space can be solved in polynomial time.

Proof. Given a problem instance $\langle \mathcal{P}, \mathcal{E}, \mathcal{O}, \mathbf{s}_0, \mathcal{G} \rangle$, the number of vertices in the graph constructed by the global search algorithm for temporal projection in Section 7 is $|S_{\mathcal{P}}| \times \prod_{1 \leq i \leq m}(1 + l_i)$, where $|S_{\mathcal{P}}|$ is the size of the state space, m is the number of chains, and l_i is the number of events in the ith chain. If $l_i = O(|\mathcal{E}|)$ and m is a constant, or $l_i = O(1)$ and $m = O(\log|\mathcal{E}|)$, in both cases the graph G is polynomial in the size of the event set $\mathcal{E}$. Therefore graph reachability and thus temporal projection is solvable in polynomial time for this special case of temporal projection. □

Theorem 5.6. *Temporal projection regarding general events with an arbitrary partial order and a constant-size state space is solvable in polynomial time.*

Proof. For this special case, the task in the PRJ1 task of temporal projection can be accomplished in the following way. First, we construct a directed graph G by (1) mapping each state to a distinct vertex in G, and (2) adding an arc (u, v) to G if one of the events can trigger a state transition from u to v. G is a graph of $O(1)$ size.

Second, we enumerate all simple directed paths which start from the initial state and end in goal states. There is only a constant number of such directed paths since the graph is of constant size. For each of these directed paths, we must determine whether there is an event sequence whose trajectory can correspond to the directed path. This task is equivalent to bipartite matching between the directed arcs in a simple directed path and the set of events. An event e can match a directed arc (u, v) if e can trigger a state transition from u to v. If each of the directed arcs in the directed path can be matched with a distinct event, this gives us an event sequence whose trajectory is the directed path. Finally, we examine the event sequence to determine whether it is consistent with the ordering constraints. Since the bipartite match problem can be solved in polynomial

time [PS82] [CLR91], this reduction together with Theorem 5.1 prove that this special case of temporal projection is solvable in polynomial time. □

Theorem 5.7. *The goal $\mathcal{G}$ is true immediately after all events have occurred if and only if for each region R, the regional subgoal $\mathcal{G}_R$ of R is true immediately after all of the events in R have occurred.*

Proof. According to the definition, all regional subgoals are subsets of the goal $\mathcal{G}$. In particular, the goal $\mathcal{G}$ equals the regional subgoal of the root region, since every condition is a local condition in this global region. Therefore the goal $\mathcal{G}$ is true after all events occur if and only if for each region R, the regional subgoal $\mathcal{G}_R$ is true after all events occur. By Lemma 5.2, a local condition X_i of a region R is true after all events occur if and only if X_i is true immediately after the events in region R have all occurred. Therefore a regional subgoal $\mathcal{G}_R$ is true after all events occur if and only if $\mathcal{G}_R$ is true immediately after the events in region R have all occurred. □

Theorem 5.8. *The set of coupling conditions of a region R is the union of two disjoint subsets: the set of abstract conditions of R and the set of subgoal conditions of R.*

Proof. The set of abstract conditions of R and the set of subgoal conditions of R are disjoint, since the abstract conditions are dependent on events not in R and the subgoal conditions of R are local conditions of R. In the following, we consider a region R of two or more child regions. (For a region composed of a single event, the proof directly follows from the definitions of these three sets of conditions.) Suppose X_i is an abstract condition of R. X_i is dependent on an event not in R and an event e in R. Since e is contained in one of R's child regions R', X_i is also an abstract condition of this child region R'. If X_i is a subgoal condition of R, then X_i must be an abstract condition of one of R's child regions. This is because X_i is dependent on R, but not a local condition of any

child region of R. Therefore, if X_i is in the union of the set of abstract conditions and the set of coupling conditions of R, then X_i must be an abstract condition of a child region R' of R, which implies that X_i is also a coupling condition of R.

On the other hand, suppose X_i is a coupling condition of R. X_i must be an abstract condition of a child region R' of R. If X_i is not an abstract condition of R, X_i is a local condition of R which is only dependent on the events in R. Since X_i is an abstract condition of a child region R' of R, X_i must be dependent on an event in R but not in R'. This implies that X_i is not a local condition of any child region of R. In other words, X_i is a subgoal condition of R. Therefore if X_i is a coupling condition of R, then X_i is either an abstract condition of R or a subgoal condition of R. □

Theorem 5.9. *The search graph $G_I = (V, E)$ constructed in procedure Temporal-Search is a directed acyclic graph. If we adopt a trivial partition that considers each individual event as a chain, the search graph G_I contains $2^{|\mathcal{P}|+|\mathcal{E}|}$ vertices. In this case, the procedure Temporal-Search runs in $O(2^{2(|\mathcal{P}|+|\mathcal{E}|)})$ time.*

Proof. When we follow a directed arc (u, v) in E, exactly one of the time counters is increased by one in v compared with the counters in u, and the other time counters remain the same. Consider the counters associated with the individual chains. There is no directed path from u to u where the counters are the same in the head and the tail, since at least one of the counters is increased by following a directed path. Therefore $G_I = (V, E)$ is a directed acyclic graph, and the single-source shortest paths problem and thus the graph reachability problem can be solved in $O(|V| + |E|)$ time [CLR91]. If we adopt the trivial partition, we have $|\mathcal{E}|$ chains, and the length l_i equals one for each chain. This gives us $|S_{\mathcal{P}}| \times \prod_{1 \le i \le m}(1 + l_i) = 2^{|\mathcal{P}|+|\mathcal{E}|}$ vertices, and at most $2^{2(|\mathcal{P}|+|\mathcal{E}|)}$ arcs. Therefore the procedure Temporal-Search runs in $O(2^{2(|\mathcal{P}|+|\mathcal{E}|)})$ time. □

Theorem 5.10. *Procedure Localized-Reasoning is sound and complete.*

Proof. By induction on the levels of the region hierarchy, we can establish the following two properties about the localized algorithm. First, for each region R, the regional subgoal $\mathcal{G}_R$ can be true immediately after the events in R have all occurred if and only if (1) procedure Abstract-Event can successfully derive the abstract event e_R and (2) at least one of the rules associated with e_R is applicable immediately before the events in R occur. Second, for each region R, procedure Generate-Sequence can generate an event sequence $\mathbf{q}$ in $\mathbf{Q}_{\mathcal{O}}^R$ immediately following which $\mathcal{G}_R$ can be true if and only if (1) $\mathbf{s}'$ is the abstract state in $S_{\mathcal{A}_R}$ immediately before the events in R occur where $\langle \mathbf{s}', t' \rangle$ is the pair of abstract states in $S_{\mathcal{A}_R}$ given as the input to procedure Generate-Sequence, and (2) at least one of the rules associated with e_R is applicable in $\mathbf{s}'$.

The induction hypothesis is composed of the two properties described above. In the basis step, we consider every region that is composed of a single event. Both properties are true in this situation directly following the localized algorithm and the definition of abstract events. In the induction step, we consider every region R in which both properties in the induction hypothesis are true for every child region of R. According to procedure Abstract-Event and the definitions of Δ_R and e_R, for each rule $\alpha \rightarrow \beta$ associated with e_R, if α is true immediately before the events in R occur, we have the following two implications: (a) the abstract events of R's child regions can occur as a sequence such that for each child region R', at least one of the rules associated with the abstract event $e_{R'}$ is applicable immediately before $e_{R'}$ occur, and (b) the incremental subgoal $\tilde{\mathcal{G}}_R$ is true immediately after the events in R occur. These two implications together with Corollary 5.2 allows us to complete the induction step for the two proposed properties.

First, suppose that we can derive the abstract event e_R for R. By implication (a) and the induction hypothesis, for each child region R' of R, the regional subgoal $\mathcal{G}_{R'}$ can be true immediately after the events in R' have all occurred. Therefore, according to implication (b) and Corollary 5.2, the regional subgoal $\mathcal{G}_R$ can be true immediately after the events in R have all occurred if one of the rules associated with e_R is applicable immediately before the events in R occur.

On the other hand, if we can not derive the abstract event e_R, this implies that Δ_R is empty and the regional subgoal $\mathcal{G}_R$ can never be achieved. This completes the induction step for the first property.

Second, suppose that (1) for a region R, $\langle \mathbf{s}', t' \rangle$ is the pair of abstract states in $S_{\mathcal{A}_R}$ given as the input to procedure Generate-Sequence and (2) at least one of the rules associated with e_R is applicable in $\mathbf{s}'$. By the implications (a) and (b) and the induction hypothesis, if one of the rules associated with e_R is applicable immediately before the events in R occur, we are able to generate an event sequence $\mathbf{q}$ in $\mathbf{Q}_{\mathcal{O}}^{R}$ such that (1) for each child region R' of R, the regional subgoal $\mathcal{G}_{R'}$ is true immediately after the events in R' have all occurred and (2) the incremental subgoal $\tilde{\mathcal{G}}_R$ is true immediately following $\mathbf{q}$. This completes the induction step for the first property.

By induction, both properties are true for all regions. In particular, consider the root region $\mathcal{E}$. Note that (1) since all conditions are local conditions in the root region, the root region has an empty set of abstract conditions; (2) $() \rightarrow ()$ is the only rule in the abstract event of the root region if we can successfully derive the abstract event; and (3) in particular, in Step 4 of procedure Localized-Reasoning, $()$ is true in $\mathbf{s}' = \langle\rangle$ immediately before the events in $\mathcal{E}$ occur where $(\mathbf{s}', \mathbf{f}') = (\langle\rangle, \langle\rangle)$ is the pair of abstract states given to procedure Generate-Sequence. Therefore, according to the two properties, procedure Localized-Reasoning always successfully (1) determines the existence of an event sequence in $\mathbf{Q}_{\mathcal{O}}^{\mathcal{E}}$that ends in a goal state and (2) generates such an event sequence if and only if one exists. □

Theorem 5.11. *Given an instance of the temporal projection problem, the procedure Localized-Reasoning runs in $O(n \times 8^{\mathcal{L}})$ time, where n is the size of the event set $\mathcal{E}$, and $\mathcal{L}$ is the interaction measure of the problem instance.*

Proof. In the procedure Localized-Reasoning, the procedure Region-Abstraction and the procedure Generate-Sequence are (recursively) called once in each region, and both procedures use the procedure Temporal-Search for local searches in individual regions. In particular, in each region the procedure Temporal-Search

is called $2^{|\mathcal{A}_R|}$ times in Step 2 of the procedure Region-Abstraction. In a region R, procedure Temporal-Search runs in $O(2^{2(b_R+c_R)})$ time according to Theorem 5.9, since there are b_R child regions and c_R coupling conditions in R. Therefore procedure Localized-Reasoning runs in $\sum_R 2^{|\mathcal{A}_R|} \times O(2^{2(b_R+c_R)})$ time, which can be written as $O((2n-1) \times 2^{\mathcal{L}} \times 2^{2\mathcal{L}})$ according to the following two observations: First, in a region hierarchy, the number of distinct regions is no more than $2n-1$; this is because in a tree with n leaves, there is at most $n-1$ internal vertices, which corresponds to a region hierarchy with n events. Second, in each region, we have

$$\mathcal{L} \geq b_R + c_R \geq c_R = |\mathcal{C}_R| = |\mathcal{U}_R \cup \mathcal{A}_R| \geq |\mathcal{A}_R|.$$

Accordingly we have a $O(n \times 8^{\mathcal{L}})$ time bound. □

Theorem 6.1. *Given an optimal policy π' for the reduced Markov decision process $\mathcal{M}'$ derived from a Markov decision process $\mathcal{M}$ by procedure* **Reduced-MDP**, *π' compactly encodes an optimal policy π for $\mathcal{M}$ with respect to a given initial state, where $\pi(\mathbf{u})$ equals $\pi'(\mathbf{u}_F)$ for every state $\mathbf{u}$ that is aggregated into an aggregate state $\mathbf{u}_F$ in a coherent fragment F.*

Proof. A variable is either active, latent, or passive in a coherent fragment. In procedure **Reduced-MDP**, states are aggregated into an aggregate state in a coherent fragment F if and only if (1) these states are indistinguishable in the values of the active variables in F and (2) only differ in the values of the passive variables in F. Note that the values of the latent variables in a coherent fragment F never change in F or the ancestors of F; the values of the latent variables in F remain the same as their initial values in the initial state. Since the passive variables in a coherent fragment F do not involve in the action dynamics in F or the descendants of F, the states aggregated into an aggregate state in coherent fragment F (1) have the same expected cumulative cost given any specific policy and (2) do not communicate with one another. Following the state-aggregation operation in coherent fragment F described in procedure **Reduced-MDP**, it is obvious that every aggregate state preserves the same expected cumulative cost

as any of its component base-level state given any specific policy. In particualr, this is true for the optimal value function for the reduced process, which maps a given aggregate state to the optimal expected cumulative cost associated with the underlying base-level states. Then according to the state-transition probability and cost function of the reduced Markov decision process described in procedure **Reduced-MDP**, it is obvious that a policy π that results in the optimal value function for the reduced process can be extended to a policy π' that results in the optimal value function for the original process simply by mapping every base-level state $\mathbf{u}$ to the action the corresponding aggreate state of $\mathbf{u}$ is mapped to. □

Theorem 7.1. *The iterative method described above improves the solution quality on each iteration, and converges to an optimal solution in a finite number of iterations.*

Proof. The iterative method is equivalent to (i) adopting the partition Q, and then (ii) applying the methods of Kushner and Chen [KC74a] that solve Markov decision processes as large linear programs using the Dantzig-Wolfe decomposition principle, which have the properties described above. A more detailed description of the algorithm and its correspondence to the methods of Kushner and Chen [KC74a] is provided in appendix B and Appendix C. □

Appendix B

Iterative Approximation and Linear Programming

In this chapter, we explore the relationship between the iterative approximation framework described in Section 5 and the use of the Dantzig-Wolfe decomposition principle in solving Markov decision processes as linear programs. This relationship then provides interesting insight into the iterative approximation framework. In the remainder of this paper, we abbreviate the Dantzig-Wolfe decomposition principle as the *DW decomposition*, and refer to the work of Kunshner and Chen [KC74a] as the *DW approach* for Markov decision processes.

B.1 Linear Programming and the Dantzig-Wolfe Decomposition Principle

To cope with linear programs of very large sizes, decomposition principles [Las70] that divide a large linear program into many correlated linear programs of smaller sizes have been well studied, among which the DW decomposition may be the most well known. In the following, we describe the general form of the DW decomposition without assuming additional structure in linear programs. We refer the readers to [Las70] [MB90] for more details about linear programming and the DW decomposition.

Consider a linear program of standard form

$$\left.\begin{array}{rcl} \min z & = & cX \\ AX & = & b \\ X & \geq & 0 \end{array}\right\} \ldots (1),$$

where X is a $N \times 1$ column vector composed of N variables $x_1, x_2, \ldots, x_N$; c is a $1 \times N$ row vector of N scalar values representing a cost function of X; A is a $M \times N$ *constraint matrix* representing M linear constraints on X; and $X \geq 0$ represents sign constraints enforcing $x_1, x_2, \ldots, x_N$ to be nonnegative.

Using the DW decomposition, we can arbitrarily partition the set of M linear constraints $AX = b$ into two smaller sets of linear constraints $A_1X = b_1$ and $A_2X = b_2$. Let m be the number of linear constraints in $A_1X = b_1$. Rather than directly solving the linear program in (1), the DW decomposition implicitly reformulate the linear program in (1) as a master problem concerning the m linear constraints in $A_1X = b_1$, and iteratively generate a series of subproblems concerning the $M-m$ linear constraints in $A_2X = b_2$ to attain an optimal solution of the linear program in (1).

B.1.1 The Master Problem in the DW Decomposition

Note that all feasible solutions to $AX = b$ also satisfy $A_2X = b_2$. Therefore each feasible solution X to $AX = b$ can be properly represented as

$$X = \sum_{1 \leq i \leq r} \lambda_i X^{(i)} + \sum_{1 \leq j \leq t} \bar{\lambda}_j \bar{X}^{(j)},$$

where

- X^i's, $1 \leq i \leq r$, and $\bar{X}^j$'s, $1 \leq j \leq t$ are $N \times 1$ column vectors representing the finite number of extreme points and extreme rays [1] [Las70] [MB90] respectively of the set $\{x | A_2X = b_2\}$, and

[1] The set of extreme rays $\{\bar{X}^1, \ldots, \bar{X}^t\}$ is any maximal set of linearly independent solutions to $A_2X = 0$.

- λ_i's and $\bar{\lambda}_j$'s must satisfy the following constraints

$$\left.\begin{array}{rcl} A_1(\sum_{1\le i\le r}\lambda_i X^i + \sum_{1\le j\le t}\bar{\lambda}_j \bar{X}^j) & = & b_1 \\ \sum_{1\le i\le r}\lambda_i & = & 1 \\ \lambda_i \ge 0, & & \forall i = 1\ldots r \\ \bar{\lambda}_j \ge 0, & & \forall j = 1\ldots t. \end{array}\right\} \ldots (2).$$

We can factor A_1 into the summations in the first equation in (2), which renders (2) as a set of linear constraints on λ_i's and $\bar{\lambda}_j$'s. Similarly, for the cost function $z = cX$ over X we can rewrite it as

$$cX = c(\sum_{1\le i\le r}\lambda_i X^{(i)} + \sum_{1\le j\le t}\bar{\lambda}_j \bar{X}^{(j)}),$$

which renders a cost function over λ_i's and $\bar{\lambda}_i$'s when we factor c into the summations in the above equation. Let m be the number of linear constraints in $A_1X = b_1$. We denote $A_1X^{(i)}$ as q_i, $A_1\bar{X}^{(j)}$ as $\bar{q}_j$, $cX^{(i)}$ as d_i and $c\bar{X}^{(j)}$ as $\bar{d}_j$, where q_i and $\bar{q}_j$ are m column vectors while d_i and $\bar{d}_j$ are scalar values. This allows us to rewrite the linear program in (1) as a linear program over the nonnegative variables λ_i's and $\bar{\lambda}_j$'s in the following form

$$\left.\begin{array}{rcl} \min z & = & \sum_{1\le i\le r}\lambda_i d_i + \sum_{1\le j\le t}\bar{\lambda}_j \bar{d}_j \\ \sum_{1\le i\le r}\lambda_i q_i + \sum_{1\le j\le t}\bar{\lambda}_j \bar{q}_j & = & b_1 \\ \sum_{1\le i\le r}\lambda_i & = & 1 \\ \lambda_i \ge 0, & & \forall i = 1\ldots r \\ \bar{\lambda}_j \ge 0, & & \forall j = 1\ldots t. \end{array}\right\} \ldots (3).$$

We refer to the linear program in (3) as the *master problem* in the DW decomposition in solving the linear program in (1). Let m be the number of linear constraints in $A_1X = b_1$. Note that we have $m + 1$ linear constraints in the master problem in addition to those sign constraints that enforce λ_i's and $\bar{\lambda}_j$'s to be nonnegative. Theoretically, we can directly apply the simplex method to solve the master problem, where the simplex method iteratively updates a basis of $m + 1$ columns of the constraint matrix of the master problem in (3). On each iteration, a profitable column is selected to enter the basis and replace an old one

in the current basis, which corresponds to an λ_i or an $\bar{\lambda}_j$ variable associated with an extreme point or an extreme ray of the set $\{X|A_2X = b_2\}$. In this way, the solution quality can be iteratively improved until an optimal solution is reached.

B.1.2 The Subproblems in the DW Decomposition

In order to solve the master problem using the simplex method, we need to determine a profitable column corresponding to an extreme point or an extreme ray of the set $\{X|A_2X = b_2\}$ to iteratively update the basis. However, it is infeasible to exhaustively enumerate all these extreme points and extreme rays in general. To cope with this difficulty, a *subproblem* is devised according to the current *simplex multipliers* [2] [Las70] [MB90] associated with the $m + 1$ linear constraints in the master problem (3). By solving the subproblem, we can find an extreme point or an extreme ray that gives us a profitable column to enter the basis.

Let $\bar{\beta}$ denote a $1 \times m$ row vector composed of the simplex multipliers associated with the m linear constraints

$$\sum_{1 \le i \le r} \lambda_i q_i + \sum_{1 \le j \le t} \bar{\lambda}_j \bar{q}_j = b_1.$$

Let β_0 denote the simplex multiplier associated with the linear constraint

$$\sum_{1 \le i \le r} \lambda_i = 1.$$

The corresponding subproblem is

$$\left.\begin{array}{rcl} \min z & = & (c - \beta A_1)X \\ A_2X & = & b_2 \\ X & \geq & 0 \end{array}\right\} \ldots(4).$$

For such a subproblem, we use z^*_β to denote the optimum solution of the subproblem, and δ_β to denote the quantity $\beta_0 - z^*_\beta$.

[2] On each iteration of the simplex method, we can determine for each linear constraint an associated simplex multiplier.

B.1.3 Conducting an Iteration of the DW Decomposition

On each iteration, we conduct the following steps to either (i) derive an entering column to enter the basis of the master problem and go through another iteration, or (ii) determine that the current solution for the master problem is optimal or within a specified tolerance ϵ of the optimum, report the current solution, and terminate the computation.

- Set up a subproblem according to the current simplex multipliers of the master problem. Solve the subproblem to determine an optimal solution z^*_β of the subproblem and $\delta_\beta = \beta_0 - z^*_\beta$.

- – If $\delta_\beta = \infty$, the cost function $z = (c - \beta A_1)X$ is unbounded along an extreme ray $\bar{X}^{(j)}$ and we can find a corresponding column

$$\begin{bmatrix} A_1\bar{X}^{(j)} \\ 0 \end{bmatrix} = \begin{bmatrix} \bar{q}^{(j)} \\ 0 \end{bmatrix}$$

  to enter the basis.

  – If δ_β is finite and greater than ϵ, z^* must be achieved by an extreme point $X^{(i)}$ and we can find a corresponding column

$$\begin{bmatrix} A_1X^{(i)} \\ 1 \end{bmatrix} = \begin{bmatrix} q^{(i)} \\ 0 \end{bmatrix}$$

  to enter the basis.

  In both situations, we determine an entering column to enter the basis. We then (i) update the current basis through an pivot operation according to the selected entering column, (ii) determine a new solution to and new simplex multipliers of the master problem, and (iii) go through another iteration of the DW decomposition.

- If δ_β is no more than ϵ, we can terminate the computation and report the current solution of the master problem as the final solution. In this

situation, the current solution is no greater than ϵ plus the optimum of the master problem. In particular, if δ_β is zero or negative, we have reached an optimal solution of the master problem.

Both the master problem and the subproblems center around the use of simplex method in solving the linear program in (3). We refer the readers to [MB90] for more details about (i) cycling prevention and (ii) the initialization of the simplex method using big-M method. We briefly summarize two important properties about the DW decomposition that are useful in the remainder of this paper.

Proposition B.1 *Using the DW decomposition, the solution quality is improved iteratively and converge to an optimum solution in a finite number of iterations.*

Proposition B.2 *For a subproblem in the DW decomposition, the value of the current solution is less than or equal to the value of an optimal solution plus δ_β where $\delta_\beta = \beta_0 - z^*_\beta$.*

B.2 The DW Decomposition and Markov Decision Processes

It is well known that we can solve Markov decision processes as linear programs [D'E63] [Der70] [KK71a] [Put94] using standard techniques like the simplex method. Kushner and Chen [KC74a] investigate the use of the DW decomposition in solving Markov decision processes as linear programs. In the following, we briefly introduce their results, which provide interesting insight into the iterative approximation framework described in Section 5.

B.2.1 Formulating Markov Decision Processes as Linear Programs

Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, c)$ be a Markov decision process with finite state space $\mathcal{S}$, finite action space $\mathcal{A}$, state transition matrix p, and cost matrix c. For convenience, let $\mathcal{S} = \{1, 2, \ldots, N\}$. X_t (A_t) is a variable indicating the state (action) at time t.

$$p_{ij}(a) = \Pr(X_t = j | X_{t-1} = i, A_{t-1} = a) \quad i, j \in \mathcal{S} \quad a \in \mathcal{A}$$
$$c_{ij}(a) = \mathrm{Cost}(X_t = j | X_{t-1} = i, A_{t-1} = a) \quad i \in \mathcal{S} \quad a \in \mathcal{A}$$

where $\Pr(.|.)$ is a conditional probability distribution and $\mathrm{Cost}(.|.)$ is a real-valued cost function. A *policy* π is a function mapping states to actions $\pi : \mathcal{S} \rightarrow \mathcal{A}$. We consider two kinds of performance criteria, namely (i) expected cumulative cost to reach a specified goal and (ii) expected discounted cumulative cost.

We further define $Cost(i, a) = \sum_j p_{ij}(a) c_{ij}(a)$. For the criterion of expected cumulative cost to reach a specified goal, a subset of $\mathcal{S}$ is designated as a target and performance is measured as the expected cost until arriving at some target states. Assuming that the target can be reached with probability 1 from any other state, we can reformulate $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, c)$ as the following linear program [D'E63] [KK71a]:

$$\left.\begin{array}{rcll} \min z & = & \sum_{i,a} Cost(i,a) E_{i,a} & \\ \sum_a E_{i,a} & = & \xi_i + \sum_{j,a} P_{ji}(a) E_{j,a}, & \forall i \in \mathcal{S} \\ E_{i,a} & \geq & 0, & \forall i \in \mathcal{S}, \forall a \in \mathcal{A} \end{array}\right\} \ldots (5),$$

where $E_{i,a}$ denote the average number of time that action a is taken in state i, and ξ_i is the probability that state i is the initial state.

Similarly, we have the following linear program for the criterion of expected discounted cumulative cost [D'E63] [KK71a].

$$\left.\begin{array}{rcll} \min z & = & \sum_{i,a} Cost(i,a) E_{i,a} & \\ \sum_a E_{i,a} & = & \xi_i + \gamma \sum_{j,a} P_{ji}(a) E_{j,a}, & \forall i \in \mathcal{S} \\ E_{i,a} & \geq & 0 & \forall i \in \mathcal{S}, \forall a \in \mathcal{A} \end{array}\right\} \ldots (6),$$

where $E_{i,a}$ denote the discounted average number of time that action a is taken in state i, and ξ_i is the probability that state i is the initial state.

Given a solution to (5) or (6), we can determine a nondeterministic policy where $E_{i,a}/\sum_a E_{i,a}$ is the probability of mapping state i to action a.

B.2.2 The DW Approach in Solving Markov Decision Processes

Kushner and Chen [KC74a] investigate the use of the DW decomposition in solving Markov decision processes formulated as the linear programs in (5) and (6). Given a Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, c)$, the following kind of partition $Q = \{U, K_1, \ldots, K_h\}$ is required in applying their techniques:

- For any two states i and j in two different K_l's, $1 \leq l \leq h$, both $p_{ij}(a)$ and $p_{ji}(a)$ are zero for any action a in $\mathcal{A}$.
- The states in two different K_l's, $1 \leq l \leq h$, can only communicate with one another through the states in U. $\mathcal{S}$ divides into isolated pieces K_l's, $l \geq 1$ when we remove U from $\mathcal{S}$.

Note that each state in $\mathcal{S}$ contributes a linear constraint to the linear programs in (5) or (6). Given a partition $Q = \{U, K_1, \ldots, K_h\}$ as required, we can partition the set of linear constraints into $A_1X = b_1$ and $A_2X = b_2$, where (i) $A_1X = b_1$ composed of the constraints contributed by the states in U and (ii) $A_2X = b_2$ composed of the constraints contributed by the states in $K_l, 1 \leq l \leq h$. This gives us a master problem involving the states in U and subproblems involving the states in K_l's.

Instead of solving subproblems directly, their dual linear programs are considered. Due to the special structure of the partition Q and the linear programs in (5) or (6), the dual linear program of a given subproblem can be reduced to the local Markov decision processes over individual regions K_l's, $1 \leq l \leq h$ rather than the entire state space $\mathcal{S}$. We briefly summarize the use of DW approach in solving Markov decision processes as linear programs in the following and refer the readers to [KC74a] for more details.

1. Choose a partition $Q = \{U, K_1, \ldots, K_h\}$ such that $\mathcal{S}$ divides into isolated pieces K_l's, $1 \leq l \leq h$ when we remove U from $\mathcal{S}$.

2. Derive the simplex multipliers associated with the master problem that is concerned with U, and determine the corresponding subproblem.

3. Reduce the subproblem to local Markov decision processes over K_l's, $1 \leq l \leq h$, and solve these Markov decision processes.

4. Manipulate the solutions to the local Markov decision processes derived in step 3 to generate an entering column for the master problem. Then update the basis and the simplex multipliers accordingly.

5. Check the stopping criteria. If the stopping criteria are met, stop and report a solution; otherwise, go to step 2.

Appendix C

More Details about the Iterative Approximation Method

In this part, we provide the details of the iterative method sketched in Chapter 7. We establish the relationship between the iterative method and the use of the DW decomposition in solving Markov decision processes as linear programs. This relationship then provides interesting insight into the iterative approximation framework, and also demonstrate the convergence of the iterative method described in Chapter 5.

C.1 Relationship between the DW Approach and Iterative Approximation

Consider the steps sketched in appendix B.2 about the DW approach that solves Markov decision processes as linear programs using the DW decomposition. They can be well interpreted as an iterative method as we sketched in Chapter 7. This demonstrates that the iterative method converges to the optimum in a finite number of iterations.

- Given a Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, c)$ and an arbitrary partition $P = \{R_1, \ldots, R_h\}$ of the state space $\mathcal{S}$, there is always a closely related partition that allows the use the DW approach. As depicted in Figure 7.5,

$U = \bigcup_{R_l} \text{Periphery}(R_l)$, which is the union of the peripheries of the regions R_i's, and each $K_l = \text{Kernel}(R_l)$, which is the kernel of each individual region R_l, $1 \leq l \leq h$, naturally yield such a partition $Q = \{U, K_1 \ldots, K_h\}$. When we remove U from $\mathcal{S}$, $\mathcal{S}$ divides into disjoint pieces K_l's, $1 \leq l \leq h$.

- Using such a partition $Q = \{U, K_1 \ldots, K_h\}$ in the DW approach, the number of linear constraints in the master problem is $|U| + 1$. The simplex multipliers associated with this master problem can be interpreted as the β_i's assigned to each state i in the peripheries of the regions in P, which determine the the $\bar{\beta}$ vector described in Section 7.3. On each iteration, such a $\bar{\beta}$ vector determine a subproblem. that is further reduced to smaller Markov decision processes over K_l's, $1 \leq l \leq h$. Each of these smaller Markov decision processes can be interpreted as the kind of local Markov decision process $\mathcal{M}_{\bar{\beta}|K_l}$, $1 \leq l \leq h$, described in Section 7.3.

- In Step 2 and Step 3 of the DW approach, local Markov decision processes over K_l's, $1 \leq l \leq h$ are derived and solved according to the current simplex multipliers. In Step 4 the solutions to these local Markov decision processes in turn allow us to update the simplex multipliers and then reiterate Step 2 and Step 3. In this way, the DW approach provides a control mechanism to iteratively solve the local Markov decision processes and update the $\bar{\beta}$ vector to improve the solution quality.

- In other words, the DW approach that solves Markov decision processes as linear programs can be interpreted as a particular iterative method. According to Proposition B.1 and Proposition B.2 in Section B.1, the DW decomposition iteratively improves the solution quality of a linear program and converges to an optimal solution in a finite number of iterations. Therefore the corresponding iterative method also converges to an optimal solution of the corresponding Markov decision process in a finite number iterations.

C.2 Overview of the Iterative Method

By appropriately interpreting the DW approach in [KC74a] under the iterative approximation framework, it yields the following iterative method.

1. Given a Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, c)$, we choose an arbitrary (but fixed) partition P of the state space $\mathcal{S}$ in advance which divides the state space into disjoint regions.

2. On the tth iteration, we decompose the original Markov decision process into smaller Markov decision processes over the kernels of the regions in P, and derive a policy $\Pi^{(t)}$ by gluing together the policies of these smaller Markov decision processes.

3. The information associated with $\Pi^{(t)}$ also tells us (i) a bound on the value of the current solution, and (ii) whether we should go through another iteration of Step 2.

4. Let m be the number of the states in the peripheries of the regions in partition P. We need to keep a policy repository of at most $m + 1$ such $\Pi^{(t)}$'s at a time. As soon as policy $\Pi^{(t)}$ is generated, it properly replaces one of the policy $\Pi^{(i)}, i < t$, in the repository. The current solution is a combination of such policies $\Pi^{(t)}$'s stored in the current policy repository.

5. If we need to go through another iteration, the information associated with the current policy repository tells us how to go through another iteration of step 2. The solution quality is improved iteratively, and converge to optimum.

In the following, we (i) provide the entire picture of the iterative method, and (ii) indicate its equivalence to the DW approach that uses the DW decomposition to solve Markov decision processes as linear programs [KC74a]. In the remainder of this paper, we focus on the criterion of expected cumulative cost to reach target states. For the expected discounted cumulative cost criterion, it is all the same except that the transition probability $p_{ij}(a)$ is multiplied by a discount factor γ

and the set of target states is empty. This is true because the linear programming formulations in (5) and (6) for these two performance criteria are basically the same except for the presence of the discount factor in the linear program in (6).

C.3 Partitioning a State Space into Local Regions

- We adopt an arbitrary but fixed partition $P = \{R_1, \ldots, R_h\}$ of the state space throughout all iterations of the iterative method. In particular, the union of the peripheries of the regions R_l's, $U = \bigcup_{R_l \in P} \text{Periphery}(R_l)$, and the kernel of each region R_l, $K_l = R_l - U$, play important roles in the iterative method.

- Let m be the number of the states in U. We number these m states in the coupling region U from 1 to m, and number the other states in $K_i, i > 0$ from $m + 1$ to N. Using such a fixed numbering on states, we can more conveniently name the elements of the matrices and the vectors that are related to these states.

C.4 The Abstract Action Space

An abstract action is a local strategy imposed on a region R to encourage certain pattern of system evolution when leaving region R to enter R's adjacent regions. An abstract action of a region R is parameterized by a set of bias values assigned to the states in U.

- $\beta_i, 1 \leq i \leq m$ denotes the bias value assigned to a state i in U as described in Section 7.3. In addition, we define β_0 to be a measure of the current solution quality. Let

$$\beta = [\beta_1, \ldots, \beta_m] \text{ and } \bar{\beta} = [\beta_1, \ldots, \beta_m, \beta_0].$$

- Intuitively, we can interpret β_i as a measure of the expected cumulative cost since the first time of leaving region R from state i. A big β_i has the effect of discouraging the use of state i as an exit to leave R to enter R's adjacent regions.

- Refer to Section C.8 for the initialization of $\bar{\beta}$ and refer to Section C.9 for the iterative modifications of $\bar{\beta}$. In the remainder of this paper, we use $\bar{\beta}^{(t)}$, $\beta_i^{(t)}$, and $\beta_0^{(t)}$ respectively to denote the specific $\bar{\beta}$, β_i, and β_0 appearing on the tth iteration of the computation.

Remark The $\bar{\beta}$ vector is composed of the simplex multipliers on a specific iteration of the master problem in the DW decomposition. Consider the $m+1$ constraints in the master problem, the first m of which concern with the states in U. On each iteration, the simplex multipliers associated with these m constraints then provide these bias values β_i's, $1 \leq i \leq m$. The simplex multipliers associated with the $(m+1)$th constraint in the master problem provides the β_0 value.

C.5 Deriving Local Policies

C.5.1 Deriving the Local Policies over the Kernels

Given a Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, c)$, and a particular $\bar{\beta}$ vector, we can derive local Markov decision processes over the kernels K_l's. For each region R_l in the partition P, we define a Markov decision process $\mathcal{M}_{\bar{\beta}|K_l} = (K_l \bigcup U, \mathcal{A}, \tilde{p}, \tilde{c})$.

1. $K_l \cup U$ is the (local) state space with respect to $\mathcal{M}_{\bar{\beta}|K_l}$, where the states outside the kernel K_l are considered as additional target states. *In other words, the union of U and the set original target states of $\mathcal{M}$ forms the set of target states of $\mathcal{M}_{\bar{\beta}|K_l}$.*

2. $\tilde{p}_{ij}$ is the (local) conditional probability distributions about action outcomes. Let $\tilde{p}_{ij}(a)$ and $p_{ij}(a)$ denote the probabilities of ending up in state

j if we take action a when we are in state i with respect to $\mathcal{M}_{\bar{\beta}|k_l}$ and $\mathcal{M}$ respectively. We define

- $\tilde{p}_{ii}(a) = 1$ if i is a target state of $\mathcal{M}_{\bar{\beta}|K_l}$, and
- $\tilde{p}_{ij}(a) = p_{ij}(a)$ for for a non-target state i in K_l.

3. $\tilde{c}_{ij}$ specifies the biased action costs with respect to $\mathcal{M}_{\bar{\beta}|K_l}$, which is properly biased according to $\bar{\beta}$. Let $\tilde{c}_{ij}(a)$ and $c_{ij}(a)$ denote the action costs of ending up in state j if we take action a when we are in state i with respect to $\mathcal{M}_{\bar{\beta}|k_l}$ and $\mathcal{M}$ respectively. We define

   - $\tilde{c}_{ij}(a) = 0$ where i is a target state of $\mathcal{M}_{\bar{\beta}|K_l}$,
   - $\tilde{c}_{ij}(a) = c_{ij}(a)$ where i is a non-target state and j is in K_l, and
   - $\tilde{c}_{ij}(a) = c_{ij}(a) + \beta_j$ where i is a non-target state and j is in U.

4. In the following, we use

   - $Cost(i,a) = \sum_j p_{ij}(a)c_{ij}(a)$ to denote the cost of taking action a in state i with respect to $\mathcal{M}$,
   - $ActionBias(i,a) = \sum_{j \in U} \tilde{p}_{ij}(a)\beta_j$ to denote the bias regarding taking action a in state i, and
   - $BiasCost(i,a) = \sum_j \tilde{p}_{ij}(a)\tilde{c}_{ij}(a)$ to denote the biased cost of taking action a in state i with respect to $\mathcal{M}_{\bar{\beta}|K_l}$.

   We then have:

   - For a target state i of $\mathcal{M}_{\bar{\beta}|_R}$, $BiasCost(i,a) = 0$.
   - For a non-target state i in K_l,

$$\begin{aligned} BiasCost(i,a) &= \sum_j \tilde{p}_{ij}(a)\tilde{c}_{ij}(a) = \sum_j p_{ij}(a)\tilde{c}_{ij}(a) \\ &= \sum_{j \notin U} p_{ij}(a)c_{ij}(a) + \sum_{j \in U} p_{ij}(a)(c_{ij}(a) + \beta_j) \\ &= \sum_j p_{ij}(a)c_{ij}(a) + \sum_{j \in U} p_{ij}(a)\beta_j \\ &= Cost(i,a) + ActionBias(i,a). \end{aligned}$$

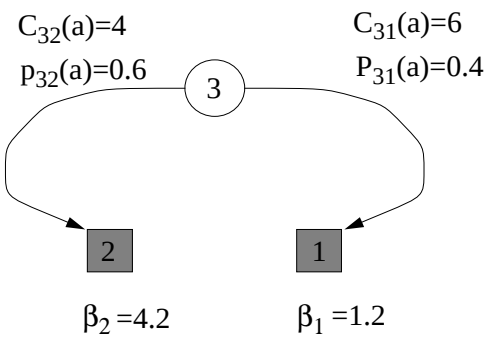

Figure C.1: The relationship between actual and biased action cost

Figure C.1 depicts three states 1, 2 and 3. State 1 and state 2 are in the peripheries of regions while state 3 is in the kernel of a region. When taking action a in state 3, we end in state 1 (2) with a cost of 6 (4) and with probability 0.4 (0.6). The bias associated with state 1 (2) is 1.2 (4.2). The biased action cost of taking action a in state 3 is then determined by

$$\begin{aligned} BiasCost(3,a) &= Cost(3,a) + Bias(3,a) \\ &= (0.6 \times 4 + 0.4 \times 6) + (0.6 \times 4.2 + 0.4 \times 1.2) \\ &= 7.8. \end{aligned}$$

Given $\mathcal{M}_{\bar{\beta}|K_l}$, we can derive the optimal local policy and the resulting local values as follows.

- $\pi_{\bar{\beta}|K_l}$ is a local policy that is optimal with respect to $\mathcal{M}_{\bar{\beta}|K_l}$, which maps the non-target states in K_l to actions [1].

- We use the term $LocalValue(i)$ to denote the value of a state i in K_l with respect to the local policy $\pi_{\bar{\beta}|K_l}$, which are determined by the following system of equations: For each i in K_l,

$$LocalValue(i) = BiasCost(i, \pi_{\bar{\beta}|k_l}(i)) + \sum_{j \in K_l} P_{ij}(\pi_{\bar{\beta}|k_l}(i)) \times LocalValue(j).$$

[1]The actions taken in the target states of $\mathcal{M}_{\bar{\beta}|K_l}$ are not important since are modeled as absorbing states with zero action cost.

Remark The biased action costs actually correspond to the coefficients in the the cost function $(c - \beta A_1)X$ of the subproblem on a particular iteration of the DW approach, where c is the vector of actual action costs; A_1 is the constraint submatrix that gives rise to the master problem; and β are the simplex multipliers concerning those m states in U. $ActionBias(i,a)$'s corresponds to the components of $-\beta A_1$. The dual linear program of the subproblem is reduced to the local Markov decision processes described above.

C.5.2 Deriving the Local Policies over the Peripheries

A local policy $\pi_{\bar{\beta}|K_l}$ maps the states in K_l, the kernel of region R_l, to actions. Given the set of local policies $\{\pi_{\bar{\beta}|K_l}\}$, we can also determine a local policy $\pi_{\bar{\beta}|U}$ that maps the states in U, the union of the peripheries, into actions as follows.

- We first determine the biased cost of taking an action a in a state i in U as follows:

  $$ActionBias(i,a) = -\beta_i + \sum_{j \in U} P_{ij}(a)\beta_j,$$

  and

  $$BiasCost(i,a) = Cost(i,a) + ActionBias(i,a).$$

- We then determine

  $$MinStay = \min_{i,a} BiasCost(i,a) + \sum_{j \notin U} P_{ij}(a) LocalValue(j).$$

- – Let

  $$(i^*, a^*) = \arg\min_{i,a}[BiasCost(i,a) + \sum_{j \notin U} P_{ij}(a) \times LocalValue(j)].$$

  – If $MinStay$ is negative, let $\pi_{\bar{\beta}|U}$ map the state i^* to action a^* ($\pi_{\bar{\beta}|U}(i^*) = a^*$) and arbitrarily map the other states in U to actions.

  – If $MinStay$ is positive or zero, let $\pi_{\bar{\beta}|U}$ arbitrarily map the states in U to actions.

Remark The dual linear program associated with the subproblem on an iteration of the DW decomposition is reduced to the local Markov decision processes. $MinStay$ measures the feasibility of the dual linear program. If $MinStay$ is negative, the dual linear program is infeasible and the optimal solution of the subproblem is unbounded; otherwise, the dual linear program is bounded. Deriving these local policies corresponds to solving the dual linear program, which allows us to find an entering column for the master problem.

C.6 Further Information Associated with Local Policies

On the tth iteration, we can derive a global policy $\Pi^{(t)}$ by gluing together (i) the set of local policies $\{\pi_{\bar{\beta}|K_l}\}$ over the kernels K_l's, and (ii) the local policy $\pi_{\bar{\beta}|U}$ over the peripheries. $\Pi^{(t)}$ can provide the following information:

- A weight vector $Weight^{(t)}$, which allows us to combine the policies in the policy repository to generate a solution policy if we decide to terminate at this moment.
- A feedback vector $Feedback^{(t)}$, which is a feedback on current solution quality that guides us to update the policy repository and conduct another iteration of computation.
- A value $\Delta^{(t)}$, which indicates the rate of the improvement in solution quality if $\Pi^{(t)}$ is incorporated to generate a new solution policy.

Again, let

$$\begin{aligned} MinStay &= \min_{i,a} BiasCost(i,a) + \sum_{j \notin U} P_{ij}(a) LocalValue(j), \text{and} \\ (i^*, a^*) &= \arg\min_{i,a} (BiasCost(i,a) + \sum_{j \notin U} P_{ij}(a) \times LocalValue(j)). \end{aligned}$$

We can derive $Weight^{(t)}$, $Feedback^{(t)}$, and $\Delta^{(t)}$ in following way.

- $Weight^{(t)}$ is a $N \times 1$ column vector, where $Weight^{(t)}(i)$ is a measure of the utility of taking action $\Pi^{(t)}(i)$ in state i.

  - First for each state i in U, we determine $Weight^{(t)}(i)$ as follows:

    * If $MinStay$ is negative,
      set $Weight^{(t)}(i^*)$ to be 1 and $Weight^{(t)}(i)$ to be 0 for the other states i's in U.

    * If $MinStay$ is positive or zero,
      set $Weight^{(t)}(i)$ to be zero for each state i in U.

  - Second for the kernel K_l of a region R_l,

    * if $MinStay$ is negative,
      we determine $Weight^{(t)}(i)$ for the states in K_l according to the following equations:

      $$\forall i \in K_l, \quad Weight^{(t)}(i) = p_{i^* i}(a^*) + \sum_{j \in K_l} P_{ji}(\Pi^{(t)}(j)) Weight^{(t)}(j);$$

    * if $MinStay$ is positive or zero,
      we determine $Weight^{(t)}(i)$ for the states in K_l according to the following equations:

      $$\forall i \in K_l, \quad Weight^{(t)}(i) = \xi_i + \sum_{j \in K_l} P_{ji}(\Pi^{(t)}(j)) Weight^{(t)}(j).$$

- $Feedback^{(t)}$ is a $(m+1) \times 1$ column vector. For each state i in U $(1 \leq i \leq m)$, we can determine $Feedback^{(t)}(i)$'s in the following way:

  $$Feedback^{(t)}(i) = Weight^{(t)}(i) - \sum_{j \in \mathcal{S}} P_{ji}(\Pi^{(t)}(j)) \times Weight^{(t)}(j).$$

  If $MinStay$ is negative,

  $$Feedback^{(t)} = \begin{bmatrix} Feedback^{(t)}(1) \\ \vdots \\ Feedback^{(t)}(m) \\ 1 \end{bmatrix};$$

otherwise,

$$Feedback^{(t)} = \begin{bmatrix} Feedback^{(t)}(1) \\ \vdots \\ Feedback^{(t)}(m) \\ 0 \end{bmatrix}.$$

- We can determine the unit profit associated with this policy by :

$$\Delta^{(t)} = \sum_{i \in \mathcal{S}} Cost(i, \Pi^{(t)}(i)) \times Weight^{(t)}(i).$$

Remark $Weight^{(t)}$ is a solution to the subproblem that is determined by the simplex multipliers $\bar{\beta}^{(t)}$. $Feedback^{(t)}$ is the corresponding entering column for the master problem. $\Delta^{(t)}$ is the unit cost of this entering column. When $MinStay$ is negative, the subproblem is unbounded and the entering column $Feedback^{(t)}$ is related to an extreme ray; otherwise, the subproblem is bounded and the entering column $Feedback^{(t)}$ is related to an extreme point.

C.7 Updating the Policy Repository

We keep a policy repository that is an array of up to $m+1$ different global policies $\Pi^{(i)}$'s generated on the previous iterations. We denote the rth policy stored in the policy repository as $\tilde{\Pi}^{(r)}$.

On the tth iteration, we maintain the following information in the policy repository:

- B is a $(m+1) \times (m+1)$ basis matrix. The rth column of B is the feedback vector of $\tilde{\Pi}^{(r)}$, the rth policy stored in the policy repository.

- C_B is a $1 \times (m+1)$ row vector. The ith element in C_B represents the rate of improvement in solution quality regarding $\tilde{\Pi}^{(i)}$, the ith policy stored in the policy repository.

- Note that $\Pi^{(t)}$ and $Weight^{(t)}$ can be reproduced from $\bar{\beta}^{(t)}$ by solving the corresponding local Markov decision processes when needed. Instead of

explicitly storing $\Pi^{(t)}$ and $Weight^{(t)}$, we can just store $\bar{\beta}^{(t)}$ in the policy repository to compactly represent $\Pi^{(t)}$ and $Weight^{(t)}$.

On the tth iteration, we update the information in the policy repository in the following way:

- Let $Y = B^{-1}\bar{\xi}$ and $Z = B^{-1}Feedback^{(t)}$, and

$$r = \arg\min_{i, Z_i > 0} (Y_i / Z_i),$$

  where $\bar{\xi} = \{\xi_1, \ldots, \xi_m, 1\}$, ξ_i is the probability that the state i in U is the initial state, and Y_i (Z_i) is the ith element of Y (Z).

- On the tth iteration, we bring in the newly generated $\Pi^{(t)}$ to replace $\tilde{\Pi}^{(r)}$, the rth policy stored in the policy repository. We store $\bar{\beta}^{(t)}$ as the rth element in the repository to represent $\Pi^{(t)}$.

- We update B by replacing the rth column of B by $Feedback^{(t)}$. Similarly, we update C_B by replacing the rth element of C_B by $\Delta^{(t)}$.

Remark The policy repository corresponds to the simplex tableau when we carry out the simplex method on the master problem of the DW decomposition. On the tth iteration, $\Pi^{(t)}$ generates an entering column $Feedback^{(t)}$ to enter the basis. The rule of deciding the leaving policy $\tilde{\Pi}^{(r)}$ actually corresponds to the ratio test in the simplex method. In other words, an update on the policy repository actually corresponds to an pivot operation on the simplex tableau.

C.8 Bounds, Stopping Criteria, and New Abstract Actions

At the end of the tth iteration, we can determine (i) a bound on the quality of the current solution, and (ii) whether we should terminate the computation or go through another iteration of computation.

Bounds on the solution quality :

Let

$$\delta_\beta^{(t)} = \beta_0^{(t)} - \sum_{i \in \mathcal{S}} BiasCost(i, \Pi^{(t)}(i)) Weight^{(t)}(i),$$

and

$$\delta^* = \min_{1 \le i \le t} \delta_\beta^{(i)}.$$

The value of the current solution is better than or equal to the optimum plus δ^*.

Stopping Criteria :

If δ^* is no more than ϵ, we have reached an ϵ-optimal solution. In particular if δ^* is less than or equal to zero, we have reached an optimal solution. In both situations, we should generate a solution policy of the required quality in the way described in Section B.9.2; otherwise, we can derive new abstract actions of local regions as follows and go through a new iteration.

New Abstract Actions and Convergence :

If we determine to go through one more iteration, we derive a new bias vector by

$$\bar{\beta}^{(t+1)} = C_B B^{-1}.$$

This gives us the new abstract actions for individual regions for the next iteration. The solution quality are guaranteed to be improved from iteration to iteration. δ^* converges to 0 as we approach an optimal solution.

Remark The bounds on solution quality and the convergence property directly come from Proposition B.1 and Proposition B.2 in Section B.1 about the DW decomposition. The stopping criteria actually corresponds to the optimality criteria of the master problem in the DW approach. The new bias vector corresponds to the new simplex multipliers derived by a pivot operation on the simplex tableau of the master problem.

C.9 The Initialization and Termination of the Iterative Method

C.9.1 Initializing the Policy Repository and Abstract Actions

- At the beginning, we let the policy repository contain $m+1$ dummy polices and set

$$\begin{aligned} C_B &= [M, M, \ldots, M], \text{and} \\ B &= I \end{aligned}$$

  where I is an identity matrix and M is a large positive value [2].

- Consequently we have $\bar{\beta}^{(0)} = C_B B^{-1} = [M, M, \ldots, M]$, which determines the initial abstract actions for individual regions. We then iteratively conduct the computation steps described in Section C.5 to Section C.8 until all the dummy policies are replaced with actual policies.

Remark Such an initialization corresponds to the derivation of an initial solution for the master problem using big-M method.

- The $m+1$ dummy policies correspond to $m+1$ artificial variables added into the master problem. The basis matrix B is initialized as the the $m+1$ columns associated with these $m+1$ artificial variables, which is an identity matrix. Every artificial variable is associated the cost M, which is a large positive value. M is exactly the value used in big-M method to initiate the simplex method. We refer the readers to [MB90] for the details of the big-M method and the issue of choosing an appropriate M.

- When applying the big-M method, some of the artificial variables may leave and enter the basis more than one time. Similarly, we allow the ith dummy

[2] M is exactly the value used in big-M method to initiate the simplex method. We refer the readers to [MB90] about the choice of an appropriate M.

policy to reenter the policy repository after it has been excluded from the policy repository. This happens when (i) on the rth iteration the stopping criteria $\delta_{\beta^{(r)}} < 0$ is satisfied but (ii) $M - \beta_i^{(r)}$ is also negative, which indicates the possibility of quality improvement by bringing back the dummy policy to kick off some actual policy stored in the current repository.

C.9.2 Generating a Solution Policy from the Policy Repository

When we meet the stopping criteria, we can properly combine the policies in the policy repository to produce an optimal or ϵ optimal policy in the following way.

- Let $\bar{\xi}$ denote the column vector $(\xi_1, \xi_2, \ldots, \xi_m, 1)^T$, where ξ_i is the probability that a state i in U is the initial state. Let $Ratio = B^{-1}\bar{\xi}$ and $Ratio_j$ be the jth component of $Ratio$. $\tilde{\Pi}^{(j)}$ denote the jth policy stored in the current policy repository. We determine

$$Frequency(i,a) = \sum_{1 \le j \le m+1, \tilde{\Pi}^{(j)}(i)=a} Ratio_j \times Weight^{(t)}(i).$$

  We then derive a random policy $\tilde{\Pi}$ where $Frequency(i,a)/\sum_a Frequency(i,a)$ is the conditional probability that we take action a when we are in state i.

- $\tilde{\Pi}$ can be improved into a deterministic policy Π^* by (i) first determining the values of states given the policy $\tilde{\Pi}$, and (ii) then conducting an iteration of policy improvement :

$$\Pi^*(i) = \arg\min_a [Cost(i,a) + \sum (P_{ij}(a) \times value(j))],$$

  where $value(j)$ is the value of state j according to $\tilde{\Pi}$. Output the policy Π^* as the solution policy.

Remark The $Frequency$ vector corresponds to a final solution for the master problem and actually $Frequency(i,a)$ corresponds to the variable $E_{i,a}$. Note that $E_{i,a}$ and thus $Frequency(i,a)$ also is the expected number of times when we are in state i and we take action a. Therefore $Frequency(i,a)/\sum_a Frequency(i,a)$ is the probability that we map state i to action a.

Bibliography

[ADA93] Paolo Atzeni and Valeria De Antonellis. *Relational Database Theory*. The Benjamin/Cummins Publishing Company, Redwood City, California, 1993.

[AHT90] James F. Allen, James Hendler, and Austin Tate, editors. *Readings in Planning*. Morgan Kaufmann, San Francisco, California, 1990.

[All83] James Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[AWFA87] S. Andreassen, M. Woldbye, B. Falck, and S. Andersen. Munin: A causal probabilistic network for interpretion of electromygraphic findings. In *Proceedings AAAI-87*, pages 121–124. AAAI, 1987.

[BCL+94] Jerry Burch, Edmund M. Clarke, David Long, Kenneth L. McMillan, and David L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer Aided Design*, 13(4):401–424, 1994.

[BCn89] D. P. Bertsekas and D. A. Castañon. Adaptive aggregation for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, 34(6):589–598, 1989.

[BD94] Craig Boutilier and Richard Dearden. Using abstractions for decision theoretic planning with time constraints. In *Proceedings AAAI-94*. AAAI, 1994.

[BDG95] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Exploiting structure in policy construction. In *Proceedings IJCAI 14*. IJCAII, 1995.

[BDH95] Craig Boutilier, Thomas Dean, and Steve Hanks. Planning under uncertainty: Structural assumptions and computational leverage. In *Proceedings of the Second European Workshop on Planning*, 1995.

[BDKL92] Kenneth Basye, Thomas Dean, Jak Kirman, and Moises Lejter. A decision-theoretic approach to planning, perception, and control. *IEEE Expert*, 7(4):58–65, 1992.

[Bel61] Richard Bellman. *Adaptive Control Processes*. Princeton University Press, Princeton, New Jersey, 1961.

[Ber87] Dimitri P. Bertsekas. *Dynamic Programming*. Prentice-Hall, Englewood Cliffs, N.J., 1987.

[BF88] John S. Breese and Michael R. Fehling. Decision-theoretic control of problem solving: Principles and architecture. In *Proceedings of the 1988 Workshop on Uncertainty in Artificial Intelligence*, pages 30–37, 1988.

[Bol85] Béla Bollobás. *Random Graphs*. Academic Press, London, 1985.

[BS78] Dimitri P. Bertsekas and Steven E. Shreve. *Stochastic Optimal Control: The Discrete Time Case*, volume 139 of *Mathematics in Science and Engineering*. Academic Press, New York, 1978.

[BW93] A. Barrett and D. Weld. Partial order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67:71–112, 1993.

[Byl94] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:161–204, 1994.

[Byl96] Tom Bylander. A probabilistic analysis of STRIPS planning. *Artificial Intelligence*, 1996.

[Cha87] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[Chi94] L. Chittaro. Skeptical and credulous event calculi for supporting modal queries. In *Proceedings ECAI-94*, 1994.

[Chr90] J. Christensen. A hierarchical planner that generates its own abstraction hierarchies. In *Proceedings AAAI-90*, pages 1004–1009. AAAI, 1990.

[Chv80] Vasek Chvatal. *Linear Programming*. W. H. Freeman and Company, 1980.

[CKL94] Anthony R. Cassandra, Leslie Kaelbling, and Michael Littman. Acting optimally in partially observable stochastic domains. In *Proceedings AAAI-94*, pages 1023–1028. AAAI, 1994.

[CLR91] T. H. Corman, C. E. Leiserson, and R. L. Rivest. *Algorithms*. MIT Press, Cambridge, Massachusetts, 1991.

[Coo90] Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

[CW90] Peter E. Caines and S. Wang. COCOLOG: A conditional observer and controller logic for finite machines. In *Proceedings of the 29th IEEE Conference on Decision and Control*, Hawaii, 1990.

[DB88a] Thomas Dean and Mark Boddy. Reasoning about partially ordered events. *Artificial Intelligence*, 36(3):375–399, 1988.

[DB88b] Thomas Dean and Mark Boddy. Reasoning about partially ordered events. *Artificial Intelligence*, 36(3):375–399, 1988.

[D'E63] F. D'Epenoux. Sur un probléme de production et de stockage dans l'aleatoire. *Management Science*, 10:98–108, 1963.

[Dea84] Thomas Dean. Managing time maps. In *Proceedings of the Canadian Society for Computational Studies of Intelligence*. CSCSI, 1984.

[Dea87] Thomas Dean. Planning, execution, and control. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, pages 29–1–29–10. DARPA, 1987.

[Dea88] Thomas Dean. Planning paradigms. In *Proceedings of the DARPA Knowledge-Based Planning Workshop (also appeared in the Proceedings of the Third Annual Workshop of Israel Association for Artificial Intelligence, and in the AI Magazine, Fall 1988)*. DARPA, 1988.

[Der70] Cyrus Derman. *Finite State Markovian Decision Processes*. Cambridge University Press, New York, 1970.

[DG97] Thomas Dean and Robert Givan. Model minimization in Markov decision processes. In *submitted to AAAI-97*. AAAI, 1997.

[DHW94] Denise Draper, Steve Hanks, and Daniel Weld. Probabilistic planning with information gathering and contingent execution. In *Second International Conference on AI Planning Systems*, 1994.

[DK89] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[DKKN93a] Thomas Dean, Leslie Kaelbling, Jak Kirman, and Ann Nicholson. Deliberation scheduling for time-critical sequential decision making. In *Proceedings of the Ninth Conference on Uncertainty in AI*, pages 309–316, 1993.

[DKKN93b] Thomas Dean, Leslie Kaelbling, Jak Kirman, and Ann Nicholson. Planning with deadlines in stochastic domains. In *Proceedings AAAI-93*, pages 574–579. AAAI, 1993.

[DKKN95] Thomas Dean, Leslie Kaelbling, Jak Kirman, and Ann Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76(1-2):35–74, 1995.

[DL93] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.

[DL95] Thomas Dean and Shieu-Hong Lin. Decomposition techniques for planning in stochastic domains. In *Proceedings IJCAI 14*. IJCAII, 1995.

[DW60] George Dantzig and Philip Wolfe. Decomposition principle for dynamic programs. *Operations Research*, 8(1):101–111, 1960.

[DW91] Thomas Dean and Michael Wellman. *Planning and Control*. Morgan Kaufmann, San Mateo, California, 1991.

[ER60] P. Erdös and A. Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.

[Eva90] C. Evans. The macro-event calculus: Representing temporal granularity. In *Proceedings of PRICAI'90*, 1990.

[FN71a] Richard Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[FN71b] Richard Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[GD94] Lloyd Greenwald and Thomas Dean. Solving time-critical decision-making problems with predictable computational demands. In *Second International Conference on AI Planning Systems*, 1994.

[GD95] Lloyd Greenwald and Thomas Dean. Anticipating computational demands when solving time-critical decision-making problems. In K. Goldberg, D. Halperin, J.C. Latombe, and R. Wilson, editors, *The Algorithmic Foundations of Robotics*. A. K. Peters, Boston, MA, 1995.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[GS93] A. Gerevini and L. Schubert. Efficient temporal reasoning through timegraphs. In *Proceedings IJCAI-93*. IJCAI, 1993.

[HM84] Ronald A. Howard and James E. Matheson. Influence diagrams. In Ronald A. Howard and James E. Matheson, editors, *The Principles and Applications of Decision Analysis*. Strategic Decisions Group, Menlo Park, CA 94025, 1984.

[How60] Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, Massachusetts, 1960.

[HS94] Peter Haddawy and Meliani Suwandi. Decision-theoretic planning using inheritance abstraction. In *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, 1994.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[Jak94] Kirman Jak. *Predicting Real-Time Planner Performance by Domain Characterization*. PhD thesis, Department of Computer Science, Brown University, 1994.

[JLO90] F.V. Jensen, S.L. Lauritzen, and K.G. Olesen. Bayesian updating in recursive graphical models by local computations. *Computational Statisticals Quarterly*, 4:269–282, 1990.

[Kae93] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: A preliminary report. In *Proceedings Tenth International Conference on Machine Learning*, 1993.

[KC74a] Harold J. Kushner and Ching-Hui Chen. Decomposition of systems governed by markov chains. *IEEE Transactions on Automatic Control*, AC-19(5):501–507, 1974.

[KC74b] Harold J. Kushner and Ching-Hui Chen. Decomposition of systems governed by Markov chains. *IEEE Transactions on Automatic Control*, 19(5):501–507, 1974.

[KG77] Kenneth Kahn and G. Anthony Gorry. Mechanizing temporal knowledge. *Artificial Intelligence*, 9:87–108, 1977.

[KHW94] Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic planning. In *Proceedings AAAI-94*. AAAI, 1994.

[KK71a] H. J. Kushner and A. J. Kleinman. Mathematical programming and the control of Markov chains. *International Journal on Control*, 13(5):801–820, 1971.

[KK71b] H. J. Kushner and A. J. Kleinman. Mathematical programming and the control of markov chains. *International Journal on Control*, 13(5):801–820, 1971.

[KL94] Lydia Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration space for fast path planning. In *IEEE International Conference on Robotics & Automation*, pages 2138–2145, San Diego, 1994.

[Kno91a] Craig A. Knoblock. Search reduction in hierarchical problem solving. In *Proceedings AAAI-91*, pages 686–691. AAAI, 1991.

[Kno91b] Craig A. Knoblock. Search reduction in hierarchical problem solving. In *Proceedings AAAI-91*, pages 686–691. AAAI, 1991.

[Kor85] Richard Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

[Kor87] Richard Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

[KS86] Robert Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[KS91] H. F. Korth and A Silberschatz. *Database System Concepts.* McGraw-Hill, New York, 1991.

[KS94] S. Kirkpatrick and B. Selman. Critical behavior in the satisfiability of random boolean formuae. *Science*, 264:1297–1301, May 1994.

[Lan88] Amy L. Lansky. Localized event-based reasoning for multiagent domains. *Computational Intelligence*, 4(4), 1988.

[Las70] Leon S. Lasdon. *Optimization Theory for Large Systems.* Macmillan Company, 1970.

[Lat90a] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer, Boston, Massachusetts, 1990.

[Lat90b] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, Massachusetts, 1990.

[LD93] Shieu-Hong Lin and Thomas Dean. Exploiting locality in temporal reasoning. In *Proceedings of the Second European Workshop on Planning*, 1993.

[LD94] Shieu-Hong Lin and Thomas Dean. Exploiting locality in temporal reasoning. In E. Sandewall and C. Backstrom, editors, *Current Trends in AI Planning*, Amsterdam, 1994. IOS Press.

[LD95a] Shieu-Hong Lin and Thomas Dean. Exploiting locality in temporal reasoning. *Computational Intelligence*, 1995. to appear.

[LD95b] Shieu-Hong Lin and Thomas Dean. Generating optimal policies for high-level plans with conditional branches and loops. In *Proceedings of the Third European Workshop on Planning*, pages 205–218, 1995.

[LD96] Shieu-Hong Lin and Thomas Dean. Localized temporal projection using subgoals and abstraction. *Computational Intelligence*, 1996. To appear.

[LDK95] Michael Littman, Thomas Dean, and Leslie Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the 14th Conference on Uncertainty in AI*, 1995.

[LP81] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[LS88] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50(2):157–194, 1988.

[LY92] David Lee and Mihalis Yannakakis. Online minimization of transition systems. In *Proceedings of 24th Annual ACM Symposium on the Theory of Computing*, 1992.

[LY94] David Lee and Mihalis Yannakakis. Testing finite state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3), 1994.

[MA95] Andrew W. Moore and Christopher G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces. *Machine Learning*, 1995.

[MB83] Jitendra Malik and Thomas O. Binford. Reasoning in time and space. In *Proceedings IJCAI 8*, pages 343–345. IJCAII, 1983.

[MB90] H. D. Sherali M.S. Bazaraa, J. J. Jarvis. *Linear Programming and Network Flows*. John Wiley & Sons, New York, 1990.

[Mea92] A. Montanari and et al. Dealing with time granularity in the event calculus. In *Proceedings of FGCS'92*, 1992.

[MH95] Drew McDermott and James Hendler. Planning: What it is, what it could be, an introduction to the special issue on planning and scheduling. *Artificial Intelligence*, 76(1-2):1–16, 1995.

[MR91] David A. McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings AAAI-91*, pages 634–639. AAAI, 1991.

[NB92] Bernhard Nebel and Christer Bäckström. On the computational complexity of temporal projection and plan validation, 1992.

[NB94] Bernhard Nebel and Christer Bäckström. On the computational complexity of temporal projection planning and plan validation. *Artificial Intelligence*, 66(1):125–160, 1994.

[NK94] Ann Nicholson and Leslie Pack Kaelbling. Toward approximate planning in very large stochastic domains. In *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, 1994.

[Paz71] Azaria Paz. *Introduction to Probabilistic Automata*. Academic Press, New York, 1971.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, California, 1988.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[PT87a] Christos H. Papadimitriou and J. N. Tsitsiklis. The complexity of markov chain decision processes. *Mathematics of Operations Research*, 12:441–450, 1987.

[PT87b] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov chain decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[Put94] M. L. Puterman. *Markov Decision Processes*. John Wiley & Sons, New York, 1994.

[PW92] J. S. Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the 1992 International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114, 1992.

[RW89a] Peter Ramadge and Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[RW89b] Peter Ramadge and Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[Sac74a] Earl Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 7:231–272, 1974.

[Sac74b] Earl Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 7:231–272, 1974.

[Sch84] Paul J. Schweitzer. Aggregation methods for large Markov chains. In G. Iazola, P. J. Coutois, and A. Hordijk, editors, *Mathemaical Computer Performance and Reliability*, pages 275–302. Elsevier, Amsterdam, Holland, 1984.

[SDDF90] Ross D. Shachter, Bruce D'Ambrosio, and Brendan A. Del Favero. Symbolic probabilistic inference in belief networks. In *Proceedings AAAI-90*, pages 126–131. AAAI, 1990.

[Sha86] Ross D. Shachter. Evaluating influence diagrams. *Operations Research*, 34(6):871–882, 1986.

[SW95] David E. Smith and Mike Williamson. Representation and evaluation of plans with loops. To appear in the working notes for the 1995 Stanford Spring Symposium on Extended Theories of Action, 1995.

[Ten91] Josh D. Tenenberg. Abstraction in planning. In J. F. Allen, H. A. Kautz, R. N. Pelavin, and J. D. Tenenberg, editors, *Reasoning about Plans*, pages 213–280. Morgan Kaufmann, San Francisco, California, 1991.

[TS90] Joseph A. Tatman and Ross D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):365–379, 1990.

[TVR96] John N. Tsitsiklis and Benjamin Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

[Ull88] J. D. Ullman. *Principles of Database and Knowledge Base Systems.* Computer Science Press, Potomac, Md, 1988.

[Vil82] Marc Vilain. A system for reasoning about time. In *Proceedings AAAI-82.* AAAI, 1982.

[Wel94] Daniel S. Weld. An introduction to partial-order planning. *AI Magazine*, Winter, 1994.

[YT90] Qiang Yang and Josh D. Tenenberg. Abtweak: Abstracting a nonlinear, least commitment planner. In *Proceedings AAAI-90*, pages 204–209. AAAI, 1990.

[ZW90] H. Zhong and W. M. Wonham. On the consistency of hierarchical supervision in discrete-event systems. *IEEE Transactions on Automatic Control*, 35(10):1125–1134, 1990.