Stochastic Planning with Concurrent, Durative Actions

Mausam

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

2007

Program Authorized to Offer Degree: Department of Computer Science & Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Mausam

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of the Supervisory Committee:

_____

Daniel S. Weld

Reading Committee:

_____

Daniel S. Weld

_____

Dieter Fox

_____

Rajesh Rao

Date: _____

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with fair use as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Proquest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, 1-800-521-0600, to whom the author has granted the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature_____

Date_____

University of Washington

Abstract

Stochastic Planning with Concurrent, Durative Actions

Mausam

Chair of the Supervisory Committee:
Professor Daniel S. Weld
Computer Science & Engineering

Controlling intelligent agents in complex, uncertain real-world environments poses requirements that are not addressed in classical planning formulations. Such problems typically involve ability to reason with uncertain action effects and are frequently formulated as Markov Decision Processes (MDPs). MDPs, while an otherwise expressive model, have two limitations – 1) MDP algorithms do not scale well, since they vary polynomially with the size of the state space. Typically the state space is exponential in the number of domain features, making state of the art MDP algorithms impractical for large problems. 2) MDPs allow only for sequential, non-durative actions. This poses severe restrictions in modeling and solving a real world planning problem, since these assumptions are rarely true in practice.

In this thesis we present solutions to both these problems. For the former, we present a novel probabilistic planner based on the novel approach to *hybridizing* two algorithms. In particular, we hybridize GPT, an exact MDP solver with MBP, a planner that plans using a qualitative (non-deterministic) model of uncertainty. Whereas exact MDP solvers produce optimal solutions, qualitative planners are fast and highly scalable. Our hybridized planner, HYBPLAN, is able to obtain the best of both techniques — speed, quality and scalability. Moreover HYBPLAN has excellent *anytime* properties and makes effective use of available time and memory.

For the latter, we extend the MDP model to incorporate — 1) simultaneous action execution, 2) durative actions, and 3) stochatic durations. We develop several algorithms

to combat the computational explosion introduced by these features. The key theoretical ideas used in building these algorithms are — modeling a complex problem as an MDP in extended state/action space, pruning of irrelevant actions, sampling of relevant actions, using informed heuristics to guide the search, hybridizing different planners to achieve benefits of both, approximating the problem and replanning. Our empirical evaluation illuminates the different merits in using various algorithms, *viz.*, optimality, empirical closeness to optimality, theoretical error bounds, and speed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

ADMISSIBLE HEURISTIC: an optimistic value function – overestimating for a maximization problem and underestimating for a minimization problem.

APPLICABILITY FUNCTION: set of actions whose preconditions are satisfied in any state.

ANYTIME ALGORITHM: an algorithm that generates the best answer within the scope of available information that have been explored up to the allowed time.

BDD: Binary Decision Diagrams. a compact representation for Boolean functions.

BELLMAN BACKUP/UPDATE: a unit operation for MDP algorithms in which values from successor states are backed up to the previous state.

COMDP: Concurrent MDP. an MDP in which a set of actions can be co-executed.

DECISION EPOCH: any time-point when a new action is allowed to start execution.

DURATIVE ACTIONS: actions that have non-zero durations.

GPT: General Planning Tool. a state space heuristic planner that is capable of solving both deterministic and probabilistic planning problems.

HAPPENING: zero or a time point when an action effect actually occurs or a new precondition is definitely needed or an existing precondition is no longer needed.

HEURISTIC FUNCTION: a value function that guides a heuristic search.

MBP: Model Based Planner. a planner for planning problems with qualitative uncertainty. It employs model checking algorithms using BDDs for planning.

MDP: Markov Decision Process. a mathematical framework useful for formulating probabilistic planning problems.

MUTEX: two actions are mutex if they cannot be co-executed either because their preconditions conflict or their effects or one's preconditions conflicts an effect of the other's.

PIVOT: zero or a time point when an action effect might occur, or a new precondition might be needed, or an existing precondition may no longer be needed.

RTDP: Real Time Dynamic Programming. a sampling based solution for probabilistic planning.

PDDL: Planning Domain Description Language. an expressive language employed for inputting a planning domain.

PLANNING: the computational problem formulating the deliberation about which actions to execute in order to achieve the objective.

PRECONDITION: a Boolean formula that should hold in the world state for successful execution of an action.

PROBABILISTIC PLANNING: planning in which the outcomes of actions is uncertain. also known as stochastic planning.

PRUNED RTDP: RTDP with provably sub-optimal actions removed from the applicability set of each state.

SAMPLED RTDP: RTDP in which only a few actions are randomly sampled for backing in the Bellman update.

STOCHASTIC PLANNING: planning in which the outcomes of actions is uncertain. also known as probabilistic planning.

TEMPORAL PLANNING: planning with an explicit representation of time in the domain.

ACKNOWLEDGMENTS

This dissertation is the culmination of almost five years of research. In these five years I have graduated from a young and under-confident student, unsure of his research potential, to a mature and confident scientist, ready to mentor others in their research. Indeed, this transition has been neither sudden nor painless. However, because of the immense support and guidance I have received from numerous people, it has probably been much easier for me than for many others.

The first person, I would like to thank, is my advisor, Prof. Daniel Weld. Dan has been a continuous source of inspiration, both in technical matters and in professional life in general. He has an unreal quality of tailoring his guidance to the needs of the student. He was especially sensitive and patient with me in my days as a beginning researcher and gave me lots of confidence when I needed it the most. It is natural for two people who have worked together for an extended period of time to experience the occasional disagreement, even minor skirmishes. Yet, even after more than five years of working with Dan, I cannot recollect a single day when his advice resulted in discontentment on my part. Instead I continued to grow as a researcher, scientist and student of the craft under his gentle but firm guidance. I feel deeply indebted to him and will feel gratitude for the rest of my life.

I wish to also thank Nicolas Meuleau, my mentor during my internship at NASA Ames for his guidance in suggesting directions for future research, as well as for his guidance on career paths. Much advice was also provided by seasoned researchers David Smith, Jeremy Frank, Eric Hansen, and Sylvie Thiebaux. I thank them for all their well wishes and guidance. Thanks also to Piergiorgio Bertoli, Ronen Brafman, and Subbarao Kambhampati for very fruitful collaborations.

On the occasions when I was faced with a mathematical or theoretical challenge in my work, it was often advice from my close friends and colleagues, Sumit Sanghai, Nilesh Dalvi

and Parag that brought in a much-required fresh perspective. I am grateful for their advice and for the many brainstorming sessions. Thanks also to my peer group from Arizona State University including Dan Bryce, J. Benton, Menkes Van den Briel, Minh Do, and Will Cushing with whom I had many fruitful discussions at technical conferences.

All of my research builds on the work done by Blai Bonet and Hector Geffner. I thank Blai Bonet for being so open in sharing his code and ideas with me. If the code of GPT (the general planning tool) had not been so well designed and documented, it would have taken me a lot of time before I could give empirical proof for my ideas.

This dissertation is a result of several writes and rewrites. I received invaluable suggestions from several colleagues and researchers on different parts of the work. Daniel Lowd and Pradeep Shenoy consistently found time to read my drafts and offered numerous ideas for improvement. I am also thankful to Alicen Smith, Benson Limketkai, Deepak Verma, Jayant Madhavan, Jiun-Hung Chen, Julie Letchner, Krzysztof Gajos, Maria Fox, Raphael Hoffman, Stanley Kok, Tina Loucks, all the anonymous reviewers and my reading committee members, Dieter Fox and Rajesh Rao, for providing suggestions on the previous drafts.

I would like to thank my friends, Amol Prakash, Arijit Mahalanabis, Deepak Verma, Gaurav Chanda, Harsha Madhyastha, Jaswinder Sethi, Nilesh Dalvi, Parag, Pragya Singh, Shobhit Gupta, Sumit Sanghai, Vijay Kumar, my music gurus Sharad Gadre and Ramesh Gangolli, the other members of Basanti Ka Badla cricket team, the Pratidhwani light music wing and Seattle's hindustani classical music community, and several others for providing various sources of hobbies that helped me achieve a good work-life balance.

Last but not the least, the moral and emotional support offered by my family – my mother, Dr. Ranjana Agrawal, my grandmother, Smt. Shanti Agrawal, and my aunts, Dr. Rashmi Agrawal and Dr. Poonam Agrawal, is invaluable. There are no words to do justice to all that I have received from these four ladies through my entire life. They have taught me the value of the pursuit of knowledge, an inquisitive attitude, sincerity towards work, an appreciation of the human psyche, an enhanced sense of priorities, and a faith in life. I am eternally grateful to them for making me who I am today, and who I will be in the future.

DEDICATION

To my mother
who has been my inspiration
in science, maths, and life.

Chapter 1

INTRODUCTION

Planning is the reasoning side of acting. It is an abstract, explicit deliberation process that chooses and organizes actions by anticipating their expected outcomes. . . Automated Planning is an area of Artificial Intelligence (AI) that studies this deliberation process computationally [31].

Automated planning is, therefore, a paramount area of computer science with applications to all intelligent, autonomous agents. All robots like the NASA Mars rover, assistbot assisting elderly or sick, helperbots helping us in our day to day activities, *etc.* as well as softbots, like semantic web agents, schedule planners *etc.* need automated planning as a core component for any high-level decision making.

In fact, successful and scalable algorithms for automated planning are what separates the current level of AI from its future. Much of science fiction fantasies, as well as futuristic visions of AI imagine powerful and versatile machine agents automatically making their own decisions in different and often difficult situations. Indeed, those cannot be much of a reality if the computational problem of deliberation towards a goal, a.k.a automated planning, does not make sufficient progress.

The Automated Planning problem has been studied in its many versions since the early ages of AI. The most popular formulation is *Classical Planning*, a formulation that tries to abstract out a planning problem with minimum complications by employing several assumptions. These assumptions, commonly known as classical assumptions, include static, closed and fully observable environment, perfect sensors, deterministic, instantaneous and sequential actions, and complete goal satisfaction with no reward for partial goal achievement.

In this overly restrictive model a lot of progress has been obtained over the last several years. Different techniques including state space search with efficient heuristics [9], partial

order plan search [55], planning graph search [8], planning as satisfiability [38], planning as constraint satisfaction [20], *etc.* have made significant impact towards the sizes of the classical problems that can be successfully planned for [29]. Indeed, there is now an aggressive focus on relaxing the classical assumptions, since almost no real world problem is a classical planning problem.

Recent progress achieved by planning researchers has yielded new algorithms which relax, individually, many of the classical assumptions. For example, successful temporal planners like SGPlan, SAPA, *etc.* [16, 22] are able to model actions that take time, and probabilistic planners like GPT, LAO*, SPUDD, *etc.* [11, 34, 36], although slow, can deal with actions with probabilistic outcomes, *etc.* However, in order to apply automated planning to many real-world domains we must eliminate larger groups of the assumptions in concert. For example, [14] notes that optimal control for a NASA Mars rover requires reasoning about uncertain, concurrent, durative actions and a mixture of discrete and metric fluents. While today's planners can handle large problems with *deterministic* concurrent durative actions, and MDPs provide a clear framework for *non-concurrent* durative actions in the face of uncertainty, few researchers have considered concurrent, uncertain, durative actions — the focus of this work.

As an example consider the NASA Mars rovers, Spirit and Oppurtunity. They have the goal of gathering data from different locations with various instruments (color and infrared cameras, microscopic imager, Mossbauer spectrometers *etc.*) and transmitting this data back to Earth. Concurrent actions are essential since instruments can be turned on, warmed up and calibrated, while the rover is moving, using other instruments or transmitting data. Similarly, uncertainty must be explicitly confronted as the rover's movement, arm control and other actions cannot be accurately predicted. Furthermore, all of their actions, *e.g.*, moving between locations and setting up experiments, take time. In fact, these temporal durations are themselves uncertain — the rover might lose its way and take a long time to reach another location, *etc.* To be able to solve the planning problems encountered by a rover, our planning framework needs to explicitly model all these domain constructs — concurrency, actions with uncertain outcomes and uncertain durations.

In this work we present a unified formalism that models all these domain features to-

gether. *Concurrent Markov Decision Processes* (CoMDPs) extend MDPs by allowing multiple actions per decision epoch. We use CoMDPs as the base to model all planning problems involving concurrency. Problems with durative actions, *concurrent probabilistic temporal planning* (CPTP), are formulated as CoMDPs in an extended state space. The formulation is also able to incorporate the uncertainty in durations in form of probabilistic distributions.

Solving these planning problems poses several computational challenges: concurrency, extended durations, and uncertainty in those durations all lead to explosive growth in the state space, action space and branching factor. We develop two techniques, Pruned RTDP and Sampled RTDP to address the blowup from concurrency. We also develop the "DUR" family of algorithms to handle stochastic durations. These algorithms explore different points in the running time *vs.* solution-quality tradeoff. The different algorithms propose several speedup mechanisms such as — 1) pruning of provably sub-optimal actions in a Bellman backup, 2) intelligent sampling from the action space, 3) admissible and inadmissible heuristics computed by solving non-concurrent problems, 4) hybridizing two planners to obtain a hybridized planner that finds good quality solution in intermediate running times, 5) approximating stochastic durations by their mean values and replanning, 6) exploiting the structure of multi-modal duration distributions to achieve higher quality approximations.

Moreover, we also study the MDP formulation of probabilistic planning, over which all of our algorithms are based upon. We find that MDPs, while a general model with scope for many expressiveness extensions, have severe computational limitations, since MDP algorithms behave exponentially in the number of domain features. Towards the end of scaling these algorithms we present a novel idea of hybridizing two planners — GPT and MBP. In essence, this hybridization is an instance of combining traditional, qualitative representation and algorithms (MBP) with modern, probabilistic ones (GPT). The hybridized planner, HYBPLAN, is able to obtain the benefits of both obtaining high quality solutions and low running times.

The rest of the thesis is organized as follows: In Chapter 2 we discuss the fundamentals of MDPs and the real-time dynamic programming (RTDP) solution method. In Chapter 3 we describe the model of Concurrent MDPs. Chapter 4 investigates the theoretical properties of the temporal problems. Chapter 5 explains our formulation of the CPTP problem for

deterministic durations. The algorithms are extended for the case of stochastic durations in Chapter 6. We describe our hybridized planner for MDPs in Chapter 7. Each chapter is supported with an empirical evaluation of the techniques presented in that chapter. In Chapter 8 we survey the related work in the area. We conclude with future directions of research in Chapters 9 and 10.

Chapter 2

BACKGROUND

Planning problems under probabilistic uncertainty are often modeled using Markov Decision Processes (MDPs). Different research communities have looked at slightly different formulations of MDPs. These versions typically differ in objective functions (maximizing reward *vs.* minimizing cost), horizons (finite, infinite, indefinite) and action representations (DBN *vs.* parametrized action schemata). All these formulations are very similar in nature, and so are the algorithms to solve them. Though, the methods proposed in the thesis are applicable to all the variants of these models, for clarity of explanation we assume a particular formulation, known as the *stochastic shortest path problem* [7].

We define a *Markov decision process* ($\mathcal{M}$) as a tuple $\langle \mathcal{S}, \mathcal{A}, Ap, \mathcal{P}r, \mathcal{C}, \mathcal{G}, s_0 \rangle$ in which

- $\mathcal{S}$ is a finite set of discrete states. We use factored MDPs, *i.e.*, $\mathcal{S}$ is compactly represented in terms of a set of state variables.

- $\mathcal{A}$ is a finite set of actions.

- Ap defines an applicability function. $Ap : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, denotes the set of actions that can be applied in a given state ($\mathcal{P}$ represents the power set).

- $\mathcal{P}r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function. We write $\mathcal{P}r(s'|s, a)$ to denote the probability of arriving at state s' after executing action a in state s.

- $\mathcal{C} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \Re^+$ is the cost model. We write $\mathcal{C}(s, a, s')$ to denote the cost incurred when the state s' is reached after executing action a in state s.

- $\mathcal{G} \subseteq \mathcal{S}$ is a set of absorbing goal states, *i.e.*, the process ends once one of these states is reached.

- s_0 is a start state.

We assume full observability, *i.e.*, the execution system has complete access to the new state *after* an action has been performed. We seek to find an optimal, stationary policy — *i.e.*, a function π: $\mathcal{S} \rightarrow \mathcal{A}$ that minimizes the expected cost (over an indefinite horizon) incurred to reach a goal state. A policy π and its execution starting from the start state induces an execution structure $Exec[\pi]$: a directed graph whose nodes are states from $\mathcal{S}$ and transitions are labeled with the actions performed due to π. We denote this graph to be *free of absorbing cycles* if for every non-goal node there always exists a path to reach a goal (*i.e.*, the probability to reach a goal is always greater than zero). A policy free of absorbing cycles is also known as a *proper* policy.

Note that any *cost function*, J: $\mathcal{S} \rightarrow \Re$, mapping states to the expected cost of reaching a goal state defines a *greedy policy* as follows:

$$\pi_J(s) = \underset{a \in Ap(s)}{\operatorname{argmin}} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a) \left\{ \mathcal{C}(s,a,s') + J(s') \right\} \tag{2.1}$$

The *optimal* policy derives from the optimal cost function, J^*, which satisfies the following pair of *Bellman equations*.

$$J^*(s) = 0, \text{ if } s \in \mathcal{G} \text{ else}$$
$$J^*(s) = \min_{a \in Ap(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a) \left\{ \mathcal{C}(s,a,s') + J^*(s') \right\} \tag{2.2}$$

For example, Figure 2.1 defines a simple MDP where four state variables $(x_1, \ldots, x_4)$ need to be set using toggle actions. Some of the actions, *e.g.*, toggle-x_3 are probabilistic.

2.0.1 Value Iteration and Labeled RTDP

Various algorithms have been developed to solve MDPs. *Value iteration* is a dynamic programming approach in which the optimal cost function (the solution to equations 2.2) is calculated as the limit of a series of approximations, each considering increasingly long action sequences. If $J_n(s)$ is the cost of state s in iteration n, then the cost of state s in the next iteration is calculated with a process called a *Bellman backup* as follows:

State variables : $x_1, x_2, x_3, x_4, p_{12}$

Action	Precondition	Effect	Probability
toggle-x_1	$\neg p_{12}$	$x_1 \leftarrow \neg x_1$	1
toggle-x_2	p_{12}	$x_2 \leftarrow \neg x_2$	1
toggle-x_3	true	$x_3 \leftarrow \neg x_3$	0.9
		no change	0.1
toggle-x_4	true	$x_4 \leftarrow \neg x_4$	0.9
		no change	0.1
toggle-p_{12}	true	$p_{12} \leftarrow \neg p_{12}$	1

Goal : $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1$

Figure 2.1: Probabilistic STRIPS definition of a simple MDP with potential parallelism

$$J_{n+1}(s) = \min_{a \in Ap(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a) \left\{ \mathcal{C}(s,a,s') + J_n(s') \right\} \tag{2.3}$$

Value iteration terminates when $\forall s \in \mathcal{S},\ |J_n(s) - J_{n-1}(s)| \leq \epsilon$, and this termination is guaranteed for $\epsilon > 0$. Furthermore, in the limit, the sequence of $\{J_i\}$ is guaranteed to converge to the optimal cost function, J^*, regardless of the initial values as long as a goal can be reached from every reachable state with non-zero probability. In practice, however, we can solve problems with reachable dead-ends by treating a very high J_n value as infinity. Once some states get an infinite value, that information is passed to their predecessors through Bellman backups. Thus, in practice, value iteration converges inspite of reachable dead-ends.

Unfortunately, value iteration tends to be quite slow, since it explicitly updates every state, and $|\mathcal{S}|$ is exponential in the number of domain features. One optimization restricts search to the part of state space reachable from the initial state s_0. Two algorithms exploiting this *reachability analysis* are LAO* [34] and our focus: RTDP [4].

RTDP, conceptually, is a lazy version of value iteration in which the states get updated in proportion to the frequency with which they are visited by the repeated executions of the greedy policy. An RTDP *trial* is a path starting from s_0, following the greedy policy and

updating the costs of the states visited using Bellman backups; the trial ends when a goal is reached or the number of updates exceeds a threshold. RTDP repeats these trials until convergence. Note that common states are updated frequently, while RTDP wastes no time on states that are unreachable, given the current policy. The complete convergence (at every state) is slow because less likely (but potentially important) states get updated infrequently. Furthermore, RTDP is not guaranteed to terminate.

Labeled RTDP (LRTDP) fixes these problems with a clever labeling scheme that focuses attention on states where the value function has not yet converged [10]. Specifically, states are gradually labeled *solved* signifying that the value function has converged for them. The algorithm back-propagates this information starting from the goal states and the algorithm terminates when the start state gets labeled. Labeled RTDP is guaranteed to terminate, and is guaranteed to converge to the optimal cost function (for states reachable using the optimal policy) if the initial cost function is admissible. This algorithm is implemented in GPT (the General Planning Tool) [11].

MDPs are a powerful framework to model stochastic planning domains. However, MDPs make two unrealistic assumptions — 1) all actions need to be executed sequentially, and 2) all actions are instantaneous. Unfortunately, there are many real-world domains where these assumptions are unrealistic. For example, concurrent actions are essential for a Mars rover, since instruments can be turned on, warmed up and calibrated while the rover is moving, and using other instruments for transmitting data. Moreover, the action durations are non-zero and stochastic — the rover might lose its way while navigating and may take a long time to reach its destination; it may make multiple attempts before finding the accurate arm placement. In this thesis we successively relax these two assumptions and build models and algorithms that can scale up in spite of the additional complexities imposed by the more general models.

Chapter 3

CONCURRENT MARKOV DECISION PROCESSES

We define a new model, *Concurrent MDP* (CoMDP), which allows multiple actions to be executed in parallel. This model is different from semi-MDPs and generalized state semi-MDPs [66] in that it does not incorporate action durations explicitly. CoMDPs focus on adding concurrency in an MDP framework. The input to a CoMDP is slightly different from that of an MDP – $\langle \mathcal{S}, \mathcal{A}, Ap_{\|}, \mathcal{P}r_{\|}, \mathcal{C}_{\|}, \mathcal{G}, s_0 \rangle$. The new applicability function, probability model and cost ($Ap_{\|}$, $\mathcal{P}r_{\|}$ and $\mathcal{C}_{\|}$ respectively) encode the distinction between allowing sequential executions of single actions versus the simultaneous executions of sets of actions.

3.1 The Model

The set of states ($\mathcal{S}$), set of actions ($\mathcal{A}$), goals ($\mathcal{G}$) and the start state (s_0) follow the input of an MDP. The difference lies in the fact that instead of executing only one action at a time, we may execute multiple of them. Let us define an *action combination*, A, as a set of one or more actions to be executed in parallel. With an action combination as a new unit operator available to the agent, the CoMDP takes the following new inputs

- $Ap_{\|}$ defines the new applicability function. $Ap_{\|} : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{A}))$, denotes the set of action *combinations* that can be applied in a given state.

- $\mathcal{P}r_{\|} : \mathcal{S} \times \mathcal{P}(\mathcal{A}) \times \mathcal{S} \rightarrow [0, 1]$ is the transition function. We write $\mathcal{P}r_{\|}(s'|s, A)$ to denote the probability of arriving at state s' after executing action combination A in state s.

- $\mathcal{C}_{\|} : \mathcal{S} \times \mathcal{P}(\mathcal{A}) \times \mathcal{S} \rightarrow \Re^{+}$ is the cost model. We write $\mathcal{C}_{\|}(s, A, s')$ to denote the cost incurred when the state s' is reached after executing action combination A in state s.

In essence, a CoMDP takes an action combination as a unit operator instead of a single action. Our approach is to convert a CoMDP into an equivalent MDP ($\mathcal{M}_{\|}$) given by

$\langle \mathcal{S}, \mathcal{P}(\mathcal{A}), Ap_{\|}, \mathcal{P}r_{\|}, \mathcal{C}_{\|}, \mathcal{G}, s_0 \rangle$ and solve it using the known MDP algorithms.

3.2 Case Study: CoMDP over Probabilistic STRIPS

In general a CoMDP could require an exponentially larger input than does an MDP, since the transition model, cost model and the applicability function are all defined in terms of action combinations as opposed to actions. A compact input representation for a general CoMDP is an interesting, open research question for the future. In this work, we consider a special class of compact CoMDP – one that is defined naturally via a domain description very similar to the *probabilistic STRIPS* [12] representation for MDPs.

Given a domain encoded in probabilistic STRIPS we can compute a safe set of co-executable actions. Under this safe semantics, the probabilistic dynamics gets defined in a consistent way as we describe below.

3.2.1 Applicability Function

We first discuss how to compute the sets of actions that can be executed in parallel since some actions may conflict with each other. We adopt the classical planning notion of mutual exclusion [8] and apply it to the *factored* action representation of probabilistic STRIPS. Two distinct actions are *mutex* (may not be executed concurrently) if in any state one of the following occurs:

1. they have inconsistent preconditions

2. an outcome of one action conflicts with an outcome of the other

3. the precondition of one action conflicts with the (possibly probabilistic) effect of the other.

4. the effect of one action possibly modifies a feature on which an action's transition function is conditioned upon.

Additionally, an action is never mutex with itself. In essence, non-mutex actions don't interact — the effects of executing the sequence $a_1; a_2$ equals those for $a_2; a_1$ — and so the semantics for parallel executions is clear.

Example: Continuing with Figure 2.1, toggle-x_1, toggle-x_3 and toggle-x_4 can execute in parallel but toggle-x_1 and toggle-x_2 are mutex as they have conflicting preconditions. Similarly, toggle-x_1 and toggle-p_{12} are mutex as the effect of toggle-p_{12} interferes with the precondition of toggle-x_1. If toggle-x_4's outcomes depended on toggle-x_1 then they would be mutex too, due to point 4 above. For example, toggle-x_4 toggle-x_1 will be mutex if the effect of toggle-x_4 was as follows: "if toggle-x_1 then the probability of $x_4 \leftarrow \neg x_4$ is 0.9 else 0.1". □

The applicability function is defined as the set of action-combinations, A, such that each action in A is independently applicable in s and all of the actions are pairwise non-mutex with each other. Note that pairwise concurrency is sufficient to ensure problem-free concurrency of all multiple actions in A. Formally $Ap_{\|}$ can be defined in terms of our original definition Ap as follows:

$$Ap_{\|}(s) = \{A \subseteq \mathcal{A} | \forall a, a' \in A,\ a, a' \in Ap(s) \wedge \quad \neg \mathrm{mutex(a, a')}\} \tag{3.1}$$

3.2.2 Transition Function

Let $A = \{a_1, a_2, \ldots, a_k\}$ be an action combination applicable in s. Since none of the actions are mutex, the transition function may be calculated by choosing any arbitrary order in which to apply them as follows:

$$\mathcal{P}r_{\|}(s'|s, A) = \sum_{s_1, s_2, \ldots s_k \in \mathcal{S}} \ldots \sum \mathcal{P}r(s_1|s, a_1)\mathcal{P}r(s_2|s_1, a_2) \ldots \mathcal{P}r(s'|s_{k-1}, a_k) \tag{3.2}$$

While we define the applicability function and the transition function by allowing only a *consistent* set of actions to be executable concurrently, there are alternative definitions possible. For instance, one might be willing to allow executing two actions together if the probability that they conflict is very small. A conflict may be defined as two actions asserting contradictory effects or one negating the precondition of the other. In such a case, a new state called *failure* could be created such that the system transitions to this state in case

of a conflict. And the transition may be computed to reflect a low probability transition to this failure state.

Although we impose that the model be conflict-free, most of our techniques don't actually depend on this assumption explicitly and extend to general CoMDPs.

3.2.3 Cost model

We make a small change to the probabilistic STRIPS representation. Instead of defining a single cost ($\mathcal{C}$) for each action, we define it additively as a sum of *resource* and *time* components as follows:

- Let t be the *durative* cost, *i.e.*, cost due to time taken to complete the action.
- Let r be the *resource* cost, *i.e.*, cost of resources used for the action.

Assuming additivity we can think of cost of an action $\mathcal{C}(s, a, s') = t(s, a, s') + r(s, a, s')$, to be sum of its time and resource usage. Hence, the cost model for a combination of actions in terms of these components may be defined as:

$$\mathcal{C}_{\|}(s, \{a_1, a_2, ..., a_k\}, s') = \sum_{i=1}^{k} r(s, a_i, s') + \max_{i=1..k} \{t(s, a_i, s')\} \tag{3.3}$$

For example, a Mars rover might incur lower cost when it preheats an instrument while changing locations than if it executes the actions sequentially, because the total time is reduced while the energy consumed does not change.

3.3 Solving a CoMDP with MDP algorithms

We have taken a concurrent MDP that allowed concurrency in actions and formulated it as an equivalent MDP, $\mathcal{M}_{\|}$, in an extended action space. For the rest of the paper we will use the term CoMDP to also refer to the equivalent MDP $\mathcal{M}_{\|}$.

3.3.1 Bellman equations

We extend Equations 2.2 to a set of equations representing the solution to a CoMDP:

$$J^*_{\|}(s) = 0, \text{ if } s \in \mathcal{G} \text{ else}$$
$$J^*_{\|}(s) = \min_{A \in Ap_{\|}(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r_{\|}(s'|s, A) \left\{ \mathcal{C}_{\|}(s, A, s') + J^*_{\|}(s') \right\} \tag{3.4}$$

These equations are the same as in a traditional MDP, except that instead of considering single actions for backup in a state, we need to consider all applicable action combinations. Thus, only this small change must be made to traditional algorithms (*e.g.*, value iteration, LAO*, Labeled RTDP). However, since the number of action combinations is worst-case exponential in $|\mathcal{A}|$, efficiently solving a CoMDP requires new techniques. Unfortunately, there is no easy structure to exploit, since an optimal action for a state from a classical MDP solution may not even appear in the optimal action *combination* for the associated concurrent MDP.

Theorem 1 *All actions in an optimal combination for a CoMDP ($\mathcal{M}_{\|}$) may individually be sub-optimal for the MDP $\mathcal{M}$.*

Proof: In the domain of Figure 2.1 let us have an additional action toggle-x_{34} that toggles both x_3 and x_4 with probability 0.5 and toggles exactly one of x_3 and x_4 with probability 0.25 each. Let all the actions take 1 time unit each, and therefore cost of any action combination is 1 as well. Let the start state be $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 0$ and $p_{12} = 1$. For the MDP $\mathcal{M}$ the only optimal action for the start state is toggle-x_{34}. However, for the CoMDP $\mathcal{M}_{\|}$ the optimal combination is {toggle-x_3, toggle-x_4}. □

3.4 Pruned Bellman Backups

Recall that during a trial, Labeled RTDP performs Bellman backups in order to calculate the costs of applicable actions (or in our case, action combinations) and then chooses the best action (combination); we now describe two pruning techniques that reduce the number of backups to be computed. Let $Q_{\|}(s, A)$ be the expected cost incurred by executing an action combination A in state s and then following the greedy policy, *i.e.*

$$Q_{\|_n}(s, A) = \sum_{s' \in \mathcal{S}} \mathcal{P}r_{\|}(s'|s, A) \left\{ \mathcal{C}_{\|}(s, A, s') + J_{\|_{n-1}}(s') \right\} \tag{3.5}$$

A Bellman update can thus be rewritten as:

$$J_{\|_n}(s) = \min_{A \in Ap_{\|}(s)} Q_{\|_n}(s, A) \tag{3.6}$$

3.4.1 Combo Skipping

Since the number of applicable action combinations can be exponential, we'd like to prune suboptimal combinations. The following theorem imposes a lower bound on $Q_{\|}(s, A)$ in terms of the costs and the $Q_{\|}$-values of single actions. For combo-skipping to work, the costs of the actions may depend only on the action and not the starting or ending state, *i.e.*, $\mathcal{C}(s, a, s') = \mathcal{C}(a) \forall s, s'$.

Theorem 2 *Let $A = \{a_1, a_2, \ldots, a_k\}$ be an action combination which is applicable in state s. For a CoMDP over probabilistic STRIPS, if costs are dependent only on actions and $Q_{\|_n}$ values are monotonically non-decreasing then*

$$Q_{\|}(s, A) \geq \max_{i=1..k} Q_{\|}(s, \{a_i\}) + \mathcal{C}_{\|}(A) - \left(\sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\}) \right)$$

Proof:

$$\begin{aligned} Q_{\|_n}(s, A) &= \mathcal{C}_{\|}(A) + \sum_{s'} \mathcal{P}r_{\|}(s'|s, A) J_{\|_{n-1}}(s') && \text{(using Eqn. 3.5)} \\ \Rightarrow \sum_{s'} \mathcal{P}r_{\|}(s'|s, A) J_{\|_{n-1}}(s') &= Q_{\|_n}(s, A) - \mathcal{C}_{\|}(A) && (3.7) \end{aligned}$$

$$\begin{aligned} Q_{\|_n}(s, \{a_1\}) &= \mathcal{C}_{\|}(\{a_1\}) + \sum_{s''} \Pr(s''|s, a_1) J_{\|_{n-1}}(s'') \\ &\leq \mathcal{C}_{\|}(\{a_1\}) + \sum_{s''} \Pr(s''|s, a_1) \left[\mathcal{C}_{\|}(\{a_2\}) + \sum_{s'''} \Pr(s'''|s'', a_2) J_{\|_{n-2}}(s''') \right] \\ & \qquad \text{(using Eqns. 3.5 and 3.6)} \\ &= \mathcal{C}_{\|}(\{a_1\}) + \mathcal{C}_{\|}(\{a_2\}) + \sum_{s'''} \mathcal{P}r_{\|}(s'''|s, \{a_1, a_2\}) J_{\|_{n-2}}(s''') \\ &\leq \sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\}) + \sum_{s'} \mathcal{P}r_{\|}(s'|s, A) J_{\|_{n-k}}(s') \qquad \text{(repeating for all actions in } A) \end{aligned}$$

$$= \sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\}) + [Q_{\|_{n-k+1}}(s, A) - \mathcal{C}_{\|}(A)] \qquad \text{(using Eqn. 3.7)}$$

Replacing n by $n + k - 1$

$$\begin{aligned} Q_{\|_n}(s, A) &\geq Q_{\|_{n+k-1}}(s, \{a_1\}) + \mathcal{C}_{\|}(A) - \left(\sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\})\right) \\ &\geq Q_{\|_n}(s, \{a_1\}) + \mathcal{C}_{\|}(A) - \left(\sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\})\right) \qquad \text{(monotonicity of } Q_{\|_n}) \\ &\geq \max_{i=1..k} Q_{\|_n}(s, \{a_i\}) + \mathcal{C}_{\|}(A) - \left(\sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\})\right) \end{aligned}$$

□

The proof above uses equation 3.2 from CoMDP over probabilistic STRIPS. The following corollary can be used to prune suboptimal action combinations:

Corollary 3 *Let $\lceil J_{\|_n}(s) \rceil$ be an upper bound of $J_{\|_n}(s)$. If*

$$\lceil J_{\|_n}(s) \rceil < \max_{i=1..k} Q_{\|_n}(s, \{a_i\}) + \mathcal{C}_{\|}(A) - \left(\sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\})\right)$$

then, A cannot be optimal for state s in this iteration.

Proof: Let $A_n^* = \{a_1, a_2, \ldots, a_k\}$ be the optimal combination for state s in this iteration n. Then,

$$\begin{aligned} \lceil J_{\|_n}(s) \rceil &\geq J_{\|_n}(s) \\ J_{\|_n}(s) &= Q_{\|_n}(s, A*_n) \end{aligned}$$

Combining with Theorem 2

$$\lceil J_{\|_n}(s) \rceil \geq max_{i=1..k} Q_{\|_n}(s, \{a_i\}) + \mathcal{C}_{\|}(A_n^*) - \left(\sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\})\right) \square$$

Corollary 3 justifies a pruning rule, *combo-skipping*, that preserves optimality in any iteration algorithm that maintains cost function monotonicity. This is powerful because all

Bellman-backup based algorithms preserve monotonicity when started with an admissible cost function. To apply combo-skipping, one must compute all the $Q_{\|}(s, \{a\})$ values for single actions a that are applicable in s. To calculate $\lceil J_{\|_n}(s) \rceil$ one may use the optimal combination for state s in the previous iteration (A_{opt}) and compute $Q_{\|_n}(s, A_{opt})$. This value gives an upper bound on the value $J_{\|_n}(s)$.

Example: Consider Figure 2.1. Let a single action incur unit cost, and let the cost of an action combination be: $\mathcal{C}_{\|}(A) = 0.5 + 0.5|A|$. Let state s = (1,1,0,0,1) represent the ordered values $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 0$, and $p_{12} = 1$. Suppose, after the n^{th} iteration, the cost function assigns the values: $J_{\|_n}(s) = 1$, $J_{\|_n}(s_1$=(1,0,0,0,1)) = 2, $J_{\|_n}(s_2$=(1,1,1,0,1)) = 1, $J_{\|_n}(s_3$=(1,1,0,1,1)) = 1. Let A_{opt} for state s be {toggle-x_3, toggle-x_4}. Now, $Q_{\|_{n+1}}(s, \{\text{toggle-}x_2\}) = \mathcal{C}_{\|}(\{\text{toggle-}x_2\}) + J_{\|_n}(s_1) = 3$ and $Q_{\|_{n+1}}(s, A_{opt}) = \mathcal{C}_{\|}(A_{opt}) + 0.81\times 0 + 0.09\times J_{\|_n}(s_2) + 0.09\times J_{\|_n}(s_3) + 0.01\times J_{\|_n}(s) = 1.69$. So now we can apply Corollary 3 to skip combination {toggle-x_2, toggle-x_3} in this iteration, since using toggle-x_2 as a_1, we have $\lceil J_{\|_{n+1}}(s) \rceil = Q_{\|_{n+1}}(s, A_{opt}) = 1.69 \leq 3 + 1.5 - 2 = 2.5$. □

Experiments show that combo-skipping yields considerable savings. Unfortunately, combo-skipping has a weakness — it prunes a combination for only a *single iteration.* In contrast, our second rule, *combo-elimination*, prunes irrelevant combinations altogether.

3.4.2 Combo Elimination

We adapt the action elimination theorem from traditional MDPs [7] to prove a similar theorem for CoMDPs.

Theorem 4 *Let A be an action combination which is applicable in state s. Let $\lfloor Q^*_{\|}(s, A) \rfloor$ denote a lower bound of $Q^*_{\|}(s, A)$. If $\lfloor Q^*_{\|}(s, A) \rfloor > \lceil J^*_{\|}(s) \rceil$ then A is never the optimal combination for state s.*

Proof: Because a CoMDP is an MDP in a new action space, the original proof in [7] holds after replacing an action by an 'action combination'. □

In order to apply the theorem for pruning, one must be able to evaluate the upper and lower bounds. By using an admissible cost function when starting RTDP search (or in value iteration, LAO* *etc.*), the current cost $J_{\|_n}(s)$ is guaranteed to be a lower bound of the

optimal cost; thus, $Q_{\|_n}(s, A)$ will also be a lower bound of $Q^*_{\|}(s, A)$. Thus, it is easy to compute the left hand side of the inequality. To calculate an upper bound of the optimal $J^*_{\|}(s)$, one may solve the MDP $\mathcal{M}$, *i.e.*, the traditional MDP that forbids concurrency. This is much faster than solving the CoMDP, and yields an upper bound on cost, because forbidding concurrency restricts the policy to use a strict subset of legal action combinations. Notice that combo-elimination can be used for all general MDPs and is not restricted to only CoMDPs over probabilistic STRIPS.

Example: Continuing with the previous example, let A={toggle-x_2} then $Q_{\|_{n+1}}(s, A) = \mathcal{C}_{\|}(A) + J_{\|_n}(s_1) = 3$ and $\lceil J^*_{\|}(s) \rceil = 2.222$ (from solving MDP $\mathcal{M}$). As $3 > 2.222$, A can be eliminated for state s in all remaining iterations. □

Used in this fashion, combo-elimination requires the additional overhead of optimally solving the single-action MDP $\mathcal{M}$. Since algorithms like RTDP exploit state-space reachability to limit computation to relevant states, we do this computation incrementally, as new states are visited by our algorithm.

Combo-elimination also requires computing the current value of $Q_{\|}(s, A)$ (for the lower bound of $Q^*_{\|}(s, A)$); this differs from combo-skipping which avoids this computation. However, once combo-elimination prunes a combination, it never needs to be reconsidered. Thus, there is a tradeoff: should one perform an expensive computation, hoping for long-term pruning, or try a cheaper pruning rule with fewer benefits? Since Q-value computation is the costly step, we adopt the following heuristic: "First, try combo-skipping; if it fails to prune the combination, attempt combo-elimination; if it succeeds, never consider it again". We also tried implementing some other heuristics, such as: 1) If some combination is being skipped repeatedly, then try to prune it altogether with combo-elimination. 2) In every state, try combo-elimination with probability p. Neither alternative performed significantly better, so we kept our original (lower overhead) heuristic.

Since combo-skipping does not change any step of labeled RTDP and combo-elimination removes provably sub-optimal combinations, *pruned* labeled RTDP maintains convergence, termination, optimality and efficiency, when used with an admissible heuristic.

3.5 Sampled Bellman Backups

Since the fundamental challenge posed by CoMDPs is the explosion of action combinations, sampling is a promising method to reduce the number of Bellman backups required per state. We describe a variant of RTDP, called *sampled RTDP*, which performs backups on a random set of action combinations[1], choosing from a distribution that favors combinations that are *likely to be optimal*. We generate our distribution by:

1. using combinations that were previously discovered to have low $Q_{\|}$-values (recorded by *memoizing* the best combinations per state, after each iteration)

2. calculating the $Q_{\|}$-values of all applicable single actions (using current cost function) and then biasing the sampling of combinations to choose the ones that contain actions with low $Q_{\|}$-values.

Algorithm 1 Sampled Bellman Backup(state, m) *//returns the best combination found*

```
1: list l = ∅      //a list of all applicable actions with their values
2: for all action ∈ A do
3:     compute Q_||(state, {action})
4:     insert ⟨a, 1/Q_||(state, {action})⟩ in l
5: for all i ∈ [1..m] do
6:     newcomb = SampleComb(state, i, l);
7:     compute Q_||(state, newcomb)
8: clear memoizedlist[state]
9: compute Q_min as the minimum of all Q_|| values computed in line 7
10: store all combinations A with Q_||(state, A) = Q_min in memoizedlist[state]
11: return the first entry in memoizedlist[state]
```

This approach exposes an exploration / exploitation trade-off. Exploration, here, refers to testing a wide range of action combinations to improve understanding of their relative merit.

[1] A similar action sampling approach was also used in the context of space shuttle scheduling to reduce the number of actions considered during value function computation [67].

Function 2 SampleComb(state, i, l) *//returns i^{th} combination for the sampled backup*

if $i \leq$ size(memoizedlist[state]) **then**
  return i^{th} entry in memoizedlist[state] *//return the combination memoized in previous iteration*
newcomb = $\emptyset$
repeat
  randomly sample an action a from l proportional to its value
  insert a in newcomb
  remove all actions mutex with a from l
  if l is empty **then**
    done = true
  else if |newcomb| == 1 **then**
    done = false *//sample at least 2 actions per combination*
  else
    done = true with prob. $\frac{|\text{newcomb}|}{|\text{newcomb}|+1}$
until done
return newcomb

Exploitation, on the other hand, advocates performing backups on the combinations that have previously been shown to be the best. We manage the tradeoff by carefully maintaining the distribution over combinations. First, we only memoize best combinations per state; these are always backed-up in a Bellman update. Other combinations are constructed by an incremental probabilistic process, which builds a combination by first randomly choosing an initial action (weighted by its individual $Q_{\|}$-value), then deciding whether to add a non-mutex action or stop growing the combination. There are many implementations possible for this high level idea. We tried several of those and found the results to be very similar in all of them. Algorithm 1 describes the implementation used in our experiments. The algorithm takes a state and a total number of combinations m as an input and returns the best combination obtained so far. It also memoizes all the best combinations for the state in memoizedlist. Function 2 is a helper function that returns the i^{th} combination that is either one of the best combinations memoized in the previous iteration or a new sampled

combination. Also notice line 10 in Function 2. It forces the sampled combinations to be at least size 2, since all individual actions have already been backed up (line 3 of Algo 1).

3.5.1 Termination and Optimality

Since the system doesn't consider every possible action combination, sampled RTDP is not guaranteed to choose the best combination to execute at each state. As a result, even when started with an admissible heuristic, the algorithm may assign $J_{\|_n}(s)$ a cost that is greater than the optimal $J^*_{\|}(s)$ — *i.e.*, the $J_{\|_n}(s)$ values are no longer admissible. If a better combination is chosen in a subsequent iteration, $J_{\|_{n+1}}(s)$ might be set a lower value than $J_{\|_n}(s)$, thus sampled RTDP is not *monotonic*. This is unfortunate, since admissibility and monotonicity are important properties required for termination[2] and optimality in labeled RTDP; indeed, sampled RTDP loses these important theoretical properties. The good news is that it is extremely useful in practice. In our experiments, sampled RTDP usually terminates quickly, and returns costs that are extremely close to the optimal.

3.5.2 Improving Solution Quality

We have investigated several heuristics in order to improve the quality of the solutions found by sampled RTDP. Our heuristics compensate for the errors due to partial search and lack of admissibility.

- **Heuristic 1:** Whenever sampled RTDP asserts convergence of a state, do not immediately label it as converged (which would preclude further exploration [10]); instead first run a complete backup phase, using all the admissible combinations, to rule out any easy-to-detect inconsistencies.

- **Heuristic 2:** Run sampled RTDP to completion, and use the cost function it produces, $J^s()$, as the initial heuristic estimate, $J_0()$, for a subsequent run of pruned RTDP. Usually, such a heuristic, though inadmissible, is highly informative. Hence, pruned RTDP terminates quite quickly.

[2] To ensure termination we implemented the policy: *if number of trials exceeds a threshold, force monotonicity on the cost function.* This will achieve termination but will reduce quality of solution.

- **Heuristic 3:** Run sampled RTDP before pruned RTDP, as in Heuristic 2, except instead of using the $J^s()$ cost function directly as an initial estimate, scale linearly downward — *i.e.*, use $J_0() := cJ^s()$ for some constant $c \in (0,1)$. Hopefully, the estimate will be admissible (though there is no guarantee). In our experience, $c = 0.9$ suffices, and the run of pruned RTDP yields the optimal policy very quickly.

Experiments showed that Heuristic 1 returns a cost function that is close to optimal. Adding Heuristic 2 improves this value moderately, and a combination of Heuristics 1 and 3 invariably returns the optimal solution.

3.6 Experiments: Concurrent MDP

Concurrent MDP is our fundamental formulation, modeling concurrent actions in a general planning domain. We first compare the various techniques to solve CoMDPs, *viz.*, pruned and sampled RTDP. In following chapters we use these techniques to model problems with durative actions.

We tested our algorithms on CoMDPs over probabilistic STRIPS in three domains. The first domain was a probabilistic variant of NASA Rover domain from the 2002 AIPS Planning Competition, in which there are multiple objects to be photographed and various rocks to be tested with resulting data communicated back to the base station. Cameras need to be focused, and arms need to be positioned before usage. Since the rover has multiple arms and multiple cameras, the domain is highly parallel. The cost function includes both resource and time components, so executing multiple actions in parallel is cheaper than executing them sequentially. We generated problems with 20-30 state variables having up to 81,000 reachable states and the average number of applicable combinations per state, $Avg(Ap(s))$, which measures the amount of concurrency in a problem, is up to 2735.

We also tested on a probabilistic version of a machineshop domain with multiple subtasks (*e.g.*, roll, shape, paint, polish *etc.*), which need to be performed on different objects using different machines. Machines can perform in parallel, but not all are capable of every task. We tested on problems with 26-28 state variables and around 32,000 reachable states. $Avg(Ap(s))$ ranged between 170 and 2640 on the various problems.

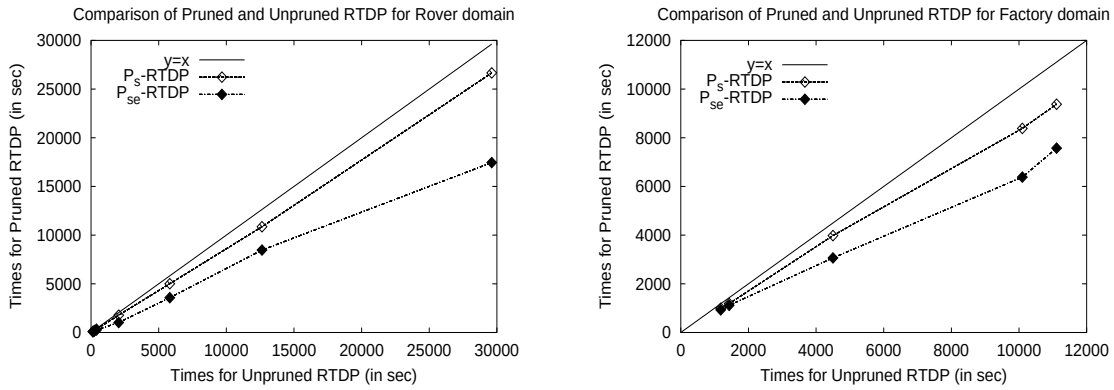

Figure 3.1: (a,b): Pruned vs. Unpruned RTDP for Rover and MachineShop domains respectively. Pruning non-optimal combinations achieves significant speedups on larger problems.

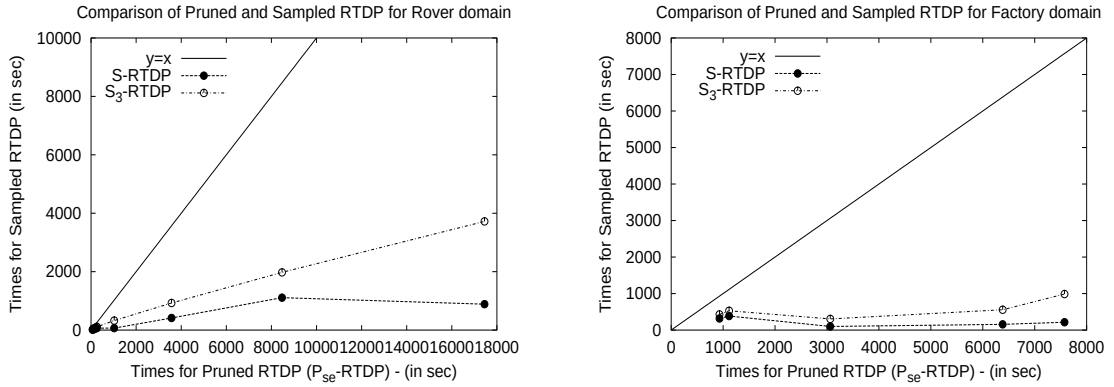

Figure 3.2: (a,b): Sampled vs Pruned RTDP for Rover and MachineShop domains respectively. Random sampling of action combinations yields dramatic improvements in running times.

Finally, we tested on an artificial domain similar to Figure 2.1 but much more complex. In this domain, some Boolean variables need to be toggled; however, toggling is probabilistic in nature. Moreover, certain pairs of actions have conflicting preconditions and thus, by varying the number of mutex actions we may control the domain's degree of parallelism. All the problems in this domain had 19 state variables and about 32,000 reachable states, with $Avg(Ap(s))$ between 1024 and 12287.

We used Labeled RTDP, as implemented in GPT [11], as the base MDP solver. It is implemented in C++. We implemented[3] various algorithms, unpruned RTDP (U-RTDP), pruned RTDP using only combo skipping (P_s-RTDP), pruned RTDP using both combo skipping and combo elimination (P_{se}-RTDP), sampled RTDP using Heuristic 1 (S-RTDP) and sampled RTDP using both Heuristics 1 and 3, with value functions scaled with 0.9 (S_3-RTDP). We tested all of these algorithms on a number of problem instantiations from our three domains, generated by varying the number of objects, degrees of parallelism, and distances to goal. The experiments were performed on a 2.8 GHz Pentium processor with a 2 GB RAM.

We observe (Figure 3.1(a,b)) that pruning significantly speeds the algorithm. But the comparison of P_{se}-RTDP with S-RTDP and S_3-RTDP (Figure 3.2(a,b)) shows that sampling has a dramatic speedup with respect to the pruned versions. In fact, pure sampling, S-RTDP, converges extremely quickly, and S_3-RTDP is slightly slower. However, S_3-RTDP is still much faster than P_{se}-RTDP. The comparison of qualities of solutions produced by S-RTDP and S_3-RTDP *w.r.t.* optimal is shown in Table 3.1. We observe that solutions produced by S-RTDP are always nearly optimal. Since the error of S-RTDP is small, scaling it by 0.9 makes it an admissible initial cost function for the pruned RTDP; indeed, in all experiments, S_3-RTDP produced the optimal solution.

Figure 3.3(a,b) demonstrates how running times vary with problem size. We use the product of the number of reachable states and the average number of applicable action combinations per state as an estimate of the size of the problem (the number of reachable states in all artificial domains is the same, hence the x-axis for Figure 3.3(b) is $Avg(Ap(s))$).

---

[3]The code may be downloaded at *http://www.cs.washington.edu/ai/comdp/comdp.tgz*

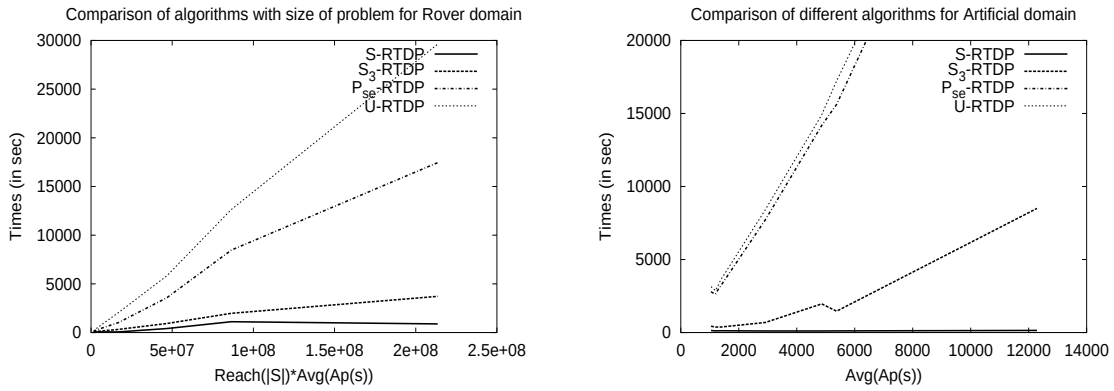

Figure 3.3: (a,b): Comparison of different algorithms with size of the problems for Rover and Artificial domains. As the problem size increases, the gap between sampled and pruned approaches widens considerably.

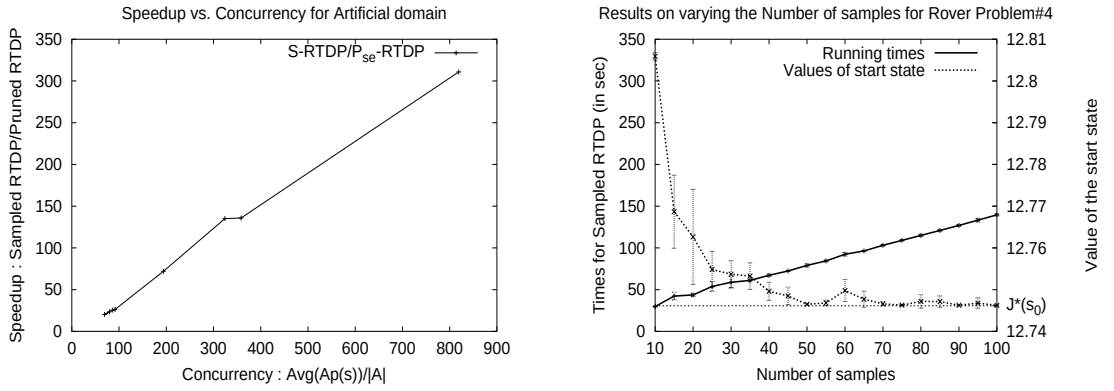

Figure 3.4: (a): Relative Speed vs. Concurrency for Artificial domain. (b) : Variation of quality of solution and efficiency of algorithm (with 95% confidence intervals) with the number of samples in Sampled RTDP for one particular problem from the Rover domain. As number of samples increase, the quality of solution approaches optimal and time still remains better than P_{se}-RTDP (which takes 259 sec. for this problem).

Table 3.1: Quality of solutions produced by Sampled RTDP

Problem	$J(s_0)$ (S-RTDP)	$J^*(s_0)$ (Optimal)	Error
Rover1	10.7538	10.7535	$<$0.01%
Rover2	10.7535	10.7535	0
Rover3	11.0016	11.0016	0
Rover4	12.7490	12.7461	0.02%
Rover5	7.3163	7.3163	0
Rover6	10.5063	10.5063	0
Rover7	12.9343	12.9246	0.08%
Artificial1	4.5137	4.5137	0
Artificial2	6.3847	6.3847	0
Artificial3	6.5583	6.5583	0
MachineShop1	15.0859	15.0338	0.35%
MachineShop2	14.1414	14.0329	0.77%
MachineShop3	16.3771	16.3412	0.22%
MachineShop4	15.8588	15.8588	0
MachineShop5	9.0314	8.9844	0.56%

From these figures, we verify that the number of applicable combinations plays a major role in the running times of the concurrent MDP algorithms. In Figure 3.4(a), we fix all factors and vary the degree of parallelism. We observe that the speedups obtained by S-RTDP increase as concurrency increases. This is a very encouraging result, and we can expect S-RTDP to perform well on large problems involving high concurrency, even if the other approaches fail.

In Figure 3.4(b), we present another experiment in which we vary the number of action combinations sampled in each backup. While solution quality is inferior when sampling only a few combinations, it quickly approaches the optimal on increasing the number of samples. In all other experiments we sample 40 combinations per state.

Chapter 4

CHALLENGES FOR TEMPORAL PLANNING

While the CoMDP model is powerful enough to model concurrency in actions, it still assumes each action to be instantaneous. We now incorporate actual action durations in the modeling the problem. This is essential to increase the scope of current models to real world domains.

Before we present our model and the algorithms we discuss several new theoretical challenges imposed by explicit action durations. Note that the results in this chapter apply to a wide range of planning problems:

- regardless of whether durations are uncertain or fixed
- regardless of whether effects are stochastic or deterministic.

Actions of uncertain duration are modeled by associating a distribution (possibly conditioned on the outcome of stochastic effects) over execution times. We focus on problems whose objective is to achieve a goal state while minimizing total expected time (*make-span*), but our results extend to cost functions that combine make-span and resource usage. This raises the question of *when* a goal counts as achieved. We require that:

Assumption 1 *All executing actions terminate before the goal is considered achieved.*

Assumption 2 *An action, once started, cannot be terminated prematurely.*

We start by asking the question "Is there a restricted set of time points such that optimality is preserved even if actions are started only at these points?"

Definition *Any time point when a new action is allowed to start execution is called a* decision epoch. *A time point is a* pivot *if it is either 0 or a time when a new effect might occur (e.g., the end of an action's execution) or a new precondition may be needed or an*

existing precondition may no longer be needed. A happening *is either 0 or a time when an effect actually occurs or a new precondition is definitely needed or an existing precondition is no longer needed.*

Intuitively, a happening is a point where a change in the world state or action constraints actually "happens" (*e.g.*, by a new effect or a new precondition). When execution crosses a pivot (a possible happening), information is gained by the agent's execution system (*e.g.*, did or didn't the effect occur) which may "change the direction" of future action choices. Clearly, if action durations are deterministic, then the set of pivots is the same as the set of happenings.

Example: Consider an action a whose durations follow a uniform integer duration between 1 and 10. If it is started at time 0 then all timepoints 0, 1, 2,..., 10 are pivots. If in a certain execution it finishes at time 4 then 4 (and 0) is a happening (for this execution). □

The planning community has developed an expressive domain description language called $\text{PDDL}_{2.1}$ [27]. This language allows effects to occur instantaneously at beginning or end of an action, and the conditions to hold either at the beginning, or at the end or over the whole execution of the action. Indeed, many other complex domains that allow intermediate action effects or preconditions may be compiled in a $\text{PDDL}_{2.1}$ language [28]. We define a $\text{PDDL}_{2.1}$ action more formally, as follows:

Definition *An action is a $PDDL_{2.1}$ action if the following hold:*

- *The effects are realized instantaneously either* (at start) *or* (at end)*, i.e., at the beginning or the at the completion of the action (respectively).*

- *The preconditions may need to hold instaneously before the start* (at start)*, before the end* (at end) *or over the complete execution of the action* (over all)*.*

We now prove that the set of pivots as decision epochs is insufficient for a $\text{PDDL}_{2.1}$ planning problem.

Theorem 5 *For a $PDDL_{2.1}$ domain restricting decision epochs to pivots causes incompleteness (i.e., a problem may be incorrectly deemed unsolvable).*

```
:action a
  :duration 4
  :condition (over all P) (at end Q)
  :effect (at end Goal)
:action b
  :duration 2
  :effect (at start Q) (at end (not P))
```

Figure 4.1: A domain to illustrate that an expressive action model may require arbitrary decision epochs for a solution. In this example, b needs to start at 3 units after a's execution to reach *Goal*.

Proof: Consider the deterministic temporal planning domain in Figure 4.1 that uses $\text{PDDL}_{2.1}$ notation [27]. If the initial state is P=true and Q=false, then the only way to reach *Goal* is to start a at time t (*e.g.*, 0), and b at some timepoint in the open interval $(t+2, t+4)$. Clearly, no new information is gained at any of the time points in this interval and none of them is a pivot. Still, they are required for solving the problem. □

Intuitively, the instantaneous start and end effects of two $\text{PDDL}_{2.1}$ actions may require a certain relative alignment within them to achieve the goal. This alignment may force one action to start somewhere (possibly at a non-pivot point) in the midst of the other's execution, thus requiring intermediate decision epochs to be considered.

Temporal planners may be classified as having one of two architectures: constraint-posting approaches in which the times of action execution are gradually constrained during planning (*e.g.*, Zeno and LPG [56, 30]) and extended state-space methods (*e.g.*, TP4 and SAPA [35, 21]). Theorem 5 holds for both architectures but has strong computational implications for state-space planners because limiting attention to a subset of decision epochs can speed these planners. (The theorem also shows that planners like SAPA and Prottle [42] are incomplete.) Fortunately, an assumption restricts the set of decision epochs considerably.

Definition *An action is a TGP-style action*[1] *if all of the following hold:*

[1] While the original TGP [62] considered only deterministic actions of fixed duration, we use the phrase

- *The effects are realized at some unknown point during action execution, and thus can be used only once the action has completed.*

- *The preconditions must hold at the beginning of an action.*

- *The preconditions and the features on which its transition function is conditioned must remain unchanged while the action is being executed, unless the action itself is modifying them.*

Thus, two TGP-style actions may not execute concurrently if they clobber each other's preconditions or effects. For the case of TGP-style actions the set of happenings is nothing but the set of time points when some action terminates. TGP pivots are the set of points when an action might terminate. (Of course both these sets additionally include zero).

Theorem 6 *If all actions are TGP-style, then the set of decision epochs may be restricted to pivots without sacrificing completeness or optimality.*

Proof Sketch: By contradiction. Suppose that no optimal policy satisfies the theorem; then there must exist a path through the optimal policy in which one must start an action, a, at time t even though there is no action which could have terminated at t. Since the planner hasn't gained any information at t, a case analysis (which requires actions to be TGP-style) shows that one could have started a earlier in the execution path without increasing the make-span. The detailed proof is discussed in the Appendix A. □

In the case of deterministic durations, the set of happenings is same as the set of pivots; hence the following corollary holds:

Corollary 7 *If all actions are TGP-style with deterministic durations, then the set of decision epochs may be restricted to happenings without sacrificing completeness or optimality.*

When planning with uncertain durations there may be a huge number of pivots; it is useful to further constrain the range of decision epochs.

"TGP-style" in a more general way, without these restrictions

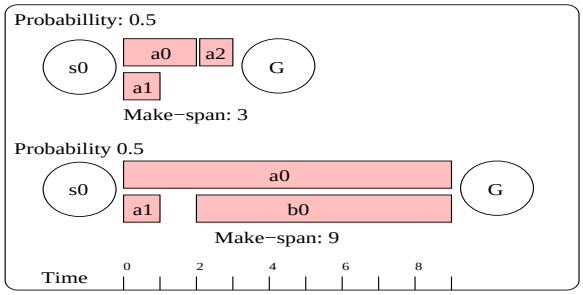

Figure 4.2: Pivot decision epochs are necessary for optimal planning in face of nonmonotonic continuation. In this domain, *Goal* can be achieved by $\langle\{a_0, a_1\}; a_2\rangle$ or $\langle b_0\rangle$; a_0 has duration 2 or 9; and b_0 is mutex with a_1. The optimal policy starts a_0 and then, if a_0 does *not* finish at time 2, it starts b_0 (otherwise it starts a_1).

Definition *An action has* independent duration *if there is no correlation between its probabilistic effects and its duration.*

Definition *An action has* monotonic continuation *if the expected time until action termination is nonincreasing during execution.*

Actions without probabilistic effects, by nature, have independent duration. Actions with monotonic continuations are common, *e.g.* those with uniform, exponential, Gaussian, and many other duration distributions. However, actions with bimodal or multi-modal distributions don't have monotonic continuations (for an example, see Figure 9.1s.

Conjecture 8 *If all actions are TGP-style, have independent duration and monotonic continuation, then the set of decision epochs may be restricted to happenings without sacrificing completeness or optimality.*

If an action's continuation is nonmonotonic then failure to terminate can increase the expected time remaining and cause another sub-plan to be preferred (see Figure 4.2). Similarly, if an action's duration isn't independent then failure to terminate changes the probability of its eventual effects and this may prompt new actions to be started.

By exploiting these theorems and conjecture we may significantly speed planning since we are able to limit the number of decision epochs needed for decision-making. We use this theoretical understanding in our models. First, for simplicity, we consider only the case of

TGP-style actions with deterministic durations. In Chapter 6, we relax this restriction by allowing stochastic durations, both unimodal as well as multimodal.

Chapter 5

TEMPORAL PLANNING WITH DETERMINISTIC DURATIONS

We use the abbreviation *CPTP* (short for *Concurrent Probabilistic Temporal Planning* to refer to the probabilistic planning problem with durative actions. A CPTP problem has an input model similar to that of CoMDPs except that action costs ($\mathcal{C}(s, a, s')$) are replaced by their *deterministic* durations ($\Delta(a)$), *i.e.*, the input is of the form $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}r, \Delta, \mathcal{G}, s_0 \rangle$. We study the objective of minimizing the expected time (*make-span*) of reaching a goal. From now, we make the following assumptions:

Assumption 3 *All action durations are integer-valued.*

This assumption has a negligible effect on expressiveness because one can convert a problem with rational durations into one that abides Assumption 3 by scaling all durations by the g.c.d. of the denominators. In case of irrational durations, one can always find an arbitrarily close approximation to the original problem by approximating the irrational durations by rational numbers.

For reasons discussed in the previous chapter we adopt the TGP temporal action model of [62], rather than the more complex $\text{PDDL}_{2.1}$ [27]. Specifically,

Assumption 4 *All actions follow the TGP model.*

These restrictions are consistent with our previous definition of concurrency. Specifically, the mutex definitions (of CoMDPs over probabilistic STRIPS) hold and are required under these assumptions. As an illustration, consider Figure 5.1. It describes a situation in which two actions with interfering preconditions and effects can not be executed concurrently. To see why not, suppose initially p_{12} was false and two actions toggle-x_1 and toggle-p_{12} were started at time 2 and 4, respectively. As $\neg p_{12}$ is a precondition of toggle-x_1, whose duration is 5, it needs to remain false until time 7. But toggle-p_{12} may produce its effects anytime

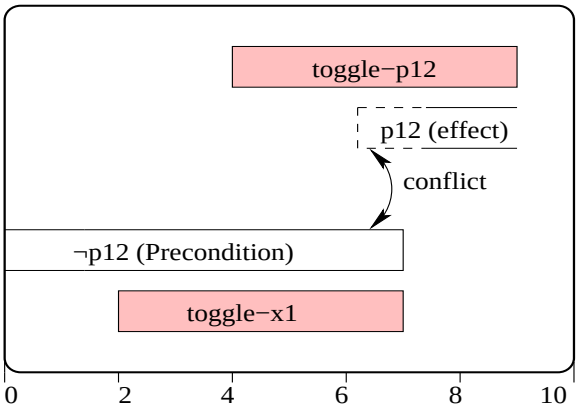

Figure 5.1: A sample execution demonstrating conflict due to interfering preconditions and effects. (The actions are shaded to disambiguate them with preconditions and effects)

between 4 and 9, which may conflict with the preconditions of the other executing action. Hence, we forbid the concurrent execution of toggle-x_1 and toggle-p_{12} to ensure a completely predictable outcome distribution.

Because of this definition of concurrency, the dynamics of our model remains consistent with Equation 3.2. Thus the techniques developed for CoMDPs derived from probabilistic STRIPS actions may be used.

5.1 Formulation as a CoMDP

We can model a CPTP problem as a CoMDP, and thus as an MDP, in more than one way. We list the two prominent formulations below. Our first formulation, *aligned epoch* CoMDP models the problem approximately but solves it quickly. The second formulation, *interleaved epochs* models the problem exactly but results in a larger state space and hence takes longer to solve using existing techniques. In subsequent sections we explore ways to speed up policy construction for the interleaved epoch formulation.

5.1.1 Aligned Epoch Search Space

A simple way to formulate CPTP is to model it as a standard CoMDP over probabilistic STRIPS, in which action costs are set to their durations and the cost of a combination is

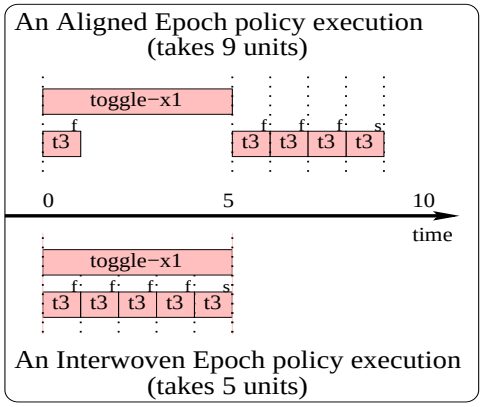

Figure 5.2: Comparison of times taken in a sample execution of an interwoven-epoch policy and an aligned-epoch policy. In both trajectories the toggle-x_3 (t3) action fails four times before succeeding. Because the aligned policy must wait for all actions to complete before starting any more, it takes more time than the interwoven policy, which can start more actions in the middle.

the maximum duration of the constituent actions (as in Equation 3.3). This formulation introduces a substantial approximation to the CPTP problem. While this is true for deterministic domains too, we illustrate this using our example involving stochastic effects. Figure 5.2 compares the trajectories in which the toggle-x_3 (t3) actions fails for four consecutive times before succeeding. In the figure, "f" and "s" denote failure and success of uncertain actions, respectively. The vertical dashed lines represent the time-points when an action is started.

Consider the actual executions of the resulting policies. In the aligned-epoch case (Figure 5.2 top), once a combination of actions is started at a state, the next decision can be taken only when the effects of *all actions* have been observed (hence the name *aligned-epochs*). In contrast, Figure 5.2 bottom) shows that at a decision epoch in the optimal execution for a CPTP problem, many actions may be midway in their execution. We have to explicitly take into account these actions and their remaining execution times when making a subsequent decision. Thus, the actual state space for CPTP decision making is substantially different from that of the simple aligned-epoch model.

Note that due to Corollary 7 it is sufficient to consider a new decision epoch only at a happening, *i.e.*, a time-point when one or more actions complete. Thus, using Assumption 3

we infer that these decision epochs will be discrete (integer). Of course, not *all* optimal policies will have this property. But it is easy to see that there exists at least one optimal policy in which each action begins at a happening. Hence our search space reduces considerably.

5.1.2 Interwoven Epoch Search Space

We adapt the search space representation of [35], which is similar to that of [3, 21]. Our original state space $\mathcal{S}$ in Chapter 2 is augmented by including the set of actions currently executing and the times passed since they were started. Formally, let the new interwoven state[1] $s \in \mathcal{S}_{\underline{=}}$ be an ordered pair $\langle X, Y \rangle$ where:

- $X \in \mathcal{S}$
- $Y = \{(a, \delta) | a \in \mathcal{A}, 0 \leq \delta < \Delta(a)\}$

Here X represents the values of the state variables (*i.e.* X is a state in the original state space) and Y denotes the set of ongoing actions "a" and the times passed since their start "δ". Thus the overall interwoven-epoch search space is $\mathcal{S}_{\underline{=}} = \mathcal{S} \times \bigotimes_{a \in \mathcal{A}} \Big(\{a\} \times Z_{\Delta(a)} \Big)$, where $Z_{\Delta(a)}$ represents the set $\{0, 1, \ldots, \Delta(a) - 1\}$ and $\bigotimes$ denotes the Cartesian product over multiple sets.

Also define A_s to be the set of actions already in execution. In other words, A_s is a projection of Y ignoring execution times in progress:

$$A_s = \{a | (a, \delta) \in Y \wedge s = \langle X, Y \rangle\}$$

Example: Continuing our example with the domain of Figure 5.3, suppose state s_1 has all state variables false, and suppose the action toggle-x_1 was started 3 units ago from the current time. Such a state would be represented as $\langle X_1, Y_1 \rangle$ with X_1=(F, F, F, F, F) and Y_1=$\{$(toggle-x_1,3)$\}$ (the five state variables are listed in the order: x_1, x_2, x_3, x_4 and p_{12}). The set A_{s_1} would be $\{$toggle-$x_1\}$.

---

[1] We use the subscript $_{\underline{=}}$ to denote the *interwoven* state space ($\mathcal{S}_{\underline{=}}$), value function ($J_{\underline{=}}$), *etc.*.

State variables : $x_1, x_2, x_3, x_4, p_{12}$

Action	$\Delta(a)$	Precondition	Effect	Probability
toggle-x_1	5	$\neg p_{12}$	$x_1 \leftarrow \neg x_1$	1
toggle-x_2	5	p_{12}	$x_2 \leftarrow \neg x_2$	1
toggle-x_3	1	true	$x_3 \leftarrow \neg x_3$	0.9
			no change	0.1
toggle-x_4	1	true	$x_4 \leftarrow \neg x_4$	0.9
			no change	0.1
toggle-p_{12}	5	true	$p_{12} \leftarrow \neg p_{12}$	1

Goal : $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1$

Figure 5.3: The domain of Example 2.1 extended with action durations.

To allow the possibility of simply waiting for some action to complete execution, that is, deciding at a decision epoch not to start any additional action, we augment the set $\mathcal{A}$ with a *no-op* action, which is applicable in all states $s = \langle X, Y \rangle$ where $Y \neq \emptyset$ (*i.e.* states in which some action is still being executed). For a state s, the no-op action is mutex with all non-executing actions, *i.e.*, those in $\mathcal{A} \setminus A_s$. In other words, at any decision epoch either a no-op will be started or any combination not involving no-op. We define no-op to have a variable duration[2] equal to the time after which another already executing action completes ($\delta_{next}(s, A)$ as defined below).

The interwoven applicability set can be defined as:

$$\underline{\underline{Ap}}(s) = \begin{cases} Ap_{\parallel}(X) \text{ if } Y = \emptyset \text{ else} \\ \{noop\} \cup \{A | A \cup A_s \in Ap_{\parallel}(X) \text{ and } A \cap A_s = \emptyset\} \end{cases}$$

Transition Function: We also need to define the probability transition function, $\underline{\underline{\mathcal{P}r}}$, for the interwoven state space. At some decision epoch let the agent be in state $s = (X, Y)$. Suppose that the agent decides to execute an action combination A. Define Y_{new} as the set similar to Y but consisting of the actions just starting; formally $Y_{new} = \{(a, \Delta(a)) | a \in A\}$. In this system, the next decision epoch will be the next time that an executing action ter-

[2] A precise definition of the model will create multiple no-op$_t$ actions with different constant durations t and the no-op$_t$ applicable in an interwoven state will be the one with $t = \delta_{next}(s, A)$.

minates. Let us call this time $\delta_{next}(s, A)$. Notice that $\delta_{next}(s, A)$ depends on both executing and newly started actions. Formally,

$$\delta_{next}(s, A) = \min_{(a,\delta) \in Y \cup Y_{new}} \Delta(a) - \delta$$

Moreover, multiple actions may complete simultaneously. Define $A_{next}(s, A) \subseteq A \cup A_s$ to be the set of actions that will complete exactly in $\delta_{next}(s, A)$ timesteps. The Y-component of the state at the decision epoch after $\delta_{next}(s, A)$ time will be

$$Y_{next}(s, A) = \{(a, \delta + \delta_{next}(s, A)) | (a, \delta) \in Y \cup Y_{new}, \Delta(a) - \delta > \delta_{next}(s, A)\}$$

Let $s{=}\langle X, Y \rangle$ and let $s'{=}\langle X', Y' \rangle$. The transition function for CPTP can now be defined as:

$$\mathcal{P}r_{\underline{\underline{\ }}}(s'|s, A){=}\begin{cases} \mathcal{P}r_{\|}(X'|X, A_{next}(s, A)) \text{ if } Y'{=}Y_{next}(s, A) \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

In other words, executing an action combination A in state $s = \langle X, Y \rangle$ takes the agent to a decision epoch $\delta_{next}(s, A)$ ahead in time, specifically to the first time when some combination $A_{next}(s, A)$ completes. This lets us calculate $Y_{next}(s, A)$: the new set of actions still executing with their times elapsed. Also, because of TGP-style actions, the probability distribution of different state variables is modified independently. Thus the probability transition function due to CoMDP over probabilistic STRIPS can be used to decide the new distribution of state variables, as if the combination $A_{next}(s, A)$ were taken in state X.

Example: Continuing with the previous example, let the agent in state s_1 execute the action combination $A = \{\text{toggle-}x_4\}$. Then $\delta_{next}(s_1, A) = 1$, since toggle-$x_4$ will finish the first. Thus, $A_{next}(s_1, A){=} \{\text{toggle-}x_4\}$. $Y_{next}(s_1, A) = \{(\text{toggle-}x_1, 4)\}$. Hence, the probability distribution of states after executing the combination A in state s_1 will be

- $((F, F, F, T, F), Y_{next}(s_1, A))$ probability $= 0.9$
- $((F, F, F, F, F), Y_{next}(s_1, A))$ probability $= 0.1$

Start and Goal States: In the interwoven space, the start state is $\langle s_0, \emptyset \rangle$ and the new set of goal states is $\mathcal{G}_{\underline{\underline{\ }}} = \{\langle X, \emptyset \rangle | X \in \mathcal{G}\}$.

By redefining the start and goal states, the applicability function, and the probability transition function, we have finished modeling a CPTP problem as a CoMDP in the interwoven state space. Now we can use the techniques of CoMDPs (and MDPs as well) to solve our problem. In particular, we can use our Bellman equations as described below.

Bellman Equations: The set of equations for the solution of a CPTP problem can be written as:

$$\underline{\underline{J}}^*(s) = 0, \text{ if } s \in \underline{\underline{\mathcal{G}}} \text{ else} \tag{5.1}$$

$$\underline{\underline{J}}^*(s) = \min_{A \in \underline{\underline{Ap}}(s)} \left\{ \delta_{next}(s, A) + \sum_{s' \in \underline{\underline{\mathcal{S}}}} \underline{\underline{\mathcal{P}r}}(s'|s, A) \underline{\underline{J}}^*(s') \right\}$$

We will use DUR_{samp} to refer to the sampled RTDP algorithm over this search space. The main bottleneck in naively inheriting algorithms like DUR_{samp} is the huge size of the interwoven state space. In the worst case (when all actions can be executed concurrently) the size of the state space is $|\mathcal{S}| \times (\prod_{a \in \mathcal{A}} \Delta(a))$. We get this bound by observing that for each action a, there are $\Delta(a)$ number of possibilities: either a is not executing or it is and has remaining times $1, 2, \ldots, \Delta(a) - 1$.

Thus we need to reduce or abstract/aggregate our state space in order to make the problem tractable. We now present several heuristics which can be used to speed the search.

5.2 Heuristics

We present both two admissible and one inadmissible heuristics that can be used as the initial cost function for DUR_{samp} algorithm. The first heuristic (maximum concurrency) solves the underlying MDP and is thus quite efficient to compute. However it is typically less informative than our second heuristic (eager effects) which requires the solution of a relaxed CoMDP in a state space larger than the underlying MDP state space. The inadmissible heuristic (average concurrency) is very efficient to compute, and also tends to be more informed than the maximum concurrency heuristic.

5.2.1 Maximum Concurrency Heuristic

We prove that the optimal expected cost in a traditional (serial) MDP divided by the maximum number of actions that can be executed in parallel is a lower bound for the expected make-span of reaching a goal in a CPTP problem. Let $J(X)$ denote the value of a state $X \in \mathcal{S}$ in a traditional MDP with costs of an action equal to its duration. Let $Q(X, A)$ denote the expected cost to reach the goal if initially all actions in the combination A are executed and the greedy serial policy is followed thereafter. Formally, $Q(X, A) = \sum_{X' \in \mathcal{S}} \mathcal{P}r_{\|}(X'|X, A)J(X')$. Let $\underline{\underline{J}}(s)$ be the value for equivalent CPTP problem with s as in our interwoven-epoch state space. Let *concurrency* of a state be the maximum number of actions that could be executed in the state concurrently. We define *maximum concurrency of a domain* (c) as the maximum number of actions that can be concurrently executed in any world state in the domain. The following theorem can be used to provide an admissible heuristic for CPTP problems.

Theorem 9 *Let* $s = \langle X, Y \rangle$,

$$\begin{aligned} \underline{\underline{J}}^*(s) &\geq \frac{J^*(X)}{c} \qquad \textit{for } Y = \emptyset \\ \underline{\underline{J}}^*(s) &\geq \frac{Q^*(X, A_s)}{c} \quad \textit{for } Y \neq \emptyset \end{aligned} \tag{5.2}$$

Proof Sketch: Consider any trajectory of make-span L (from a state $s = \langle X, \emptyset \rangle$ to a goal state) in a CPTP problem using its optimal policy. We can make all concurrent actions sequential by executing them in the chronological order of being started. As all concurrent actions are non-interacting, the outcomes at each stage will have similar probabilities. The maximum make-span of this sequential trajectory will be cL (assuming c actions executing at all points in the semi-MDP trajectory). Hence $J(X)$ using this (possibly non-stationary) policy would be at most $c\underline{\underline{J}}^*(s)$. Thus $J^*(X) \leq c\underline{\underline{J}}^*(s)$. The second inequality can be proven in a similar way. □

There are cases where these bounds are tight. For example, consider a deterministic planning problem in which the optimal plan is concurrently executing c actions each of unit duration (make-span = 1). In the sequential version, the same actions would be taken sequentially (make-span = c).

Following this theorem, the maximum concurrency (MC) heuristic for a state $s = \langle X, Y \rangle$ is defined as follows:

$$\text{if } Y = \emptyset \; H_{MC}(s) = \frac{J^*(X)}{c} \quad \text{else } H_{MC}(s) = \frac{Q^*(X, A_s)}{c}$$

The maximum concurrency c can be calculated by a static analysis of the domain and is a one-time expense. The complete heuristic function can be evaluated by solving the MDP for all states. However, many of these states may never be visited. In our implementation, we do this calculation on demand, as more states are visited, by starting the MDP from the current state. Each RTDP run can be seeded by the previous value function, thus no computation is thrown away and only the relevant part of the state space is explored. We refer to DUR_{samp} initiated with MC heuristic by $\text{DUR}^{\text{MC}}_{\text{samp}}$.

5.2.2 Average Concurrency Heuristic

Instead of using maximum concurrency c in the above heuristic we use the average concurrency in the domain (c_a) to get the average concurrency (AC) heuristic. We call the resulting algorithm $\text{DUR}^{\text{AC}}_{\text{samp}}$. The AC heuristic is not admissible, but in our experiments it is typically a more informed heuristic. Moreover, in the case where all the actions have the same duration, the AC heuristic equals the MC heuristic.

5.2.3 Eager Effects Heuristic

Given the CPTP problem, we can generate a relaxed CoMDP by making the effects of actions, which would otherwise be visible only in the future, be known right away — thus the name eager effects (EE). A state for this relaxed CoMDP is $\langle X, \delta \rangle$ where X is a world state and δ is an integer. Intuitively, $\langle X, \delta \rangle$ signifies that the agent will *reach* state X after time δ units. Thus, we have discarded the information about which actions are executing and when they will individually end; we only record that all of them will have ended after time δ units and that the agent will reach the state X (possibly with some probability).

The applicable set of a relaxed state is defined as $Ap_{EE}(\langle X, \delta \rangle) = Ap_{\|}(X)$. Note that this new problem really is a relaxation because certain actions are applicable that would be mutex to the currently executing actions (in the original problem). We explain this in detail

in the discussion of Theorem 10. The goal states in the relaxed problem are $\{\langle X, 0\rangle | X \in \mathcal{G}\}$, *i.e.* all states that are goals in the underlying MDP and no action is executing.

Finally, the transition probabilities are redefined. The state-component of the resulting relaxed states denotes that the effects of the combination currently started have been realized. Whereas, the time component does not advance to the end of all actions, rather it advances to the completion of the shortest action, generating a new decision epoch for starting new actions.

Formally, suppose that we execute a combination A in state $s = \langle X, \delta\rangle$. Let δ^{EE}_{last} be the length of the longest action in A. Let δ^{EE}_{first} be the the length of the shortest action in A. Define δ^{EE}_{next} as

$$\begin{aligned}
\delta^{EE}_{next} &= \delta^{EE}_{last} - \delta^{EE}_{first} \qquad \text{if } \delta = 0 \\
&= \delta^{EE}_{last} - \delta \qquad \text{if } 0 < \delta \leq \delta^{EE}_{first} \\
&= \delta^{EE}_{last} - \delta^{EE}_{first} \qquad \text{if } \delta^{EE}_{first} < \delta \leq \delta^{EE}_{last} \\
&= \delta - \delta^{EE}_{first} \qquad \text{if } \delta > \delta^{EE}_{last}
\end{aligned}$$

The transition function can now be defined using the above definition of δ^{EE}_{next}.

$$\begin{aligned}
\mathcal{P}r_{EE}(\langle X', \delta'\rangle | \langle X, \delta\rangle, A) &= 0 \qquad \text{if } \delta' \neq \delta^{EE}_{next} \\
&= \mathcal{P}r_{\|}(X' | X, A) \text{ if } \delta' = \delta^{EE}_{next}
\end{aligned}$$

The cost of executing a combination in the relaxed state represents the duration by which the current time moves forward. It is equal to $\max(\delta, \delta^{EE}_{last}) - \delta^{EE}_{next}$.

Based on the solution of the relaxed CoMDP we can compute a heuristic value for our original CPTP problem. Let $s = \langle X, Y\rangle$ be a state in the interwoven-epoch space. Let J^*_{EE} be the optimal cost function for the relaxed CoMDP. Then, the *EE* heuristic function is computed as follows:

$$H_{EE}(s) = \sum_{X' \in \mathcal{S}} \mathcal{P}r_{\|}(X' | X, A_s) J^*_{EE}(\langle X', \delta^{EE}_{last}\rangle)$$

Theorem 10 *The* EE *heuristic value is non-overestimating, thus admissible.*

The admissibility stems from the fact that, in the relaxed problem, we have eased two essential features of the original domain. First, we have assumed that the present state contains the results of actions that would actually complete in the future. So, there is actually more information in the relaxed problem, than in the original problem; thus the decisions taken are more informed and lead to a goal in less time. Secondly, since we lose the information of which actions were executing in the domain, we have to allow for all applicable combinations in the MDP state. That is, all the actions that were mutex with the actions executing (in the real problem) are also allowed. Thus, this is a relaxation of the original problem, and the time taken to reach the goal will be shorter. Hence, overall the heuristic value is admissible.

We further speed the convergence of the relaxed problems by initializing its value function with simple heuristics. It is easy to show that the following inequalities hold:

$$\begin{aligned} J^*_{EE}(\langle X, \delta \rangle) &\geq \delta \\ J^*_{EE}(\langle X, \delta \rangle) &\geq J^*_{EE}(\langle X, \delta' \rangle) \qquad \text{if } \delta' \leq \delta \\ J^*_{EE}(\langle X, \delta \rangle) &\geq J^*_{EE}(\langle X, \delta' \rangle) - (\delta' - \delta) \quad \text{if } \delta' > \delta \end{aligned}$$

So for any state $\langle X, \delta \rangle$ we can set the initial value function to δ or max it with other values computed using the above equations for the states $\langle X, \delta' \rangle$ that have already been visited. We can use the current values of these states instead of J^*_{EE} to compute these seed values.

5.2.4 Comparing the Admissible Heuristics

Theorem 11 *Neither of the two heuristics (eager effects or maximum concurrency) dominates the other.*

Proof: Consider a deterministic problem in which two parallel sets of actions in the order a_1, a_2, a_3 and b_1, b_2, b_3 need to be executed to achieve the goal. Let a_1, a_3, b_2, and b_3 be of

duration n and the rest be unit duration. If the maximum concurrency in the domain is 2, then H_{MC} value of start state is $(4n+2)/2$ which is also the optimal value $(2n+1)$. The H_{EE} value of the start state calculates to $n+2$. This is an example of a problem in which the *MC* heuristic is more informative. If however in a similar problem, the only actions that could be executed concurrently are a_3 and b_3 then the maximum concurrency remains 2. So the H_{MC} does not change, although the optimal plan is now longer. But the H_{EE} value calculates to $3n+2$ which is optimal. □

In spite of the theorem, in practice *EE* is consistently more informative than *MC* on the domains we tried. But, the computation times required for the two heuristics are quite different. *MC* requires the computation of the underlying MDP which is a relatively easy problem to solve. Whereas, *EE* requires the computation of a problem which has a larger search space than even the underlying CoMDP. Thus the computation of *EE* heuristic can take a long time, at times to the extent that the advantage of the more informative heuristic is lost in the complex heuristic computation.

5.3 Hybridized Algorithm

We present an approximate method to solve CPTP problems. While there can be many kinds of possible approximation methods, our technique exploits the intuition that it is best to focus computation on the most probable branches in the current policy's reachable space. The danger of this approach is the chance that, during execution, the agent might end up in an unlikely branch, which has been poorly explored; indeed it might blunder into a dead-end in such a case. This is undesirable, because such an apparently attractive policy might have a true expected make-span of infinity. Since, we wish to avoid dead-ends, we explore the desirable notion of *propriety*.

Definition *Propriety: A policy is* proper *at a state if it is guaranteed to lead, eventually, to the goal state (i.e., it avoids all dead-ends and cycles) [4]. We define a planning algorithm* proper *if it always produces a proper policy (when one exists) for the initial state.*

We now describe an anytime approximation algorithm, which quickly generates a proper policy and uses any additional available computation time to improve the policy, focusing

on the most likely trajectories.

5.3.1 Hybridized Planner

Our algorithm, DUR_{hyb}, is created by *hybridizing* two other policy creation algorithms. Indeed, our novel notion of hybridization is both general and powerful, applying to many MDP-like and other problems; however, in this section we focus on the use of hybridization for CPTP. We revisit the applicability of hybridization for MDPs in Chapter 7 and for other problems in Section 9.7.

Hybridization uses an anytime algorithm like RTDP to create a policy for frequently visited states, and uses a faster (and presumably suboptimal) algorithm for the infrequent states. For the case of CPTP, our algorithm hybridizes the RTDP algorithms for interwoven-epoch and aligned-epoch models. With aligned-epochs, RTDP converges relatively quickly, because the state space is smaller, but the resulting policy is suboptimal *for the CPTP problem*, because the policy waits for *all* currently executing actions to terminate before starting any new actions. In contrast, RTDP for interwoven-epochs generates the optimal policy, but it takes much longer to converge. Our insight is to run RTDP on the interwoven space long enough to generate a policy which is good on the common states, but stop well before it converges in every state. Then, to ensure that the rarely explored states have a proper policy, we substitute the aligned policy, returning this *hybridized* policy.

Thus the key question is how to decide which states are well explored and which are not. We define the *familiarity* of a state s to be the number of times it has been visited in previous RTDP trials. Any reachable state whose familiarity is less than a constant, k, has an aligned policy created for it. Furthermore, if a dead-end state is reached using the greedy interwoven policy, then we create an aligned policy for the immediate precursors of that state. If a cycle is detected[3], then we compute an aligned policy for all the states which are part of the cycle.

We have not yet said how the hybridized algorithm terminates. Use of RTDP helps us in defining a very simple termination condition with a parameter that can be varied to achieve

[3] In our implementation cycles are detected using simulation.

Algorithm 3 Hybridized Algorithm $\mathrm{DUR}_{\mathrm{hyb}}(r, k, m)$

1: **for all** $s \in \underline{\underline{\mathcal{S}}}$ **do**
2: initialize $\underline{\underline{J}}(s)$ with an admissible heuristic
3: **repeat**
4: perform m RTDP trials
5: compute hybridized policy (π_{hyb}) using interwoven-epoch policy for k-familiar states and aligned-epoch policy otherwise
6: clean π_{hyb} by removing all dead-ends and cycles
7: $\underline{\underline{J}}^{\pi}\langle s_0, \emptyset\rangle \leftarrow$ evaluation of π_{hyb} from the start state
8: **until** $\left(\frac{\underline{\underline{J}}^{\pi}(\langle s_0,\emptyset\rangle) - \underline{\underline{J}}(\langle s_0,\emptyset\rangle)}{\underline{\underline{J}}(\langle s_0,\emptyset\rangle)} < r \right)$
9: **return** hybridized policy π_{hyb}

the desired *closeness* to optimality as well. The intuition is very simple. Consider first, optimal labeled RTDP. This starts with an admissible heuristic and guarantees that the value of the start state, $\underline{\underline{J}}(\langle s_0, \emptyset\rangle)$, remains admissible (thus less than or equal to optimal). In contrast, the hybridized policy's make-span is always longer than or equal to optimal. Thus as time progresses, these values approach the optimal make-span from opposite sides. Whenever the two values are within an *optimality ratio (r)*, we know that the algorithm has found a solution, which is close to the optimal.

Finally, evaluation of the hybridized policy is done using simulation, which we perform after a fixed number of m RTDP trials. Algorithm 3 summarizes the details of the algorithm. One can see that this combined policy is proper for two reasons: 1) if the policy at a state is from the aligned policy, then it is proper because the RTDP for the aligned-epoch model was run to convergence, and 2) for the rest of the states it has explicitly ensured that there are no cycles or dead-ends.

5.4 Experiments: Planning with Deterministic Durations

Continuing from Section 3.6, in this set of experiments we evaluate the various techniques for solving problems involving explicit deterministic durations. We compare the computation time and solution quality of five methods: interwoven Sampled RTDP with no heuristic

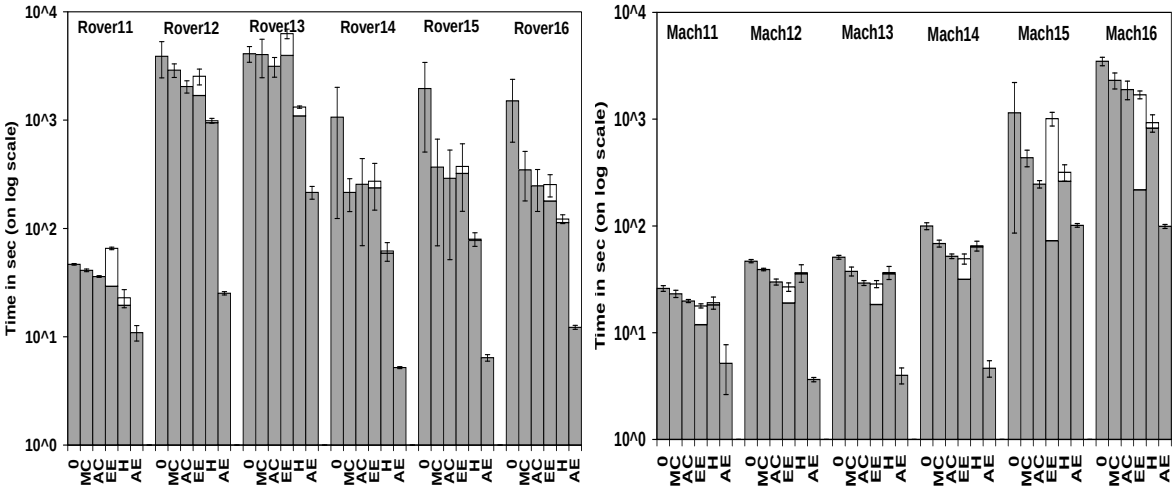

Figure 5.4: (a,b): Running times (on a log scale) for the Rover and Machineshop domain, respectively. For each problem the six bars represent the times taken by the algorithms: $\mathrm{DUR}_{\mathrm{samp}}$ (0), $\mathrm{DUR}^{\mathrm{MC}}_{\mathrm{samp}}$ (AE), $\mathrm{DUR}^{\mathrm{AC}}_{\mathrm{samp}}$ (AC), $\mathrm{DUR}^{\mathrm{EE}}_{\mathrm{samp}}$ (EE), $\mathrm{DUR}_{\mathrm{hyb}}$ (H), and $\mathrm{DUR}_{\mathrm{AE}}$ (AE), respectively. The white bar on $\mathrm{DUR}^{\mathrm{EE}}_{\mathrm{samp}}$ denotes the portion of time taken by heuristic computation and on $\mathrm{DUR}_{\mathrm{hyb}}$ denotes the portion of time taken by aligned-epoch RTDP.

($\mathrm{DUR}_{\mathrm{samp}}$), with the maximum concurrency ($\mathrm{DUR}^{\mathrm{MC}}_{\mathrm{samp}}$), average concurrency ($\mathrm{DUR}^{\mathrm{AC}}_{\mathrm{samp}}$), and eager effects ($\mathrm{DUR}^{\mathrm{EE}}_{\mathrm{samp}}$) heuristics, the hybridized algorithm ($\mathrm{DUR}_{\mathrm{hyb}}$) and Sampled RTDP on the aligned-epoch model ($\mathrm{DUR}_{\mathrm{AE}}$). We test on our Rover, MachineShop and Aritificial domains. We also use our Artificial domain to see if the relative performance of the techniques varies with the amount of concurrency in the domain.

5.4.1 Experimental Setup

We modify the domains used in Section 3.6 by additionally including action durations. For NASA Rover and MachineShop domains, we generate problems with 17-26 state variables and 12-18 actions, whose duration range between 1 and 20. The problems have between 15,000-700,000 reachable states in the interwoven-epoch state space, $\underline{\underline{\mathcal{S}}}$.

We use Artificial domain for control experiments to study the effect of degree of parallelism. All the problems in this domain have 14 state variables and 17,000-40,000 reachable states and durations of actions between 1 and 3.

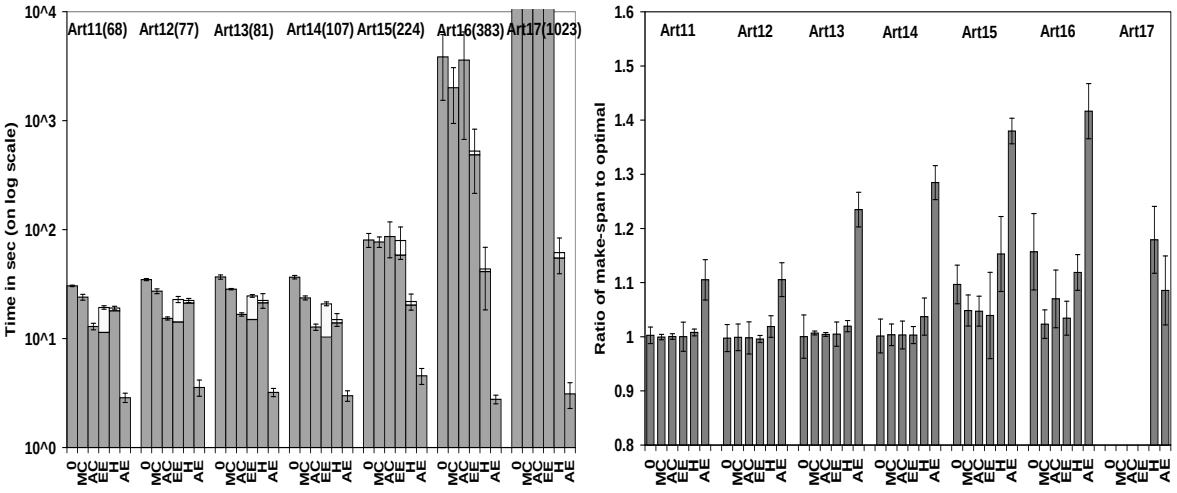

Figure 5.5: (a,b): Comparison of the different algorithms (running times and solution quality respectively) for the Artificial domain. As degree of parallelism increases the problems become harder; the largest problem is solved only by $\mathrm{DUR}_{\mathrm{hyb}}$ and $\mathrm{DUR}_{\mathrm{AE}}$.

We use our implementation of Sampled RTDP[4] and implement all heuristics: maximum concurrency (H_{MC}), average concurrency (H_{AC}), and eager effects (H_{EE}), for the initialization of the value function. We calculate these heuristics on demand for the states visited, instead of computing the complete heuristic for the whole state space at once. We also implement the hybridized algorithm in which the initial value function was set to the H_{MC} heuristic. The parameters r, k, and m are kept at 0.05, 100 and 500, respectively. We test each of these algorithms on a number of problem instances from the three domains, which we generate by varying the number of objects, degrees of parallelism, durations of the actions and distances to the goal.

5.4.2 Comparison of Running Times

Figures 5.4(a, b) and 5.5(a) show the variations in the running times for the algorithms on different problems in Rover, Machineshop and Artificial domains, respectively. The first four bars represent the base Sampled RTDP without any heuristic, with H_{MC}, with H_{AC},

[4]Note that policies returned by $\mathrm{DUR}_{\mathrm{samp}}$ are not guaranteed to be optimal. Thus all the implemented algorithms are approximate. We can replace $\mathrm{DUR}_{\mathrm{samp}}$ by pruned RTDP ($\mathrm{DUR}_{\mathrm{prun}}$) if optimality is desired.

and with H_{EE} respectively. The fifth bar represents the hybridized algorithm (using the H_{MC} heuristic) and the sixth bar is computation of the aligned-epoch Sampled RTDP with costs set to the maximum action duration. The white region in the fourth bar represents the time required for the H_{EE} computation. The white region in the fifth bar represents the time taken for the aligned-epoch RTDP computations in the hybridized algorithm. The error bars represent 95% confidence intervals on the running times. Note that the plots are on a log scale.

We notice that $\mathrm{DUR}_{\mathrm{AE}}$ solves the problems extremely quickly; this is natural since the aligned-epoch space is much smaller. Use of both H_{MC} and H_{AC} always speeds search in the $\mathcal{S}_{\overline{=}}$ model. The white region in the fourth bar represents the time required for the H_{EE} computation. Comparing the heuristics amongst themselves, we find that average concurrency heuristic mostly performs faster than maximum concurrency — presumably because H_{AC} is a more informed heuristic in practice, although at the cost of being inadmissible. We find a couple of cases in which H_{AC} doesn't perform better; this could be because it is focusing the search in the incorrect region, given its inadmissible nature. For the the Rover domain H_{EE} does not perform as well as H_{MC} whereas for Machineshop domain for most problems H_{EE} outperforms even H_{AC}. For the Artificial domain, the performance typically lies in between H_{MC} and H_{AC}.

For the Rover domain, the hybridized algorithm performs fastest. In fact, the speedups are dramatic compared to other methods. In other domains, the results are more comparable for small problems. However, for large problems in these two domains, hybridized outperforms the others by a huge margin. In fact for the largest problem in Artificial domain, none of the heuristics are able to converge (within a day) and only $\mathrm{DUR}_{\mathrm{hyb}}$ and $\mathrm{DUR}_{\mathrm{AE}}$ converge to a solution.

Table 5.1 shows the speedups obtained by various algorithms compared to the basic $\mathrm{DUR}_{\mathrm{samp}}$. In the Rover and Artificial domains the speedups obtained by $\mathrm{DUR}_{\mathrm{hyb}}$ and $\mathrm{DUR}_{\mathrm{AE}}$ are much more prominent than in the Machineshop domain. Averaging over all domains, H produces a 10x speedup and AE produces more than a 100x speedup.

Table 5.1: The ratio of the time taken by $\mathrm{DUR}_{\mathrm{samp}}$ with no heuristics to that of each algorithm. Our heuristics produce 2-3 times speedups. The hybridized algo produces about a 10x speedup. Aligned epoch search produces 100x speedup, but sacrifices solution quality.

Algos	Speedup compared with $\mathrm{DUR}_{\mathrm{samp}}$			
	Rover	Machineshop	Artificial	Average
$\mathrm{DUR}_{\mathrm{samp}}^{\mathrm{MC}}$	3.016764	1.545418	1.071645	1.877942
$\mathrm{DUR}_{\mathrm{samp}}^{\mathrm{AC}}$	3.585993	2.173809	1.950643	2.570148
$\mathrm{DUR}_{\mathrm{samp}}^{\mathrm{EE}}$	2.99117	1.700167	2.447969	2.379769
$\mathrm{DUR}_{\mathrm{hyb}}$	10.53418	2.154863	16.53159	9.74021
$\mathrm{DUR}_{\mathrm{AE}}$	135.2841	16.42708	241.8623	131.1911

Table 5.2: Overall solution quality produced by all algorithms. Note that all algorithms except $\mathrm{DUR}_{\mathrm{AE}}$ produce policies whose quality is quite close to optimal. On average $\mathrm{DUR}_{\mathrm{AE}}$ produces make-spans that are about 125% of the optimal.

Algos	Average Quality			
	Rover	Machineshop	Artificial	Average
$\mathrm{DUR}_{\mathrm{samp}}$	1.059625	1.065078	1.042561	1.055704
$\mathrm{DUR}_{\mathrm{samp}}^{\mathrm{MC}}$	1.018405	1.062564	1.013465	1.031478
$\mathrm{DUR}_{\mathrm{samp}}^{\mathrm{AC}}$	1.017141	1.046391	1.020523	1.028019
$\mathrm{DUR}_{\mathrm{samp}}^{\mathrm{EE}}$	1.021806	1.027887	1.012897	1.020863
$\mathrm{DUR}_{\mathrm{hyb}}$	1.059349	1.075534	1.059201	1.064691
$\mathrm{DUR}_{\mathrm{AE}}$	1.257205	1.244862	1.254407	1.252158

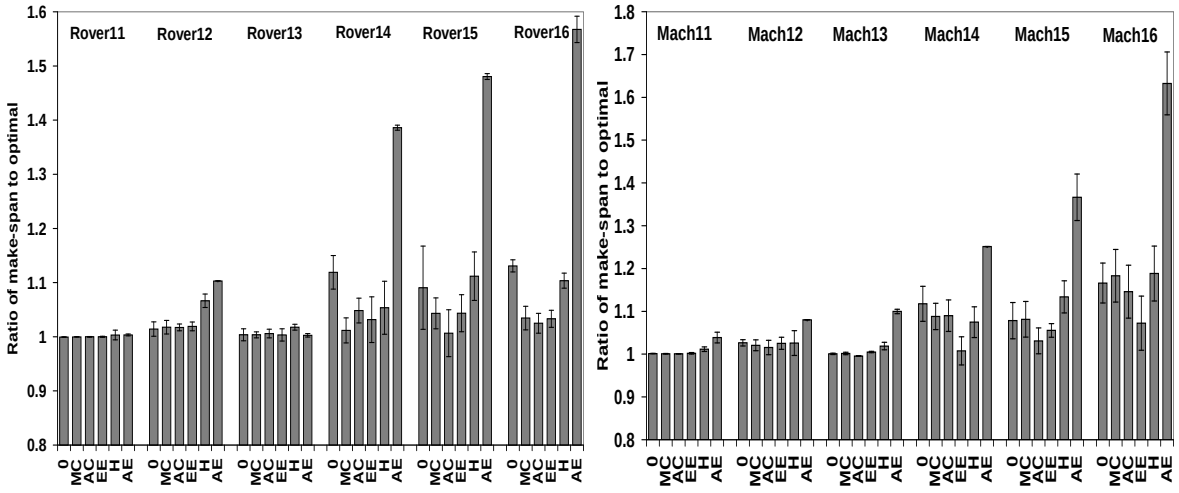

Figure 5.6: (a,b): Comparison of make-spans of the solution found with the optimal(plotted as 1 on the y-axes) for Rover and Machineshop domains, respectively. All algorithms except DUR_{AE} produce solutions quite close to the optimal.

5.4.3 Comparison of Solution Quality

Figures 5.6(a, b) and 5.5(b) show the quality of the policies obtained by the same six methods on the same domains. We measure quality by simulating the generated policy across multiple trials, and reporting the average time taken to reach the goal. We plot the ratio of the so-measured expected make-span to the optimal expected make-span[5]. Table 5.2 presents solution qualities for each method, averaged over all problems in a domain. We note that the aligned-epoch policies usually yield significantly longer make-spans (e.g., 25% longer); thus one must make a quality sacrifice for their speedy policy construction. In contrast, the hybridized algorithm extorts only a small sacrifice in quality in exchange for its speed.

5.4.4 Variation with Concurrency

Figure 5.5(a) represents our attempt to see if the relative performance of the algorithms changed with increasing concurrency. Along the top of the figure, by the problem names,

[5] In some large problems, the optimal algorithm did not converge. For those, we take as optimal, the best policy found in our runs.

are numbers in brackets; these list the average number of applicable combinations in each MDP state, $Avg_{s \in \mathcal{S}} |Ap(s)|$, and range from 68 to 1023 concurrent actions. Note that for the difficult problems with a lot of parallelism, $\mathrm{DUR}_{\mathrm{samp}}$ slows dramatically, regardless of heuristic. In contrast, the $\mathrm{DUR}_{\mathrm{hyb}}$ is still able to quickly produce a policy, and at almost no loss in quality (Figure 5.5(b)).

Chapter 6

OPTIMAL PLANNING WITH UNCERTAIN DURATIONS

We now extend the techniques of previous chapter for the case when action durations are not deterministic. As before, we consider TGP-style actions and a discrete temporal model. We assume independent durations, and monotonic continuations, but Section 6.3 relaxes the latter, extending our algorithms to handle multimodal duration distributions. As before we aim to minimize the expected time required to reach a goal.

6.1 Formulating as a CoMDP

We now formulate our planning problem as a CoMDP similar to Section 5.1. While some of the parameters of the CoMDP can be used directly from our work on deterministic durations, we need to recompute the transition function.

State Space: Both the aligned epoch state space as well as the interwoven epoch space, as defined in Section 5.1 are adequate to model this planning problem. To determine the size of the interwoven space, we replace the duration of an action by its *max* duration. Let $\Delta_M(a)$ denote the maximum time within which action a will complete. The overall interwoven-epoch search space is $\underline{\underline{\mathcal{S}}} = \mathcal{S} \times \bigotimes_{a \in \mathcal{A}} \left(\{a\} \times Z_{\Delta_M(a)} \right)$, where $Z_{\Delta_M(a)}$ represents the set $\{0, 1, \ldots, \Delta_M(a) - 1\}$ and $\bigotimes$ denotes the Cartesian product over multiple sets.

Action Space: At any state we may apply a combination of actions with the applicability function reflecting the fact that the combination of actions is safe *w.r.t* itself (and *w.r.t.* already executing actions in case of interwoven space) as in the previous sections. While the previous state space and action space work well for our problem, the transition function definition needs to change, since we now need to take into account the uncertainty in durations.

Transition Function: Uncertain durations require significant changes to the probability transition function ($\underline{\underline{\mathcal{P}r}}$) for the interwoven space from the definitions of Section 5.1.2.

Since our assumptions justify Conjecture 8, we need only consider happenings when choosing decision epochs.

Algorithm 4 ComputeTransitionFunc(s=$\langle X, Y\rangle$,A)

1: $\mathcal{Y} \leftarrow Y \cup \{(a, 0)\} \quad \forall a \in A$
2: $mintime \leftarrow \min_{(a,\delta)\in\mathcal{Y}}$ minimum remaining time for a
3: $maxtime \leftarrow \min_{(a,\delta)\in\mathcal{Y}}$ maximum remaining time for a
4: **for all** integer $t \in [mintime, maxtime]$ **do**
5: $\quad A_t \leftarrow$ set of actions that could possibly terminate at t
6: $\quad$ **for all** non-empty subsets $Asub_t \subseteq A_t$ **do**
7: $\quad\quad p_c \leftarrow$ prob. that exactly $Asub_t$ terminates at t. (see Equation 6.1).
8: $\quad\quad \mathcal{W} \leftarrow \{(X_t, p_w) \mid X_t$ is a world state; p_w is the probability that $Asub_t$ terminates yielding $X_t\}$.
9: $\quad\quad$ **for all** $(X_t, p_w) \in \mathcal{W}$ **do**
10: $\quad\quad\quad Y_t \leftarrow \{(a, t + \delta) \mid (a, \delta) \in \mathcal{Y}, a \notin Asub_t\}$
11: $\quad\quad\quad$ insert $(\langle X_t, Y_t\rangle, p_w \times p_c)$ in *output*
12: **return** *output*

The computation of transition function is described in Algorithm 4. Although the next decision epoch is determined by a happening, we still need to *consider* all pivots for the next state calculations as all these are potential happenings. *mintime* is the minimum time when an executing action could terminate, *maxtime* is the minimum time by which it is guaranteed that at least one action would terminate. For all times between *mintime* and *maxtime* we compute the possible combinations that could terminate then and the resulting next interwoven state. The probability, p_c, (line 7) may be computed using the following formula:

$$p_c = \prod_{(a,\delta_a)\in\mathcal{Y}, a\in Asub_t} (\text{prob. } a \text{ terminates at } t+\delta_a | a \text{ hasn't terminated till } \delta_a) \times \prod_{(b,\delta_b)\in\mathcal{Y}, b\notin Asub_t} (\text{prob. } b \text{ doesn't terminate at } t+\delta_b | b \text{ hasn't terminated till } \delta_b) \quad (6.1)$$

Considering all pivots makes the algorithm computationally intensive because there may be many pivots and many action combinations could end at each one, and with many

outcomes each. In our implementation, we cache the transition function so that we do not have to recompute the information for any state.

Start and Goal States: The start state and goal set that we developed for the deterministic durations work unchanged when the durations are stochastic. So, the start state is $\langle s_0, \emptyset \rangle$ and the goal set is $\underline{\underline{\mathcal{G}}} = \{\langle X, \emptyset \rangle | X \in \mathcal{G}\}$.

Thus we have modeled our problem as a CoMDP in the interwoven state space. We have redefined the start and goal states, and the probability transition function. Now we can use the techniques of CoMDPs to solve our problem. In particular, we can use our Bellman equations as below.

Bellman Equations: Define $\delta_{el}(s, A, s')$ as the time elapsed between two interwoven states s and s' when combination A is executed in s. The set of equations for the solution of our problem can be written as:

$$\underline{\underline{J}}^*(s) = 0, \text{ if } s \in \underline{\underline{\mathcal{G}}} \text{ else} \tag{6.2}$$

$$\underline{\underline{J}}^*(s) = \min_{A \in \underline{\underline{Ap}}(s)} \sum_{s' \in \underline{\underline{\mathcal{S}}}} \underline{\underline{\mathcal{P}r}}(s'|s, A) \left\{ \delta_{el}(s, A, s') + \underline{\underline{J}}^*(s') \right\}$$

Compare these equations with Equation 5.1. There is one difference besides the new transition function — the time elapsed is within the summation sign. This is because time elapsed depends also on the next interwoven state.

Having modeled this problem as a CoMDP we again use our algorithms of Chapter 5. We use ΔDUR to denote the family of algorithms for the CPTP problems involving stochastic durations. The main bottleneck in solving these problem, besides the size of the interwoven state space, is the high branching factor.

6.1.1 Policy Construction: RTDP & Hybridized Planning

Since we have modeled our problem as a CoMDP in the new interwoven space, we may use pruned RTDP ($\Delta\text{DUR}_{\text{prun}}$) and sampled RTDP ($\Delta\text{DUR}_{\text{samp}}$) for policy construction. Since the cost function in our problem (δ_{el}) depends also on the current and the next state,

combo-skipping does not apply for this problem. Thus $\Delta\text{DUR}_{\text{prun}}$ refers to RTDP with only combo-elimination.

Furthermore, only small adaptations are necessary to incrementally compute the (admissible) *maximum concurrency* (MC) and (more informed, but inadmissible) *average concurrency* (AC) heuristics. For example, for the serial MDP (in the RHS of Equation 5.2) we now need to compute the average duration of an action and use that as the action's cost.

Likewise, we can further speed planning by hybridizing ($\Delta\text{DUR}_{\text{hyb}}$) RTDP algorithms for interwoven and aligned-epoch CoMDPs to produce a near-optimal policy in significantly less time. The dynamics of aligned epoch space is same as that in Section 5 with one exception. The cost of a combination, in the case of deterministic durations, was simply the max duration of the constituent actions. The novel twist stems from the fact that uncertain durations require computing this *cost* of an action combination as the mean of the max of the possible duration outcomes. For example, suppose two actions both with uniform duration distributions between 1 to 3 are started concurrently. The probabilities that both actions will finish by time 1, 2 and 3 are 1/9, 3/9, and 5/9 respectively. Thus the expected duration of completion of the combination (let us call it Δ_{AE}) is $1{\times}1/9 + 2{\times}3/9 + 3{\times}5/9 = 2.44$.

6.2 Expected-Duration Planner

When modeled as a CoMDP in the full-blown interwoven space, stochastic durations cause a cancerous growth in the branching factor. In general, if n actions are started each with m possible durations and each having r probabilistic effects, then there are $(m-1)[(r+1)^n - r^n - 1] + r^n$ potential successors. This number may be computed as follows: for each duration between 1 and $m-1$ any subset of actions could complete and each action could result in r outcomes. Hence, total number of successors per duration is $\sum_{i\in[1..n]}{}^nC_i r^i = (r+1)^n - r^n - 1$. Moreover, if none of the actions finish until time $m-1$ then at the last step all actions terminate leading into r^n outcomes. So, total number of successors is $(m-1)[(r+1)^n - r^n - 1] + r^n$. Thus, the branching factor is multiplicative in the duration uncertainty and exponential in the concurrency.

To manage this computational tumor we must curb the branching factor. One method is to ignore duration distributions. We can assign each action a *constant* duration equal to

the mean of its distribution, then apply a deterministic-duration planner such as DUR_{samp}. However, when executing the deterministic-duration policy in a setting where durations are actually stochastic, an action will likely terminate at a time *different* than its mean, expected duration. The $\Delta\text{DUR}_{\text{exp}}$ planner addresses this problem by augmenting the deterministic-duration policy created to account for these unexpected outcomes.

6.2.1 Online Version

The procedure is easiest to understand in its *online* version (Algorithm 5): wait until the unexpected happens, pause execution, and re-plan. If the original estimate of an action's duration is implausible, we compute a revised deterministic estimate in terms of $\mathcal{E}_a(min, max)$ — the expected value of a's duration distribution restricted between times min and max. Recall that Δ_M denotes the max duration of an action. Thus, $\mathcal{E}_a(0, \Delta_M(a))$ will compute the expected duration of a.

Example: Let the duration of an action a follow a uniform distribution between 1 and 15. The expected value that gets assigned in the first run of the algorithm ($\lceil \mathcal{E}_a(0, \Delta_M(a)) \rceil$) is 8. While running the algorithm, suppose the action didn't terminate by 8 and we reach a state where a has been running for, say, 9 time units. In that case, a revised expected duration for a would be ($\lceil \mathcal{E}_a(8, 15) \rceil$) = 12. Similarly, if it doesn't terminate by 12 either then the next expected duration would be 14, and finally 15. In other words for all states where a has been executing for times 0 to 8, it is expected to terminate at 8. For all times between 8 and 12 the expected completion is at 12, for 12 to 14 it is 14 and if it doesn't terminate at 14 then it is 15.□

6.2.2 Offline Version

This algorithm also has an *offline version* in which re-planning for all contingencies is done ahead of time and for fairness we used this version in the experiments. Although the offline algorithm plans for all possible action durations, it is still much faster than the other algorithms. The reason is that each of the planning problems solved is now significantly smaller (less branching factor, smaller reachable state space), and all the previous compu-

Algorithm 5 Online $\Delta\text{DUR}_{\text{exp}}$

build a deterministic-duration policy from the start state s_0
repeat
  execute action combination specified by policy
  wait for interrupt
    case: action a terminated as expected {//*do nothing*}
    case: action a terminates early
      extend policy from current state
    case: action a didn't terminate as expected
      extend policy from current state revising
      a's duration as follows:
        $\delta \leftarrow$ time elapsed since a started executing
        $nextexp \leftarrow \lceil \mathcal{E}_a(0, \Delta_M(a)) \rceil$
        while $nextexp < \delta$ **do**
          $nextexp \leftarrow \lceil \mathcal{E}_a(nextexp, \Delta_M(a)) \rceil$
        endwhile
      a's revised duration $\leftarrow nextexp - \delta$
  endwait
until goal is reached

tation can be succinctly stored in the form of the ⟨interwoven state, value⟩ pairs and thus reused. Algorithm 6 describes this offline planner and the subsequent example illustrate the savings.

Algorithm 6 Offline $\Delta\text{DUR}_{\text{exp}}$

1: build a deterministic-duration policy from the start state s_0; get current $\underline{\underline{J}}$ and $\underline{\underline{\pi}}$ values
2: insert s_0 in the queue open
3: **repeat**
4: state = open.pop()
5: **for all** currstate s.t. $\mathcal{P}r_{\underline{\underline{\cdot}}}(\text{currstate}|\text{state}, \underline{\underline{\pi}}^*(\text{state})) > 0$ **do**
6: **if** currstate is not goal and currstate is not in the set visited **then**
7: visited.insert(currstate)
8: **if** $\underline{\underline{J}}$(currstate) has not converged **then**
9: if required, change the expected durations of the actions that are currently executing in currstate.
10: solve a deterministic-duration planning problem with the start state currstate
11: insert currstate in the queue open
12: **until** open is empty

Line 9 of Algorithm 6 assigns a new expected duration for all actions that are currently running in the current state and have not completed by the time of their previous termination point. This reassignment follows the similar case in the online version (line 13).

Example: Consider a domain with two state-variables, x_1 and x_2, with two actions set-x_1 and set-x_2. The task is to set both variables (initially they are both false). Assume that set-x_2 always succeeds whereas set-x_1 succeeds with only 0.5 probability. Moreover, let both actions have a uniform duration distribution of 1, 2, or 3. In such a case a complete interwoven epoch search could touch 36 interwoven states (each state variable could be true or false, each action could be "not running", "running for 1 unit", and "running for 2 units"). Instead, if we build a deterministic duration policy then each action's deterministic duration will be 2, and so the total number of states touched will be from the 16 interwoven states (each action could now only be "not running" or "running for 1 unit").

Now, suppose that the deterministic planner decides to execute both actions in the start

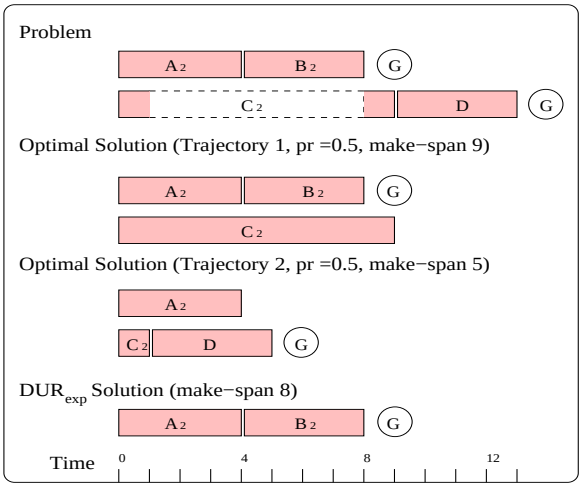

Figure 6.1: An example of a domain where the $\Delta\text{DUR}_{\text{exp}}$ algorithm does not compute an optimal solution.

state. Having committed to this combination, it is easy to see that certain states will never be reached. For example, the state $\langle(\neg x_1, \neg x_2), \{(\text{set}{-}\text{x}_1, 2)\}\rangle$ can never be visited, since once set-x_2 completes it is guaranteed that x_2 will be set. In fact, in our example, only 3 new states will initiate offline replanning (line 10 in Algo 6), *viz.*, $\langle(x_1, \neg x_2), \{(\text{set}{-}\text{x}_2, 2)\}\rangle$, $\langle(\neg x_1, \neg x_2), \{(\text{set}{-}\text{x}_2, 2)\}\rangle$, and $\langle(\neg x_1, x_2), \{(\text{set}{-}\text{x}_1, 2)\}\rangle$ □

6.2.3 Properties

Unfortunately, our $\Delta\text{DUR}_{\text{exp}}$ algorithm is not guaranteed to produce an optimal policy. How bad are the policies generated by the expected-duration planner? The experiments show that $\Delta\text{DUR}_{\text{exp}}$ typically generates policies which are extremely close to optimal. Even the worst-case pathological domain we are able to construct leads to an expected make-span which is only 50% longer than optimal (in the limit). This example is illustrated below.

Example: We consider a domain which has actions $A_{2:n}, B_{2:n}, C_{2:n}$ and D. Each A_i and B_i takes time 2^i. Each C_i has a probabilistic duration: with probability 0.5, C_i takes 1 unit of time, and with the remaining probability, it takes $2^{i+1}+1$ time. Thus, the expected duration of C_i is $2^i + 1$. D takes 4 units. In sub-problem SP_i, the goal may be reached by executing A_i followed by B_i. Alternatively, the goal may be reached by first executing C_i and

then recursively solving the sub-problem SP_{i-1}. In this domain, the $\Delta\text{DUR}_{\text{exp}}$ algorithm will always compute $\langle A_i; B_i \rangle$ as the best solution. However, the optimal policy starts both $\{A_i, C_i\}$. If C_i terminates at 1, the policy executes the solution for SP_{i-1}; otherwise, it waits until A_i terminates and then executes B_i. Figure 6.1 illustrates the sub-problem SP_2 in which the optimal policy has an expected make-span of 7 (*vs.* $\Delta\text{DUR}_{\text{exp}}$'s make-span of 8). In general, the expected make-span of the optimal policy on SP_n is $\frac{1}{3}[2^{n+2}+2^{4-n}]+2^{2-n}+2$. Thus, $\lim_{n\to\infty} \frac{exp}{opt} = \frac{3}{2}$.□

6.3 Multi-Modal Duration Distributions

The planners of the previous two sections benefited by considering the small set of happenings instead of pivots, an approach licensed by Conjecture 8. Unfortunately, this simplification is not warranted in the case of actions with multi-modal duration distributions, which can be common in complex domains where all factors can't be modeled explicitly. For example, the amount of time for a Mars rover to transmit data might have a bimodal distribution — normally it would take little time, but if a dust storm were in progress (unmodeled) it could take much longer. To handle these cases we model durations with a mixture of Gaussians parameterized by the triple $\langle amplitude, mean, variance \rangle$.

6.3.1 CoMDP Formulation

Although we cannot restrict decision epochs to happenings, we need not consider *all* pivots; they are required only for actions with multi-modal distributions. In fact, it suffices to consider pivots in *regions* of the distribution where the expected-time-to-completion increases. In all other cases we need consider only happenings.

Two changes are required to the transition function of Algorithm 4. In line 3, the *maxtime* computation now involves time until the next pivot in the increasing remaining time region for all actions with multi-modal distributions (thus forcing us to take a decision at those points, even when no action terminates). Another change (in line 6) allows a *non-empty* subset $Asub_t$ for $t = maxtime$. That is, next state is computed even without any action termination. By making these changes in the transition function we reformulate our

problem as a CoMDP in the interwoven space and thus solve, using our previous methods of pruned/sampled RTDP, hybrid algorithm or expected-duration algorithm.

6.3.2 Archetypal-Duration Planner

We also develop a multi-modal variation of the expected-duration planner, called $\Delta\text{DUR}_{\text{arch}}$. Instead of assigning an action a single deterministic duration equal to the expected value, this planner assigns it a probabilistic duration with various outcomes being the means of the different modes in the distribution and the probabilities being the probability mass in each mode. This enhancement reflects our intuitive understanding for multi-modal distributions and the experiments confirm that $\Delta\text{DUR}_{\text{arch}}$ produces solutions having shorter make-spans than those of $\Delta\text{DUR}_{\text{exp}}$.

6.4 Experiments: Planning with Stochastic Durations

We now evaluate our techniques for solving planning problems involving stochastic durations. We compare the computation time and solution quality (make-span) of our five planners for domains with and without multi-modal duration distributions. We also re-evaluate the effectiveness of the maximum- (MC) and average-concurrency (AC) heuristics for these domains.

6.4.1 Experimental Setup

We modify our Rover, MachineShop, and Artificial domains by additionally including uncertainty in action durations. For this set of experiments, our largest problem had 4 million world states of which 65536 were reachable. Our algorithms explored up to 1,000,000 distinct states in the interwoven state space during planning. The domains contained as many as 18 actions, and some actions had as many as 13 possible durations.

6.4.2 Comparing Running Times

We compare all algorithms with and without heuristics and reaffirm that the heuristics significantly speed up the computation on all problems; indeed, some problems are too large to be solved without heuristics. Comparing them amongst themselves we find that AC

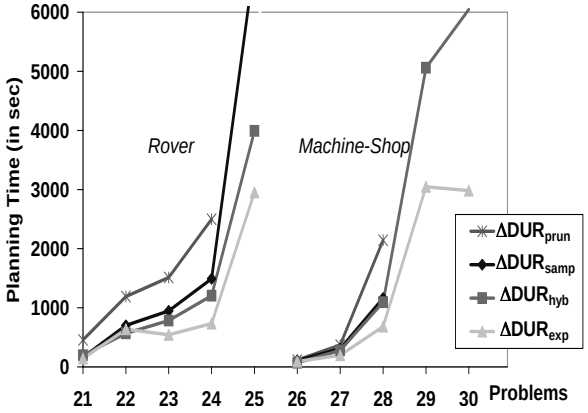

Figure 6.2: Planning time comparisons for Rover and MachineShop domains: Variation along algorithms when all initialized by the average concurrency (AC) heuristic; $\Delta\text{DUR}_{\text{exp}}$ performs the best.

Table 6.1: All three planners produce near-optimal policies as shown by this table of ratios to the optimal make-span.

Algos	Average Quality of Make-Span		
	Rover	MachineShop	Artificial
$\Delta\text{DUR}_{\text{samp}}$	1.001	1.000	1.001
$\Delta\text{DUR}_{\text{hyb}}$	1.022	1.011	1.019
$\Delta\text{DUR}_{\text{exp}}$	1.008	1.015	1.046

beats MC — regardless of the planning algorithm; this isn't surprising since AC sacrifices admissibility.

Figure 6.2 reports the running times of various algorithms (initialized with the AC heuristic) on the Rover and Machine-Shop domains when all durations are unimodal. $\Delta\text{DUR}_{\text{exp}}$ out-performs the other planners by substantial margins. As this algorithm is solving a comparatively simpler problem, fewer states are expanded and thus the approximation scales better than others — solving, for example, two Machine-Shop problems, which were too large for most other planners. In most cases hybridization speeds planning by significant amounts, but it performs better than $\Delta\text{DUR}_{\text{exp}}$ only for the artificial domain.

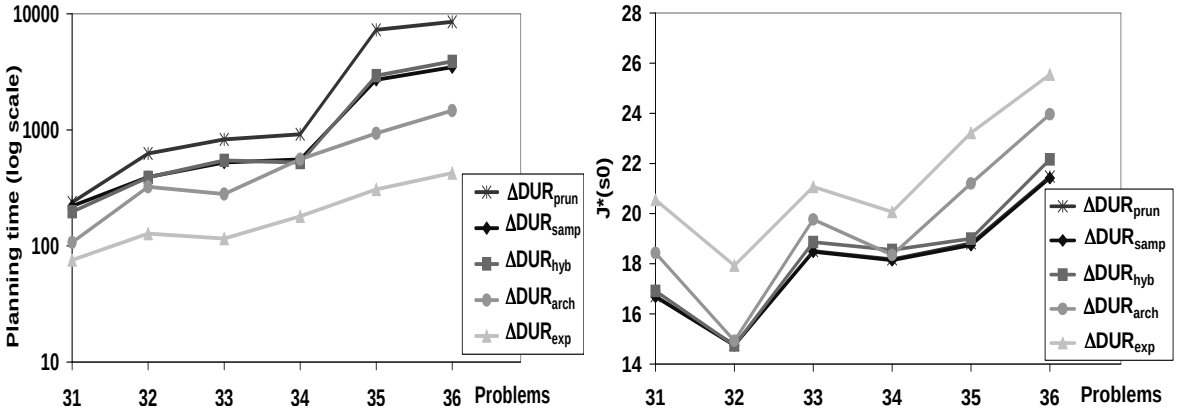

Figure 6.3: Comparisons in the Machine-Shop domain with multi-modal distributions. (a) Computation Time comparisons: $\Delta\text{DUR}_{\text{exp}}$ and $\Delta\text{DUR}_{\text{arch}}$ perform much better than other algos. (b) Make-spans returned by different algos: Solutions returned by $\Delta\text{DUR}_{\text{samp}}$ are almost optimal. Overall $\Delta\text{DUR}_{\text{arch}}$ finds a good balance between running time and solution quality.

6.4.3 Comparing Solution Quality

We measure quality by simulating the generated policy across multiple trials. We report the ratio of average expected make-span and the optimal[1] expected make-span for domains with all unimodal distributions in Table 6.1. We find that the make-spans of the inadmissible heuristic AC are at par with those of the admissible heuristic MC. The hybridized planner is approximate with a user-defined bound. In our experiments, we set the bound to 5% and find that the make-spans returned by the algorithm are quite close to the optimal and do not always differ by 5%. $\Delta\text{DUR}_{\text{exp}}$ has no quality guarantees, still the solutions returned on the problems we tested upon are nearly as good as other algorithms. Thus, we believe that this approximation will be quite useful in scaling to larger problems without losing solution quality.

6.4.4 Multimodal Domains

We develop multi-modal variants of our domains; *e.g.*, in the Machine-Shop domain, time for fetching paint was bimodal (if in stock, paint can be fetched fast, else it needs to be

[1]If the optimal algorithm doesn't converge, we use the best solution found across all runs as "optimal".

ordered). There was an alternative but costly paint action that doesn't require fetching of paint. Solutions produced by $\Delta\text{DUR}_{\text{samp}}$ made use of pivots as decision epochs by starting the costly paint action in case the fetch action didn't terminate within the first mode of the bimodal distribution (*i.e.* paint was out of stock).

The running time comparisons are shown in Figure 6.3(a) on a log-scale. We find that $\Delta\text{DUR}_{\text{exp}}$ terminates extremely quickly and $\Delta\text{DUR}_{\text{arch}}$ is not far behind. However, the make-span comparisons in Figure 6.3(b) clearly illustrate the approximations made by these methods in order to achieve planning time. $\Delta\text{DUR}_{\text{arch}}$ exhibits a good balance of planning time and solution quality.

Chapter 7

A HYBRIDIZED PLANNER FOR MDPS

While MDPs are a very general framework, popular optimal algorithms (e.g., LAO* [34], Labeled RTDP [10]) do not scale to large problems. In fact, (labeled) RTDP is popular for quickly producing a relatively good policy for *discounted* reward-maximization problems, since the range of infinite horizon total rewards is finite. However, many planning problems, such as those in the planning competition, are undiscounted cost minimization problems with absorbing goals. Intermediate RTDP policies on these problems often contain absorbing cycles. This implies that the expected cost to reach the goal is infinite! Clearly, RTDP is *not* an anytime algorithm for our problems because ensuring that all trajectories reach a goal takes a long time.

Many researchers have argued for planning with a qualitative, non-deterministic model of uncertainty (in contrast to numeric probabilities). Such contingent planners (*e.g.*, MBP [5]) cannot make use of quantitative likelihood (and cost) information. Because they solve a much simpler problem, these planners are able to scale to much larger problems than probabilistic planners. By stripping an MDP of probabilities and costs, one could use a qualitative contingent planner to quickly generate a policy, but the quality of the resulting solution will likely be poor. Thus it is natural to ask "can we develop an algorithm with the benefits of both frameworks?".

Using two (or more) algorithms to obtain the benefits of both is a generic idea that forms the basis of many proposed algorithms, *e.g.*, bound and bound [44] and algorithm portfolios [32]. Various schemes hybridize multiple algorithms differently and are aimed at different objectives. For example, algorithm portfolios run multiple algorithms in parallel and thus reduce the total time to obtain a solution; bound and bound uses a solution from one algorithm as a bound for the other algorithm. In our work we employ a tighter notion of hybridization, where we explicitly incorporate solutions to sub-problems from one algorithm

into the partial solution of the other.

We present a novel algorithm, HYBPLAN, which hybridizes two planners: GPT and MBP. GPT (General Planning Tool) [11] is an exact MDP solver using labeled real time dynamic programming (RTDP). MBP (Model Based Planner) [5], is a non-deterministic planner exploiting binary decision diagrams (BDDs). Our hybridized planner enjoys benefits of both algorithms, *i.e.*, *scalability* and *quality*. To the best of our knowledge, this is the first attempt to bridge the efficiency-expressiveness gap between the planners dealing with qualitative *vs.* probabilistic representations of uncertainty.

HYBPLAN has excellent *anytime* properties — it produces a legal solution very fast and then successively improves on this solution[1]. Moreover, HYBPLAN can effectively handle interleaved planning and execution in an online setting, makes use of the available time and memory efficiently, is able to converge within a desired optimality bound, and reduces to optimal planning given an indefinite time and memory. Our experiments demonstrate that HYBPLAN is competitive with the state-of-the-art planners, solving problems in the International Planning Competition that most other planners could not solve.

7.1 Background: The Model-Based Planner

An MDP *without* cost and probability information translates into a planning problem with qualitative uncertainty. A strong-cyclic solution to this problem — one that admits loops but is free of absorbing cycles — can be used as a legal solution to the original MDP, though it may be highly sub-optimal. However, because we are solving a much easier problem, the algorithms for solving this relaxed problem are highly scalable. One such algorithm is implemented within MBP.

The Model-Based Planner, MBP [5], relies on effective BDD-based representation techniques to implement a set of sound and complete plan search and verification algorithms. All the algorithms within MBP are designed to deal with qualitatively non-deterministic planning domains, defined as Moore machines using MBP's input language SMV. The planner is very general and is capable of accommodating domains with various state observability

[1] While RTDP is popular as an anytime algorithm itself, for problems with absorbing goals it may not return any legal policy for as much as 10 min. or more.

(*e.g.*, fully observable, conformant) and different kinds of planning goals (*e.g.*, reachability goals, temporal goals).

MBP has two variants of strong-cyclic algorithm denoted by "global" and "local". Both share the representation of the policy π as a binary decision diagram, and the fact that it is constructed by backward chaining from the goal. The general idea is to iteratively regress from a set of solution states (the current policy π), first admitting any kind of loop introduced by the backward step, and then removing those "bad" loops for which no chance of goal achievement exists. On top of this, the "local" variant of the algorithm prioritizes solutions with no loops, in order to retrieve strong solutions whenever possible. The search ends either when π covers the initial state, or when a fixed point is reached. Further details can be found in [17].

7.2 HybPlan*: A Hybridized Planner*

Our novel planner, HybPlan, hybridizes GPT and MBP. On the one hand, GPT produces cost-optimal solutions to the original MDP; on the other, MBP ignores probability and cost information, but produces *a* solution extremely quickly. HybPlan combines the two to produce high-quality solutions in intermediate running times.

At a high level, HybPlan invokes GPT with a maximum amount of time, say *hybtime*. GPT preempts itself after running for this much time and passes the control back to HybPlan. At this point GPT has performed several RTDP trials and might have labeled some states solved. However, the whole cost function J_n has not converged and the start state is not yet solved. Despite this, the current greedy partial[2] policy (as given by Equation 2.1) contains much useful information. HybPlan combines this partial policy (π_{GPT}) with the policy from MBP (π_{MBP}) to construct a *hybridized* policy (π_{HYB}) that is defined for all states reachable following π_{HYB}, and is guaranteed to lead to the goal. We may then evaluate π_{HYB} by computing $J_{\mathrm{HYB}}(s_0)$ denoting the expected cost to reach a goal following this policy. In case we are dissatisfied with π_{HYB}, we may run some more RTDP trials and repeat the process. We describe the pseudo-code for the planner in Algorithm 7.

[2]It is partial because some states reachable by the greedy policy might not even be explored yet.

Algorithm 7 HYBPLAN(*hybtime, threshold*)

1: $deadends \leftarrow \emptyset$
2: **repeat**
3: run GPT for *hybtime*
4: $open \leftarrow \emptyset$; $closed \leftarrow \emptyset$
5: $open$.insert(s_0)
6: **while** *open* is non-empty **do**
7: remove s from *open*
8: $closed$.insert(s)
9: **if** s is labeled *solved* inside GPT **then**
10: $\pi_{\mathrm{HYB}}(s) \leftarrow \pi_{\mathrm{GPT}}(s)$
11: **else**
12: **if** $visit(s) > threshold$ **then**
13: $\pi_{\mathrm{HYB}}(s) \leftarrow \pi_{\mathrm{GPT}}(s)$
14: **else**
15: ASSIGNMBPSOLUTION(s)
16: **for all** s' s.t. $\mathcal{P}r(s'|s, \pi_{\mathrm{HYB}}(s)) > 0$, $s' \notin closed$ **do**
17: **if** $s' \in deadends$ **then**
18: ASSIGNMBPSOLUTION(s)
19: **else**
20: $open$.insert(s')
21: remove absorbing cycles from $Exec[\pi_{\mathrm{HYB}}]$
22: evaluate π_{HYB} by computing $J_{\mathrm{HYB}}(s_0)$
23: **if** π_{HYB} is the best policy found so far, cache it
24: **until** all resources exhaust or desired error bound is achieved

Function 8 AssignMbpSolution(s)

```
if MBP(s) succeeds then
  π_HYB(s) ← π_MBP(s)
  for all s' s.t. Pr(s'|s, π_HYB(s)) > 0, s' ∉ closed do
    open.insert(s')
else
  if s = s_0 then
    problem is unsolvable; exit()
  else
    closed.remove(s)
    deadends.insert(s)
    for all s' s.t. π_HYB(s') leads directly to s do
      AssignMbpSolution(s')
```

The construction of the hybridized policy starts from the start state and uses the popular combination of *open* and *closed* lists denoting states for which an action needs to be assigned and have been assigned, respectively. Additionally we maintain a *deadends* list that memoizes all states starting from which we cannot reach a goal (dead-end).

Deciding between GPT and MBP (lines 9 to 15): For every state s, HybPlan decides whether to assign the action using π_{GPT} or π_{MBP}. If s is already labeled *solved* then we are certain that GPT has computed the optimal policy starting from s (lines 8-9). Otherwise, we need to assess our confidence in $\pi_{\text{GPT}}(s)$. We quantitatively estimate our confidence on GPT's greedy policy by keeping a count on the number of times s has been updated inside labeled RTDP. Intuitively, smaller number of visits to a state s corresponds to our low confidence on the quality of $\pi_{\text{GPT}}(s)$ and we may prefer to use $\pi_{\text{MBP}}(s)$ instead. A user-defined *threshold* decides on how quickly we start trusting GPT.

MBP returning with failure (Function 8): Sometimes MBP may return with a failure implying that no solution exists from the current state. Clearly, the choice of the action (in the previous step) is faulty because that step led to this dead-end. The procedure AssignMbpSolution recursively looks at the policy assignments at previous levels and debugs π_{HYB} by assigning solutions from π_{MBP}. Additionally we memoize the dead-end

states to reduce future computation.

Cleaning, Evaluating and Caching $\pi_{\mathbf{HYB}}$ (lines 21-23): We can formulate the evaluation of π_{HYB} as the following system of linear equations:

$$J_{\mathrm{HYB}}(s) = 0, \text{ if } s \in \mathcal{G} \text{ else}$$
$$J_{\mathrm{HYB}}(s) = \mathcal{C}(\pi(s)) + \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s, \pi(s)) J_{\mathrm{HYB}}(s') \tag{7.1}$$

These equations are tricky to solve, because it is still possible that there is an absorbing cycle in $Exec[\pi_{\mathrm{HYB}}]$. If the rank of the coefficient matrix is less than the number of equations then there is an absorbing cycle. We can convert the coefficient matrix into row-echelon form by using row transformations and in parallel perform the same transformations on the identity matrix. When we find a row with all zeros the non-zero entries of the same row in the transformed identity matrix reveals the states in the original system that form absorbing cycle(s). We pick one of these states, assign the MBP action for it and repeat this computation. Note that this system of equations is on a very small fraction of the overall state space (which are in $Exec[\pi_{\mathrm{HYB}}]$), hence this step is not expensive. As we find intermediate hybridized policies, we cache the best (*i.e.*, the one with minimum $J_{\mathrm{HYB}}(s_0)$) policy found so far (line 23).

Termination (line 24): We may terminate differently in different situations: given a fixed amount of time, we may stop GPT when the available time is about to expire and follow it with a hybridized policy computation and terminate. Given a fixed amount of memory, we may do the same with memory. If we need to terminate within a desired fraction of the optimal, we may repeat hybridized policy computation at regular intervals and when we find a policy whose error bound is within the desired limits, we terminate.

Error Bound: We develop a simple procedure to bound the error in π_{HYB}. Since labeled RTDP is always started with an admissible heuristic, GPT's current cost function J_n remains a lower bound of the optimal ($J_n \leq J^*$). However the hybridized policy is clearly worse than the optimal and hence $J_{\mathrm{HYB}} \geq J^*$. Thus, $\frac{J_{\mathrm{HYB}}(s_0) - J_n(s_0)}{J_n(s_0)}$ bounds the error of π_{HYB}.

Properties of HybPlan: HybPlan uses the current greedy policy from GPT, combines it with solutions from MBP for states that are not fully explored in GPT and ensures that the

final hybridized policy is *proper*, *i.e.*, free of absorbing cycles. Thus HybPlan has excellent anytime properties, *i.e.*, once $\pi_{\text{MBP}}(s_0)$ has returned with success, HybPlan is capable of improving the quality of the solution as the time available for the algorithm increases. If infinite time and resources are available for the algorithm, then the algorithm reduces to GPT, whereas if the available resources are extremely limited, then it reduces to MBP. In all other cases, the hybridized planner demonstrates intermediate behavior.

7.2.1 Two Views of the Hybridized Planner

Our hybridized planner may be understood in two ways. The first view is MBP-centric. If we run HybPlan without any GPT computation then only π_{MBP} will be outputted. This solution will be a legal but possibly low quality. HybPlan successively improves the quality of this basic solution from MBP by plugging in additional information from GPT.

An alternative view is GPT-centric. We draw from the intuition that, in GPT, the partial greedy policy (π_{GPT}) improves gradually and eventually gets defined for all relevant states accurately. But before convergence, the current greedy policy may not even be defined for many states and may be inaccurate for others, which have not been explored enough. HybPlan uses this partial policy as much as reasonable and completes it by adding in solutions from MBP; thus making the final policy consistent and useful. In essence, both views are useful: each algorithm patches the other's weakness.

7.2.2 Implementation of HybPlan

We now address two different efficiency issues in implementing HybPlan. First, instead of pre-computing an MBP policy for the whole state space, we do this computation on demand. We modify MBP so that it can efficiently solve the sub-problems without repeating any computation. We modify MBP's "local" strong cyclic planning algorithm in the following ways —

1. We cache the policy table (π_{MBP}) produced by previous planning episodes,

2. At each planning episode, we analyze the cached result π_{MBP}, and if the input state is solved already, search is skipped,

3. We perform search by taking π_{MBP} as a starting point (rather than the goal).

Second, in our implementation we do not evaluate π_{HYB} by solving the system of linear equations. Instead, we approximate this by averaging repeated simulations of the policy from the start state. If any simulation exceeds a maximum trajectory length we guess that the hybrid policy has an absorbing cycle and we try to break the cycle by recursively assigning the action from π_{MBP} for a state in the cycle. This modification speeds up the overall algorithm. Although in theory this takes away the guarantee of reaching the goal with probability 1 since there could be some low probability trajectory not explored by the simulation that may contain an absorbing cycle, in practice this modification is sufficient as even the planning competition relies on policy simulation for evaluating planners. In our experiments, our hybridized policies reach the goal with probability 1.

We finally remark that while GPT takes as input a planning problem in probabilistic PDDL format, MBP's input is a domain in SMV format. Our translation from PDDL to SMV is systematic but only semi-automated; we are implementing a fully automated procedure.

7.3 Experiments

We evaluate HybPlan on the speed of planning, quality of solutions returned, anytime behavior, and scalability to large problems. We also perform a sensitivity experiment testing the algorithm towards sensitivity to the parameters.

Methodology

We compare HybPlan with GPT and MBP. In the graphs we plot the expected cost of the cached policy for both HybPlan and GPT as a function of time. The first value in the HybPlan curve is that of MBP (since initially for each state s, $visit(s) = 0$, and thus $\pi_{\text{HYB}} = \pi_{\text{MBP}}$). We also plot the current $J_n(s_0)$ value from labeled RTDP. As this admissible value increases, the error bound of the solution reduces.

We run the experiments on three large probabilistic PDDL domains. The first two domains are probabilistic variants of the Rovers and MachineShop domains from the 2002 AIPS Planning Competition. The third is the Elevators domain from the 2006 ICAPS Planning

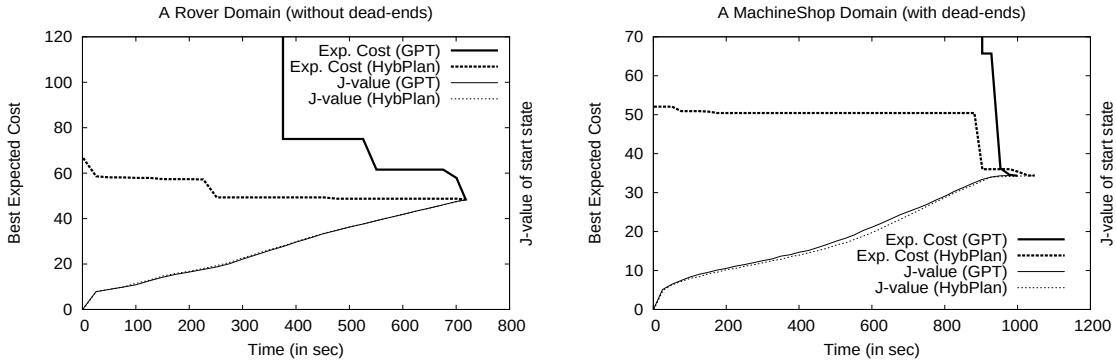

Figure 7.1: Anytime properties of HybPlan: On one Y-axis we show the expected cost of the cached policy and on the other the $J_n(s_0)$ values. J_n converges when the two curves meet. We find that HybPlan's policy is superior to GPT's greedy policy. The first time when GPT's policy has non-infinite expected cost occurs much later in the algorithm.

Competition. The largest problem we attempted was in the Elevators domain and had 606 state variables.

For our experiments we terminate when either labeled RTDP terminates or when the memory goes out of bound. For most experiments, we initialize HybPlan with *hybtime* = 25 sec and *threshold* = 50. We also perform experiments to analyze the sensitivity to these parameters.

7.3.1 Anytime Property

We first evaluate the anytime properties of HybPlan for moderate sized planning problems. We show results on two problems, one from the Rovers domain and the other from the MachineShop domain (Figures 7.1(a), 7.1(b)). These problems had 27 state variables and about 40 actions each. We observe that π_{HYB} has a consistently better expected cost than did π_{GPT}, and that the difference between the two algorithms is substantial. For example, in Figure 7.1(b) the first time, when π_{GPT} has a non-infinite expected cost (*i.e.*, all simulated paths reach the goal), is after 950 seconds; whereas HybPlan always constructs a valid policy. This also validates our claim that for our problems, RTDP is not anytime as its

initial policies are not free of absorbing goals.

Figure 7.1(a) is for a domain without any dead-ends whereas in the domain of Figure 7.1(b) there are some 'bad' actions from which the agent can never recover. While HybPlan obtains greater benefits for domains with dead-ends, for all the domains the anytime nature of HybPlan is superior to the GPT. Also notice the $J_n(s_0)$ values in Figure 7.1. HybPlan sometimes takes marginally longer to converge because of overheads of hybrid policy construction. Clearly this overhead is insignificant.

Recall that the first expected cost of π_{HYB} is the expected cost of following the MBP policy. Clearly an MBP policy is not of very high quality and is substantially improved as time progresses. Also, the initial policy is computed very quickly.

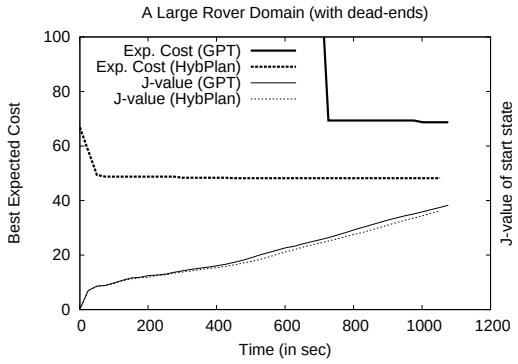

Figure 7.2: Plot of expected cost on a problem too large for GPT to converge.

7.3.2 Scaling to Large Problems

If we desire to run the algorithm until convergence then HybPlan is no better than GPT. For large problems, however, running until convergence is not a practical option due to limited resources. For example, in Figure 7.2 we show experiments on a larger problem from the rovers domain where the memory requirements exceed our machine's 2 GB. (typically, the memory fills up after the algorithm explores about 600,000 states) In such cases hybridization provides even more benefits (Table 7.1). For many of the large domains GPT is in fact unable

Table 7.1: Scalability of HybPlan: Best quality solutions found (before memory exhausts) by GPT, MBP and HybPlan for large problems. HybPlan outperforms the others by substantial margins.

Problems	Time before memory exhausts	Expected Cost		
		GPT	MBP	HybPlan
Rover5	$\sim$ 1100 sec	55.36	67.04	48.16
Rover2	$\sim$ 800 sec	∞	65.22	49.91
Mach9	$\sim$ 1500 sec	143.95	66.50	48.49
Mach6	$\sim$ 300 sec	∞	71.56	71.56
Elev14	$\sim$ 10000 sec	∞	46.49	44.48
Elev15	$\sim$ 10000 sec	∞	233.07	87.46

to output a single policy with finite expected cost. While one might use MBP directly for such problems, by hybridizing the two algorithms we are able to get consistently higher quality solutions.

Notice the Elevator problems in Table 7.1. There are 606 variables in this domain. These problems were the largest test problems in the Elevators domain for the Planning Competition 2006 and few planners could solve it. Thus HybPlan's performance is very encouraging.

7.3.3 Sensitivity to Parameters

HybPlan is controlled by two parameters: *hybtime* and *threshold*. We evaluate how sensitive HybPlan is to these parameters. We find that increasing *hybtime* reduces the total algorithm time but the difference is marginal, implying that the overhead of hybrid policy construction is not significant. However, smaller values result in repeated policy construction and this helps in finding a good quality solution early. Thus small values of *hybtime* are overall more effective.

Varying *threshold* does not affect the overall algorithm time. But it does marginally affect the first time a good solution is observed. Increasing *threshold* implies that an MBP policy is used until a state is sufficiently explored in GPT. For Rovers domain this translates to some extra time before a good policy is observed. For MachineShop we observe the opposite

behavior suggesting that we are better off using MBP policies for less explored regions of the space. While overall the algorithm is only marginally sensitive to this parameter, the differences in two domains lead us to believe that the optimal values are domain-dependent and in general, intermediate values of *threshold* are preferable.

Finally, we also compare with symbolic LAO* [24] to determine whether our speedup is due primarily to the use of a qualitative uncertainty representation or instead to exploitation of symbolic techniques. We observe that for our large problems symbolic LAO* does not converge even after many hours, since backing up the whole ADD takes a huge amount of time. Thus, we conclude that the speedup is due to hybridization with a simpler, qualitative model of uncertainty.

7.3.4 Discussion

Although the computation of a proper policy is a strength of HYBPLAN, it is also a weakness because there exist problems (which we term "improper"), which may not contain even a single proper policy. For these improper problems the algorithm (in its present form) will deem the problem unsolvable, because we have specifically chosen GPT and the strong-cyclic planning algorithm of MBP. Instead, we could hybridize other algorithms (*e.g.*, PARAGRAPH [41] or GPT supplemented with a high but non-infinite cost of reaching a dead-end) with MBP's *weak* planning algorithms. For better results we could combine MBP's strong-cyclic and weak planning algorithms sequentially — if strong-cyclic planner returns failure then apply weak planner. A planner hybridized in this manner would be able to handle these improper problems comfortably and will also guarantee reaching a goal if it is possible.

7.4 The General Framework of Algorithm Hybridization

Having seen two instances of hybridized planners, for CPTP and for MDPs (chapters 5.3 and 7), we now show that this form of algorithm hybridization is quite general and can be employed to several planning and search scenarios.

Any general search optimization problem may be described in terms of four fields attributes — (1) STATES, on which the search is carried out; (2) OPERATORS, denoting the transitions between states; (3) GOALS, the desired subset of STATES; (4) COST, a function

that evaluates the quality of a solution. We wish to compute a solution π : STATES$\rightarrow$ OPERATORS, that takes the agent to a state in GOALS.

As input, algorithm hybridization takes two algorithms for the same search-optimization problem and a confidence function (defined below). The first algorithm (let us call it AQUALITY) is possibly slow, but produces a high quality solution. For MDPs, this is labeled RTDP, which produces the optimal policy.

The second algorithm (AFAST) is expected to be fast but may produce low quality solutions, *e.g.*, a heuristic or *adhoc* algorithm. For MDPs this is a qualitative contingent planner, like MBP. Algorithm hybridization combines the π_{AQUALITY} and multiple solutions from AFAST (π_{AFAST}) to produce a *good* quality solution (π_{HYB}) in time that is *intermediate* between the running times of the two algorithms.

The two main requirements for hybridization to work are:

1. If AQUALITY is not run to completion, one must still be able to extract a partial solution, (π_{AQUALITY}), from the partial execution.

2. AFAST fails only when no solution is possible.

The third input to our the hybridization procedure is a *confidence* function, which estimates the quality of AQUALITY's partial solution for each search state. Hybridization uses this confidence value to *judiciously* combine π_{AQUALITY} with π_{AFAST} to obtain π_{HYB}. We present the pseudo-code for the technique below:

In line 2 of the hybridized algorithm we let AQUALITY run for some time, *i.e.*, expanding some more states[3]. In line 4 we compute a *confidence* value for each state. As explained above this value denotes our confidence in the partial solution of AQUALITY for this state. For our example, if all the descendants of a state have been expanded then we have 100% confidence on the partial solution for this state. On the contrary, if none of the children have been expanded then we have zero confidence. We may also assign intermediate values in a similar fashion. The hybridized algorithm constructs a solution by taking the solution

[3]We can hybridize iterative algorithms as well. For those line 2 implies running some more iterations.

Algorithm 9 Hybridized Algorithm

1: **repeat**
2: run AQUALITY for *hybtime*
3: **for all** states $s \in$ STATES expanded in AQUALITY **do**
4: compute *confidence*(n)
5: **if** *confidence*$(s) >$ *threshold* **then**
6: $\pi_{\text{HYB}}(s) \leftarrow \pi_{\text{AQUALITY}}(s)$
7: **else**
8: ASSIGNFASTSOLUTION(s)
9: **for all** states (s') unexplored and reachable from the current solution: ASSIGNFASTSOLUTION(s)
10: (optionally) clean and evaluate π_{HYB}. (see Algorithm 11).
11: **until** termination

Function 10 ASSIGNFASTSOLUTION(s)

if AFAST(s) succeeds **then**
$\pi_{\text{HYB}}(s) \leftarrow \pi_{\text{AFAST}}(s)$
else
for all s' s.t. $\pi_{\text{HYB}}(s')$ leads directly to s **do**
ASSIGNFASTSOLUTION(s')

from AQUALITY for all states that we have high confidence for, and solutions from AFAST otherwise.

Sometimes AFAST may return with a failure implying that no solution exists from the current state. Clearly, the choice of the operator in the previous step is faulty which led to this dead-end. The procedure ASSIGNFASTSOLUTION recursively looks at the operators at previous levels and debugs π_{HYB}.

The hybridized solution constructed until line 9 may still not be error-free since it is constructed using two separate sources. For example, if we are trying to reach a goal then the solution path might have cycles due to a less informed confidence function. In an AND/OR graph, some of the AND branches might not reach the goal. We can optionally check the solution for consistency and recursively debug any errors by assigning solution from AFAST, whenever required. If we need any termination guarantees then we may evaluate the hybridized solution and terminate when we are satisfied with the solution. Algorithm 11 suggests a way to clean the hybridized policy in case of a regular OR graph search. Note that for this algorithm the only states we need to consider are those that are part of the hybridized solution. This set could be much smaller than STATES, *e.g.*, when an initial state is known.

Algorithm 11 Clean and Evaluate Hybridized Solution(π_{HYB})

1: **while** $\exists$ a cycle c in π_{HYB} **do**
2: pick a state $s \in c$
3: ASSIGNFASTSOLUTION(s)
4: **repeat**
5: evaluate COST of π_{HYB} for each state
6: pick a state $s, s.t.\ \text{COST}_\pi(s) = \infty$
7: ASSIGNFASTSOLUTION(s)
8: **until** $\forall s\ \text{COST}_\pi(s) < \infty$

The hybridized algorithm is anytime, *i.e.*, one which is capable of producing different (typically increasingly better quality) solutions as the time available for the algorithm increases. If infinite time and resources are available for the algorithm, then the hybridized

algorithm reduces to AQUALITY, whereas if the available resources are extremely limited, then it reduces to AFAST. In all other cases, the hybridized algorithm demonstrates intermediate behavior.

Memory Efficient Hybridized Algorithm If available memory is at a premium and we are running a memory efficient version of the search algorithm, which only stores the current frontier of the breadth first search, we may still be able to exploit the advantages of algorithm hybridization. Unfortunately, we may not be able to make use of the *confidence* function since the internal states are not handy. Thus for all states in the frontier, we may compute the solution from AFAST and output the solution that minimizes the cost to reach the frontier state (using AQUALITY) plus cost of the solution from AFAST starting at that state. As an optimization, we may just consider *a few* states from the frontier (instead of all) and optimize over them. These states may be picked greedily or randomly.

The idea of algorithm hybridization is quite general and can be applied to various other settings. Please refer to Section 9.7 for a detailed discussion.

Chapter 8

RELATED WORK

8.1 Concurrent Probabilistic Temporal Planning

This thesis (chapters 3 - 6) extends our prior work, originally reported in [47, 48, 49, 50] and chapter 7 extends [46].

Temporal planners may be classified as using constraint-posting or extended state-space methods (discussed earlier in Chapter 4). While the constraint approach is promising, few (if any) *probabilistic* planners have been implemented using this architecture; one exception is Buridan [39], which performed poorly. In contrast, the MDP community has proven the state-space approach successful. Since the powerful deterministic temporal planners, which have won the various planning competitions, also use the state-space approach, we adopt it for our algorithms that combine temporal planning with MDPs. It may be interesting to incorporate constraint-based approaches in a probabilistic paradigm and compare against the techniques of this paper.

8.1.1 Comparison with Semi-MDPs

A *Semi-Markov Decision Process* is an extension of MDPs that allows durative actions to take variable time. A discrete time semi-MDP can be solved by solving a set of equations that is a direct extension of Equations 2.2. The techniques for solving discrete time semi-MDPs are natural generalizations of those for MDPs. The main distinction between a semi-MDP and our formulation of concurrent probabilistic temporal planning with stochastic durations concerns the presence of concurrently executing actions in our model. A semi-MDP does not allow for concurrent actions and assumes one executing action at a time. By allowing concurrency in actions and intermediate decision epochs, our algorithms need to deal with large state and action spaces, which is not encountered by semi-MDPs.

Furthermore, Younes and Simmons have shown that in the general case, semi-MDPs

Table 8.1: A table listing various planners that implement different subsets of concurrent, stochastic, durative actions.

	concurrent		*non-concurrent*	
	durative	*non-durative*	*durative*	*non-durative*
stochastic	ΔDUR, Tempastic, GSMDP, Prottle FPG, Aberdeen *et al.*,	Concurrent MDP, Factorial MDP, Paragraph	Time Dependent MDP, IxTeT, CIRCA, Foss & Onder	MDP (RTDP, LAO*, *etc.*)
deterministic	Temporal Planning (TP4, SAPA, MIPS TLPlan, *etc.*)	Step-optimal planning (GraphPlan, SATPlan)	Planning with Numerical Resources (Sapa, Metric-FF, CPT)	Classical Planning (HSP, FF, *etc.*)

are incapable of modeling concurrency. A problem with concurrent actions and stochastic continuous durations needs another model known as Generalized Semi-Markov Decision Process (GSMDP) for a precise mathematical formulation [66].

8.1.2 Concurrency and Stochastic, Durative Actions

Tempastic [65] uses a rich formalism (*e.g.* continuous time, exogenous events, and expressive goal language) to generate concurrent plans with stochastic durative actions. Tempastic uses a completely non-probabilistic planner to generate a plan which is treated as a candidate policy and repaired as failure points are identified. This method does not guarantee completeness or proximity to the optimal. Moreover, no attention was paid towards heuristics or search control making the implementation impractical.

GSMDPs [66] extend continuous-time MDPs and semi-Markov MDPs, modeling asynchronous events and processes. Both of Younes and Simmons's approaches handle a strictly more expressive model than ours due to their modeling of continuous time. They solve GSMDPs by approximation with a standard MDP using phase-type distributions. The approach is elegant, but its scalability to realistic problems is yet to be demonstrated. In particular, the approximate, discrete MDP model can require many states yet still behave very differently than the continuous original.

Prottle [42] also solves problems with an action language more expressive than ours:

effects can occur in the middle of action execution and dependent durations are supported. Prottle uses an RTDP-type search guided by heuristics computed from a probabilistic planning graph; however, it plans for a finite horizon — and thus for an acyclic state space. It is difficult to compare Prottle with our approach because Prottle optimizes a different objective function (probability of reaching a goal), outputs a finite-length conditional plan as opposed to a cyclic plan or policy, and is not guaranteed to reach the goal.

FPG [2] learns a separate neural network for each action individually based on the current state. In the execution phase the decision, *i.e.*, whether an action needs to be executed or not, is taken independently of decisions regarding other actions. In this way FPG is able to effectively sidestep the blowup caused by exponential combinations of actions. In practice it is able to very quickly compute high quality solutions.

Rohanimanesh and Mahadevan [60] investigate concurrency in a hierarchical reinforcement learning framework, where abstract actions are represented by *Markov options*. They propose an algorithm based on value-iteration, but their focus is calculating joint termination conditions and rewards received, rather than speeding policy construction. Hence, they consider *all* possible Markov option combinations in a backup.

Aberdeen *et al.* plan with concurrent, durative actions with *deterministic* durations in a specific military operations domain. They apply various domain-dependent heuristics to speed the search in an extended state space [1].

8.1.3 Concurrency and Stochastic, Non-durative Actions

Meuleau *et al.* and Singh & Cohn deal with a special type of MDP (called a factorial MDP) that can be represented as a set of smaller weakly coupled MDPs — the separate MDPs are completely independent except for some common resource constraints, and the reward and cost models are purely additive [52, 61]. They describe solutions in which these sub-MDPs are independently solved and the sub-policies are merged to create a global policy. Thus, concurrency of actions of different sub-MDPs is a by-product of their work. Singh & Cohn present an optimal algorithm (similar to combo-elimination used in DUR_{prun}), whereas Meuleau *et al.*'s domain specific heuristics have no such guarantees. All of the work

in Factorial MDPs assumes that a weak coupling exists and has been identified, but factoring an MDP is a hard problem in itself.

Paragraph [43] formulates the planning with concurrency as a regression search over the probabilistic planning graph. It uses techniques like nogood learning and mutex reasoning to speed policy construction.

Guestrin *et al.* solve the multi-agent MDP problem by using a linear programming (LP) formulation and expressing the value function as a linear combination of basis functions. By assuming that these basis functions depend only on a few agents, they are able to reduce the size of the LP [33].

8.1.4 Stochastic, Non-concurrent, Durative Actions

Many researchers have studied planning with stochastic, durative actions in *absence* of concurrency. For example, Foss and Onder [26] use *simple temporal networks* to generate plans in which the objective function has no time component. Simple Temporal Networks allow effective temporal constraint reasoning and their methods can generate temporally contingent plans.

Boyan and Littman [13] propose *Time-dependent MDPs* to model problems with (non-concurrent) actions having time-dependent, stochastic durations; their solution generates piece-wise linear value functions.

NASA researchers have developed techniques for generating non-concurrent plans with uncertain continuous durations using a greedy algorithm which incrementally adds branches to a straight-line plan [14, 19]. While they handle continuous variables and uncertain continuous effects, their solution is heuristic and the quality of their policies is unknown. Also, since they consider only limited contingencies, their solutions are not guaranteed to reach the goal.

IxTeT is a temporal planner that uses constraint based reasoning within partial order planning [40]. It embeds temporal properties of actions as constraints and does not optimize make-span. CIRCA is an example of a system that plans with uncertain durations where each action is associated with an unweighted set of durations [53].

8.1.5 Deterministic, Concurrent, Durative Actions

Planning with deterministic actions is a comparitively simpler problem and much of the work in planning under uncertainty is based on the previous, deterministic planning research. For instance, our interwoven state representation and transition function are extensions of the extended state representations in TP4, SAPA, and TLPlan [35, 22, 3].

Other planners, like MIPS and AltAlt^p, have also investigated fast generation of parallel plans in deterministic settings [23, 54] and [37] extends it to problems with disjunctive uncertainty.

8.2 Hybridized Planning

Hybridizing planners were introduced recently in the planning community independently by us and McMahan *et al.* [48, 51]. We first used hybridization in the context of concurrent probabilistic temporal planning by hybridizing interwoven epoch and aligned epoch planning. McMahan *et al.* used a sub-optimal policy to achieve policy guarantees in an MDP framework (an extension of bound and bound to MDPs). However, they did not provide any method to obtain this sub-optimal policy. Our hybridization scheme(chapter 7) is the first to present a principled hybridized planner for MDPs.

The strength of our work is in the coupling of a probabilistic and qualitative contingent planner. While there have been several attempts to use *classical* planners to obtain probabilistic solutions (*e.g.*, generate, test and debug [65], FF-Replan), they have severe limitations because conversion to classical languages results in decoupling various outcomes of the same action. Thus the classical planners have no way to reject an action that has two outcomes – one good and one bad. However, this kind of information is preserved in domains with qualitative uncertainty and combining them with probabilistic planners creates a time-quality balanced algorithm.

Chapter 9

FUTURE WORK

Having presented a comprehensive set of techniques to handle probabilistic outcomes, concurrent and durative actions in a single formalism, we now direct our attention towards different relaxations and extensions to the proposed model. In particular, we explore other objective functions, infinite horizon problems, continuous-valued duration distributions, temporally expressive action models, degrees of goal satisfaction and interruptibility of actions.

9.1 Extension to Other Cost Functions

For the planning problems with durative actions (chapters 4 and beyond) we focused on make-span minimization problems. However, our techniques are quite general and are applicable (directly or with minor variations) to a variety of cost metrics. As an illustration, consider the mixed cost optimization problem in which in addition to the duration of each action, we are also given the amount of resource consumed per action, and we wish to minimize the the sum of make-span and total resource usage. Assuming that the resource consumption is unaffected by concurrent execution, we can easily compute a new max-concurrency heuristic. The mixed-cost counterpart for Equations 5.2 is:

$$\begin{aligned} \underline{J^*}(s) &\geq \frac{J^*_t(X)}{c} + J^*_r(X) \qquad \text{for } Y = \emptyset \\ \underline{J^*}(s) &\geq \frac{Q^*_t(X, A_s)}{c} + Q^*_r(X, A_s) \quad \text{for } Y \neq \emptyset \end{aligned} \tag{9.1}$$

Here, J_t is for the single-action MDP assignng costs to be durations and J_r is for the single action MDP assigning costs to be resource consumptions. A more informed average concurrency heuristic can be similarly computed by replacing maximum concurrency by average concurrency. The hybridized algorithm follows in the same fashion, with the fast algorithm being a CoMDP solved using techniques of Chapter 3.

On the same lines, if the objective function is to minimize make-span given a certain maximum resource usage, then the total amount of resource remaining can be included in the state-space for all the CoMDPs and underlying single-action MDPs *etc.* and the same techniques may be used.

9.2 Infinite Horizon Problems

Until now we have designed the techniques for the case of *indefinite* horizon problems, in which an absorbing state is defined as is reachable. For other problems an alternative formulation is preferred that allows for infinite execution but discounts the future costs by multiplying them by a discount factor in each step. Again, our techniques can be suitably extended for such scenario. For example, Theorem 2 gets modified to the following:

$$Q_{\|}(s, A) \geq \gamma^{1-k} Q_{\|}(s, \{a_1\}) + \mathcal{C}_{\|}(A) - \left(\sum_{i=1}^{k} \gamma^{i-k} \mathcal{C}_{\|}(\{a_i\}) \right)$$

Recall that this theorem provides us with the pruning rule, combo-skipping. Thus, we can use Pruned RTDP with the new pruning rule.

9.3 Extensions to Continuous Duration Distributions

Until now we have confined ourselves to actions with discrete durations (refer to Assumption 3). We now investigate the effects of dealing directly with continuous uncertainty in the duration distributions. Let $f_i^T(t)dt$ be the probability of action a_i completing between times $t+T$ and $t+T+dt$, conditioned on action a_i not finishing until time T. Similarly, define $F_i^T(t)$ to be the probability of the action finishing *after* time $t+T$.

Let us consider the extended state $\langle X, \{(a_1, T)\}\rangle$, which denotes that action a_1 started T units ago in the world state X. Let a_2 be an applicable action that is started in this extended state. Define $M = \min(\Delta_M(a_1) - T, \Delta_M(a_2))$, where Δ_M denotes the maximum possible duration of execution for each action. Intuitively, M is the time by which at least one action will complete. Then

$$Q_{\underline{\underline{n+1}}}(\langle X, \{(a_1, T)\}\rangle, a_2) = \int_0^M f_1^T(t) F_2^0(t) \left[t + J_{\underline{\underline{n}}}(\langle X_1, \{a_2, t\}\rangle) \right] dt +$$

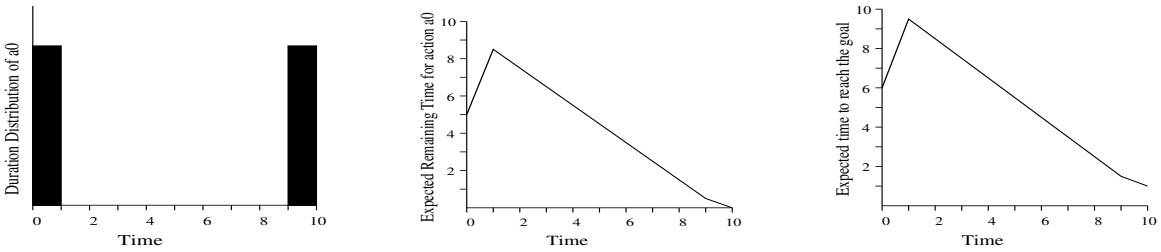

Figure 9.1: If durations are continuous (real-valued) rather than discrete, there may be an infinite number of potentially important decision epochs. In this domain, a crucial decision epoch could be required at any time in $(0, 1]$ — depending on the length of possible alternate plans.

$$\int_0^M F_1^T(t) f_2^0(t) \left[t + J_{\underline{\underline{n}}}(\langle X_2, \{a_1, t+T\}\rangle) \right] dt \qquad (9.2)$$

Here X_1 and X_2 are world states obtained by applying the deterministic actions a_1 and a_2 respectively on X. Recall that $J_{\underline{\underline{n+1}}}(s) = \min_a Q_{\underline{\underline{n+1}}}(s, a)$. For a fixed point computation of this form, we desire that J_{n+1} and J_n have the same functional form[1]. Going by the equation above this seems very difficult to achieve, except perhaps for very specific action distributions in some special planning problems. For example, if all distributions are constant or if there is no concurrency in the domain, then these equations are easily solvable. But for more interesting cases, solving these equations is a challenging open question.

Furthermore, dealing with continuous *multi-modal* distributions worsens the decision epochs explosion. We illustrate this with the help of an example.

Example: Consider the domain of Figure 4.2 except that let action a_0 have a bimodal distribution, the two modes being uniform between 0-1 and 9-10 respectively as shown in Figure 9.1(a). Also let a_1 have a very small duration. Figure 9.1(b) shows the expected remaining termination times if a_0 terminates at time 10. Notice that due to bimodality, this expected remaining execution time increases between 0 and 1. The expected time to reach the goal using plan $\langle \{a_0, a_1\}; a_2 \rangle$ is shown in the third graph. Now suppose, we have started $\{a_0, a_1\}$, and we need to choose the next decision epoch. It is easy to see that the optimal decision epoch could be any point between 0 and 1 and would depend on the alternative routes to the goal. For example, if duration of b_0 is 7.75, then the optimal time-point to

[1] This idea has been exploited in order to plan with continuous resources [25].

start the alternative route is 0.5 (right after the expected time to reach the goal using first plan exceeds 7.75).

Thus, the choice of decision epochs depends on the expected durations of the alternative routes. But these values are not known in advance, in fact these are the ones being calculated in the planning phase. Therefore, choosing decision epochs ahead of time does not seem possible. This makes the optimal continuous multi-modal distribution planning problem mostly intractable for any reasonable sized problem.

9.4 Generalizing the TGP Action Model

The assumption of TGP style actions enables us to compute optimal policies, since we can prune the number of decision epochs. In the case of complex action models like $\text{PDDL}_{2.1}$ [27], all old, deterministic state-space planners are incomplete. For the same reasons, our algorithms are incomplete for problems in $\text{PPDDL}_{2.1}$. Recently, Cushing *et al.* has introduced Tempo, a state-space planner, which uses lifting over time in to achieve completeness [18]. In pursuit of finding a complete, state-space, probabilistic planner for complex action models, a natural step is to consider a Tempo-like representation in a probabilistic setting. While working out the details seems relatively straightforward, the important research challenge will be to find the right heuristics to streamline the search so that the algorithm can scale.

9.5 Other Extensions

There are several other extensions to the basic framework that we have suggested. Each different construct introduces additional structure and we need to exploit the knowledge in order to design fast algorithms. Many times, the basic algorithms proposed in this thesis may be easily adapted to such situations, sometimes they may be not. We list two of the important extensions below.

- **Notion of Goal Satisfaction:** Different problems may require slightly different notions of when a goal is reached. For example, we have assumed thus far that a goal is not "officially achieved" until all executed actions have terminated. Alternatively, one might consider a goal to be achieved if a satisfactory world state is reached, even though

some actions may be in the midst of execution. There are intermediate possibilities in which a goal requires some *specific* actions to necessarily end. Just by changing the definition of the goal set, these problems can be modeled as a CoMDP. The hybridized algorithm and the heuristics can be easily adapted for this case.

- **Interruptible Actions:** We have assumed that, once started, an action cannot be terminated. However, a richer model may allow preemptions, as well as the continuation of an interrupted action. The problems, in which all actions could be interrupted at will, have a significantly different flavor. Interrupting an action is a new kind of decision and requires a full study of when might an action termination be useful. To a large extent, planning with these is similar to finding different concurrent paths to the goal and starting all of them together, since one can always interrupt all the executing paths as soon as the goal is reached. For instance, example in Figure 4.2 no longer holds since b_0 can be started at time 1, and later terminated as needed to shorten the make-span.

9.6 Effect of Large Durations

A weakness of all extended-state space approaches, both in deterministic as well as probabilistic settings, is the dependence on absolute durations (or to be more accurate, the greatest common divisor of action durations). For instance, if the domain has an action a with a large duration, say 100 and another concurrently executable action with duration 1, then all world states will be explored with the tuples $(a, 1)$, $(a, 2)$, ..., $(a, 98)$, $(a, 99)$. In general, many of these states will behave similarly and there will be certain decision boundaries that will be important. "Start b if a has been executing for 50 units and c otherwise" is one example of such a decision boundary. Instead of representing all these flat discrete states individually, planning in an aggregate space in which each state represents several extended states will help alleviate this inefficiency.

However, it is not obvious how to achieve such an aggregation automatically, since adapting the well-known methods for aggregation do not hold in our case. For instance, SPUDD [36] uses algebraic decision diagrams to represent abstract states that have the same J-value.

Aggregating the same valued states may not be enough for us, since the expected time of completion depends linearly on the amount of time left for the longest executing action. So, all the states which differ only by the amount of time an action has been executing will not be able to aggregate together. In a similar way, Feng *et al.* [25] use piecewise constant and piecewise linear representations to adaptively discretize continuous variables. In our case, we have $|\mathcal{A}|$ of such variables. While only a few of them that are executing are active at a given time, modeling a sparse high-dimensional value function is not easy either. Being able to exploit this structure due to action durations is an essential future direction in order to scale the algorithms to complex real world domains.

9.7 Applications of Algorithm Hybridization

The idea of algorithm hybridization is quite general and can be applied to several other planning and search settings. We list several problems that could make good use of our idea of hybridization::

1. Traveling Salesman Problem : TSP may be viewed as search in a space where each state remembers all cities visited and the current city. An approximate solution to the TSP is by computing the minimum spanning tree and visiting each city along the MST. This gives a 2-approximate solution. We may hybridize the two approaches to find a solution with even better quality.

2. Partially Observable Markov Decision Processes (POMDP): The scalability issues in POMDPs are much more pronounced than in MDPs due to the exponential blowup of the continuous belief-state representation. We could hybridize GPT or point-based value iteration based methods [57, 58] with disjunctive planners like MBP, BBSP, or POND [6, 59, 15] to gain the benefits of both the probabilistic and disjunctive representations.

3. Deterministic Over-subscription Planning: The objective of over-subscription planning problem is to maximize return by achieving as many goals as possible given the resource constraints [63]. The problem may be modeled as a search in the state space extended

with the goals already achieved. A heuristic solution to the problem could be to achieve the goals greedily in the order of decreasing returns. We can hybridize the two algorithms to obtain a better than greedy, faster than optimal solution for the problem.

4. Probabilistic Planning with Continuous Resources: A Hybrid AO* algorithm on an abstract state space solves this problem. Here each abstract node consists of the discrete component of the state space and contains the value function for all values of continuous resources [45]. This Hybrid AO* algorithm may be hybridized with an algorithm that solves a simpler problem assuming deterministic resource consumption for each action with the deterministic value being the average of consumption distribution.

5. Concurrent MDP: A concurrent MDP is an MDP with a factored action representation and is used to model probabilistic planning problems with concurrency [47]. A concurrent MDP algorithm and the equivalent single action MDP algorithm could be hybridized together to obtain a fast concurrent MDP solver.

Chapter 10

CONCLUSIONS

Although concurrent and durative actions with stochastic effects characterize many real-world domains, few planners can handle all these challenges in concert. This thesis proposes a unified state-space based framework to model and solve such problems. State space formulations are popular both in deterministic temporal planning as well as in probabilistic planning. However, each of these features bring in additional complexities to the formulation and afford new solution techniques. We develop the "DUR" family of algorithms to alleviates these complexities. We evaluate the techniques on the running times and qualities of solutions produced. Moreover, we study the theoretical properties of these domains and also identify key conditions under which fast, optimal algorithms are possible. Finally, we also develop HybPlan, a new planner for MDPs, that combines two popular existing approaches and obtains the benefits of both. We make the following contributions:

1. We define Concurrent MDPs (CoMDP) — an extension of the MDP model to formulate a stochastic planning problem with concurrent actions. A CoMDP can be cast back into a new MDP with an extended action space. Because this action space is possibly exponential in the number of actions, solving the new MDP naively may take a huge performance hit. We develop the general notions of *pruning* and *sampling* to speed up the algorithms. Pruning refers to pruning of the provably sub-optimal action-combinations for each state, thus performing less computation but still guaranteeing optimal solutions. Sampling-based solutions rely on an intelligent sampling of action-combinations to avoid dealing with their exponential number. This method converges orders of magnitude faster than other methods and produces near-optimal solutions.

2. We formulate the planning with concurrent, durative actions as a CoMDP in two modified state spaces — aligned epoch, and interwoven epoch. While aligned epoch

based solutions run very fast, interwoven epoch algorithms yield a much higher quality solutions. We also define two heuristic functions — maximum concurrency (MC), and average concurrency (AC) to guide the search. MC is an admissible heuristic, whereas AC, while inadmissible, is typically more-informed leading to better computational gains. We call our algorithms the "DUR" family of algorithms. The subscripts *samp* or *prun* refer to sampling and pruning respectively, optional superscripts AC or MC refer to the heuristic employed, if any and an optional "Δ" before DUR notifies a problem with stochastic durations. For example, Labeled RTDP for a deterministic duration problem employing sampling and started with AC heuristic will be abbreviated as $\mathrm{DUR}^{\mathrm{AC}}_{\mathrm{samp}}$.

3. We also develop the general technique of hybridizing two planners. Hybridizing interwoven-epoch and aligned-epoch CoMDPs yields a much more efficient algorithm, $\mathrm{DUR}_{\mathrm{hyb}}$. The algorithm has a parameter, which can be varied to trade-off speed against optimality. In our experiments, $\mathrm{DUR}_{\mathrm{hyb}}$ quickly produces near-optimal solutions. For larger problems, the speedups over other algorithms are quite significant. The hybridized algorithm can also be used in an anytime fashion thus producing good-quality proper policies (policies that are guaranteed to reach the goal) within a desired time.

4. Uncertainty in durations leads to more complexities because in addition to state and action spaces, there is also a blowup in the branching factor and in the number of decision epochs. We bound the space of decision epochs in terms of pivots (times when actions may potentially terminate) and conjecture further restrictions, thus making the problem tractable. We also propose two algorithms, the expected duration planner ($\Delta\mathrm{DUR}_{\mathrm{exp}}$) and the archetypal duration planner ($\Delta\mathrm{DUR}_{\mathrm{arch}}$), which successively solve small planning problems each with no or limited duration uncertainty, respectively. $\Delta\mathrm{DUR}_{\mathrm{arch}}$ is also able to make use of the additional structure offered by multi-modal duration distributions. These algorithms perform much faster than other techniques. Moreover, $\Delta\mathrm{DUR}_{\mathrm{arch}}$ offers a good balance between planning time *vs.* solution quality tradeoff.

5. Besides our focus on stochastic actions, we expose important theoretical issues related with durative actions which have repercussions to deterministic temporal planners as well. In particular, we prove that all common state-space temporal planners are incomplete in the face of expressive action models, *e.g.*, $\text{PDDL}_{2.1}$, a result that may have a strong impact on the future temporal planning research [18].

6. We present a novel probabilistic planning algorithm (HYBPLAN) that combines two popular planners in a principled way. By hybridizing an optimal (but slow) planner (GPT) with a fast (but sub-optimal) planner (MBP) we obtain the best of both worlds. We empirically test HYBPLAN on a suite of medium and large problems, finding that it has significantly better anytime properties than RTDP. Given limited resources, it is able to solve much larger problems while still computing high quality solutions. Furthermore, HYBPLAN is competitive with the best planners in the 2006 International Planning Competition. Our notion of hybridization is a general one, and we discuss several other planning and search applications where we believe that the idea will be effective.

Overall, this thesis proposes a large set of techniques that are useful in modeling and solving planning problems employing stochastic effects, concurrent executions and durative actions with duration uncertainties. The algorithms range from fast but suboptimal solutions, to relatively slow but optimal. Various algorithms that explore different intermediate points in this spectrum are also presented. We hope that our techniques will be useful in scaling the planning techniques to real world problems in the future and will play their part towards making the dreams of AI, a reality.

BIBLIOGRAPHY

[1] D. Aberdeen, S. Thiebaux, and L. Zhang. Decision-theoretic military operations planning. In *ICAPS'04*, 2004.

[2] Douglas Aberdeen and Olivier Buffet. Concurrent probabilistic temporal planning with policy-gradients. In *ICAPS'07*, 2007.

[3] F. Bacchus and M. Ady. Planning with resources and concurrency: A forward chaining approach. In *IJCAI'01*, pages 417–424, 2001.

[4] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[5] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a Model Based Planner. In *ICAI-2001 workshop on Planning under Uncertainty and Incomplete Information*, pages 93–97, 2001.

[6] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Strong planning under partial observability. *Artificial Intelligence*, 170:337–384, 2006.

[7] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[8] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):281–300, 1997.

[9] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.

[10] B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *ICAPS'03*, pages 12–21, 2003.

[11] B. Bonet and H. Geffner. mGPT: A probabilistic planner based on heuristic search. *JAIR*, 24:933, 2005.

[12] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *J. Artificial Intelligence Research*, 11:1–94, 1999.

[13] Justin A. Boyan and Michael L. Littman. Exact solutions to time-dependent MDPs. In *NIPS'00*, page 1026, 2000.

[14] John Bresina, Richard Dearden, Nicolas Meuleau, David Smith, and Rich Washington. Planning under continuous time and resource uncertainty : A challenge for AI. In *UAI'02*, 2002.

[15] D. Bryce, S. Kambhampati, and D. E. Smith. Planning graph heuristics for belief space search. *JAIR*, 26:35–99, 2006.

[16] Y. Chen, B. W. Wah, and C. Hsu. Temporal planning using subgoal partitioning and resolution in sgplan. *JAIR*, 26:323, 2006.

[17] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

[18] William Cushing, Subbarao Kambhampati, Mausam, and Daniel S. Weld. When is temporal planning really *temporal*? In *IJCAI'07*, 2007.

[19] Richard Dearden, Nicolas Meuleau, Sailesh Ramakrishnan, David E. Smith, and Rich Washington. Incremental Contingency Planning. In *ICAPS'03 Workshop on Planning under Uncertainty and Incomplete Information*, 2003.

[20] Minh B. Do and Subbarao Kambhampati. Solving planning graph by compiling it into CSP. In *AIPS'00*, pages 82–91, 2000.

[21] Minh B. Do and Subbarao Kambhampati. Sapa: A domain-independent heuristic metric temporal planner. In *ECP'01*, 2001.

[22] Minh B. Do and Subbarao Kambhampati. Sapa: A scalable multi-objective metric temporal planner. *JAIR*, 20:155–194, 2003.

[23] S. Edelkamp. Taming numbers and duration in the model checking integrated planning system. *Journal of Artificial Intelligence Research*, 20:195–238, 2003.

[24] Z. Feng and E. Hansen. Symbolic heuristic search for factored Markov decision processes. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 2002.

[25] Zhengzhu Feng, Richard Dearden, Nicolas Meuleau, and Rich Washington. Dynamic programming for structured continuous Markov decision processes. In *UAI'04*, page 154, 2004.

[26] Janae Foss and Nilufer Onder. Generating temporally contingent plans. In *IJCAI'05 Workshop on Planning and Learning in Apriori Unknown or Dynamic Domains*, 2005.

[27] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR Special Issue on 3rd International Planning Competition*, 20:61–124, 2003.

[28] Maria Fox and Keith Halsey. An investigation into the expressive power of PDDL2.1. In *ECAI'04*, pages 328–342, 2004.

[29] A. Gerevini, B. Bonet, and R. Givan. 5th International Planning Competition, 2006.

[30] A. Gerevini and I. Serina. LPG: A planner based on local search for planning graphs with action graphs. In *AIPS'02*, page 281, 2002.

[31] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. Morgan Kaufmann, 2004.

[32] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001.

[33] C. Guestrin, D. Koller, and R. Parr. Max-norm projections for factored MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 673–682, 2001.

[34] E. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001.

[35] Patrick Haslum and Hector Geffner. Heuristic planning with time and resources. In *ECP'01*, 2001.

[36] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288, 1999.

[37] R. M. Jensen and M. Veloso. OBDD=based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research*, 13:189, 2000.

[38] H. Kautz and B. Selman. Planning as satisfiability. In *ECAI'92*, 1992.

[39] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286, 1995.

[40] Philippe Laborie and Malik Ghallab. Planning with sharable resource constraints. In *IJCAI'95*, page 1643, 1995.

[41] I. Little and S. Thiebaux. Concurrent probabilistic planning in the graphplan framework. In *ICAPS'06*, 2006.

[42] Iain Little, Douglas Aberdeen, and Sylvie Thiebaux. Prottle: A probabilistic temporal planner. In *AAAI'05*, 2005.

[43] Iain Little and Sylvie Thiebaux. Concurrent probabilistic planning in the graphplan framework. In *ICAPS'06*, 2006.

[44] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[45] Mausam, E. Benazara, R. Brafman, N. Meuleau, and E. Hansen. Planning with continuous resources in stochastic domains. In *IJCAI'05*, page 1244, 2005.

[46] Mausam, Piergiorgio Bertoli, and Daniel Weld. A hybridized planner for stochastic domains. In *IJCAI'07*, 2007.

[47] Mausam and Daniel Weld. Solving concurrent Markov decision processes. In *AAAI'04*, 2004.

[48] Mausam and Daniel Weld. Concurrent probabilistic temporal planning. In *ICAPS'05*, pages 120–129, 2005.

[49] Mausam and Daniel Weld. Challenges for temporal planning with uncertain durations. In *ICAPS'06*, 2006.

[50] Mausam and Daniel Weld. Probabilistic temporal planning with uncertain durations. In *AAAI'06*, 2006.

[51] H. Brendan McMahan, Maxim Likhachev, and Geoffrey J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML'05*, 2005.

[52] Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Kaelbling, Thomas Dean, and Craig Boutilier. Solving very large weakly coupled Markov Decision Processes. In *AAAI'98*, pages 165–172, 1998.

[53] D. Musliner, D. Murphy, and K. Shin. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence*, 74:83–127, 1991.

[54] R. S. Nigenda and S. Kambhampati. Altalt-p: Online parallelization of plans with heuristic state search. *Journal of Artificial Intelligence Research*, 19:631–657, 2003.

[55] J.S. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *KR'92*, pages 103–114, 1992.

[56] J.S. Penberthy and D. Weld. Temporal planning with continuous change. In *AAAI'94*, page 1010, 1994.

[57] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI'03*, pages 1025–1032, 2003.

[58] P. Poupart. Exploiting structure to efficiently solve large scale partially observable Markov decision processes. Ph.d. thesis, University of Toronto, 2005.

[59] J. Rintanen. Conditional planning in the discrete belief space. In *IJCAI'05*, 2005.

[60] Khashayar Rohanimanesh and Sridhar Mahadevan. Decision-Theoretic planning with concurrent temporally extended actions. In *UAI'01*, pages 472–479, 2001.

[61] Satinder Singh and David Cohn. How to dynamically merge markov decision processes. In *NIPS'98*. The MIT Press, 1998.

[62] D. Smith and D. Weld. Temporal graphplan with mutual exclusion reasoning. In *IJCAI'99*, pages 326–333, Stockholm, Sweden, Aug 1999. San Francisco, CA: Morgan Kaufmann.

[63] D. E. Smith. Choosing objectives in over-subscription planning. In *ICAPS'04*, page 393, 2004.

[64] Vincent Vidal and Hector Geffner. Branching and pruning: An optimal temporal pocl planner based on constraint programming. *AIJ*, 170(3):298–335, 2006.

[65] Håkan L. S. Younes and Reid G. Simmons. Policy generation for continuous-time stochastic domains with concurrency. In *ICAPS'04*, page 325, 2004.

[66] Håkan L. S. Younes and Reid G. Simmons. Solving generalized semi-markov decision processes using continuous phase-type distributions. In *AAAI'04*, page 742, 2004.

[67] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI'95*, pages 1114–1120. Morgan Kaufmann, 1995.

APPENDIX A: PROOF OF THEOREM 6

We now prove the statement of Theorem 6, *i.e.*, if all actions are *TGP-style* then the set of pivots suffices for optimal planning. In the proof make use of the fact that if all actions are *TGP-style* then a consistent execution of any concurrent plan requires that any two executing actions be non-mutex (refer to Section 5 for an explanation on that). In particular, none of their effects conflict and a precondition of one does not conflict with the effects of the another.

We prove our theorem by contradition. Let us assume that for a problem each optimal solution requires at least one action to start at a non-pivot. Let us consider one of those optimal plans, in which the first non-pivot point at which an action needs to start at a non-pivot is minimized. Let us name this time point t and let the action that starts at that point be a. We now prove by a case analysis that we may, as well, start a at time $t-1$ without changing the nature of the plan. If $t-1$ is also a non-pivot then we contradict the hypothesis that t is the *minimum* first non-pivot point. If $t-1$ is pivot then our hypothesis is contradicted because a does not "need to" start at a non-pivot.

To prove that a can be left-shifted by 1 unit, we take up one trajectory at a time (recall that actions could have several durations) and consider all actions playing a role at $t-1$, t, $t+\Delta(a)-1$, and $t+\Delta(a)$, where $\Delta(a)$ refers to the duration of a in this trajectory. Considering these points suffice, since the system state does not change at any other points on the trajectory. We prove that the execution of none of these actions is affected by this left shift. There are the following twelve cases:

1. $\forall$actions b that start at $t-1$: b can't end at t (t is a non-pivot). Thus a and b do execute concurrently after t, implies that a and b are non-mutex. Thus a and b may as well start together.

2. $\forall$actions b that continue execution at $t-1$: Use the argument similar to case 1 above.

3. $\forall$actions b that end at $t-1$: Because b is *TGP-style*, its effects are realized in the open interval ending at $t-1$. Therefore, start of a does not conflict with the end of b.

4. $\forall$actions b that start at t: Because a and b start together they are not dependent on each other for preconditions. Also, they are non-mutex, thus their starting times can be shifted in either direction.

5. $\forall$actions b that continue execution at t: If b was started at $t-1$ refer to case 1 above. If not, t and $t-1$ are both similar points for b.

6. $\forall$actions b that end at t: Case not possible due to the assumption that t is a non-pivot.

7. $\forall$actions b that start at $t+\Delta(a)-1$: Since a continued execution at this point, a and b are non-mutex. Thus a's effects do not clobber b's preconditions. Hence, b can still be executed after realizing a's effects.

8. $\forall$actions b that continue execution at $t+\Delta(a)-1$: a and b are non-mutex, so a may end earlier without any effect of b.

9. $\forall$actions b that end at $t+\Delta(a)-1$: a and b were executing concurrently. Thus they are non-mutex. So they may end together.

10. $\forall$actions b that start at $t+\Delta(a)$: b may still start at $t+\Delta(a)$, since the state of $t+\Delta(a)$ doesn't change.

11. $\forall$actions b that continue execution at $t+\Delta(a)$: If b was started at $t+\Delta(a)-1$ refer to case 7 above, else there is no state change at $t+\Delta(a)$ to cause any effect on b.

12. $\forall$actions b that end at $t+\Delta(a)$: a and b are non-mutex because they were executing concurrently. Thus, a's effects don't clobber b's preconditions. Hence, a may end earlier.

Since a can be left shifted in all the trajectories, therefore the left-shift is legal. Also, if there are multiple actions a that start at t they may each be shifted one by one using the same argument. Hence Proved. □

APPENDIX B: THE DOMAIN DESCRIPTIONS

The Rover Domain Description: Unimodal durations

```
(define (domain NasaRover)
   (:model (:dynamics :probabilistic) (:feedback :complete))
   (:types Rover Store Camera Objective Rock Hand)
   (:predicates
     (have_rock_analysis Rover Rock)
     (communicated_rock_data Rock)
     (full Store)
     (ready_to_drop Store)
     (calibrated Camera Objective)
     (have_image Rover Objective)
     (communicated_image_data Objective)
     (store_of Store Rover)
     (on_board Camera Rover)
     (free_h Hand)
     (free_c Camera)
     (good Hand)
     (hand_of Hand Rover)
     (ready Hand Rock)
   )


(:action sample_rock_good
:parameters
   ?x - Rover
   ?s - Store
   ?r - Rock
   ?h - Hand
:precondition (:and (store_of ?s ?x) (hand_of ?h ?x) (good ?h))
:effect
   (:probabilistic
     (0.9
       (:when (:and (ready ?h ?r) (:not (full ?s)) (:not (ready_to_drop ?s)))
         (:set (full ?s) true) (:set (have_rock_analysis ?x ?r) true) (:set (free_h ?h)
true) (:set (ready ?h ?r) false))
       (:when (:or (:not (ready ?h ?r)) (full ?s) (ready_to_drop ?s))
         (:set (ready ?h ?r) false) (:set (free_h ?h) true) (:set (full ?s) true))
```

```
     )
     (0.05 (:set (ready ?h ?r) false) (:set (free_h ?h) true))
   )
:duration :normal 8.0 1.0
)
(:action sample_rock
:parameters
   ?x - Rover
   ?s - Store
   ?r - Rock
   ?h - Hand
:precondition (:and (store_of ?s ?x) (hand_of ?h ?x) (:not (good ?h)))
:effect
   (:probabilistic
     (0.7
       (:when (:and (ready ?h ?r) (:not (full ?s)) (:not (ready_to_drop ?s)))
         (:set (full ?s) true) (:set (have_rock_analysis ?x ?r) true) (:set (free_h ?h)
true) (:set (ready ?h ?r) false))
       (:when (:or (:not (ready ?h ?r)) (full ?s) (ready_to_drop ?s))
         (:set (ready ?h ?r) false) (:set (free_h ?h) true) (:set (full ?s) true))
     )
     (0.05 (:set (ready ?h ?r) false) (:set (free_h ?h) true))
   )
:duration :normal 9.0 1.0
)
(:action drop
:parameters
   ?y - Store
:effect
   (:when (ready_to_drop ?y) (:set (full ?y) false) (:set (ready_to_drop ?y) false))
:duration :constant 3.0
)
(:action calibrate
:parameters
   ?i - Camera
   ?t - Objective
:effect (:set (calibrated ?i ?t) true)
:duration :uniform 6.0 11.0
)
(:action turn_on_dropping
:parameters
   ?s - Store
```

```
:effect (:set (ready_to_drop ?s) true)
:duration :constant 2.0
)
(:action turn_on_good_hand
:parameters
   ?h - Hand
   ?r - Rock
:precondition (:not (good ?h))
:effect (:when (free_h ?h) (:set (ready ?h ?r) true) (:set (free_h ?h) false))
:duration :constant 3.0
)
(:action turn_on_hand
:parameters
   ?h - Hand
   ?r - Rock
:precondition (good ?h)
:effect (:probabilistic
   (0.8 (:when (free_h ?h) (:set (ready ?h ?r) true) (:set (free_h ?h) false)))
)
:duration :constant 2.0
)
(:action take_image
:parameters
   ?r - Rover
   ?o - Objective
   ?i - Camera
:precondition (:and (calibrated ?i ?o) (on_board ?i ?r) )
:effect
   (:probabilistic
     (0.9
       (:when (calibrated ?i ?o) (:set (have_image ?r ?o) true) (:set (calibrated ?i ?o)
false))
       (:when (:not (calibrated ?i ?o)) (:set (have_image ?r ?o) false)))
     (0.1 (:set (have_image ?r ?o) false) (:set (calibrated ?i ?o) false))
   )
:duration :normal 8.0 2.0
)
(:action communicate_rock_data
:parameters
   ?x - Rover
   ?r - Rock
:precondition (have_rock_analysis ?x ?r)
```

```
:effect
   (:probabilistic
     (0.6 (:set (communicated_rock_data ?r) true))
     (0.4 (:set (communicated_rock_data ?r) false))
   )
:duration :uniform 4.0 7.0
)
(:action communicate_image_data
:parameters
   ?r - Rover
   ?o - Objective
:precondition (have_image ?r ?o)
:effect
   (:probabilistic
     (0.6 (:set (communicated_image_data ?o) true))
     (0.4 (:set (communicated_image_data ?o) false))
   )
:duration :uniform 5.0 6.0
)
)

(define (problem p1)
   (:domain NasaRover)
   (:objects
     r1 - Rover
     s1 - Store
     x1 x2 - Rock
     h1 h2 - Hand
   )
   (:init
     (:set (have_rock_analysis r1 x1) false)
     (:set (have_rock_analysis r1 x2) false)
     (:set (communicated_rock_data x1) false)
     (:set (communicated_rock_data x2) false)
     (:set (full s1) false)
     (:set (ready_to_drop s1) false)
     (:set (store_of s1 r1) true)
     (:set (free_h h1) true)
     (:set (free_h h2) true)
     (:set (good h1) true)
     (:set (good h2) false)
     (:set (hand_of h1 r1) true)
```

```
    (:set (hand_of h2 r1) true)
    (:set (ready h1 x1) false)
    (:set (ready h1 x2) false)
    (:set (ready h2 x1) false)
    (:set (ready h2 x2) false)
  )
  (:goal (:and (communicated_rock_data x1) (communicated_rock_data x2))))
```

The MachineShop Domain Description: Multimodal durations

```
(define (domain MachineShop)
   (:model (:dynamics :probabilistic) (:feedback :complete))
   (:types Piece Machine)
   (:predicates
     (shaped Piece)
     (painted Piece)
     (smooth Piece)
     (canpolpaint Machine)
     (canlatroll Machine)
     (cangrind Machine)
     (at Piece Machine)
     (on Piece Machine)
     (hasimmersion Machine)
     (free Machine)
   )

(:action spraypaint
:parameters
  ?x - Piece
  ?m - Machine
:precondition (canpolpaint ?m)
:effect
   (:probabilistic
     (0.8
       (:when (on ?x ?m)
         (:set (painted ?x) true))
     )
   )
:duration :uniform 6.0 8.0
)
(:action immersionpaint
:parameters
```

```
  ?x - Piece
  ?m - Machine
:precondition (canpolpaint ?m)
:effect
   (:probabilistic
     (0.57
       (:when (:and (on ?x ?m) (hasimmersion ?m))
         (:set (painted ?x) true))
     )
     (0.38
       (:when (:and (on ?x ?m) (hasimmersion ?m))
         (:set (painted ?x) true) (:set (hasimmersion ?m) false))
     )
     (0.02
       (:when (:and (on ?x ?m) (hasimmersion ?m))
         (:set (hasimmersion ?m) false))
     )
   ) :duration :constant 4.0
)
(:action lathe
:parameters
  ?x - Piece
  ?m - Machine
:precondition (canlatroll ?m)
:effect
   (:probabilistic
     (0.9
       (:when (on ?x ?m)
         (:set (shaped ?x) true) (:set (painted ?x) false) (:set (smooth ?x) false))
     )
   )
:duration :uniform 2.0 5.0
)
(:action grind
:parameters
   ?x - Piece
  ?m - Machine
:precondition (cangrind ?m)
:effect
   (:probabilistic
     (0.9
       (:when (on ?x ?m)
```

```
         (:set (smooth ?x) true))
     )
   )
:duration :normal 4.0 1.0
)
(:action buyimmersion
:parameters ?m - Machine
:effect
   (:set (hasimmersion ?m) true)
:duration :multinormal 2 1.0 2.0 1.0 1.0 10.0 1.0
)
(:action place
:parameters
   ?x - Piece
  ?m - Machine
:effect    (:when (:and (at ?x ?m) (free ?m))
     (:set (on ?x ?m) true) (:set (free ?m) false)
   )
:duration :constant 1.0
)
(:action move
:parameters
   ?x - Piece
  ?m1 - Machine
  ?m2 - Machine
:precondition (:not (= ?m1 ?m2))
:effect
   (:probabilistic
     (0.9
       (:when (on ?x ?m1)
         (:set (at ?x ?m2) true) (:set (at ?x ?m1) false) (:set (on ?x ?m1) false) (:set
(free ?m1) true))
       (:when (:and (:not (on ?x ?m1)) (at ?x ?m1))
         (:set (at ?x ?m2) true) (:set (at ?x ?m1) false))
     )
     (0.1
       (:when (on ?x ?m1)
         (:set (on ?x ?m1) false) (:set (free ?m1) true))
     )
   )
:duration :constant 3.0
)
```

```
)

(define (problem p1)
   (:domain MachineShop)
   (:objects
     x1 - Piece
     m1 m2 - Machine
   )
   (:init
     (:set (at x1 m2) true)
     (:set (canpolpaint m1) true)
     (:set (canlatroll m2) true)
     (:set (cangrind m2) true)
     (:set (free m1) true)
     (:set (free m2) true)
   )
   (:goal (:and (shaped x1) (painted x1) (free m1) (free m2) (smooth x1))))
```

CURRICULUM VITAE: MAUSAM

Research Interests

Artificial Intelligence: Markov Decision Processes, Automated Planning, Heuristic Search, Machine Learning, Probabilistic Reasoning, Information extraction over the Web, First Order Probabilistic Models, Temporal Reasoning.

Education

2004 – 2007 **Ph.D.** Advisor : Daniel S. Weld
Dissertation: *Stochastic Planning with Concurrent, Durative Actions*
Computer Science and Engineering. University of Washington, Seattle.

2001 – 2004 **Master of Science**. Advisor : Daniel S. Weld
M.S. Thesis: *Solving Concurrent Markov Decision Processes*
Computer Science and Engineering. University of Washington, Seattle.

1997 – 2001 **Bachelor of Technology (G.P.A.: 9.45/10, Dept. Rank 1)**
Computer Science and Engineering. Indian Institute of Technology, Delhi.

Research Experience and Collaborations

Jun'02 – present **Research Assistant**, University of Washington, Seattle
Decision-Theoretic approaches to Concurrent Probabilistic Temporal Planning. Relational MDPs, Hierarchical MDPs.

Apr'06 – present **Collaboration with Yochan Research Group**, Arizona St. Univ., Tempe
Decision Epoch Temporal Planners.
(Collaborators: Subbarao Kambhampati, William Cushing)

Mar'06 – present **Collaboration with Istituto per la Ricerca Scientifica e Tecnologica**, Italy
Hybridizing Planners for Probabilistic Planning. (Collaborator: Piergiorgio Bertoli)

Oct – Dec'04 **Intern, NASA Ames Research Center**, Mountain View, CA

Probabilistic Planning with Continuous Resources.
(Mentors: Ronen Brafman, Nicolas Meuleau)

June – Aug'00 **Intern, Max Planck Institut fur Informatik, Saarbrucken, Germany**
Fast solutions to sorting problems. (Mentor: Peter Sanders)

Teaching and Mentoring Experience

Mar'05 – Jan'06 **Co-advisor, Jiun-Hung Chen, Master's Thesis.**
Naive Bayes POMDPs.

Nov'03, May'04 **Guest Lecturar in the AI courses**
Markov Decision Processes, Probabilistic Planning.

Oct'01 – May'02 **Teaching Assistant**, University of Washington
Digital Hardware Design, Data Structures, Introduction to Programming I.

Jan – May'01 **Teaching Assistant**, Indian Institute of Technology, Delhi
Data Structures.

Tutorial

- Mausam, David E. Smith, Sylvie Thiebaux. "Probabilistic Temporal Planning". A half-day tutorial accepted at *International Conference on Automated Planning and Scheduling (ICAPS).* Providence, RI, USA. September 2007.

Publications

Refereed Journal Papers

- Mausam, Daniel S. Weld. "Planning with Durative Actions in Stochastic Domains". Accepted for publication in *Journal of Artificial Intelligence Research.*

Refereed Conference Papers

- William Cushing, Subbarao Kambhampati, Karthik Talamadupula, Daniel S. Weld, Mausam. "Evaluating Temporal Planning Domains". *International Conference on*

Automated Planning and Scheduling (ICAPS). Providence, RI, USA. September 2007.

- Mausam, Piergiorgio Bertoli, Daniel S. Weld. "A Hybridized Planner for Stochastic Domains". *International Joint Conference on Artificial Intelligence (IJCAI).* Hyderabad, A.P., India. January 2007.

- William Cushing, Subbarao Kambhampati, Mausam, Daniel S. Weld. "When is Temporal Planning *Really* Temporal?". *International Joint Conference on Artificial Intelligence (IJCAI).* Hyderabad, A.P., India. January 2007.

- Mausam, Daniel S. Weld. "Probabilistic Temporal Planning with Uncertain Durations". *National Conference on Artificial Intelligence (AAAI).* Boston, MA, USA. July 2006.

- Mausam, Daniel S. Weld. "Challenges for Temporal Planning with Uncertain Durations". *International Conference on Automated Planning and Scheduling (ICAPS).* The English Lake District, UK. June 2006.

- Mausam, Emmanuelle Benazara, Ronen Brafman, Nicolas Meuleau, Eric Hansen. "Planning with Continuous Resources in Stochastic Domains". *International Joint Conference on Artificial Intelligence (IJCAI).* Edinburgh, Scotland. August 2005.

- Mausam, Daniel S. Weld. "Concurrent Probabilistic Temporal Planning". *International Conference on Automated Planning and Scheduling (ICAPS).* Monterey, CA, USA. June 2005.

- Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, Deepak Verma. "Adversarial Classification". *International Conference on Knowledge Discovery and Data Mining (SIGKDD).* Seattle, WA, USA. August 2004.

- Mausam, Daniel S. Weld. "Solving Concurrent Markov Decision Processes". *National Conference on Artificial Intelligence (AAAI).* San Jose, CA, USA. July 2004.

Refereed Workshop Papers

- Emmanuelle Benazara, Ronen Brafman, Nicolas Meuleau, Mausam, Eric Hansen. "An AO* Algorithm for Planning with Continuous Resources". *Workshop on Planning under Uncertainty for Autonomous Systems, at ICAPS.* Monterey, CA, USA. June 2005.

- Mausam, Daniel S. Weld. "Concurrent Probabilistic Temporal Planning: Initial Results". *Workshop on Learning and Planning in Markov Processes – Advances and Challenges, at AAAI.* San Jose, CA, USA. July 2004.

- Pedro Domingos, Yeuhi Abe, Corin Anderson, AnHai Doan, Dieter Fox, Alon Halevy, Geoff Hulten, Henry Kautz, Tessa Lau, Lin Liao, Jayant Madhavan, Mausam, Don Patterson, Matthew Richardson, Sumit Sanghai, Daniel Weld, Steve Wolfman. "Research on Statistical Relational Learning at the University of Washington". *Workshop on Learning Statistical Models from Relational Data, at IJCAI. Acapulco, Mexico.* August 2003.

- Mausam, Daniel S. Weld. "Solving Relational MDPs with First-Order Machine Learning". *Workshop on Planning under Uncertainty and Incomplete Information, at ICAPS.* Trento, Italy. June 2003.

Technical Reports and Articles

- Mausam. "Artificial Intelligence – Trends and Implications to the Society". Delta (IET Newsletter, Delhi Network). February 2007.

- Mausam. "A Comparison of Hierarchical Reinforcement Learning Techniques" Generals Exam. University of Washington. 2005.

- Mausam, Daniel S. Weld. "Solving Concurrent Markov Decision Processes" UW-CSE-TR-04-03-03. University of Washington. 2004.

- Mausam, Gaurav Singh Kushwaha. "Cache Efficiency and Sorting Algorithms" B.Tech project, Indian Institute of Technology. 2001.

Academic Achievements

- **Recipient of the silver medal for being first rank in the computer science department at IIT in the year 2000-2001.**

- **Recipient of Shashank Vikram Garg Award for the Best All Rounder in the batch at IIT for the year 1999-2000.**

- Recipient of Merit Award in all eight semesters, awarded by IIT Delhi for being the top 7% according to the semester GPA.(1997-2000)

- Recipient of National Talent Search Examination Scholarship, awarded by National Council for Educational research and Training, India.(1995)

- Recipient of National Science Talent Search Examination Scholarship (96-97), organised by Universal Trust. (All India 2nd Rank).

- Recipient of National Science Talent Search Examination Scholarship (95-96), organised by Universal Trust. (All India 2nd Rank).

- CBSE (Central Board of Secondary Education) Merit Scholarship Holder for being in the top 0.1% in senior High School examination. (1997).

- Secured 11th rank out of about 150,000 (All India) in IIT Entrance Examination, 7th rank (All India) in Roorkee Entrance Examination. (1997)

Professional Service and other Activities

- **Program Committee Member** National Conference on Artificial Intelligence (AAAI'05), International Conference on Automated Planning and Scheduling (ICAPS'07), Workshop on Planning and Learning in Apriori Unknown or Dynamic Domains at IJCAI'05.

- **Reviewer** Journal of Artificial Intelligence Research (JAIR), Journal of Machine Learning Research (JMLR), International Joint Conference on Artificial Intelligence (IJCAI'05, IJCAI'07), International Conference on Automated Planning and Scheduling (ICAPS'06), Ubiquitous Computing (UbiComp'07).

- **Member, Graduate Student Admissions Committee**, Dept. of Computer Science (U.W.), 2003.

- **Student Mentor**, Dept. of Computer Science (U.W.), 2005-06.