January 2009

Scheduling Policy Design using Stochastic Dynamic Programming

Robert Glaubius
Washington University in St. Louis

WASHINGTON UNIVERSITY IN ST. LOUIS

School of Engineering and Applied Science

Department of Computer Science and Engineering

Thesis Examination Committee:
William D. Smart, Chair
Christopher Gill
Yixin Chen
Sally Goldman
Csaba Szepesvári
Kurt Thoroughman

SCHEDULING POLICY DESIGN USING STOCHASTIC DYNAMIC PROGRAMMING

by

Robert Glaubius

A dissertation presented to the
Graduate School of Arts and Sciences
of Washington University in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2009
Saint Louis, Missouri

ABSTRACT OF THE DISSERTATION

Scheduling Policy Design using Stochastic Dynamic Programming

by

Robert Glaubius

Doctor of Philosophy in Computer Science

Washington University in St. Louis, 2009

Research Advisor: Professor William D. Smart

Scheduling policies for open soft real-time systems must be able to balance the competing concerns of meeting their objectives under exceptional conditions while achieving good performance in the average case. Balancing these concerns requires modeling strategies that represent the range of possible task behaviors, and solution techniques that are capable of effectively managing uncertainty in order to discover scheduling policies that are effective across the range of system modes. We develop methods for solving a particular class of task scheduling problems in an open soft real-time setting involving repeating, non-preemptable tasks that contend for a single shared resource. We enforce timeliness by optimizing performance with respect to the proportional progress of tasks in the system.

We model this scheduling problem as an infinite-state Markov decision process, and provide guarantees regarding the existence of optimal solutions to this problem. We derive several methods for approximating optimal scheduling policies and provide

theoretical justification and empirical evidence that these solutions are good approximations to the optimal solution. We consider cases in which task models are known, and adapt reinforcement learning methods to learn task models when they are not available.

Acknowledgments

I must first express my deep gratitude to my wife for her love and understanding. Without her support and motivation, this dissertation would likely never have been begun, let alone completed. From the years living hundreds of miles apart at the beginning of our graduate careers, through the late nights, stressful days, and the moments of bleak doubt and indecision, she has helped me to keep everything in perspective.

Thanks especially to my research advisor, Bill Smart, for supplying me with ideas and permitting me the freedom to pursue them along their often chaotic trajectories. When it became clear that my initial dissertation topic was prohibitively ambitious, it was his guidance, understanding, and acuity that led to the fruitful collaboration that resulted in the research detailed herein. Finally, Bill's efforts in managing to juggle assisting in reading and editing this dissertation, along with his other professional responsibilities, just short months after his second child was born merits special note.

Chris Gill has been invaluable in his capacities as a collaborator, advisor, instructor, and editor. His thoughts and ideas helped to inspire this work, and his suggestions have helped to moor this research in reality, otherwise those ties might have been tenuous indeed, as the theory at times seems to have its own volition.

Terry Tidwell has also played a substantial role in realizing this dissertation, both as a collaborator and as a friend. Many of the early developments of concepts and ideas that related in dissertation were refined in coordination with Terry in the late hours prior to conference deadlines.

Thanks to Justin Meden, David Pilla, and Braden Sidoti, students in the Research Experience for Undergraduates program, who devoted a substantial portion of their time and effort evaluating and testing some of the ideas in this dissertation.

Thanks to the past and contemporary members of the Media and Machines laboratory at Washington University in St. Louis, particularly Nathan Jacobs, Tom Erez,

Michael Dixon, Nisha Sudarsanam, and Frederick Heckel, who have helped articulate and refine ideas, evaluated and constructively criticized talks and papers, and generally provided an excellent environment for performing research.

My thanks to Berthe Choueiry, my advisor during my introduction to research as an undergraduate student at the University of Nebraska. Without her guidance and I may never have discovered the desire to pursue a career in research.

Finally, my family is due substantial thanks as well; my parents' encouragement to persist, to keep learning, and to constantly challenge myself have led me on a path I could not have imagined a decade ago. Their support has not been purely figurative: they have carried me (and my belongings) many times along the way.

Robert Glaubius

Washington University in Saint Louis
December 2009

This research has been supported by National Science grants CNS-0716764 (Cybertrust) and CCF-0448562 (CAREER)

Contents

List of Figures

Chapter 1

Introduction

Open soft real-time systems, such as mobile robots, must respond adaptively to varying operating conditions. For example, the discovery of a physical obstacle in the robot's environment may require an immediate action in order to avoid a collision and maintain the physical integrity of the robot. Such a system must be able to select an appropriate response from among multiple alternatives and act on that decision within specific timing constraints – in this case, the system has to decide that it needs to avoid the obstacle, and how best to do so before a collision occurs.

Such systems often provide fail-safe actions in the event of an anticipated or observed timing failure. For example, a robot may pause its motion if the robot finds itself within an unsafe distance of an obstacle, or if it has not been able to perform a detection action within some predefined interval. While such an action can prevent a catastrophic system failure, it may be suboptimal in the sense that some other non-exceptional course of action may address the immediate issue while allowing the system to continue making progress towards its objectives. Reactive, dynamic scheduling mechanisms are thus necessary to ensure timely decisions and actions while attaining the best possible performance.

The crux is that multiple behaviors, or *tasks*, (*e.g.*, obstacle avoidance or mission objectives) may contend for control of a shared resource (*e.g.*, the robot's sensors, actuators, computers, and communications links). For the system to be useful, we have to ensure progress of each of these tasks while guaranteeing the safety and reliability of the system.

While existing real-time systems theory describes how to schedule computation and communication resources for sensors and actuators, new methods are needed to address concerns that arise in domains like that described above. For example, behaviors may require use of a robot actuator for a long and unpredictable duration, since the time to move the actuator depends on the how close the actuator is to the required attitude and is influenced by a series of time-variable mechanical processes. Additionally, we can not preempt these behaviors, since restoring the actuator to a previously preempted state may be more time consuming than completing the behavior itself. These concerns render many real-time scheduling algorithms, which require explicit knowledge of task behavior and the ability to preempt at will, inapplicable. We use the following example robotics application to make these concerns more concrete.

Robot PTU Scheduling: Consider a mobile robot with two sensors, a camera and a contact sensor consisting of a set of bumpers. The bumpers allow the robot to detect when it makes physical contact with an obstacle. These are used to implement a fail safe action, triggering the robot to stop moving while considering its course of action. The other sensor is a camera, mounted on a pan-tilt unit (PTU) at about 1.75 meters above the ground – roughly at the height of the average adult human head when standing. The camera serves two functions. When the pan-tilt unit is parallel to the ground plane, the camera is used to carry out the robot's mission: capturing images of people's faces. Otherwise, the camera can be used to augment obstacle avoidance by detecting obstacles before the robot contacts them.

The pan-tilt unit is the shared resource in this domain. We need to balance safety against the robot's mission by interleaving face-finding with checking the ground ahead of the robot for obstacles. It should be clear that the right balance between these two depends on a number of factors, including the volatility of the environment and the speed of the robot. In a volatile environment there may be many other actors or moving objects that we need to avoid, so a larger obstacle avoidance budget is appropriate. Similarly, if the robot is moving quickly through even a non-volatile environment, it would behoove us to allocate more time to obstacle avoidance, while if the robot is standing still we may be able to eliminate most of the obstacle avoidance budget.

Ideally, this system would spend all of its time finding people in the environment. However, performing fail-safe actions may cause the robot to spend less time carrying out this mission objective than would be the case if it intermittantly used the camera to detect obstacles. We therefore are interested in effectively sharing the PTU between these tasks in order to maximize the time dedicated to finding faces in the environment, while minimizing the number of fail-safe actions taken. This is challenging, since uncertainty in the environment translates into unpredictable task behavior.

This research considers the general problem of scheduling repeating tasks on a single, mutually-exclusive resource, such as the PTU in our example. We distinguish between tasks and their *instances*, or *jobs*; for example, the obstacle avoidance task consists of jobs that check the environment for obstacles and plans a route to avoid them. Three principal considerations guide our choices when designing scheduling policies for these systems. First, tasks can not be preempted: once an instance of a task acquires the resource, it retains it until the instance completes. Second, we assume that task instances have stochastic, highly-variable duration. Finally, we assume that tasks can be run at any time – that is, whenever the scheduler is given control, each task has some job that is ready to run.

Preemption tends to make scheduling easier, as it allows individual jobs to be broken into smaller pieces, thus avoiding hard bin-packing problems [4]. However, preemption does not make sense in the domains we consider. For example, each job of an image capture task may require adjusting a pan-tilt unit. Preempting an image capture job may leave the PTU in an arbitrary state, which may result in a longer duration for the next task that acquires the resource. Further, restoring the PTU state to its attitude at preemption may be more expensive that simply running the next job in the same task.

We assume that the system is *open*: we do not have complete or accurate specifications of the task behavior in advance. Task behavior in such a system can only effectively be modeled statistically. That is, a suitable model of task behavior may consist of a probability distribution describing their arrival rates, as in queuing theory, or we may have a distribution describing task duration. By contrast, in a closed system

the task implementation is provided in advance, and its behavior can be exhaustively modeled, accurately simulated, and then abstracted into a simpler model.

Timeliness of task completion in the considered domains is soft, in the sense that if some task takes longer than expected to run, the worst-case is that the fail-safe mechanism takes over. This may be contrasted with the hard case, in which a missed deadline may manifest as a complete, catastrophic system failure. In the hard case, it is appropriate to model stochasticity in task execution by a deterministic reduction to worst-case performance. This is overly conservative for the soft real-time setting [78]; since late completion only degrades performance, we can trade off worst-case performance against expected case performance fruitfully.

Task durations are random variables. We make several simplifying assumptions regarding the distribution of task durations:

Bounded support: there is a finite upper bound, or *worst-case execution time*, on the maximum duration of each job.

Inter-task independence: Durations are independent between tasks.

Intra-task independence: The duration of jobs of the same task are identically and independently distributed.

These assumptions regarding job durations are fairly common in the discussion of soft, periodic real-time systems [59, 2].

Our assumptions regarding job availability differ from those of classical periodic real-time systems. In classical systems, the jobs of a task arrive at regular intervals or are governed by some minimum inter-arrival rate [21]. Timeliness is then enforced by establishing and meeting deadlines on the completion of each job.

We model job availability by assuming that whenever one job of a task relinquishes the shared resource, the next job of that task becomes available immediately. This does not lend itself naturally to a deadline formulation, however. Instead, our objective is to make progress on each task proportional to some portion of the shared resource, called its resource *utilization target*. More specifically, our goal is to see that in *any* sufficiently large observation interval, the difference between the time that each task spent occupying the resource and the time that it was intended to spend is small.

This notion of *proportional progress* [12] ensures timeliness by forcing each task to make progress relative to the rest at a roughly uniform rate. Further, proportional fairness promotes *temporal isolation* between tasks [2, 82], limiting the amount of interference between tasks.

New methods are necessary to address the concerns raised in the application domains we have described. Most scheduling algorithms for real-time systems are extensions to the earliest deadline first (EDF) or rate monotonic (RM) scheduling algorithms [58]. While EDF in particular is optimal for the deterministic, preemptive case, both are known to be suboptimal when preemption is not allowed [59]. Further, reduction to the deterministic case is overly conservative, and tends to lead to poor resource utilization. Finally, introducing the usual notion of deadlines and periods into our problem model is likely to result in worse resource utilization as the shared resource may idle unnecessarily while waiting for periods to complete.

Achieving optimal behavior in these domains requires methods for deriving dynamic scheduling policies that leverage feedback from the system in order to make good scheduling decisions. Our focus is on the design and analysis of scheduling policies obtained by solving stochastic dynamic programming problems. Thus, we do not analyze the behavior of classical real-time scheduling algorithms, but instead focus on constructing problem representations that are *sufficient* to represent optimal scheduling policies, and then use techniques from stochastic dynamic programming [72, 14] and reinforcement learning [44, 87] to compute or approximate those policies.

1.1 Task Scheduling Model

Over the course of this research we have proposed a simple task scheduling problem model [94, 38, 36]. In this model, we have n repeating tasks $(J_i)_{i=1}^{n}$that share access to a single mutually exclusive resource. Each task J_i consists of an infinite sequence of identical jobs $(J_{i,j})_{j=0}^{\infty}$. When a job is dispatched, it occupies the resource for some stochastic duration. Once a job $J_{i,j}$ releases the resource, the subsequent job $J_{i,(j+1)}$ immediately becomes available.

Suppose $J_{i,j}$ is a job of task J_i. The duration of job $J_{i,j}$ is a random variable $t_{i,j}$ distributed according to the *duration distribution* $P_{i,j}$, with $\mathbb{P}\{t_{i,j} = t\} \equiv P_{i,j}(t)$. $P_{i,j}$ is supported on the positive integers; specifically, for all $t > 0$, $P_{i,j}(t) \geq 0$, and $\sum_{t=1}^{\infty} P_{i,j}(t) = 1$. This explicitly requires that tasks advance the system time by at least one quantum when dispatched. We make several simplifying assumptions regarding these distributions:

(A1) The durations $t_{i,j}$ and $t_{k,l}$ of any pair of jobs $J_{i,j}$ and $J_{k,l}$ are independently distributed; that is, $\mathbb{P}\{t_{i,j} = t | t_{k,l} = t'\} = P_{i,j}(t)$.

(A2) The durations $t_{i,j}$ and $t_{i,k}$ of jobs $J_{i,j}$ and $J_{i,k}$ of task J_i are identically distributed; that is, $P_{i,j}(t) = P_{i,k}(t)$.

(A3) Every job $J_{i,j}$ has worst-case execution time $T_{i,j} < \infty$ such that $\sum_{t=1}^{T_{i,j}} P_{i,j}(t) = 1$.

These assumptions appear in the standard formulation of real-time systems with periodic tasks [59, 2]. Assumption A1 asserts that one job does not influence another. This means that we can expect a job to run according to the same distribution regardless of what jobs were scheduled before it. This is a strong assumption that is likely to be violated in practice. However, without this assumption there is little that we can do to schedule well. In the worst case behaving correctly in any given system state requires considering the complete execution history to make a decision, which rapidly becomes intractable. At best we can hope to study empirically how varying degrees of dependence effect the performance of the scheduling policies we obtain.

Assumption A2 is subject to similar concerns. If the task in question is an interrupt handler, for example, we would expect repeated successive instances to behave according to about the same distribution. However, in the robot example, consecutive instances of obstacle detection are neither identical nor independent, since we may only need to move the PTU for the first dispatch.

Practically assumption A2 means that $P_{i,j}$ is identical to $P_{i,k}$ for every other job $J_{i,k}$ of task J_i. This implies that can omit the job index and just refer to the task's duration distribution P_i. This regularity plays a crucial role in making the scheduling problem tractable.

Assumption A3 implies that there is a finite worst-case execution time for each job,

$$T_{i,j} = \max\{t | P_{i,j}(t) > 0\}$$

By assumption A2, since the duration of jobs from the same task obey the same distribution, those jobs have identical worst-case execution time $T_{i,j} = T_i$. The latter two assumptions allow us to talk about task behaviors rather than considering the individual behaviors of jobs. Throughout this dissertation we therefore discuss task timing behavior to characterize the behavior of each job of that task.

In addition to these strong assumptions, we make the weaker assumption that instance durations are positive integers. This allows us to treat the system time resolution in terms of a discrete time quantum. This assumption is fundamental to the discrete modeling techniques described in Chapter 3, but can be relaxed to the non-negative real numbers when performing direct search in policy space using the methods in Chapter 5.

We have explicitly ignored deadlines in this system. Our fundamental measure of timeliness is the proportional utilization of the resource. We specify our scheduling criterion in terms of the relative progress of each task, measured as the amount of time that each task holds the resource. This proportional progress criterion is determined by specifying a utilization target $\mathbf{u}$ as a system parameter. The utilization target is a positive, rational-valued n-vector. Component u_i is the relative target resource utilization for task J_i. $\mathbf{u}$ is subject to a total utilization constraint

$$\sum_{i=1}^{n} u_i = 1 \tag{1.1}$$

requiring that the resource time is budgeted completely among all tasks, and an interval constraint

$$\forall i = 1 \ldots n,\ 0 < u_i < 1. \tag{1.2}$$

The interval constraint means that each task must be allotted some share, but no task can be allocated the entire resource. Pragmatically, we can reduce the case $u_i = 0$ to a scheduling problem with $(n-1)$ tasks, and the case $u_i = 1$ can be reduced to the trivial problem of scheduling a single task.

Let $\mathbf{x}(t)$ be an integer-valued n-vector. This corresponds to the state of the scheduled system after t quanta have elapsed. Each component $x_i(t)$ equal to the number of time quanta during which task J_i held the resource in the time interval $[0, t)$ (system initialization is defined as time 0). We refer to $\mathbf{x}(t)$ as the system's *utilization state*, or just *state*. Then $x_i(t)/t$ is the percentage of the time spent by the resource on task J_i. A minimum criterion for a proportionally fair scheduling policy is that $x_i(t)/t$ should approach u_i as t grows large. However, we are not just concerned with steady state, asymptotic behavior, but also want to make sure that we quickly approach and maintain $\mathbf{x}(t)$ near the target utilization. More specifically, in any time interval $[t, t')$ we want to ensure that

$$|(t' - t)u_i - (x_i(t') - x_i(t))| \tag{1.3}$$

is small for each task.

In Chapter 2, we model this scheduling problem as a Markov Decision Process (MDP). We show that, given an appropriate choice of cost function based on achieving and maintaining the desired target utilization, there is an optimal scheduling policy. In Chapter 3, we present several methods for approximating the optimal policy using finite-state representations.

1.2 Literature Review

In order to ground this research in the context of real-time systems, it is useful first to illustrate some common modeling assumptions and considerations of that field. The periodic task model of Liu and Layland [58] is the basis for many system models studied in the literature on real-time systems. Periodic tasks consist of an infinite sequence of identical jobs that require access to a preemptable, mutually exclusive shared resource. This access is subject to timing constraints. The earliest moment that a job is ready to use resource is its *arrival* or *release time*; the task's *period* is the interval between consecutive arrivals of its jobs. A job is *available* after its release and prior to its completion; completion should occur prior to the job's *deadline*. Each job is assumed to have a known, finite worst-case execution time.

An optimal real-time scheduling algorithm produces schedules that meet all deadlines whenever possible. Liu and Layland [58] introduced the Rate Monotonic (RM) and Earliest-Deadline-First (EDF) scheduling algorithms for the periodic setting. EDF is optimal for many generalizations of the periodic setting, including the basic setting described above. RM is suboptimal, but is the optimum among algorithms that make scheduling decisions according to a static task ordering. EDF dynamically prioritizes available jobs according to increasing deadlines, so that the job with least deadline is run first. While EDF's optimality makes it more satisfying from a theoretical perspective, RM remains relevant because of its relative ease and efficiency of implementation on practical systems. Thus, these two algorithms provide the foundation for most real-time scheduling theory and implementation.

In our research, we relax several of the periodic task modeling assumptions. Jobs are released upon completion of the previous job of the same task, so that arrivals are aperiodic and depend upon previous scheduling decisions and task durations. This makes establishing job deadlines somewhat arbitrary, so instead we focus on enforcing timeliness and temporal isolation via proportional fairness. Finally, since we are concerned with tasks that manipulate physical actuators, preemption is not available. While many of these relaxations individually have been considered in previous work, this research appears to be the first that addresses all of them at once.

Stochastic Task Release: Job release times are often subject to variability in practice. Mok's sporadic task model [66] extended the periodic task model to accommodate variability in release times by assuming only a minimum bound on the time between consecutive releases of jobs of the same task. This model has since become widely studied and extensively used [59, 21, 78]. Jeffay and Goddard [42] introduced Rate-Based Execution, a further generalization of the sporadic task model for the soft real-time setting in which only job release rates are known in advance. They showed that EDF is optimal for both preemptive and non-preemptive resources in this model.

Stochastic Task Duration: Atlas and Bestavros [5, 6] proposed Statistical Rate Monotonic scheduling (SRMS), which extends RM scheduling to systems of periodic tasks with stochastic durations. In SRMS each task is represented by its period,

duration distribution, and desired quality-of-service. Quality-of-service is defined as the probability that any randomly selected job of the corresponding task completes before its deadline. The algorithm associates a periodically replenished budget with each task; a released job is scheduled only if its duration is less than its remaining budget. This budget mechanism prevents tasks from interfering with one another – *i.e.*, it enforces *temporal isolation* – since it places an upper bound on individual task's resource use over time. One limitation of this technique is that a job's duration must be known before it is scheduled in order to enforce budgets.

The Constant Bandwidth Server (CBS) of Abeni and Buttazzo [1, 2, 20] addresses this limitation. The CBS was introduced as a means of interleaving execution of soft, stochastic, aperiodic real-time tasks into a hard real-time system. Each soft task is associated with a server; the fraction of resource time not allocated to hard real-time tasks is divided among these servers according to user-specified computation budget and period parameters. The server associates a hard deadline with the soft task it serves. This deadline is a function of the server's period and remaining budget. If a job would overrun its server-assigned deadline, it is preempted and must wait until its server's budget is replenished. This allows the CBS to accommodate stochastic task durations while enforcing temporal isolation between soft and hard real-time tasks. Tuning CBS parameters has since been used as a means to optimize additional aspects of system performance [57].

Real-time queuing theory [55] is a more general model for repeating tasks that accommodates stochastic job arrivals, durations, and deadlines. This theory adapts methods from queuing theory to the study of timeliness behavior of real-time scheduling policies. These tools are primarily useful for analyzing statistics of fixed scheduling policies, particularly deadline miss rates. For example, Doytchinov *et al.* [27] and Kargahi and Movaghar [47] study the performance of EDF in this setting; see the latter for references to similar analyses of first-come-first-serve scheduling. This analysis tends to focus on highly structured arrival, duration, and deadline distributions under steady state conditions. Manolache *et al.* [63] use a related approach to estimate performance of arbitrary fixed scheduling policies under more restrictive conditions; one particularly interesting aspect of this work is that their computational methods are simplified by identifying and collapsing equivalent model states based on intervals in which scheduling priorities are static.

Proportional Fairness: SRMS and CBS enforce temporal isolation using a fairness mechanism. Proportional fairness has received substantial attention, particularly in multiprocessor scheduling, as a means of pruning poor scheduling alternatives to focus attention on a narrower range containing *feasible* schedules that meet all deadlines whenever they exist.

Baruah *et al.* introduced two optimal algorithms, PF [12] and PD [13], for proportionally fair, or *Pfair* multiprocessor scheduling in the periodic task model. Under Pfair scheduling, each task J_i is assigned a weight, defined as the ratio of its worst-case execution time T_i to its period p_i, T_i/p_i. These schedules enforce the *Pfair condition*, which bounds the *lag*

$$|t \cdot T_i/p_i - t \cdot x_i(t)| < 1$$

at every time quantum t, where $x_i(t)$ is the cumulative resource utilization of task J_i at time t. The utilization target $t \cdot T_i/p_i$ is the resource usage of task J_i under an idealized *fluid schedule* that divides time among tasks at infinitesimal resolution. The Pfair condition requires that task's actual usage is as close as possible to this ideal given that the resource is allocated in discrete quanta. Anderson and his collaborators have since developed a more efficient Pfair algorithm, PD^2 [4], and extended it to the sporadic task model and further application-specific generalizations thereof [3, 82, 83, 26]. To the best of our knowledge, enforcing fairness with stochastic task durations has not been considered.

Pfair scheduling makes extensive use of preemption [12]; jobs are fragmented to the extent that scheduling decisions are made explicitly at the single-quantum *subtask* level [4]. Without preemption, we can only hope to enforce a Pfair-like condition at a coarser resolution. In our research, we enforce timeliness by maintaining each task J_i's resource utilization near a user-specified utilization target u_i. We can characterize our methods as optimizing with respect to the Pfair condition in stochastic, non-preemptive systems by finding policies that minimize the lag

$$|t \cdot u_i - x_i(t)|$$

for each task J_i at every time t.

Cho, Ravindran, and Jensen [22] also derived an optimal scheduling algorithm that approximates the fluid schedule. This is achieved by decomposing time into intervals between task period boundaries. It is then possible to bound the change in a task's accumulated resource utilization above and below within this interval under all feasible schedules; these bounds necessarily encompass the fluid schedule. An optimal schedule can be obtained by making scheduling decisions that respect the geometry of this bounding region. This scheduling algorithm reduces, but does not eliminate, preemptions.

Non-preemptive Scheduling: Non-preemptive systems have received substantially less attention in the literature than have preemptive systems. This is because preemptive semantics are supported in readily-available off-the-shelf systems, and because optimal non-preemptive scheduling is known to be NP-hard [43]. Part of the challenge of non-preemptive scheduling is that an optimal schedule may need to leave the resource idle even if there are jobs ready; for example, this may be the case if running any ready job would force a job that is not yet available to miss its deadline. Thus, much of the work on non-preemptive scheduling incorporates the assumption that schedules must be *work conserving* – the resource can not idle if jobs are ready to run on it.

A non-preemptive version of EDF, EDF_{np}, is optimal in this setting. However, admission control under EDF_{np}, the problem of determining whether a new task can be scheduled, remains hard. Thus, most of the work on non-preemptive real-time scheduling focuses on deriving sufficient conditions for schedulability under EDF_{np} [10, 11, 40]. EDF is known to perform poorly in overload situations that are likely to arise under uncertainty, and it also does not maintain temporal isolation between tasks. Therefore, it does not satisfy our scheduling criteria, but does provide a useful basis for comparison.

1.3 Overview

This dissertation is organized as follows. Above, we described our task scheduling problem model. In Chapter 2, we provide some necessary background on Markov Decision Processes (MDPs), classical results that are used throughout this dissertation. We then model the task scheduling problem as an infinite-state MDP with unbounded costs, and derive some results that guarantee the existence of an optimal scheduling policy and conditions that allow us to represent the MDP more compactly.

In Chapter 3, we present several approximation methods that find good scheduling policies over restricted subsets of the infinite-state MDP derived in Chapter 2. This includes a truncated MDP, selected so that we will always find a policy that satisfies some bounds on deviation from target utilization whenever one exists. We also propose the Expanding State Policy Iteration algorithm, which exploits the structure of the task scheduling problem in order to construct minimal state representations that are necessary and sufficient to find good scheduling policies. We investigate the performance of these algorithms empirically on simulated problem instances.

Chapter 4 applies methods for reinforcement learning to extend the methods of previous chapters to domains where accurate task models are not provided in advance. One key aspect of reinforcement learning is the need to balance exploitation against exploration; that is, a reinforcement learning agent must determine when it is appropriate to act optimally with respect to its current information, or if instead it should choose an apparently suboptimal sequence of actions in order to learn more about the controlled system. We find that the structure of the task scheduling MDP eliminates most of the benefit of exploration, as exploiting current information reduces the number of suboptimal actions performed. We derive a PAC bound on the number of suboptimal actions taken before learning task models with low error.

One drawback of the methods described in Chapter 3 is that they rely on explicit enumeration of a large number of system states. This number of states grows exponentially in the number of tasks, so that in order for these methods to be useful in practice, either the number of tasks must be small or we must be able to aggregate them into abstract behaviors. In order to address this limitation to the scalability of our approach, in Chapter 5 we leave behind explicit representation of the scheduling

MDP and instead focus on direct search over a parameterized class of scheduling policies. While in general this class does not contain the optimal solution to the corresponding MDP formulation, these methods tend to produce policies that perform much better than available heuristic scheduling policies, particularly as the number of tasks grows large.

We conclude in Chapter 6 with a discussion of some of the open questions that arose over the course of this research, and propose several lines of future investigation that appear particularly significant.

Chapter 2

Scheduling as a Sequential Decision Problem

In Section 1.1 we defined the Task Scheduling Problem. This problem consists of determining a scheduling policy that selects which task will gain exclusive access to a shared resource at any moment in time. Our objective is to maintain the relative resource utilization of each task near some target share. In this section we model the task scheduling problem as a Markov Decision Process, or MDP. This framework allows us to quantify and compare the long-term utility of different scheduling policies, and ultimately to derive optimal ones.

In Section 2.1 we provide a brief development of the theory of MDPs and some algorithms for deriving control policies for them. In Section 2.2 we define the task scheduling problem as an MDP. In Section 2.3 we formally demonstrate the existence of an optimal solution to the task scheduling MDP, and in Section 2.4 we examine some of the special structure of the task scheduling MDP that we will exploit in order to obtain solution methods. Finally, in Section 2.5 we discuss related results and ideas.

2.1 Markov Decision Processes

Sequential decision problems arise when an agent or controller must make decisions repeatedly while maximizing measures of long-term utility. Two complications that

make these problems challenging are uncertainty and the need to reason about delayed rewards. We encounter the former in the task scheduling problem because our tasks have stochastic duration. The latter may occur whenever a small amount of "unfairness" in the short term makes it easier to balance utilization over the longer term. For example, it may be better to run an overutilized task with short duration rather than a slightly underutilized task with long duration.

A Markov Decision Process, or MDP, is a popular tool for modeling these kinds of problems. These problems arise in machine learning, operations research, and economics to name a few, and MDPs have been used to model and solve Reinforcement Learning problems in a wide range of applications, such as helicopter control [70, 68], mobile robotics [81, 52], elevator control [23], job shop scheduling [99], and playing Backgammon [93] and Go [79], among numerous others. The design and analysis of MDPs is a mature field; while we discuss standard results that are useful for understanding our algorithms and results, the interested reader is directed to the preponderance of literature on the subject; in particular, the books by Puterman [72] and by Bertsekas and Tsitsiklis [14], and the book chapter by Rust [77] provide detailed developments.

An MDP is a four-tuple $(\mathcal{X}, \mathcal{A}, P, R)$ consisting of a collection of states $\mathcal{X}$ and actions $\mathcal{A}$, a transition system P that establishes the conditional probabilities $P(y|x, a)$ of transitioning from state x to y on action a, and a reward function R that specifies the immediate utility of acting in each state. In this work we will take R as a deterministic function mapping state-action-state triples to a real-valued reward, so that $R(x, a, y)$ is the reward for taking action a in state x, then ending up in state y.

In order to model a sequential decision problem as an MDP, we associate a set of discrete decision epochs $k = 0, 1, \ldots, K$; in this work we consider the infinite-horizon setting where $K = \infty$. Simulation consists of observing the state of the MDP at epoch k, x_k and choosing an action a_k. The system transitions to a new state x_{k+1} according to $P(\cdot|x_k, a_k)$ and emits reward $r_{k+1} = R(x_k, a_k, x_{k+1})$. We refer to the sequence of states $(x_k)_{k=0}^{K}$ visited during simulation as a *trajectory*.

Our objective is to find a policy – a strategy for choosing actions at each state – that maximizes the expected sum of rewards obtained during simulation. In general a policy π associates a distribution over actions to each state, so that $\pi(a|x)$ is the

probability of selecting action a at state x. One classical result is that any MDP with a finite action space and stationary transitions (*i.e.*, the transition probabilities do not change between decision epochs) has an optimal deterministic policy [72]. With this in mind, we will restrict most of our treatment to deterministic policies. If a policy π is deterministic we overload notation to let $\pi(x)$ be the recommended action at state x.

We compare policies based on the expected sum of rewards obtained while simulating a policy. However, the sum of rewards obtained during a single trajectory, $\sum_{k=0}^{\infty} r_k$, may grow without bound, which makes direct comparison of policies difficult. There are a couple of well-studied criteria for alleviating this problem. We adopt the discounted reward criteria: we *discount* the contribution of the reward at decision epoch k by γ^k, with the *discount factor* γ satisfying $0 \leq \gamma < 1$. Then the value of a policy, written V^π, is the expected sum of discounted rewards

$$V^\pi(x) = \mathbb{E}\left\{ \sum_{k=0}^{\infty} \gamma^k R(x_k, \pi(x_k), x_{k+1}) \;\middle|\; x_0 = x \right\}. \tag{2.1}$$

Practically, the discount factor helps ensure that V^π is bounded. Conceptually, it can be interpreted as a prior probability of surviving from one decision epoch to the next [44]. In the task scheduling problem, the discount factor helps balance the immediate concern of quickly reaching the utilization target against maintaining the system state near that target over the lifetime of the system.

Finding the optimal policy π^* corresponds to finding the policy with maximum value at every state. That is, $V^{\pi^*}(x) \geq V^\pi(x)$ for each state x. While there may be multiple optimal policies for a given problem, the optimal value function is unique, and is identical for all optimal policies. We denote the optimal value function V^*.

It is straightforward to show from Equation 2.1 that the value function for π satisfies the following recurrence (Howard [41]):

$$V^\pi(x) = R(x, \pi(x)) + \gamma \sum_{y \in \mathcal{X}} P(y|x, \pi(x)) V^\pi(y), \tag{2.2}$$

where $R(x, a)$ is the expected reward for taking action a in x,

$$R(x, a) = \sum_{y \in \mathcal{X}} P(y|x, a) R(x, a, y). \tag{2.3}$$

Since the recurrence in Equation 2.2 is a linear system, we can compute V^π by solving a matrix equation when $\mathcal{X}$ and $\mathcal{A}$ are finite and the model parameters R and P are known.

The optimal value function satisfies the Bellman equation,

$$V^*(x) = \max_{a \in \mathcal{A}} \left\{ R(x, a) + \gamma \sum_{y \in \mathcal{X}} P(y|x, a) V^*(y) \right\}. \tag{2.4}$$

Given the optimal value function, we can derive an optimal policy by acting greedily with respect to V^*:

$$\pi^*(x) \in \operatorname*{argmax}_{a \in \mathcal{A}} \left\{ R(x, a) + \gamma \sum_{y \in \mathcal{X}} P(y|x, a) V^*(y) \right\}. \tag{2.5}$$

In order to compute the optimal policy for an MDP, it suffices to derive the optimal value function. Solution methods are typically based on the classical *value iteration* and *policy iteration* algorithms.

Value iteration computes an approximation to the optimal value function. Beginning with some initial approximation V_0 (for example, $V_0 = 0$), the algorithm proceeds by computing the sequence of approximations V_{k+1} by applying the Bellman equation 2.4 to the previous iterate V_k,

$$V_{k+1}(x) = \max_{a \in \mathcal{A}} \left\{ R(x, a) + \gamma \sum_{y \in \mathcal{X}} P(y|x, a) V_k(y) \right\}.$$

The sequence $(V_k)_{k=0}^{\infty}$ is guaranteed to asymptotically approach V^* [72]. If we choose $V_0 = 0$, V_k corresponds to the optimal value given that the system will terminate after k decision epochs [14]. This algorithm tends to be slower than the next algorithm, policy iteration, but is conceptually useful for understanding the semantics of the

optimal value function and is theoretically useful as a basis for inductively proving claims about it.

In contrast to value iteration, policy iteration computes the optimal value function exactly in a finite number of steps, given that there are only finitely many states and actions. Each iteration consists of two steps, *policy evaluation* and *policy improvement*. Beginning with some initial policy π_0, at iteration k *policy evaluation* consists of computing the value function V^{π_k} by solving the linear system in Equation 2.2. In the *policy improvement* step, we choose a new policy π_{k+1} by choosing the greedy action according to V^{π_k},

$$\pi_{k+1}(x) = \max_{a \in \mathcal{A}} \left\{ R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^{\pi_k}(x) \right\}.$$

This is sufficient to guarantee that every $V^{\pi_{k+1}}$ pointwise dominates[1] V^{π_k}; since there are only finitely many deterministic policies, the algorithm terminates. The final value function is guaranteed to satisfy the Bellman equation 2.4 [72], and so is optimal.

Value and policy iteration are the basis of many algorithms for solving MDPs, but most analysis considers the case where there are only finitely many states and actions. Finite state and action spaces admit exact, tabular representations of the intermediate value functions produced by these algorithms. In the following sections we formulate the task scheduling problem as a Markov Decision Process with finitely many actions but an infinite, discrete state space, then investigate some aspects of the problem structure that we may exploit in order to compute exact optimal scheduling policies.

2.2 Task Scheduling MDP

In previous work [94] we proposed an MDP formulation of the task scheduling problem based on the system model described in Section 1.1. The set of problem states corresponds to the accumulated resource utilization of each task. Actions in the model correspond to the decision to dispatch each task, and modify the state by increasing the accumulated utilization of the running task according to its observed

[1] A function $f : \mathcal{X} \to \mathbb{R}$ pointwise dominates $g : \mathcal{X} \to \mathbb{R}$ if and only if for every x in $\mathcal{X}$, $f(x) \geq g(x)$.

duration. In the task scheduling setting it is a bit more natural to talk about costs rather than rewards; we penalize the system for reaching states that are "unfair" with respect to the utilization target. For example, we may wish to make sure that two threads receive an equal share by establishing a utilization target of $\frac{1}{2}$ for each thread; there is some penalty attached to one task receiving twice as much resource as the other, but the penalty for this is less than when one task receives four times as much resource as the other.

The set of decision epochs in this system corresponds to the points in time when a task relinquishes the shared resource, and the scheduler must decide which task to dispatch next. We have two distinct notions of time: (1) system time, which is measured in quanta and denoted by t when discussing task durations, or $\tau(\mathbf{x})$ as the elapsed time when the system is in state $\mathbf{x}$, and (2) model time, which is measured in decision epochs and denoted using k.

We introduce a bit of notation before describing these components more formally. We denote vectors $\mathbf{u}$ and $\mathbf{v}$ using bold face fonts. $\mathbf{0}$ is the vector with all zero components; the dimension of this vector will be apparent from context. For n-vectors $\mathbf{u} = (u_1, \ldots, u_n)$ and $\mathbf{v} = (v_1, \ldots, v_n)$, we say $\mathbf{u}$ is less than or equal to $\mathbf{v}$, written $\mathbf{u} \preceq \mathbf{v}$, if and only if $u_i \leq v_i$ for all $i = 1, \ldots, n$. We define the operators $\prec$, $\succeq$, and $\succ$ analogously. These comparison operators can also be applied to real-valued functions as well, since these can be interpreted as members of a vector space with an infinite number of dimensions.

The set of actions in the task scheduling MDP $\mathcal{A}$ is the set of integers between 1 and n, inclusive,

$$\mathcal{A} = \{1, 2, \ldots, n\}. \tag{2.6}$$

Choosing to execute action i in $\mathcal{A}$ corresponds to the decision to dispatch task J_i.

We define the MDP state space $\mathcal{X}$ as the cumulative resource utilization of each task. This is just an n-vector $\mathbf{x}$ with non-negative integer components,

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{x} \succeq \mathbf{0}\}. \tag{2.7}$$

If $\mathbf{x}$ is in $\mathcal{X}$, we call it a *utilization state* in order to distinguish it from model refinements we will discuss in greater detail later in this dissertation. We refer to $\mathcal{X}$ as the collection of *utilization states*, or as the *utilization state space*.

Suppose that $\mathbf{x}$ is a utilization state. Then $\mathbf{x} = (x_1, \ldots, x_n)$, and each component x_i corresponds to the cumulative resource utilization of task J_i. This is the number of time quanta in which J_i occupied the shared resource. The total utilization $\tau(\mathbf{x})$ at $\mathbf{x}$ is the sum of task utilizations,

$$\tau(\mathbf{x}) = \sum_{i=1}^{n} x_i; \tag{2.8}$$

this is the total number of time quanta that have passed when the system is in $\mathbf{x}$. At time $\tau(\mathbf{x})$ the target utilization point[2] is $\tau(\mathbf{x})\mathbf{u}$ – the utilization vector scaled by the number of elapsed time quanta; we say that task J_i is *overutilized* when $x_i > \tau(\mathbf{x})u_i$; when $x_i < \tau(\mathbf{x})u_i$, the task is *underutilized*.

When we run task J_i in utilization state $\mathbf{x}$, the successor state $\mathbf{y}$ is stochastically determined according to P_i, J_i's distribution over durations. If J_i occupies the resource for t quanta, then $\mathbf{x}$ and $\mathbf{y}$ differ by t in the i^{th} component: $\mathbf{y} = \mathbf{x} + t\Delta_i = (x_1, \ldots, x_i + t, \ldots, x_n)$. Δ_i is the vector with component i equal to one and all other components equal zero. We can write $\Delta_i = (\delta_{i,1}, \ldots, \delta_{i,n})$ using the Kronecker delta,

$$\delta_{i,j} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}. \tag{2.9}$$

Using this notation, we define the probability of transitioning into utilization state $\mathbf{y}$ from $\mathbf{x}$ on task J_i

$$P(\mathbf{y}|\mathbf{x}, i) = \begin{cases} P_i(t) & \mathbf{y} - \mathbf{x} = t\Delta_i \\ 0 & otherwise. \end{cases} \tag{2.10}$$

Figure 2.1 illustrates this transition system using an example problem with two tasks. Task J_1 stochastically runs for one or two quanta; its transitions are shown in grey with open arrowheads, advancing to the right. Task J_2 deterministically runs for

[2]We usually can not call $\tau(\mathbf{x})\mathbf{u}$ a utilization *state*, since it will may not be integer-valued for some $\mathbf{x}$. As we will see in Section 2.4, the case where $\tau(\mathbf{x})\mathbf{u}$ is a utilization state has some important and useful properties.

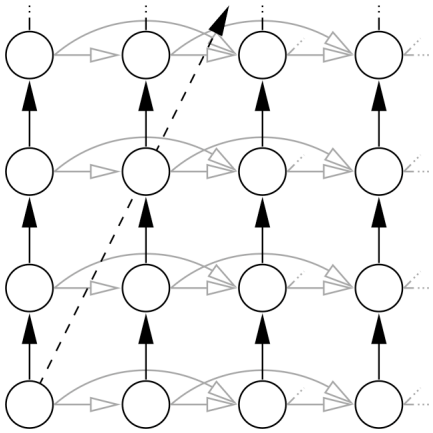


Figure 2.1: A utilization-state MDP model of a two-task scheduling problem.

a single quantum, with transitions advancing upwards, shown in black with closed arrowheads. The dashed ray points points in the direction of the target utilization vector $\mathbf{u} = (1/3, 2/3)$. Since we require that tasks always occupy the shared resource for at least one quantum, the transition system induces a directed acyclic graph over $\mathcal{X}$. Since the transition probabilities are derived from the task duration distribution independently from the state, this transition system is self-similar: we can obtain the transition system at any state by translating the system from any other state. This observation of self-similarity is a requirement of state reduction techniques based on identifying *homomorphisms* between states [24, 34, 74, 34]. These methods simplify the state space by collapsing together states have similar futures. We will use this idea later to simplify the task scheduling MDP.

As we mentioned above, the target utilization point $\tau\mathbf{u}$ after τ time quanta have passed is not necessarily a utilization state. For example, this happens when $\tau = 1$, since $\tau u_i = u_i$ is strictly between zero and one. With this in mind, we require that the cost of states is defined at any point in $\mathbb{R}^n$. We achieve this by defining a cost function c mapping points in $\mathbb{R}^n$ to real-values. Since our objective is to encourage the scheduler to keep the system near the specified utilization target $\tau\mathbf{u}$, we measure cost in terms of the distance between a state $\mathbf{x}$ and the target utilization after $\tau(\mathbf{x})$ quanta have elapsed, $\tau(\mathbf{x})\mathbf{u}$. The cost function must be minimal at the target point, and non-decreasing as $\|\mathbf{x} - \tau(\mathbf{x})\mathbf{u}\|$ increases for fixed $\tau(\mathbf{x})$. This is satisfied, for

example, by any L_p-norm based cost function c_p, $p \geq 1$,

$$c_p(\mathbf{x}) = \|\mathbf{x} - \tau(\mathbf{x})\mathbf{u}\|_p = \Big(\sum_{i=1}^{n} |x_i - \tau(\mathbf{x})u_i|^p\Big)^{1/p}, \tag{2.11}$$

since $c_p(\tau(\mathbf{x})\mathbf{u}) = 0$. Optimal policies are not invariant with respect to the choice of p, so it is important to consider the implications of any particular choice. As p increases, a large deviation in a single component, $|x_i - \tau(\mathbf{x})u_i|$, is weighted more heavily than small deviations across a number of tasks. Conversely, if $p = 1$, all deviations are weighted equally, so that the cost is exactly the sum of deviations from target utilization for each task. In order to retain flexibility in our framework, we will attempt to derive results that do not require a specific choice of p whenever possible.

The c_p cost function is a pseudonorm when extended to arbitrary $\mathbf{x}$ in $\mathbb{R}^n$:

Theorem 2.2.1. *The cost function c_p is a pseudo-norm over $\mathbb{R}^n$.*

Proof. c_p is scalable, since for any $\mathbf{x} \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$,

$$\begin{aligned} c_p(\alpha\mathbf{x}) &= \|\alpha\mathbf{x} - \tau(\alpha\mathbf{x})\mathbf{u}\|_p \\ &= \|\alpha\mathbf{x} - \alpha\tau(\mathbf{x})\mathbf{u}\|_p \\ &= |\alpha| \, \|\mathbf{x} - \tau(\mathbf{x})\mathbf{u}\|_p . \\ &= |\alpha| \, c_p(\mathbf{x}) \end{aligned}$$

c_p satisfies the triangle inequality, since

$$\begin{aligned} c_p(\mathbf{x} + \mathbf{y}) &= \|\mathbf{x} + \mathbf{y} - \tau(\mathbf{x} + \mathbf{y})\mathbf{u}\|_p \\ &= \|\mathbf{x} + \mathbf{y} - \tau(\mathbf{x})\mathbf{u} - \tau(\mathbf{y})\mathbf{u}\|_p \\ &\leq \|\mathbf{x} - \tau(\mathbf{x})\mathbf{u}\|_p + \|\mathbf{y} - \tau(\mathbf{y})\mathbf{u}\|_p \\ &= c_p(\mathbf{x}) + c_p(\mathbf{y}). \end{aligned}$$

□

c_p falls short of being a norm, since (for example) there are infinitely many states besides $\mathbf{x} = \mathbf{0}$ where c_p is zero. This result is examined in more detail in Theorem 2.4.3 in Section 2.4.

Theorem 2.2.1 implies that we can bound the difference in cost between a state and its successor by a constant. We make this concrete in Lemma 2.2.1.

Lemma 2.2.1. *For all utilization states* $\mathbf{x} \in \mathcal{X}$*, actions* $i \in \mathcal{A}$*, and positive integers* p *and* t*,*

$$c_p(\mathbf{x}) - tc_p(\Delta_i) \leq c_p(\mathbf{x} + t\Delta_i) \leq c_p(\mathbf{x}) + tc_p(\Delta_i)$$

Proof. The upper bound follows from Theorem 2.2.1 by straightforward applications of scalability and the triangle inequality:

$$c_p(\mathbf{x} + t\Delta_i) \leq c_p(\mathbf{x}) + c_p(t\Delta_i) = c_p(\mathbf{x}) + tc_p(\Delta_i).$$

We can obtain the lower bound by augmenting the cost at $\mathbf{x}$:

$$\begin{aligned} c_p(\mathbf{x}) &= \|\mathbf{x} - \tau(\mathbf{x})\mathbf{u}\|_p \\ &\leq \|\mathbf{x} - \tau(\mathbf{x})\mathbf{u} + t(\Delta_i - \mathbf{u})\|_p + \|t(\Delta_i - \mathbf{u})\|_p \\ &= c_p(\mathbf{x} + t\Delta_i) + tc_p(\Delta_i); \end{aligned}$$

we can rearrange terms in this inequality to establish the lower bound. □

This result establishes a "speed limit" of sorts on how much the cost may change while a task occupies the resource [54], provided that tasks' worst-case execution times are finite.

We define the reward function $R(\mathbf{x}, i, \mathbf{y})$ as the negative cost of the successor state $\mathbf{y}$; we switch the sign of the rewards so that maximization of rewards corresponds to minimization of costs

$$R(\mathbf{x}, i, \mathbf{y}) = -c(\mathbf{y}). \tag{2.12}$$

At this point, it will facilitate our discussion to introduce some backup operators to describe compactly the value function recurrence relations in Equations 2.2 and 2.4. Let V be a bounded, real-valued function of $\mathcal{X}$. Then we define an action-specific backup operator Γ_i that maps functions to functions,

$$(\Gamma_i V)(\mathbf{x}) = R(\mathbf{x}, i) + \gamma \sum_{\mathbf{y} \in \mathcal{X}} P(\mathbf{y}|\mathbf{x}, i)V(\mathbf{y}). \tag{2.13}$$

Using this notation, we can define the policy backup Γ_π and the Bellman operator Γ,

$$(\Gamma_\pi V)(\mathbf{x}) = (\Gamma_{\pi(\mathbf{x})} V)(\mathbf{x}) \tag{2.14}$$

$$(\Gamma V)(\mathbf{x}) = \max_{i \in \mathcal{A}} \{(\Gamma_i V)(\mathbf{x})\} \tag{2.15}$$

These operators allow us to compactly write the policy recurrence equation from Equation 2.2 as

$$V^\pi = \Gamma_\pi V^\pi;$$

similarly, we can rewrite the Bellman equation 2.4 as

$$V^* = \Gamma V^*,$$

and the value iteration algorithm can be written compactly as $V_{k+1} = \Gamma V_k$ or $V_k = \Gamma^k V_0$. Plugging in the scheduling-specific definitions of P and R, Equation 2.13 can be rewritten as

$$(\Gamma_i V)(\mathbf{x}) = \sum_{t=1}^{\infty} P_i(t)[\gamma V(\mathbf{x} + t\Delta_i) - c(\mathbf{x} + t\Delta_i)], \tag{2.16}$$

emphasizing that the value of task J_i in $\mathbf{x}$ depends only on the cost and value of states that differ only in component i. One consequence of this fact is that, while there may be infinitely many states in the problem, any one state can transition to at most nT successor states if T is the worst-case execution time among all tasks.

Finally, given the optimal value function V^*, we can easily recover an optimal policy π^* by maximizing with respect to the per-action backup operator. That is,

$$\pi^*(\mathbf{x}) \in \operatorname*{argmax}_{i \in \mathcal{A}} \{(\Gamma_i V^*)(\mathbf{x})\}. \tag{2.17}$$

We use the membership relation '$\in$' rather than equality '$=$' here to indicate that there may be more than one optimal action in a given state in general.

The utilization-state MDP described above captures the semantics of the task scheduling problem from Section 1.1, so that if an optimal scheduling policy for the MDP formulation exists, it is also optimal for the task scheduling policy given the cost function. However, this model has infinitely many states, that we can not compute

its optimal value function V^* exactly at all states. Our choice of cost function introduces an additional complication: since state costs grow without bound, it is not clear that the optimal value can be bounded even in the discounted case.

In the next section we will demonstrate that the optimal value function exists and is a bounded function over $\mathcal{X}$, despite L_p costs that decrease without bound. This result is sufficient to prove the existence of an optimal solution to the MDP model. In subsequent sections we will explore a number of approaches that enable us to compute or approximate the optimal value function over a subset of utilization states.

2.3 Existence of the Optimal Value Function

In the finite state and action case, showing that the Bellman optimality condition $V^* = \Gamma V^*$ has a unique solution is straightforward, following from the fact that Γ is a contraction map with factor γ over the Banach space of real-valued functions of state. The infinite-state case complicates matters, particularly since the cost function decreases without bound. We will establish two points: first, that the optimal value function for our infinite-state MDP is a bounded function in some Banach space, and second, that the limit $\Gamma^k V$ as k grows large is uniquely identified in that space.

We will follow an approach described by Puterman [72] for handling this situation. We define a weighted supremum norm $\|\cdot\|_\alpha$ in terms of a weight function α, so that for any function $V : \mathcal{X} \to \mathbb{R}$,

$$\|V\|_\alpha = \sup\{|V(\mathbf{x})| / \alpha(\mathbf{x}) \mid \mathbf{x} \in \mathcal{X}\}. \tag{2.18}$$

We can then define the space $\mathcal{V}_\alpha$,

$$\mathcal{V}_\alpha = \{V : \mathcal{X} \to \mathbb{R} \mid \|V\|_\alpha < \infty\}, \tag{2.19}$$

as the set of functions bounded in the α-weighted supremum norm. We can prove the existence of the optimal value function by choosing α appropriately. Following theorem 6.10.4 from Puterman [72], an appropriate choice of α is a positive, real-valued function that constrains the growth of the value function approximations $V_{t+1} = \Gamma V_t$.

This consists of requiring that the magnitude of expected rewards are bounded above by α, and that the expectation $\sum_t P_i(t)\alpha(\mathbf{x}+t\Delta_i)$ can be bounded in terms of $\alpha(\mathbf{x})$. This constrains the growth of $V(\mathbf{x})$ by bounding the contribution of future states' values to the value of $\mathbf{x}$.

We define α in terms of the cost function plus a constant term determined by the maximum difference in cost that can be accrued when invoking a task. Let μ_i be the expected duration of task J_i,

$$\mu_i = \sum_{t=1}^{\infty} tP_i(t), \tag{2.20}$$

Then

$$\alpha(\mathbf{x}) = c(\mathbf{x}) + \max_{i\in\mathcal{A}} \{\mu_i c(\Delta_i)\}. \tag{2.21}$$

We can then show that $V^* = \Gamma V^*$ has a unique fixed point in $\mathcal{V}_\alpha$. Before we prove this, we will establish a pair of lemmas. First, we will show that we can bound the expected reward and expected future state values in terms of α.

Lemma 2.3.1. *Let* $\mathbf{x}$ *be a state and let* i *be an action. Then*

1. $|R(\mathbf{x}, i)| \leq \alpha(\mathbf{x})$.
2. $\sum_{t=1}^{\infty} P_i(t)\alpha(\mathbf{x}+t\Delta_i) \leq \alpha(\mathbf{x}) + \mu_i c(\Delta_i)$.

Proof. We can show (1) by applying Lemma 2.2.1 to bound $c(\mathbf{x}+t\Delta_i)$ in terms of the cost at $\mathbf{x}$ a cost for dispatching task J_i, $tc(\Delta_i)$:

$$\begin{aligned} |R(\mathbf{x}, i)| &= \sum_{t=1}^{\infty} P_i(t)c(\mathbf{x}+t\Delta_i) \\ &\leq \sum_{t=1}^{\infty} P_i(t)[c(\mathbf{x}) + tc(\Delta_i)] \\ &= c(\mathbf{x}) + \mu_i c(\Delta_i). \end{aligned}$$

We can use this same observation to prove the second claim. Let $M = \max_{i \in \mathcal{A}} \{\mu_i c(\Delta_i)\}$.

$$\begin{aligned}\sum_{t=1}^{\infty} P_i(t)\alpha(\mathbf{x} + t\Delta_i) &= M + \sum_{t=1}^{\infty} P_i(t)c(\mathbf{x} + t\Delta_i) \\ &\leq M + c(\mathbf{x}) + \mu_i c(\Delta_i) \\ &= \alpha(\mathbf{x}) + \mu_i c(\Delta_i).\end{aligned}$$

□

We can use this result to show that the k-step Bellman operator is a contraction map on $\mathcal{V}_\alpha$. This result will allow us to show that if $\mathcal{V}_\alpha$ is a Banach space, then the sequence of value iterates $V_{t+1} = \Gamma V_t$ must converge to a unique fixed point in $\mathcal{V}_\alpha$.

Lemma 2.3.2. *There is an integer k such that Γ^k is a contraction map on $\mathcal{V}_\alpha$.*

Proof. We will begin by showing inductively that the k-stage Bellman operator satisfies a pointwise Lipschitz condition: that for any state $\mathbf{x}$, non-negative integer k, and pair of functions U and V in $\mathcal{V}_\alpha$,

$$\left|(\Gamma^k U)(\mathbf{x}) - (\Gamma^k V)(\mathbf{x})\right| \leq \gamma^k(\alpha(\mathbf{x}) + kM) \left\|U - V\right\|_\alpha,$$

where $M = \max_i\{\mu_i c(\Delta_i)\}$. This is trivial when $k = 0$, since $\Gamma^0 W = W$ for any function W. Suppose that the claim is true for some k. Then

$$\begin{aligned}\left|(\Gamma^{k+1} U)(\mathbf{x}) - (\Gamma^{k+1} V)(\mathbf{x})\right| &= \left|\max_{i \in \mathcal{A}}\{(\Gamma_i \Gamma^k U)(\mathbf{x})\} - \max_{i \in \mathcal{A}}\{(\Gamma_i \Gamma^k V)(\mathbf{x})\}\right| \\ &\leq \max_{i \in \mathcal{A}} \left|(\Gamma_i \Gamma^k U)(\mathbf{x}) - (\Gamma_i \Gamma^k V)(\mathbf{x})\right| \\ &\leq \gamma \max_{i \in \mathcal{A}} \sum_{\mathbf{y} \in \mathcal{X}} P(\mathbf{y}|\mathbf{x}, i) \left|(\Gamma^k U)(\mathbf{y}) - (\Gamma^k V)(\mathbf{y})\right| \\ &\leq \gamma \max_{i \in \mathcal{A}} \sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, i)\gamma^k(\alpha(\mathbf{y}) + kM) \left\|U - V\right\|_\alpha \\ &= \gamma^{k+1}(\alpha(\mathbf{x}) + (k+1)M) \left\|U - V\right\|_\alpha;\end{aligned}$$

the final step above holds by simplifying according to Lemma 2.3.1:

$$\begin{aligned}\max_{i\in\mathcal{A}} \sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x},i)(\alpha(\mathbf{y})+kM) &= kM + \max_{i\in\mathcal{A}} \sum_{t=1}^{\infty} P_i(t)\alpha(\mathbf{x}+t\Delta_i) \\ &\leq kM + \alpha(\mathbf{x}) + \max_{i\in\mathcal{A}}\{\mu_i c(\Delta_i)\} \\ &= (k+1)M + \alpha(\mathbf{x}),\end{aligned}$$

establishing the pointwise Lipschitz property. To complete the proof,

$$\begin{aligned}\left\|\Gamma^k U - \Gamma^k V\right\|_\alpha &= \sup_{\mathbf{x}\in\mathcal{X}} \left\{\left|(\Gamma^k U)(\mathbf{x}) - (\Gamma^k V)(\mathbf{x})\right| / \alpha(\mathbf{x})\right\} \\ &\leq \sup_{\mathbf{x}\in\mathcal{X}} \left\{\gamma^k (1 + kM/\alpha(\mathbf{x})) \left\|U - V\right\|_\alpha\right\} \\ &\leq \gamma^k \left\|U - V\right\|_\alpha \sup_{\mathbf{x}\in\mathcal{X}} \left\{1 + kM/\alpha(\mathbf{x})\right\}.\end{aligned}$$

The supremum term above is a bounded constant, since $\alpha(\mathbf{x})$ is strictly positive. Thus, when $\gamma < 1$, $\gamma^k \sup_{\mathbf{x}}\{1 + kM/\alpha(\mathbf{x})\} < 1$ for some k, so that Γ^k is a contraction on $\mathcal{V}_\alpha$. □

We will use Lemma 2.3.2 to demonstrate that the value function exists.

Theorem 2.3.1. *The Bellman equation $V^* = \Gamma V^*$ has a unique solution in $\mathcal{V}_\alpha$.*

Proof. In order to prove the claim, we need to show that $\mathcal{V}_\alpha$ is complete (*i.e.*, that it is a Banach space) and is closed under Γ. Then we can use Lemma 2.3.2 to conclude that the sequence $V_{k+1} = \Gamma V_k$ has a unique fixed point in $\mathcal{V}_\alpha$.

Showing that $\mathcal{V}_\alpha$ is a Banach space is straightforward, but we will provide an argument here for completeness. Consider a Cauchy sequence $(f_k)_{k=0}^\infty$ with each $f_k \in \mathcal{V}_\alpha$. Then by definition, for every $\varepsilon > 0$ there is an integer $K > 0$ so that if k and k' exceed K, $\left\|f_k - f_{k'}\right\|_\alpha < \varepsilon$. Pick an ε and f_k with k greater than the corresponding index K. Since $f_k \in \mathcal{V}_\alpha$, $f_k(\mathbf{x})$ is bounded for any $\mathbf{x}$. Then for any $k' > k$, $|f_k(\mathbf{x}) - f_{k'}(\mathbf{x})| \leq \left\|f_k - f_{k'}\right\|_\alpha \alpha(\mathbf{x}) < \varepsilon\alpha(\mathbf{x})$. Therefore $f_{k'}(\mathbf{x}) \in f_k(\mathbf{x}) \pm \varepsilon\alpha(\mathbf{x})$ for any $k' \geq k$, so the limit must be finite. Since $\mathbf{x}$ is arbitrary, the limit of $(f_k)_{k=0}^\infty$ is in $\mathcal{V}_\alpha$; since the sequence is arbitrary, $\mathcal{V}_\alpha$ contains the limit of any Cauchy sequence, and so is complete.

In the proof of Lemma 2.3.2 we demonstrated inductively that Γ is Lipschitz with factor $\gamma \sup_{\mathbf{x}}(1 + M/\alpha(\mathbf{x}))$. Therefore,

$$\|\Gamma V\|_\alpha \leq \gamma \sup_{\mathbf{x} \in \mathcal{X}} (1 + M/\alpha(\mathbf{x})) \|V\|_\alpha < \infty,$$

and so ΓV is in $\mathcal{V}_\alpha$. Therefore, since Γ is a k-stage contraction on $\mathcal{V}_\alpha$, by the Banach fixed point theorem we can guarantee that the sequence of functions $V_{k+1} = \Gamma V_k$ converges uniquely to the optimal value function V^* □

This demonstrates that the value function exists and is bounded, implying that some policy exists that will not accumulate arbitrarily large costs. However, the utilization state MDP model is impractical, since it requires representing the policy over an infinitely large state space. Fortunately, the MDP formulation of this problem is highly structured. The transition probabilities depend on the action but not on the current state; *i.e.*, it is just as likely that a task runs for t steps in state $\mathbf{x}$ as state $\mathbf{y}$. A second regularity in the problem has to do with our choice of cost functions: since the reward depends on the distance to the target utilization ray, there are many states with the same cost. In the next section we will exploit these properties to show that it is not necessary to represent every utilization state in order to represent the optimal value function.

2.4 State Space Periodicity

In this section, we will explore some of the properties of the MDP formulation of the task scheduling problem that make it amenable to efficient solution methods despite having an infinite number of states. In particular, we will exploit the periodic nature of the problem state space[3]. For example, notice that the cost is zero everywhere along the target utilization ray $\lambda\mathbf{u}$. In fact, since we defined the cost function c_p using the L_p-norm of the distance between a state $\mathbf{x}$ and the point $\tau(\mathbf{x})\mathbf{u}$, every point on the line $\{\mathbf{x} + \lambda\mathbf{u} \mid \lambda \in \mathbb{R}\}$ has the same cost for any $\mathbf{x}$ we choose. Coupled with

[3]It is important to note that this use of "period" differs from the standard notion in real-time systems. A periodic task in a real time system repeatedly releases identical jobs to the scheduler at regular intervals [21]. Periodicity in this work instead refers to a regularity in the state space that allows us to treat certain states occurring at regular intervals as equivalent.

the fact that transition probabilities depend only on the dispatched task but not on the current state, any states that are collinear in the direction $\mathbf{u}$ behave the same: they have the same cost, and their distributions over successor costs are the same.

The key observation is that we can describe the state space as a collection of equivalence classes. Below we show that distinguishing between members of these classes individually provides no novel information about the long-term cost. Following McCallum's *utile distinction* principle [64], we need not represent more than one state in each equivalence class. Our notion of equivalence here is based on state space periodicity, due to the fact that if the utilization ray passes through states $\mathbf{0}$ and $\mathbf{x}$, then it must necessarily pass through other utilization states at regular intervals.

Periodicity in the state space occurs whenever we can obtain a distinct state $\mathbf{y}$ by adding the scaled utilization vector to another state $\mathbf{x}$. We will show that in this case, $\mathbf{x}$ and $\mathbf{y}$ have the same optimal value, and that there is some optimal policy that selects the same action at both states. More formally, we say that the task scheduling MDP is *periodic with period* κ if and only if whenever $\mathbf{x}$ is a utilization state, $\mathbf{x} + \kappa\mathbf{u}$ is also a utilization state. We say that a period κ is minimal if and only if the MDP is periodic with period κ and for any $0 < \kappa' < \kappa$, κ' is not a period of the MDP.

It is straightforward to compute the minimal period of a task scheduling problem. Suppose that we can specify the utilization target $\mathbf{u} = (u_1, \ldots, u_n)$ as a vector of rational values. Let each component $u_i = q_i/r_i$, where q_i and r_i are integer terms of the simplest fractional form of u_i. Then the least common multiple (lcm) of the utilization denominators, $\kappa = \text{lcm}(r_1, \ldots, r_n)$, is the minimal period of any given task scheduling MDP.

Theorem 2.4.1. *Let $\mathcal{M}$ be a task scheduling MDP with rational utilization target $\mathbf{u} = (u_1, \ldots, u_n)$, and let each $u_i = r_i/q_i$ with $gcd(r_i, q_i) = 1$. Then*

$$\kappa = \text{lcm}(r_1, \ldots, r_n)$$

is the minimal period of $\mathcal{M}$.

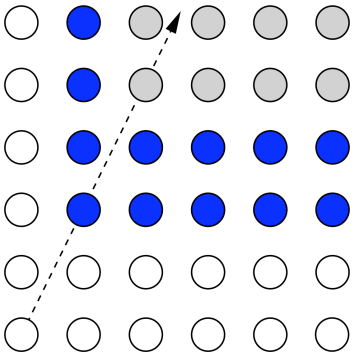


Figure 2.2: Periodic utilization states of the problem in Figure 2.1.

Proof. $\mathbf{x} + \kappa\mathbf{u}$ is integer-valued if and only if $\kappa\mathbf{u}$ is integer-valued, so it suffices to demonstrate this latter proposition. For any task J_i, κu_i is an integer, since

$$\kappa u_i = \kappa q_i / r_i = l_i q_i$$

for some positive integer l_i. Notice that if κ' is a period of $\mathcal{M}$, then κ' is a common multiple of $r_1, \ldots, r_n$, since otherwise for some action i, $\kappa' u_i$ could not be an integer. Therefore it follows that $\kappa = \text{lcm}(r_1, \ldots, r_n)$ is the minimal period of $\mathcal{M}$. □

One consequence of periodicity is that, in order for an MDP to be periodic, the utilization target components must be rational. This does not impose much of a restriction on the kinds of systems we can consider, since we are committed to some quantized digital representation of the utilization ray in practice anyway. Below we will exploit periodicity in order to obtain more compact models; the size of these models grows with the size of the period.

Figure 2.2 illustrates the periodicity of states from the example in Figure 2.1. The utilization target is $\mathbf{u} = (1/3, 2/3)$, so the problem is is periodic with period $\kappa = 3$. Each blue state can be obtained by translating a white state up and to the right by adding $\kappa\mathbf{u}$; similarly, the grey states can be generated from the blue states by adding $\kappa\mathbf{u}$ or from the white states by adding $2\kappa\mathbf{u}$.

We say that a function $f : \mathcal{X} \to \mathbb{R}$ is a *periodic function* if and only if for any utilization state $\mathbf{x}$ and positive integer λ, $f(\mathbf{x}) = f(\mathbf{x} + \lambda\kappa\mathbf{u})$. In other words, a periodic function is any function that agrees at every state that is collinear with any

given state along the utilization ray. If we know the value of a periodic function at $\mathbf{x}$ then we also know the value automatically at each $\mathbf{x} + \lambda\kappa\mathbf{u}$. If we know that a function is periodic, then there are infinitely many states that we do not need to represent in order to store that function exactly.

Below, we will demonstrate that the optimal value function is periodic, and that there is a periodic optimal policy as well. In order to justify this claim, we will show that the set of periodic functions is closed under Γ and Γ_i. We will also show that the c_p cost functions are periodic, as indicated above.

Theorem 2.4.2. *If V and c are periodic functions, then ΓV and $\Gamma_i V$ are periodic functions for any action i.*

Proof. Let $\mathbf{x}$ be a utilization state and λ be a positive integer.

$$\begin{aligned}(\Gamma_i V)(\mathbf{x} + \lambda\kappa\mathbf{u}) &= \sum_{t=1}^{\infty} P_i(t)\left[\gamma V(\mathbf{x} + \lambda\kappa\mathbf{u} + t\Delta_i) - c(\mathbf{x} + \lambda\kappa\mathbf{u} + t\Delta_i)\right] \\ &= \sum_{t=1}^{\infty} P_i(t)\left[\gamma V(\mathbf{x} + t\Delta_i) - c(\mathbf{x} + t\Delta_i)\right] \\ &= (\Gamma_i V)(\mathbf{x})\end{aligned}$$

Periodicity of ΓV follows immediately from the definition of $(\Gamma V)(\mathbf{x}) = \max_i\{(\Gamma_i V)(\mathbf{x})\}$; since the per-action backups agree, the maxima agree. □

Corollary 2.4.1. *V^* is periodic whenever c is periodic.*

Proof. $V^* = \lim_{k\to\infty}\{\Gamma^k V\}$ for any V in $\mathcal{V}_\alpha$. 0 is a periodic function in $\mathcal{V}_\alpha$. By Theorem 2.4.2, if $\Gamma^k V$ is periodic, then so is $\Gamma^{k+1}V$. By induction, V^* must be periodic. □

Corollary 2.4.2. *If c is periodic, then there is an optimal, deterministic, periodic policy π.*

Proof. Any action in $\operatorname{argmax}_i\{(\Gamma_i V^*)(\mathbf{x})\}$ is optimal at state $\mathbf{x}$. Since V^* is periodic, by Theorem 2.4.2,

$$\operatorname*{argmax}_{i\in\mathcal{A}}\{(\Gamma_i V^*)(\mathbf{x})\} = \operatorname*{argmax}_{i\in\mathcal{A}}\{(\Gamma_i V^*)(\mathbf{x} + \lambda\kappa\mathbf{u})\},$$

so that i is an optimal action at $\mathbf{x}$ if and only if it is optimal at $\mathbf{x} + \lambda\kappa\mathbf{u}$. □

Theorem 2.4.3. *For any p, c_p is periodic.*

Proof. Let $\mathbf{x}$ be an arbitrary utilization state and let λ be a positive integer. Notice that t is a linear function and that $\tau(\mathbf{u}) = 1$. Therefore,

$$\begin{aligned} c_p(\mathbf{x} + \lambda\kappa\mathbf{u}) &= \|\mathbf{x} + \lambda\kappa\mathbf{u} - \tau(\mathbf{x} + \lambda\kappa\mathbf{u})\mathbf{u}\|_p \\ &= \|\mathbf{x} + \lambda\kappa\mathbf{u} - \tau(\mathbf{x})\mathbf{u} - \lambda\kappa\mathbf{u}\|_p \\ &= \|\mathbf{x} - \tau(\mathbf{x})\mathbf{u}\|_p . \end{aligned}$$

□

Periodicity of the L_p costs also implies periodicity of some derived cost functions. For example, the family of ε-insensitive cost functions $c_{p,\varepsilon}$,

$$c_{p,\varepsilon}(\mathbf{x}) = \begin{cases} c_p(\mathbf{x}) - \varepsilon & c_p(\mathbf{x}) > \varepsilon \\ 0 & c_p(\mathbf{x}) \leq \varepsilon \end{cases}$$

is periodic. This cost function is useful if we wish to permit bounded deviations from target utilization. In fact, any composite cost function c that can be expressed as $c(c_p(\mathbf{x}))$ is periodic. This includes cost functions that threshold on $c_p(\mathbf{x})$ cost, so that the cost is zero if $c_p(\mathbf{x})$ is less than a threshold, otherwise the cost is negative. Thresholded cost functions are useful if we want to determine the existence of policies that never reach high L_p-cost states from the initial state $\mathbf{0}$, since if such a policy exists, its value at $\mathbf{0}$ is zero under thresholded costs.

It is worth noting that another natural expression of the c_p cost objective is *not* periodic. In state $\mathbf{x}$, task J_i's resource share is $x_i/\tau(\mathbf{x})$, so its deviation from target utilization is $|x_i/\tau(\mathbf{x}) - u_i|$. We could express this as a cost function proportional to $c_1(\mathbf{x})$ by defining

$$c_{1,u}(\mathbf{x}) = \sum_{i=1}^{n} |x_i/\tau(\mathbf{x}) - u_i| .$$

$c_{1,u}$ is not periodic, since $c_{1,u}(\mathbf{x}+\lambda\kappa\mathbf{u}) = c_{1,u}(\mathbf{x})/(\tau(\mathbf{x})+\lambda\kappa)$. As time progresses, $c_{1,u}$ becomes less sensitive small deviations from target utilization, as a single transition

is unable to change the utilization much. The c_p costs we employ do not suffer from this issue, and so are better suited to reducing deviation from target utilization over both the short and the long term.

Periodicity implies that the task scheduling problems have a reduced set of states that provide a complete representation of the deviations from target utilization that we care about. In the next chapter, we exploit the periodicity of the optimal value function in order to allow its computation with a more compact representation.

2.5 Discussion

MDPs with infinite state or action spaces: There is an extensive literature surrounding the problem of extending MDP solution techniques to problems with infinite state or action spaces, though a complete survey of these results is beyond the scope of this work. Boyan and Moore [18] demonstrated that it is difficult guarantee stable value function approximation when directly substituting off-the-shelf function approximation strategies in place of a tabular value function representation. Gordon [39] characterized a class of stable approximation strategies. Lagoudakis and Parr [53] adapted the policy iteration algorithm to the linear function approximation setting. Szepesvári and Munos [90, 67] studied the finite-sample performance of general implementations of fitted value iteration, a procedure analogous to value iteration that fits a general function approximation architecture to the intermediate value functions.

Finiteness of V^*: We published the proof that the value function exists in Section 2.3 in RTSS 2008 [38]. A much simpler proof is possible when the worst-case execution time among all tasks T is finite, since then for any policy π,

$$V^{\pi}(\mathbf{x}) = -\mathbb{E}\left\{ \sum_{k=0}^{\infty} \gamma^k c_p(\mathbf{x}_k) \middle| \mathbf{x}_0 = \mathbf{x} \right\},$$

with the expectation taken with respect to the sequence of actions taken by the policy π. Each state $\mathbf{x}_k$ can be expanded by considering the individual steps needed to arrive

at that state,

$$\mathbf{x}_k = \mathbf{x} + \sum_{d=1}^{k} t_d \Delta_{i_d},$$

where $i_d = \pi(\mathbf{x}_d)$ is the policy action at decision epoch d, and t_d is its duration, distributed according to P_{i_d}. Then by Theorem 2.2.1, using the trajectory $(\mathbf{x}_k)_{k=0}^{\infty}$ we can obtain a worst-case upper bound on the cost of $\mathbf{x}_k$

$$c_p(\mathbf{x}_k) \leq c_p(\mathbf{x}) + \sum_{d=1}^{k} t_d c_p(\Delta_{i_d}).$$

By linearity of expectations, we can write

$$\begin{aligned} V^{\pi}(\mathbf{x}) &= -\sum_{k=0}^{\infty} \gamma^k \mathbb{E}\left\{c_p(\mathbf{x}_k) | \mathbf{x}_0 = \mathbf{x}\right\} \\ &\geq -\sum_{k=0}^{\infty} \gamma^k \mathbb{E}\Big\{c_p(\mathbf{x}) + \sum_{d=1}^{k} t_d c_p(\Delta_{i_d}) \Big| \mathbf{x}_0 = \mathbf{x}\Big\} \\ &= -\sum_{k=0}^{\infty} \gamma^k \Big[c_p(\mathbf{x}) + \sum_{d=1}^{k} \mathbb{E}\left\{t_d c_p(\Delta_{i_d}) | \mathbf{x}_0 = \mathbf{x}\right\}\Big]. \end{aligned}$$

We can simplify this by bounding the expectation $\mathbb{E}\left\{t_d c_p(\Delta_{i_d}) | \mathbf{x}_0 = \mathbf{x}\right\}$. This can be accomplished by noticing that $c_p(\Delta_i) \leq 2$ for any action i and choice of p; we can also use the fact that t_d can not exceed the maximum worst-case execution time T. This gives us

$$\mathbb{E}\left\{t_d c_p(\Delta_{i_d}) | \mathbf{x}_0 = \mathbf{x}\right\} \leq 2T.$$

We can substitute this into the V^{π} inequality to get

$$\begin{aligned} V^{\pi}(\mathbf{x}) &\geq -\sum_{k=0}^{\infty} \gamma^k c_p(\mathbf{x}) - \sum_{k=0}^{\infty} \gamma^k \sum_{d=1}^{k} 2T \\ &= -\frac{c_p(\mathbf{x})}{(1-\gamma)} - 2T \sum_{k=0}^{\infty} \gamma^k k \\ &= -\frac{c_p(\mathbf{x})}{(1-\gamma)} - \frac{2T\gamma}{(1-\gamma)^2}. \end{aligned} \tag{2.22}$$

This provides a finite lower bound on the value of any policy at each state. Since the value can never be greater than zero, this suffices to show that every policy has finite value at each state, even though V^π is not bounded.

This proof is conceptually simpler than that provided in Section 2.3, since we did not need to introduce any additional function spaces in order to reason about the value. However, the previous results appear to be more powerful, since the additional structure of the Banach space $\mathcal{V}_\alpha$ allows us to invoke the fixed point theorem to prove that Γ and Γ_π have unique fixed points corresponding to the optimal value function and the policy value functions, respectively.

Periodicity as State Aggregation: State aggregation techniques [56] attempt to reduce the computational burden of finding a good policy for controlling an MDP. These techniques achieve this goal by combining, or aggregating, states to obtain a more compact abstract MDP with a smaller state space. Depending on the aggregation strategy used, the abstract MDP may retain only a subset of the policies for the original problem.

McCallum's utile distinction memory approach [64] was based on the principle that states should only be distinguished if doing so allows the system to achieve better performance. This occurs whenever one of the states in the aggregate has different value than the rest. The decision tree based value representation of Boutilier *et al.* [17] implicitly incorporates this principle, as the length of a path from the root to leaf, which is equivalent to an assignment to relevant state values, corresponds exactly to the set of state variables that influence the value in a particular state. The utile distinction principle leads us to consider a more compact task scheduling MDP by pruning out states that are periodically related to at least one other state until we have exactly one state from each equivalence class.

Givan *et al.* [34] proposed a notion of stochastic bisimulation. Two MDPs M and M' are in bisimulation if and only if there is a binary relation between states of each MDP involving every state in each MDP such that related states have the same expected reward and whenever x and x' are related and y and y' are related, then $P(y|x,a) = P(y'|x',a)$. This notion is perhaps most useful when a larger MDP is in stochastic bisimulation with a smaller one, as a solution to the smaller MDP is

equivalent to a solution to the larger. Periodicity in our application can be restated as bisimulation between the utilization state MDP and a related MDP over exemplars from each equivalence class. MDP homomorphisms [74, 75] can be viewed as an extension of stochastic bisimulation in that it allows a notion of action equivalence in addition to state equivalence.

The topics covered in this section are solely concerned with existential questions regarding the structure of the task scheduling problem formulated as a Markov Decision Process. In the next chapter, we approach the problem from a less abstract standpoint, focusing on practical issues, including the representation of the task scheduling MDP and computational issues that arise given that we can only work with a finite subset of the full utilization state space.

Scaled cost as a dynamic sliding window: We mentioned the distinction between using $c_p(\mathbf{x}) = \|\mathbf{x} - \tau(\mathbf{x})\mathbf{u}\|_p$, which penalizes deviation from the best possible utilization state at time $\tau(\mathbf{x})$, versus the utilization cost $c_{p,u}(\mathbf{x}) = \|\mathbf{x}/\tau(\mathbf{x}) - \mathbf{u}\|_p$, which penalizes deviation from the utilization target. The former is periodic, while the latter is not. The distinction between these costs is subtle: both reward hitting the utilization target exactly; small perturbations to the state have less impact as time passes under the latter but not under the former. One way to look at this is that our approach, trying to hit a target point at each time step, encourages us to stay on target over all time, while the latter permits a policy to deviate from target utilization if it has done a good job of maintaining target utilization over most of the history.

One way to avoid ossification of state due to accumulated history is by maintaining a sliding window over history. That is, we could retain system states corresponding to the resource utilization of each task over the last t time quanta for some choice of t. Once the system has executed for at least t quanta, the change in cost due to executing a task maintains a fixed, attainable upper bound.

It turns out that the L_p cost function performs a similar purpose, except that it uses a dynamic window length: periodicity implies that whenever the historic utilization x_i of each task J_i achieves or exceeds its periodic target κu_i, we can throw out $\kappa\mathbf{u}$ quanta of history from each task and get an equivalent state.

Chapter 3

Solution Methods

In Chapter 2 we modeled the task scheduling problem from Chapter 1 as a discrete Markov Decision Process with a countably infinite state space. We call this model the *utilization state model* because each model state represents the cumulative resource utilization of each task. This implicitly assumes that, in order to make appropriate scheduling decisions, we do not need to know exactly when in the past a task has occupied the shared resource; but only how much time it has spent occupying it.

It is straightforward to describe the task scheduling problem using the utilization state model. Whenever a dispatched task completes, the utilization state is updated by incrementing the corresponding state variable by the task's observed duration. In order to encourage solutions that keep the resource share near the utilization target, maximizing reward in the utilization state model corresponds to minimizing costs for deviating from target utilization.

We demonstrated that this model has a unique optimal value function in the infinite-horizon discounted reward case. We also noted that the optimal value function exhibits periodic behavior under appropriate conditions on the utilization target and cost function. In this chapter we will exploit these observations in order to obtain methods for approximating and solving the task scheduling problem as an MDP.

In Section 3.1 we propose a more compact model, the *wrapped state model*, that removes an infinitude of states from the utilization state model. This is achieved by removing equivalent states, then re-establishing the transition function to account for the probability of moving between collections of equivalent states. We prove that

this model retains the optimal solution to the utilization state model; however, the wrapped state model still has infinitely many states.

In Section 3.2 we demonstrate that when L_p costs are used, there are finitely many wrapped model states with cost less than any specified bound. We use this observation to truncate the state space and provide bounds on the quality of the truncated model. In Section 3.3 we propose a technique for automatically constructing a state space that is necessary for policy evaluation and improvement. We present the ESPI algorithm, which computes locally optimal solutions in this setting. We conclude this chapter with discussion of related work that has motivated these methods.

3.1 Wrapped State Model

In the previous chapter, we defined the utilization state model, consisting of a straightforward MDP formulation of the task scheduling problem. While this model is useful for grounding the semantics of the task scheduling problem in a computational approach, general direct MDP solution techniques like value iteration and policy iteration can not be applied to this model because of the size of its state space.

The periodicity property indicates that the utilization state model carries substantial redundant information. If the model is periodic with period κ, then the optimal value is the same whether the system is in state $\mathbf{x}$ or $\mathbf{x} + \kappa\mathbf{u}$. Intuitively it does not seem necessary to consider more than one of these two states. However, being able to *aggregate* these equal value states into a single exemplar state is contingent not just on states' values, but on the relationship between possible future state space trajectories from each state [24, 56]. In this section we show that we can safely aggregate equivalent states given the properties of state transitions and rewards in the utilization state model, in the sense that there is an MDP defined in terms of aggregates of states from the utilization state model so that the optimal policy for the aggregate model corresponds to an optimal policy for the utilization state model.

Congruence in modular arithmetic can be extended to vectors of positive integers as follows. Let $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ be vectors in $\mathbb{Z}_+^n$. Then $\mathbf{x}$ is congruent to $\mathbf{y}$ modulo $\mathbf{z}$ if and only if the displacement vector $\mathbf{x} - \mathbf{y}$ is a scalar multiple of $\mathbf{z}$. As in the case where

$n = 1$, congruency modulo a vector is an equivalence relation over $\mathbb{Z}_+^n$, and we write $\mathbf{x} \equiv_{\mathbf{z}} \mathbf{y}$, or just $\mathbf{x} \equiv \mathbf{y}$ when the vector modulus is clear from context.

We can use this formalization to describe periodicity more precisely. If $\mathbf{x}$ and $\mathbf{y}$ are utilization states with $\mathbf{y} = \mathbf{x} + \lambda\kappa\mathbf{u}$ for some λ, then $\mathbf{x}$ and $\mathbf{y}$ are congruent modulo $\kappa\mathbf{u}$. As we showed in the previous chapter, since $\mathbf{x} \equiv_{\kappa\mathbf{u}} \mathbf{y}$, then any periodic function f agrees at these states, including the optimal value function and some optimal policy. In this chapter we will show that we can construct an MDP model using a single exemplar from each each equivalence class; in principle, the choice of exemplar should not matter, but computationally it is useful to choose this state in a principled way.

The vectors that occur along a line in $\mathbb{R}^n$ are totally ordered under $\preceq$. To see this, if the line has direction $\mathbf{z}$ and contains $\mathbf{x}$, then any other vector $\mathbf{y}$ on this line can be written as $\mathbf{y} = \mathbf{x} + \lambda\mathbf{z}$ for some λ, with $\mathbf{y} \prec \mathbf{x}$ if λ is negative and otherwise $\mathbf{x} \succeq \mathbf{y}$. Since we restrict consideration to just vectors with non-negative integer components, there is a well-defined least vector in $\mathbb{Z}_+^n$ along any such line. This vector is the remainder $w(\mathbf{x}, \mathbf{z})$, defined

$$w(\mathbf{x}, \mathbf{z}) = \mathbf{x} - \lambda^*\mathbf{z}, \tag{3.1}$$

with

$$\lambda^* = \min_{i=1,\ldots,n} \lfloor x_i/z_i \rfloor. \tag{3.2}$$

In the context of the task scheduling problem, we treat the state $w(\mathbf{x}, \kappa\mathbf{u})$ as an exemplar of the set of states congruent to $\mathbf{x}$ modulo $\kappa\mathbf{u}$. We denote this state more concisely as $w(\mathbf{x})$. We call w the *wrapping function*, since conceptually, it wraps the utilization state space into a topological cylinder so that all equivalent states map to one another. When $n = 2$, this can be visualized by drawing the states on a sheet of paper, then tightly rolling the sheet of paper in the direction of $\mathbf{u}$ so that $\mathbf{0}$ and $\kappa\mathbf{u}$ overlap.

We define the *wrapped state model* as an MDP over these exemplar states; the state space of this model is the set

$$\mathcal{W} = \{w(\mathbf{x}) | \mathbf{x} \in \mathcal{X}\}, \tag{3.3}$$

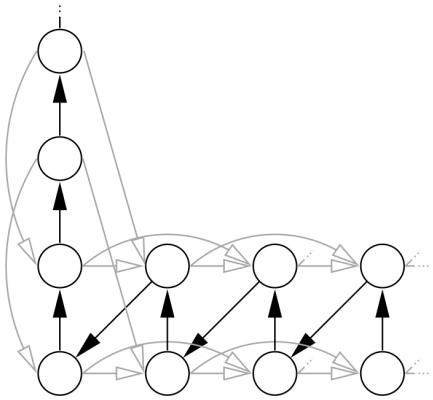


Figure 3.1: The wrapped model of the utilization state MDP from Figure 2.1.

where $\mathcal{X}$ is the utilization state space. In the previous chapter, we showed that this state space is sufficient to represent the optimal value function and an optimal policy. In this chapter, we will further show that a model defined over the wrapped state space is sufficient for computing the optimal value function of the utilization state model, once we have adjusted the transition system appropriately.

In practice, wrapping the state space removes a huge number of states. This corresponds to removing every utilization state $\mathbf{x} \succeq \kappa\mathbf{u}$. Figure 3.1 illustrates the wrapped state model for the example problem from Figure 2.1. Following the black transition out of (1,1) in the utilization state model would have moved the system upward into the state (1,2), which has cost zero; in the wrapped model, this transition is remapped to the state (0,0).

The set of exemplar states is determined by utilization targets, but is invariant to changes in the duration distribution, since the wrapping function is independent of task durations. It is worth considering how $\mathcal{W}$ changes as we vary the utilization target. For example, in the example shown in Figure 3.1, the wrapped state space consists of an infinite strip of states along the horizontal axis and the vertical axis; the width of these strips is determined by the utilization target numerators, as $\mathbf{u} = (1/3, 2/3)$. In general, the wrapped state space is contained in the union of $(n-1)$-dimensional axis-aligned hypercubes with infinite extent.

It is interesting to note how the size of the state space grows as we vary the utilization target. Of course, since the wrapped state space is infinite, the cardinality is a poor measure of size. Instead, we will consider the number of wrapped states in the

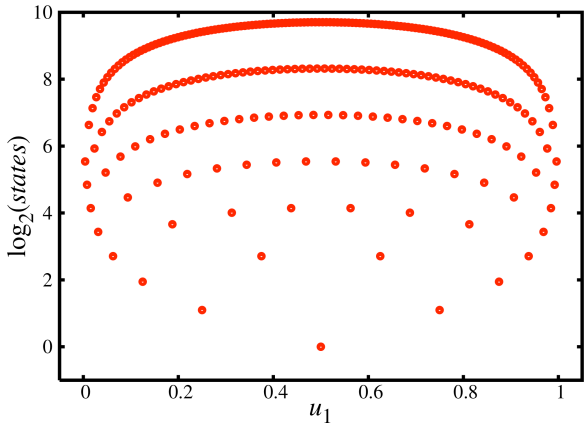


Figure 3.2: The number of states in $[\mathbf{0}, \kappa\mathbf{u})$ in two-task wrapped state scheduling MDPs with varying utilization targets. See the text for further details.

rectangle $[0, \kappa u_1) \times [0, \kappa u_2)$ for a two-task problem – in the example in Figure 3.1, this rectangle contains the states $(0, 0)$ and $(0, 1)$. If the utilization target $\mathbf{u} = (r_1/q, r_2/q)$ with r_1 and r_2 relatively prime, then there are $r_1 r_2$ states in this rectangle. Then the system period is the common denominator $q = r_1 + r_2$; as κ grows large, the number of states grows large as well.

The period κ directly describes the *resolution* of the utilization target. When κ is small, for example, if $\kappa = 2$, then the only utilization we can describe is $\mathbf{u} = (1/2, 1/2)$, the fair share case. As q grows larger, we can describe a wider variety of utilization targets; for example, we can describe the fair share case $\mathbf{u} \equiv (128/256, 128/256)$, but we can also describe situations that are almost fair, such as $(127/256, 129/256)$. While the difference in utilization targets is small, the difference in the number of states is striking. There is only one state in the region of interest in the fair share case compared to more than sixteen thousand for the $(127/256, 129/256)$ case!

Figure 3.2 explicitly quantifies the number of states for utilization targets of the form $\mathbf{u} = (u_1'/256, (256 - u_1')/256)$. Since $u_2 = 1 - u_1$, we plot u_1 against the number of states in the half-open rectangle $[\mathbf{0}, \kappa\mathbf{u})$. The number of states falls into a series of bands; the uppermost band corresponds to utilization targets that are irreducible. The next highest band corresponds to utilization targets that can be cut in half losslessly (*i.e.*, problems with period $\kappa = 128$), the next can be cut in half again, and so on. The least number of states occurs in the fair share case when $u_1 = u_2$.

Figure 3.2 implies a useful tradeoff, as we may approximate high resolution utilization targets – utilization targets that require large denominators – using lower resolution targets by scaling down the utilizations and rounding appropriately. The figure shows that reducing the resolution can dramatically decrease the number of states. Due to stochasticity in task duration distributions, even the best scheduling policies will only keep the system near target utilization, and will rarely achieve the target utilization exactly. It seems likely that stochasticity in task durations will negate most of the drawbacks from using a lower resolution utilization target, since the difference between the actual utilization of each task and its target will probably be quite variable over small intervals.

Next, we consider how to define an MDP over the wrapped state model. Moving from the utilization state model to a model over the wrapped state space leaves some "dangling" transitions whenever it is possible to transition from an exemplar state to a non-exemplar state. We can reattach any of these dangling transitions to an equivalent exemplar state that is in the wrapped state space, as we have done when mapping the example utilization state model in Figure 2.1 to the wrapped model in Figure 3.1. Lemma 3.1.1 provides a basis for this result: any two equivalent states have equivalent successors.

Lemma 3.1.1. *If* $\mathbf{x}$ *and* $\mathbf{y}$ *are equivalent utilization states in* $\mathcal{X}$*, then for any task* J_i *and duration* t*,*

$$\mathbf{x} + t\Delta_i \equiv \mathbf{y} + t\Delta_i$$

Proof. Without loss of generality, we assume that $\mathbf{x} \preceq \mathbf{y}$. Then $\mathbf{y} = \mathbf{x} + \lambda\mathbf{u}$ for some non-negative integer λ. The claim follows immediately, since

$$\mathbf{y} + t\Delta_i = \mathbf{x} + t\Delta_i + \lambda\kappa\mathbf{u} \equiv \mathbf{x} + t\Delta_i.$$

□

As a consequence of Lemma 3.1.1, any successor state $\mathbf{x}+t\Delta_i$ is equivalent to $w(w(\mathbf{x})+t\Delta_i)$, since

$$\mathbf{x} + t\Delta_i \equiv w(\mathbf{x}) + t\Delta_i \equiv w(w(\mathbf{x}) + t\Delta_i)$$

This means that the successor state distribution at $\mathbf{x}$ is captured exactly using only exemplar states.

We define the *wrapped state model* as a Markov Decision Process with state space $\mathcal{W}$, the actions from the utilization state model $\mathcal{A}$, transitions P_w, and rewards R_w. For this model to be useful, we need to define P_w and R_w so that the optimal value of any state in the wrapped model agrees with the optimal value of all equivalent states in the original model. To achieve this, we require that executing any task in any state results in the same distribution over rewards as executing the same task in an equivalent state in the original model. We can then invoke Theorem 2.4.2 to guarantee that this wrapped model captures the optimal value function.

Conceptually, we define the transition function P_w by starting with the transition system from the original problem (see Figure 2.1). Any transition to a state that does not map to itself under w (*e.g.*, $\mathbf{x} \neq w(\mathbf{x})$) is instead mapped to an equivalent state. We then remove the unreachable states. The resulting transition system is shown in Figure 3.1.

Let $\mathbf{x}$ and $\mathbf{y}$ be wrapped states in $\mathcal{W}$, and let J_i be a task. We define the transition function P_w in terms of duration distributions,

$$P_w(\mathbf{y}|\mathbf{x}, i) = \begin{cases} P_i(t) & \mathbf{y} = w(\mathbf{x} + t\Delta_i) \\ 0 & otherwise. \end{cases}$$

This maps the successors in the utilization state model back into the wrapped state space to establish the transition. This definition satisfies

$$P_w(\mathbf{y}|\mathbf{x}, i) = P(\mathbf{z} + t\Delta_i|\mathbf{z}, i) = P_i(t)$$

whenever $\mathbf{z}$ is a utilization state equivalent to $\mathbf{x}$.

Since we require the cost function c to be periodic, any two equivalent states have the same cost. Following Equation 2.12, we define the wrapped-state rewards according to

$$R_w(\mathbf{x}, i, \mathbf{y}) = -c(\mathbf{y})$$

which satisfies $R(\mathbf{x}, i, \mathbf{y}) = R_w(\mathbf{x}, i, \mathbf{y})$ because of cost periodicity. Thus, for any utilization $\mathbf{z}$ and wrapped state $\mathbf{x} = w(\mathbf{z})$, the expected rewards obey

$$\begin{aligned} R_w(\mathbf{x}, i) &= -\sum_{\mathbf{y} \in \mathcal{W}} P(\mathbf{y}|\mathbf{x}, i) c(\mathbf{y}) \\ &= -\sum_{t=1}^{\infty} P_i(t) c(w(\mathbf{x} + t\Delta_i)) \\ &= -\sum_{t=1}^{\infty} P_i(t) c(\mathbf{x} + t\Delta_i) \\ &= R(\mathbf{z}, i), \end{aligned}$$

so that the expected reward for dispatching task J_i in the wrapped state $\mathbf{x}$ matches the expected reward of all equivalent states in the original model. This formulation is sufficient to compute the optimal value function to wrapped state model, which is equivalent to the optimal value function of the original model.

Theorem 3.1.1. *Let V^* and V_w^* be the optimal value functions for the original and wrapped state models, respectively. For any utilization state $\mathbf{z}$, we have*

$$V_w^*(w(\mathbf{z})) = V^*(\mathbf{z}).$$

Proof. Let V be a periodic function over $\mathcal{X}$, $V_{w,k} = \Gamma_w^k V$, and $V_k = \Gamma^k V$. Then for any utilization state $\mathbf{z}$,

$$V_{w,0}(w(\mathbf{z})) = V(w(\mathbf{z})) = V(\mathbf{z}) = V_0(\mathbf{z}).$$

Suppose that $V_{w,k-1}(w(\mathbf{z})) = V_{k-1}(\mathbf{z})$. Then the k-step wrapped-state approximation $V_{w,k}$ agrees with the the corresponding utilization state value function, since

$$\begin{aligned}
V_{w,k}(w(\mathbf{z})) &= (\Gamma_w V_{w,k-1})(w(\mathbf{z})) \\
&= \max_{i \in \mathcal{A}} \sum_{t=1}^{\infty} P_i(t)[\gamma V_{w,k-1}(w(\mathbf{z} + t\Delta_i)) - c(w(\mathbf{z} + t\Delta_i))] \\
&= \max_{i \in \mathcal{A}} \sum_{t=1}^{\infty} P_i(t)[\gamma V_{k-1}(\mathbf{z} + t\Delta_i) - c(\mathbf{z} + t\Delta_i)] \\
&= (\Gamma V_k)(\mathbf{z}) \\
&= V_k(\mathbf{z}).
\end{aligned}$$

Therefore, since the limits of these sequences exist in $\mathcal{V}_\alpha$, we have

$$\begin{aligned}
V_w^*(w(\mathbf{z})) &= \lim_{k \to \infty} \left\{ \left(\Gamma_w^k V\right)(w(\mathbf{z})) \right\} \\
&= \lim_{k \to \infty} \left\{ (\Gamma^k V)(\mathbf{z}) \right\} \\
&= V^*(\mathbf{z}),
\end{aligned}$$

so the optimal value in the utilization state model of $\mathbf{z}$ is identical to the optimal value of its exemplar $w(\mathbf{z})$ in the wrapped model. □

Theorem 3.1.1 shows that it is sufficient to compute the value function over the wrapped states, and the proof establishes that in principle the value iteration algorithm will correctly converge to this solution. However, computing V_w^* over $\mathcal{W}$ is no more feasible than computing the optimal solution to the utilization state model, since there are also infinitely many states in the wrapped state model. In the next section we truncate the wrapped state space to obtain an MDP with only finitely many states, analyze its approximation qualities, and examine its empirical performance.

3.2 Bounded, Wrapped State Model

Wrapping the state space allows us to describe the features of the state space that are relevant for evaluating scheduling policies more compactly. However, there are still

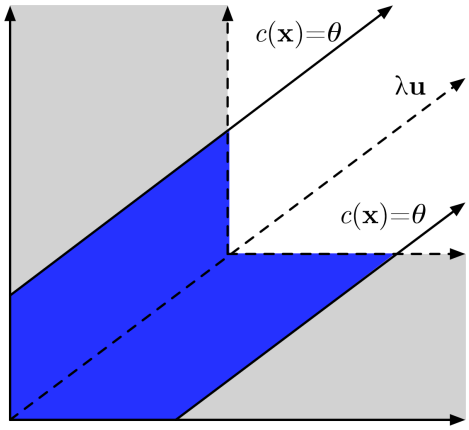


Figure 3.3: The set of utilization states that satisfy cost bound θ.

infinitely many equivalence classes in the wrapped state model. In order to obtain an MDP that we can feasibly solve, we still need to use some additional methods to restrict our attention to a finite number of states.

There are only so many "good" states in the wrapped state model. Specifically, for any cost threshold that we choose, there are only finitely many states with L_p cost below that threshold. We would expect a scheduling policy that tries to minimize cost to stay close to target utilization at all times, which means that the policy should consistently restrict itself to low-cost states. This does depend on having finite worst-case execution times for each thread, since otherwise there is always some probability that any low-cost state has a successor with arbitrarily high cost.

We can introduce some notation to formalize this first claim, that there are only finitely many states that satisfy any particular cost bound. Let θ be a non-negative real value; then we define the set of cost-bounded utilization states X_θ and cost-bounded wrapped states W_θ:

$$X_\theta = \{\mathbf{x} \in \mathcal{X} | c(\mathbf{x}) \leq \theta\}, \tag{3.4}$$

$$W_\theta = \{\mathbf{x} \in \mathcal{W} | c(\mathbf{x}) \leq \theta\}. \tag{3.5}$$

Figure 3.3 illustrates these state sets on a two-task example. Here the cost bound defines the two red rays of points $\mathbf{x}$ with cost $c(\mathbf{x}) = \theta$. The set of cost-bounded utilization states X_θ lies between these rays. The grey and blue regions contain the

wrapped state space, and the blue region between these rays contains the bounded state space W_θ.

In any periodic task scheduling problem, X_θ is infinite, since it contains the infinite set

$$X_0 = \{\lambda\kappa\mathbf{u} | \lambda \in \mathbb{Z}_+\}.$$

W_θ is finite, though; this is because X_θ intersects only finitely many equivalence classes in periodic problems, and so wrapping will remove all but a finite number of exemplar states. We formalize this result in Lemma 3.2.1.

Lemma 3.2.1. *For any θ in $[0, \infty)$ and L_p-norm cost c_p, W_θ is finite.*

Proof. Notice that we can decompose W_θ into disjoint subsets

$$W_\theta = \bigcup_{\tau=0}^{\infty} W_{\theta,\tau}$$

by defining cost-bounded sets $W_{\theta,\tau}$ consisting of wrapped states with equal cumulative utilization,

$$W_{\theta,\tau} = \{\mathbf{x} \in W_\theta | \tau(\mathbf{x}) = \tau\}.$$

Consider the union $\bigcup_{\tau=0}^{m} W_{\theta,\tau}$ for $m \in \mathbb{Z}_+$. If $\mathbf{x}$ is in $\bigcup_{\tau=0}^{m} W_{\theta,\tau}$, then every component of $\mathbf{x}$ is between 0 and m, inclusive. Then this union is a discrete subset of $[0, m]^n$, and therefore finite.

Each $W_{\theta,\tau}$ is empty for large enough τ. To see this, notice that if $\mathbf{x}$ is in $W_{\theta,\tau}$, it has some component x_i with $0 \leq x_i < \kappa u_i$, otherwise $\mathbf{x}$ would not be a wrapped state since we could subtract $\kappa\mathbf{u}$ from $\mathbf{x}$ to get an equivalent state with smaller components. As τ grows large, $|x_i - \tau u_i|$ grows large as well since x_i is bounded but τu_i grows linearly with τ. We know that action i is underutilized for large enough τ, since $\tau u_i > \kappa u_i > x_i$, so we have $|x_i - \tau u_i| = \tau u_i - x_i > \tau u_i - \kappa u_i$. Notice that

$$\tau u_i - \kappa u_i \geq \theta \equiv \tau \geq \theta/u_i + \kappa,$$

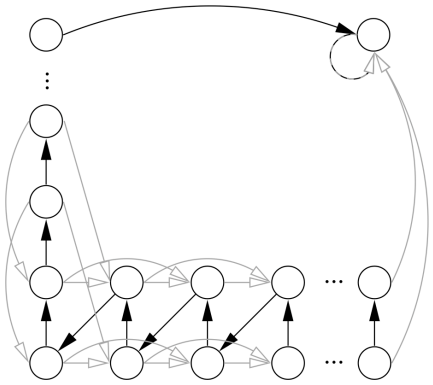


Figure 3.4: A bounded state approximation to the wrapped model from Figure 3.1.

so that if $\tau \geq \max_i\{\theta/u_i\} + \kappa$ (these inequalities are well-defined since we require $0 < u_i < 1$ for all i), then for any $\mathbf{x}$ in $W_{\theta,\tau}$ we have

$$c_p(\mathbf{x}) \geq |x_i - \tau u_i| > \tau u_i - \kappa u_i > \theta$$

so $\mathbf{x}$ can not be in $W_{\theta,\tau}$; since $\mathbf{x}$ was chosen arbitrarily, $W_{\theta,\tau}$ is empty for sufficiently large τ. Thus $W_\theta = \bigcup_{\tau=0}^{m} W_{\theta,\tau}$ for some finite m, so W_θ is finite. □

Assuming that the optimal policy never reaches states with high cost given an initial state with low cost, then the optimal policy can only visit a finite number of wrapped states. Using this assumption, our third MDP model for the task scheduling problem explicitly enforces a bound on the set of states that we will consider. In particular, we restrict consideration to the wrapped states that satisfy a user-specified cost threshold.

Figure 3.4 illustrates this construction on the example problem from Figures 2.1 and 3.1. We have replaced the continued transitions upwards and to the left with transitions to the absorbing state in the upper right. Local transitions are unchanged for the states that are far enough away from the bounds. By choosing a large enough cost for the absorbing state, we can force MDP solution methods to discover policies that always stay within the cost bounds if any such policy exists.

We obtain the *bounded state model* by constructing an MDP over the set of wrapped states in W_θ. This is achieved by preserving most of the transition and reward structure from the wrapped-state model. The main difference between the wrapped and bounded state models occur at boundary states: states in the wrapped model that

have successors with cost exceeding θ. Since these successors are not part of the wrapped state space, we are left with some dangling transitions. We abstract the removed states with a single absorbing state, and map all of the dangling transitions to the absorbing state. For example, if $\mathbf{x}$ is a wrapped state with successor $\mathbf{y} = w(\mathbf{x} + t\Delta_i)$, with $c(\mathbf{x}) \leq \theta$ but $c(\mathbf{y}) > \theta$, then the transition from $\mathbf{x}$ to $\mathbf{y}$ is replaced by a transition to the absorbing state, and the probability of transitioning to the absorbing state is increased by $P_i(t)$.

The absorbing state is an abstraction for all of the high cost states in the wrapped state model. Since our goal is to avoid high cost states, we penalize the system heavily for entering the absorbing state. In practice, we choose a cost that exceeds the cost of any successor utilization state reachable from W_θ,

$$c(\mathbf{z}_\theta) = \theta + 2T,$$

where T is the largest worst-case execution time among all tasks, since this guarantees that the bounded state model never underestimates the cost of an action.

Formally, given a cost bound θ, the bounded state MDP is defined over the states W_θ together with an absorbing state $\mathbf{z}_\theta$, along with the actions $\mathcal{A}$, the transition system P_θ, and the rewards R_θ. For any $\mathbf{x}$ and $\mathbf{y}$ in W_θ, $P_\theta(\mathbf{y}|\mathbf{x}, i) = P_w(\mathbf{y}|\mathbf{x}, i)$, while the probability of transitioning to the absorbing state in $\mathbf{x}$ is the aggregate probability of transitioning to any state with cost greater than the bounds,

$$P_\theta(\mathbf{z}_\theta|\mathbf{x}, i) = \sum_{\mathbf{y} \in \mathcal{W}/W_\theta} P_w(\mathbf{y}|\mathbf{x}, i).$$

The reward function is defined analogously to the rewards in the utilization state and wrapped state models as

$$R_\theta(\mathbf{x}, i, \mathbf{y}) = -c(\mathbf{y}).$$

The utilization state, wrapped state, and bounded state rewards are all equal when $\mathbf{y}$ is in W_θ.

While the state space W_θ is finite, its size grows exponentially in the number of tasks. The boundaries of the bounded, wrapped state space are parallel to the utilization ray, and so are not axis aligned (Figure 3.3). To get a sense for the size of the state

space, we bound it above and below using the difference of two hypercubes. In two dimensions, this corresponds to two L-shaped regions of real space, one contained completely inside the bounded state space and another completely containing the bounded state space. For vectors $\mathbf{v}$ and $\mathbf{w}$ in $\mathbb{R}^n$ with $\mathbf{v} \preceq \mathbf{w}$, let $[\mathbf{v}, \mathbf{w}]$ be the hypercube obtained by taking the Cartesian product of the intervals $\times_{i=1}^{n}[v_i, w_i]$. Then if we have bounds b_i such that $(b_i - 1)c(\Delta_i) < \theta$ and $b_i c(\Delta_i) \geq \theta$, the bounded state space is contained in

$$[\mathbf{0}, \mathbf{b} + \kappa\mathbf{u})/[\kappa\mathbf{u}, \mathbf{b} + \kappa\mathbf{u}),$$

and the cost bounds contain the set

$$[\mathbf{0}, \mathbf{b} - \mathbf{1}]/[\kappa\mathbf{u}, \mathbf{b} - \mathbf{1} + \kappa\mathbf{u}].$$

This does mean that the bounded state model is restricted to small numbers of tasks in practice. In Chapter 5, we discuss methods for approximating an optimal policy in problems involving many tasks using direct policy search techniques [88, 52, 71].

3.2.1 Approximation Quality

The bounded, wrapped state model achieves our goal of providing a finite model of the task scheduling problem; intuitively, it restricts the set of states we consider to just "good" states – states with relatively low costs. In this section we derive a prior bound on the approximation error between the optimal value function of the truncated model versus the optimal utilization state value function.

The basic idea is straightforward: we can bound the contribution of any state $\mathbf{y}$ with high cost to the value at some arbitrary state $\mathbf{x}$ if we know that, for example, $c(\mathbf{y}) \geq 2Tk + c(\mathbf{x})$. This is because any policy that reaches $\mathbf{y}$ from $\mathbf{x}$ must take at least k decision epochs to do so; then the contribution of $\mathbf{y}$ has weight at most γ^k. Even if our policy could reach $\mathbf{y}$ in k steps and stay there, the contribution to the value at $\mathbf{x}$ is at most $\gamma^k c(\mathbf{y})/(1-\gamma)$. Since L_p costs grow polynomially while high-cost states are discounted exponentially, the contribution of far away states vanishes. This means that if we want to estimate accurately the value of states in W_θ, then there is

another threshold θ' such that a bounded state model defined over $W_{\theta'}$ is sufficient to guarantee the desired precision.

With this intuition in mind, we present a more formal worst-case analysis of the quality of approximations obtained by solving the bounded state model. Our analysis is a specialization of general results from Chapter 6.10 of Puterman's book for analyzing the performance of finite-state approximation of countable state models [72]. First, we define a sequence of cost thresholds

$$(\theta_k = 2Tk)_{k=0}^{\infty}$$

we assume that the threshold θ used to define the bounded state model is in this sequence, with $\theta = \theta_m$. This in turn defines a sequence of nested wrapped state sets

$$\{\mathbf{0}\} \subset W_{\theta_0} \subset W_{\theta_1} \subset \ldots \subset W_{\theta_m} = W_\theta \subset \ldots$$

To reduce the proliferation of subscripts, we abbreviate W_{θ_k} to just W_k. It is not possible to reach any state with cost greater than θ_k after taking any sequence of within k decision epochs from the initial state $\mathbf{0}$, so if we start in $\mathbf{0}$ and dispatch k actions, the system state is in W_k. Similarly, if the system is initially in $\mathbf{x} \in W_j$, then after k decision epochs the system state is guaranteed to be in W_{j+k}.

It is useful for our discussion to look at sets of states with cost bounded above and below. Specifically, the set difference W_k/W_{k-1} is the set of states with cost in the interval $(\theta_{k-1}, \theta_k]$. We denote the set of these states as Z_k:

$$Z_k = \begin{cases} W_k/W_{k-1}, & k > 0 \\ W_0, & k = 0. \end{cases}$$

Together $(Z_k)_{k=0}^{\infty}$ partitions the wrapped state space $\mathcal{W}$ into cells with the property that if $\mathbf{x}$ is in Z_k, then its successors are in the union of Z_{k-1}, Z_k, and Z_{k+1}.

Given the cost threshold θ, for any real-valued function V over $\mathcal{W}$ we write the bounded model backup operators as

$$(\Gamma_\theta V)(\mathbf{x}) = \max_{i \in \mathcal{A}} \{(\Gamma_{\theta,i} V)(\mathbf{x})\}, \tag{3.6}$$

$$(\Gamma_{\theta,i} V)(\mathbf{x}) = \sum_{\mathbf{y} \in W_\theta} P(\mathbf{y}|\mathbf{x}, i)[\gamma V(\mathbf{y}) - c(\mathbf{y})] + \sum_{\mathbf{y} \notin W_\theta} P(\mathbf{y}|\mathbf{x}, i) v_\theta \tag{3.7}$$

where v_θ is the default, fixed value of the absorbing state $\mathbf{z}_\theta$,

$$v_\theta = -\theta_{m+1}/(1-\gamma).$$

We use this as the default value for unrepresented states because θ_{m+1} is a pessimistic estimate of the cost of states reachable from W_θ; this biases solutions towards policies that stay inside W_θ by making the alternative prohibitively expensive.

If $\mathbf{x}$ is in W_{m-1}, then its successors are entirely inside W_θ, so these backup operators are equal to their utilization or wrapped state equivalents. Let $V^*_\theta = \Gamma_\theta V^*_\theta$ be the optimal bounded state value function. Let

$$\varepsilon_k = \max_{\mathbf{x} \in Z_k} |V^*(\mathbf{x}) - V^*_\theta(\mathbf{x})|$$

be the largest magnitude error over states in Z_k. Below, we exploit the locality of the transition function to show that ε_k depends directly only on ε_{k-1}, ε_k, and ε_{k+1}. Our intuition is that ε_k is largest when k is near m, since it takes fewer steps to reach outside of W_θ from Z_k as k approaches m. To formalize this intuition, we make use of the following straightforward inequality,

$$\begin{aligned} |V^*(\mathbf{x}) - V^*_\theta(\mathbf{x})| &= |(\Gamma V^*)(\mathbf{x}) - (\Gamma_\theta V^*_\theta)(\mathbf{x})| \\ &\leq \max_{i \in \mathcal{A}} |(\Gamma_i V^*)(\mathbf{x}) - (\Gamma_{\theta,i} V^*_\theta)(\mathbf{x})| \,. \end{aligned}$$

This indicates that we can bound the approximation error in terms of backups performed with respect to a single action. This eliminates the need to consider possible

differences in the optimal policy at $\mathbf{x}$.

$$
\begin{aligned}
&|(\Gamma_i V^*)(\mathbf{x}) - (\Gamma_{\theta,i} V_\theta^*)(\mathbf{x})| \\
&= \Big|\gamma \sum_{j=0}^{m} \sum_{\mathbf{y}\in Z_j} P(\mathbf{y}|\mathbf{x},i)[V^*(\mathbf{y}) - V_\theta^*(\mathbf{y})] + \sum_{\mathbf{y}\in Z_{m+1}} P(\mathbf{y}|\mathbf{x},i)[\gamma V^*(\mathbf{y}) - c(\mathbf{y}) - \gamma v_\theta]\Big| \\
&\leq \gamma \sum_{j=0}^{m} \sum_{\mathbf{y}\in Z_j} P(\mathbf{y}|\mathbf{x},i)\,|V^*(\mathbf{y}) - V_\theta^*(\mathbf{y})| + \sum_{\mathbf{y}\in Z_{m+1}} P(\mathbf{y}|\mathbf{x},i)\,|\gamma V^*(\mathbf{y}) - c(\mathbf{y}) - \gamma v_\theta| \\
&\leq \gamma \sum_{j=0}^{m} \sum_{\mathbf{y}\in Z_j} P(\mathbf{y}|\mathbf{x},i)\varepsilon_j + \sum_{\mathbf{y}\in Z_{m+1}} P(\mathbf{y}|\mathbf{x},i)[\gamma\varepsilon_{j+1} + c(\mathbf{y})] \\
&\leq \gamma \sum_{j=0}^{m+1} \sum_{\mathbf{y}\in Z_j} P(\mathbf{y}|\mathbf{x},i)\varepsilon_j + \sum_{\mathbf{y}\in Z_{m+1}} P(\mathbf{y}|\mathbf{x},i)\theta_{m+1}
\end{aligned}
$$

We obtain the final inequality above by noting that $\mathbf{y}$ must be in W_{m+1}, or equivalently, $c(\mathbf{y}) \leq \theta_{m+1}$. We can use this inequality to bound ε_k by maximizing the right-hand side with respect to the state $\mathbf{x}$ and the action i,

$$
\varepsilon_k \leq \max_{\substack{\mathbf{x}\in Z_k \\ i\in\mathcal{A}}} \Big\{\gamma \sum_{j=0}^{m+1} \sum_{\mathbf{y}\in Z_j} P(\mathbf{y}|\mathbf{x},i)\varepsilon_j + \sum_{\mathbf{y}\in Z_{m+1}} P(\mathbf{y}|\mathbf{x},i)\theta_{m+1}\Big\}
$$

Three cases for ε_k must be considered; (1) when $k < m$, (2) when $k = m$, and (3) when $k = m + 1$. We examine each of these cases below.

ε_{m+1} : Since states in Z_{m+1} are absorbing in the bounded state model, we can bound ε_{m+1} independently of the error in any other cells. The error in this region is bounded by the largest difference between the optimal value $V^*(\mathbf{x})$ and the default value v_θ,

$$
|V^*(\mathbf{x}) - v_\theta| = \max\{V^*(\mathbf{x}) - v_\theta, v_\theta - V^*(\mathbf{x})\}.
$$

We loosely bound the first term in the maximum by observing that $V^*(\mathbf{x}) \leq 0$ since there are no positive rewards, so

$$
V^*(\mathbf{x}) - v_\theta \leq \theta_{m+1}/(1-\gamma).
$$

For the second term,

$$\begin{aligned} v_\theta - V^*(\mathbf{x}) &= -\theta_{m+1}/(1-\gamma) - V^*(\mathbf{x}), \\ &\leq -\theta_{m+1}/(1-\gamma) \\ &\quad + c(\mathbf{x})/(1-\gamma) + 2T\gamma/(1-\gamma)^2, \\ &\leq 2T\gamma/(1-\gamma)^2, \end{aligned}$$

which follows because $c(\mathbf{x})$ is bounded above by θ_{m+1} when $\mathbf{x}$ is in Z_{m+1}. Since $\theta_{m+1}/(1-\gamma)$ eventually exceeds $2T\gamma/(1-\gamma)^2$ as m grows large, we use

$$\varepsilon_{m+1} \leq \theta_{m+1}/(1-\gamma).$$

$\varepsilon_k,\ k < m$: For any $k < m$, $\varepsilon_k \leq \gamma\varepsilon_{k+1}$. We demonstrate this inductively. First, assuming that $m > 1$, then there is a state $\mathbf{x}$ in Z_0 such that

$$\varepsilon_0 \leq \gamma P(Z_0|\mathbf{x}, i)\varepsilon_0 + \gamma P(Z_1|\mathbf{x}, i)\varepsilon_1,$$

Since $P(Z_1|\mathbf{x}, i) = 1 - P(Z_0|\mathbf{x}, i)$ we can rearrange terms to get

$$\varepsilon_0/\varepsilon_1 \leq \gamma \frac{1 - P(Z_0|\mathbf{x}, i)}{1 - \gamma P(Z_0|\mathbf{x}, i)} \leq \gamma,$$

or equivalently, $\varepsilon_0 \leq \gamma\varepsilon_1$. Then if $\varepsilon_{k-1} \leq \gamma\varepsilon_k$, we can employ a similar argument to show that $\varepsilon_k \leq \gamma\varepsilon_{k+1}$, since

$$\varepsilon_k \leq \gamma \sum_{j=k-1}^{k+1} P(Z_j|\mathbf{x}, i)\varepsilon_j \leq \gamma\varepsilon_k \sum_{j=k-1}^{k} P(Z_j|\mathbf{x}, i) + \gamma P(Z_{k+1}|\mathbf{x}, i)\varepsilon_{k+1}.$$

Therefore the inductive hypothesis is valid.

ε_m : We treat this case last because it relies on both ε_{m-1} and ε_{m+1}. We have that for some $\mathbf{x}$ in Z_m,

$$\varepsilon_m \leq \gamma[1 - P(Z_{m+1}|\mathbf{x}, a)]\varepsilon_m + P(Z_{m+1}|\mathbf{x}, a)[\gamma\varepsilon_{m+1} + \theta_{m+1}].$$

Since ε_{m+1} is bounded by $\theta_{m+1}/(1-\gamma)$, the term $\gamma\varepsilon_{m+1} + \theta_{m+1}$ simplifies to $\theta_{m+1}/(1-\gamma)$. In the worst case, this means that we have

$$\varepsilon_m \leq \theta_{m+1}/(1-\gamma).$$

Putting these pieces together, we have that if k is less than m, then

$$\varepsilon_k \leq \gamma^{m-k}\theta_{m+1}/(1-\gamma).$$

Therefore as θ increases (that is, as we choose larger values of m), ε_k vanishes, and the bounded model provides a consistent approximation to the value function of the original wrapped-state MDP.

3.2.2 Empirical Results

In the previous section, we showed that the bounded state model is a consistent approximator for the optimal value function of the task scheduling problem. These bounds are primarily useful as a theoretical tool for demonstrating consistency; since they apply to arbitrary policies, they are quite loose. In this section we will look empirically at bounded state model solutions empirically to get a better sense for their practical performance.

In the following discussion, we present results from different strategies for approximating the optimal task scheduling policy using the bounded model representation. This includes experiments considering the choice of cost bounds on the derived policy quality, the effect of reducing the temporal resolution of the problem, and the effect of reducing the fidelity of the utilization target (for example, by approximating a 51%/49% share between two tasks using a 50%/50% share).

We can characterize each of these settings as varying some parameter of interest β to study its impact on approximation performance. Each parameter β induces a different problem that approximates the original task scheduling problem, and each of these problems has its own optimal policy π^*_β. There are two values that we can consider when talking about π^*_β. One is its value on the approximate problem, denoted V^*_β, and the other is $V^{\pi^*_\beta}$, the value of using π^*_β on the actual task scheduling problem.

Even if some state $\mathbf{x}$ occurs in both the approximate and the original problem, $V^*_\beta(\mathbf{x})$ is not necessarily equal to $V^{\pi^*_\beta}(\mathbf{x})$.

The quantity we are interested in studying in our approximation experiments is $|V^{\pi^*_\beta}(\mathbf{0}) - V^*(\mathbf{0})|$, the difference in value of the optimal and approximate solutions at the initial state $\mathbf{0}$. One reason that we restrict our attention to just the initial state is that it appears in the original problem and every approximation we consider, so it provides a consistent basis for comparison. The other reason is that if $V^{\pi^*_\beta}$ and V^* differ at $\mathbf{x}$, then it is necessarily the case that $V^{\pi^*_\beta}(\mathbf{x}) < V^*(\mathbf{x})$, so if $\mathbf{x}$ is reachable from $\mathbf{0}$ under π^*_β, then $V^{\pi^*_\beta}(\mathbf{0}) < V^*(\mathbf{0})$, so the initial state reflects suboptimality at all of the states we care about even when $\pi^*_\beta(\mathbf{0}) = \pi^*(\mathbf{0})$.

There are a couple of issues with using the error $|V^{\pi^*_\beta}(\mathbf{0}) - V^*(\mathbf{0})|$ as our quantity of interest. The first issue is that expected rewards, and so values, can vary tremendously between problem instances, so this error has high variance. The second issue is that we do not have access to the optimal value $V^*(\mathbf{0})$ in general.

We report results averaged across many random problem instances, and the value of the initial state can vary substantially even between problem instances with the same number of tasks. The expected error calculated across problem instances tends to be dominated by the problem instances with the greatest magnitude initial state value, and is not necessarily indicative of the actual approximation quality. With this in mind, we instead report the normalized approximation error $|[V^{\pi^*_\beta}(\mathbf{0}) - V^*(\mathbf{0})]/V^*(\mathbf{0})|$, or equivalently, $|V^{\pi^*_\beta}(\mathbf{0})/V^*(\mathbf{0}) - 1|$. This measure vanishes as the approximation approaches the true optimal value, and is monotonically increasing as the approximation gets worse. It is well-defined for all of the problem instances we consider, since $V^*(\mathbf{0})$ is zero only in the degenerate case in which some task has a duration of zero[4].

A further problem is that we can not certify that any policy we consider is actually optimal, since we can not evaluate policies that may reach infinitely many states – while we suspect that no such policy can be optimal, we have not been able to demonstrate this conclusively. Therefore, we need to use some proxy in place of $V^*(\mathbf{0})$ in our error measurements. In all of the settings we consider, there is a natural choice

[4]If some degenerate task has a duration of zero, then running that task from the initial state never accrues any cost; any task with non-zero duration has positive expected cost, so the optimal policy will be to run the degenerate task repeatedly forever

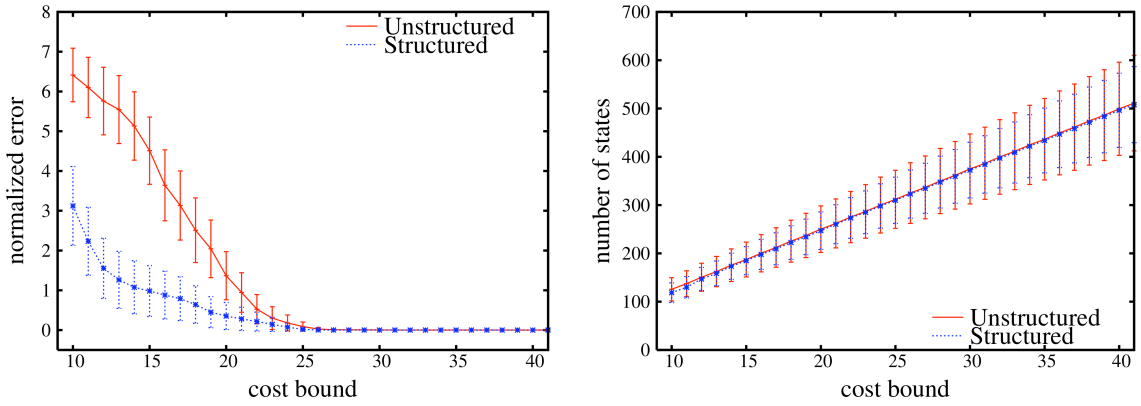
of a baseline parameter value B that serves this purpose: solutions to the largest bound model are never worse than policies derived with smaller bounds, the set of policies over coarse temporal resolution approximations is a subset of the policies for finer resolution problems, and so on, so that $V^{\pi^*_\beta} \preceq V^{\pi^*_B}$. With this in mind, we report approximation quality in terms of $|V^{\pi^*_\beta}(\mathbf{0})/V^{\pi^*_B}(\mathbf{0}) - 1|$.

Varying the cost bounds: Our first tests consider the convergence rate on random two-task problem instances. These problem instances were constructed by choosing task duration distributions and utilization targets at random. Utilization targets were selected by choosing integers q_1 and q_2 uniformly at random from between 1 and 16, inclusive, in order to establish a utilization target of $\mathbf{u} = (q_1, q_2) \cdot (q_1 + q_2)^{-1}$. The worst-case execution time for each task varied from instance to instance and was selected uniformly at random from between 8 and 32 quanta, inclusive.

One common consideration with simulated performance is whether or not results from simulated problem instances predict real-world behavior. In order to address this, we ran tests with structured and unstructured duration distributions. Unstructured distributions consisted of randomly chosen histograms, and may simulate a wide variety of phenomena, including distributions with non-convex support. Structured support is intended to model task behaviors that may be more likely in certain domains: for example, the duration of robot motor tasks is often estimated using normal distributions, as this fits well with empirical observations.

Unstructured histograms were generated at random as follows. We first select a worst-case execution time T_i for the task J_i uniformly at random between 8 and 32 time quanta. The closed interval from time 0 to T_i+1 was then discretized into T_i bins; bin t corresponds to the probability of running for t quanta, $P_i(t)$. We populated each bin by sampling uniformly at random among integers between 0 and 10, inclusive, then normalized each bin by the sum over all bins in order to obtain a discrete distribution.

Structured histograms correspond to discretized, truncated normal distributions. As above, we first select a worst-case execution time uniformly at random between 8 and 32 time quanta. Then we select the normal distribution's real-valued mean from the intervals 1 and T_a and its variance from the interval $[0, 5)$. The corresponding discrete distribution is obtained by taking the height of the normal density at the center of each bin, then normalizing by the sum over bins to get a proper discrete distribution.



(a) value comparison (b) state space size

Figure 3.5: Approximation performance for the bounded state model.

We examined the approximation performance for increasing cost bounds between $\theta = 10$ and $\theta = 100$ at unit intervals. The results of these tests are shown in Figure 3.5. We report the approximation quality in Figure 3.5(a). The approximation quality is a function of the difference between the optimal and approximate policy, and is discussed in greater detail below. Figure 3.5(b) shows the number of states in the bounded state model as we increase the bounds. We show results for cost bounds between 10 and 40 because in all of these tests the bounded state approximation converged at or before $\theta = 37$.

In Figure 3.5(b), the number of states grows linearly as the cost bounds increase. In the n-task case, the size of the bounded state space grows according to $\mathcal{O}(nf(\mathbf{u})^{n-1})$ for some utilization-dependent density constant $f(\mathbf{u})$. One way to visualize this growth is by projecting every utilization state (or equivalently, every wrapped state) onto n orthogonal, axis-aligned $(n-1)$-dimensional halfplanes that meet at the origin and extend to $+\infty$, *i.e.*, the halfplanes $p_i = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} \succeq \mathbf{0},\ x_i = 0\}$ for $i = 1, \ldots, n$. We can flatten out the union of these halfplanes to span $\mathbb{R}^{n-1}$ so that the cost of a state is just the distance to the initial state in the the flattened space. Therefore the size of wrapped state spaces with progressively larger cost bounds corresponds to selecting progressively larger balls in the flattened $(n-1)$-dimensional space; the number of states in these balls necessarily grows asymptotically on the order of $density^{n-1}$. The density of states in this space is dictated by the utilization target, becoming more dense as the period grows large.

The approximation quality for varying cost bounds is shown in Figure 3.5(a). We measure the approximation quality as the normalized error $|V^*_\theta(\mathbf{0})/V^*_{100}(\mathbf{0}) - 1|$. If the optimal policy for the model with cost bound θ never reaches an absorbing state in that model, then its value at the initial state in the bounded model is identical to its value at the initial state of the unbounded wrapped state model. In these experiments, π^*_θ stayed within the cost bounds whenever θ exceeded 36; we suspect that the optimal bounded model policy for each of these problems may be the optimal policy for the original problem, as further increasing the bounds fails to improve the policy. While this stabilization point varies from problem to problem, we conjecture that for any periodic task scheduling problem with finite worst-case execution times there is some finite bound cost model that is sufficient to represent the optimal policy at the initial state and its successors.

Structured problem instances exhibited much better approximation performance than we saw in the unstructured case. This is encouraging, since we expect that most real-world tasks will resemble more closely our structured distributions rather than the unstructured ones. We attribute this approximation performance to the likelihood of transitioning to the absorbing state as the bounds increase. Since we used discretized Gaussian distributions for these experiments, the probability of emitting larger durations gradually falls off beyond the mean. If we choose a wrapped state and look at the probability of transitioning from it to the absorbing state in bounded state models with progressively larger bounds, that probability will fall off gracefully. Under the unstructured distributions, the change in probability of reaching the absorbing state while varying cost bounds may be less smooth.

Varying the worst-case execution time (WCET): In these tests, we ran thirty trials to evaluate the effects of differing worst-case execution convergence time of bounded state approximations. In each trial we chose a mean and variance for each task; these means were real values selected uniformly at random between 1 and 32, while variances were selected similarly from between 1 and 16. We then generated 25 problem instances by truncating the distributions at $T = 8$ up to $T = 32$ in order to induce a variety of worst-case execution times.

Next, we approximated a solution to each of these problem instances by using cost bounds between 10 and 50 to determine the point of convergence. The point of

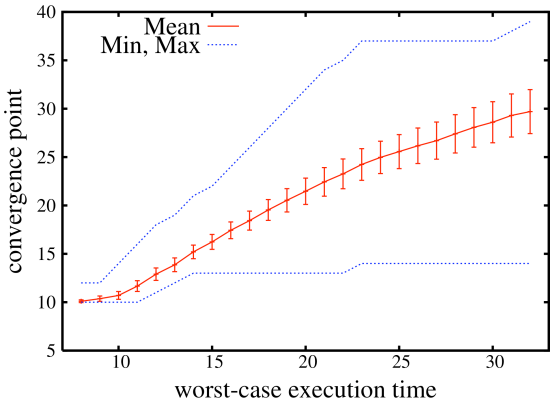


Figure 3.6: Convergence points as a function of task worst-case execution time.

convergence was determined by finding the least cost bound θ' such that for each θ between θ' and 50, $V^*_\theta(\mathbf{0}) = V^*_{50}(\mathbf{0})$. Figure 3.6 shows the results of these tests.

In Figure 3.6 we report the mean, minimum, and maximum locations of the convergence point for each worst-case execution time we considered. 95% confidence intervals about the mean are shown. We see that the convergence point grows almost linearly with worst-case execution time. The curvature in the mean curve is due to our method for choosing the distribution means; the WCET induced by truncation may differ from the actual one if a duration distribution has small enough mean and variance that the last few bins in the discretization have zero probability mass. This causes an artificial decrease in slope as truncation is irrelevant for more problems as the induced worst-case execution time increases.

Perhaps the most interesting observation is that in both sets of tests, the value function approximations appeared to converge at a finite cost bound. This is in contrast to our *a priori* performance bounds in the previous section, which guaranteed convergence to the optimal asymptotically. This suggests that for any periodic task scheduling problem with finite worst-case execution times, there may be a finite cost bound such that the bounded state model is sufficient to completely represent an optimal policy from the initial state. That is, there may be some optimal policy that only reaches finitely many states from the initial state.

This is not particularly surprising, since increasing the number of states in a model can result in a new optimal policy only when the new states themselves offer lower costs or make it easier to reach existing low cost states. We have formulated the

bounded cost model such that increasing the bounds only adds states with higher cost, but increasing the bounds would change the optimal policy only if they made it easier to reach low cost states. The limit on the difference in cost between states and their successors (Lemma 2.2.1 from Section 2.2) means that high cost states are near similarly costed states, so it seems unlikely that adding higher cost states will open up paths to beneficial states. However, we have not been able to prove that this conjecture holds overall.

Comparison to heuristic policies: Good approximations to the optimal policy are quite expensive to compute. With this in mind, we performed experiments to compare the quality of the optimal solution to a pair of heuristic policies. We looked at two policies in particular: the cost-greedy policy, and the policy that always runs the most underutilized action.

The cost-greedy policy π_g chooses actions according to

$$\pi_g(\mathbf{x}) \in \operatorname*{argmin}_{i \in \mathcal{A}} \left\{ \sum_{t=1}^{\infty} P_i(t) c(\mathbf{x} + t\Delta_i) \right\}, \tag{3.8}$$

with ties broken uniformly at random. This policy just chooses the action with the best expected next-state cost. We denote by π_u the policy that always executes the most underutilized action. This utilization policy is expressed as

$$\pi_u(\mathbf{x}) \in \operatorname*{argmax}_{i \in \mathcal{A}} \{\tau(\mathbf{x}) u_i - x_i\}; \tag{3.9}$$

with ties broken uniformly at random. In the two-task case, the decision boundary for the greedy policy is parallel to the target utilization ray, while the decision boundary for π_u is the utilization ray itself.

We compared the quality of these policies to the optimal bounded model policy for varying cost bounds using the same methodology as above. We considered two- and three-task problem instances with discretized normal duration distributions with means between 1 and 32, variances between 1 and 16, and worst-case execution times between 8 and 32, selected uniformly at random. The utilization targets for each problem instance were selected by choosing integers $\mathbf{q} \in [1, 16]^n$ uniformly at random, so that the utilization target is $\mathbf{u} = \mathbf{q} \cdot (\sum_i q_i)^{-1}$. We computed the optimal bounded

policy using bounds ranging from $\theta = 10$ to $\theta = 50$. We repeated this procedure 50 times for both the two- and three-task case.

Figure 3.7 shows the results for the two-task comparison. We measure the approximation quality in terms of the normalized approximation error $|V^{\pi}(\mathbf{0})/V^{*}_{50}(\mathbf{0}) - 1|$, where $V^{\pi}(\mathbf{0})$ can be the value of one of the heuristic policies or the value of the optimal bounded model policy for varying cost-bounds. We verified that neither the greedy policy nor the utilization policy ever had value greater than $V^{*}_{50}(\mathbf{0})$ at the initial state, so the approximation error does indeed measure how much worse these heuristic policies perform relative to the best bounded model policy considered. We report the mean and 95% confidence intervals over fifty trials.

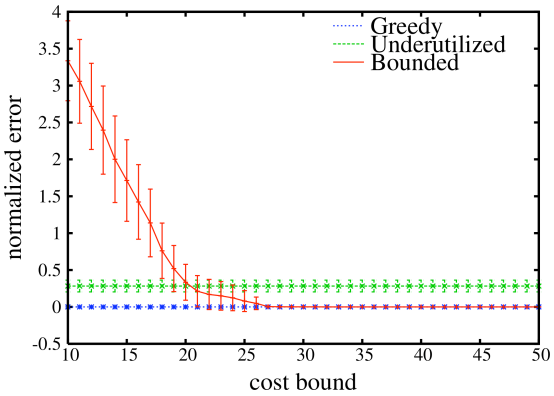
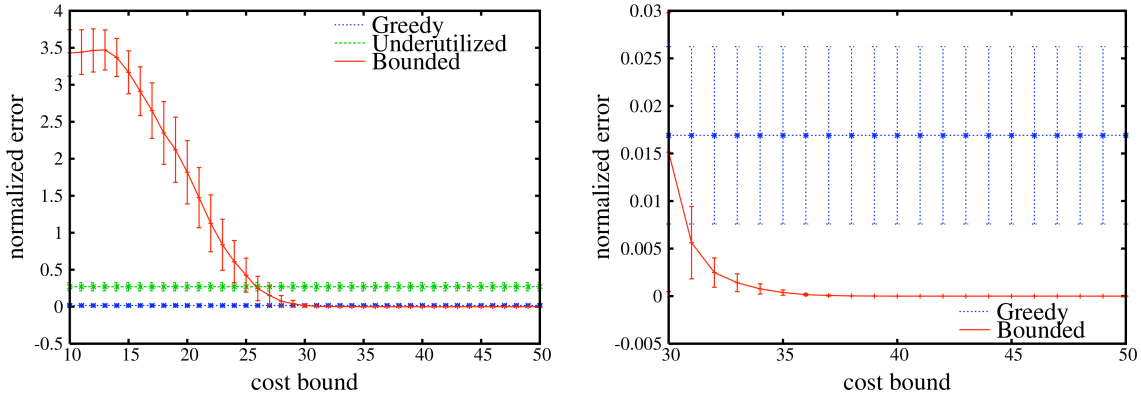


Figure 3.7: Comparison of the optimal bounded policy value to two heuristic policies on two-task problems.

The heuristic policies do not depend on the cost bound. To evaluate these policies we first computed their closure, *i.e*, the set of states reachable from the initial state. This allows us to compute exactly the value of each of these policies at the initial state. We see that the greedy policy has significantly higher value (lower error) than the utilization policy. More importantly, the greedy policy and the optimal bounded policy are identical for sufficiently large bounds. While we have not been able to prove that the greedy policy is optimal for the two-task case, empirically these policies have been identical in every problem instance that we have considered.

Optimality of the greedy policy clearly does not hold in general; moving to the three task case illustrates this quite clearly. Figure 3.8 shows the results of our experiments in this setting. In Figure 3.8(a) we see that once again the utilization policy is

significantly inferior to the greedy policy. At this scale, the greedy policy and the optimal bounded policy are quite similar. Figure 3.8(b) provides a closer comparison of the greedy and bounded optimal policies for larger bounds. Here we can see that the bounded policy is significantly better, although they are similar enough to suggest that the greedy policy is a good proxy for the optimal policy. We do, however, expect that the difference in value between the greedy and optimal policies will grow with the number of tasks.



(a) full results (b) detail

Figure 3.8: Comparison of the optimal bounded policy value to two heuristic policies on three-task problems.

Figure 3.9 plots the average number of states as a function of the cost bound in the two- and three-task cases. Notice that the vertical axis is in log scale; the three-task results reflect our expectation that the number of states grows quadratically in the cost bounds, while the number of states grows linearly in the cost bounds.

In this and in the previous experiments, we considered the performance of heuristic policies or bounded model policies as approximations to the optimal task scheduling policies. These experiments consider different models of the same task scheduling problem. In the next set of experiments, we consider approximating the original task scheduling problem with a similar task scheduling problem with a simpler utilization state model. This consists of reducing the temporal resolution when modeling task durations. This reduces the number of successors any state can have, and corresponds to aggregating temporally similar states.

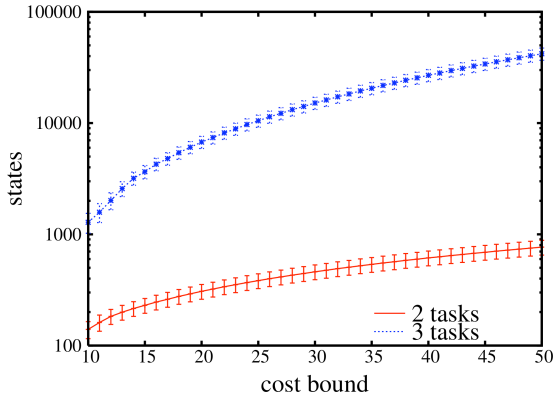


Figure 3.9: The number of bounded model states for increasing cost bounds.

Varying the temporal resolution: Above, we considered the effect of approximating one problem with another using a more easily-represented utilization target. Another factor was not considered in much depth is the temporal resolution of a problem. For example, if we have extremely large durations in tens of thousands of quanta, using a single quantum time resolution may be inappropriate, since we would require a large number of states to capture the entire distribution of successor states of any given state. In these situations, it may be appropriate to aggregate time quanta into a different time scale.

We tested this conjecture in order to evaluate empirically the performance ramifications of using a problem with coarse time resolution as an approximation to a one with finer resolution. In these tests, we constructed fifty two-task scheduling problem instances using randomized duration distributions and utilization targets. Utilization targets were selected uniformly at random by choosing integers in $\mathbf{q} \in [1, 16]^n$ to establish a utilization target of $\mathbf{u} = \mathbf{q} \cdot (\sum_i q_i)^{-1}$.

Duration distributions were constructed with a worst-case execution time of $T = 256$. We repeated these tests with unstructured histograms selected uniformly at random, and using structured discretized normal distributions. These were constructed following the procedures described in previous experiments above; normal distribution parameters were selected uniformly at random, with means between 1 and 256 and variance between 1 and 128.

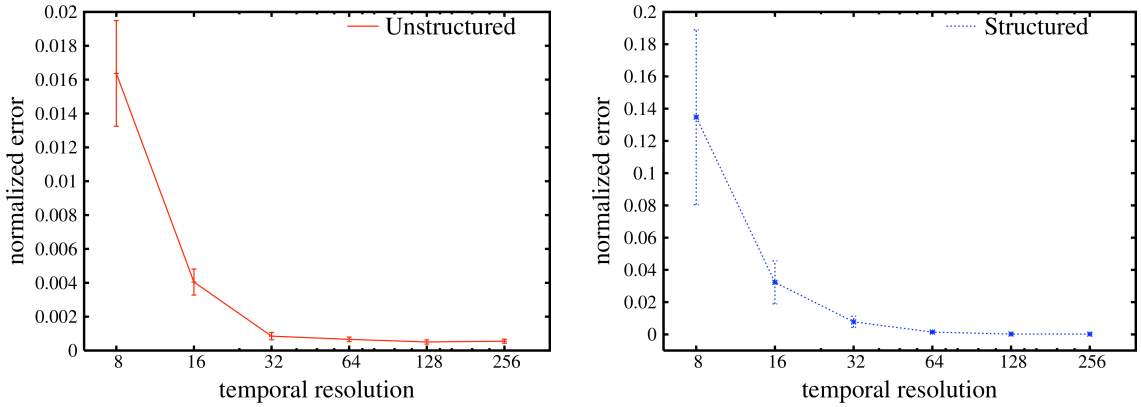
In each trial we considered a single problem instance. For each task, we took the duration distribution over the interval $[0, 256]$ and rescaled it to the interval $[0, \rho]$ for

each resolution ρ in $8, 16, 32, 64$ and 128. The probability of running for t quanta in the resolution $\rho = 2^k$ problem was the probability of running between $2^{8-k}t$ and $2^{8-k}(t+7-k)$ quanta (inclusive) in the original problem. For example, the probability for running t quanta in the 128-resolution approximation was just the probability of running for $2t$ or $2t+1$ quanta in the original, maximum resolution problem. Moving from a resolution of 2^k down to $2^{k'}$ corresponds to aggregating groups of $k - k' + 1$ adjacent utilization states. For example, the 128-resolution state $(0, 0)$ corresponds to the set of states $\{(0, 0), (0, 1), (1, 0), (1, 1)\}$ of the 256-resolution problem.

We then performed 5000 Monte-Carlo trials, following the greedy policy for the low-resolution problem on the original problem for 2000 steps. We used this to estimate the value of that policy on the original problem, since in our experience the greedy and optimal policies agree on two-task problems. This was accomplished by mapping each utilization state $\mathbf{x}$ we encountered in the original problem down to the corresponding low-resolution state $\mathbf{x}'$, looking up the greedy action at $\mathbf{x}'$, then applying that action at $\mathbf{x}$. We report the results of these experiments in Figure 3.10; Figure 3.10(a) shows the approximation performance as a function of resolution for unstructured random histograms, and Figure 3.10(b) presents our results on discretized normal duration distributions.

As in our previous experiments, we report approximation quality after normalizing the results with respect to the optimal value. Let $V^\rho(\mathbf{0})$ be the initial-state value of the ρ-resolution greedy policy on the original 256-resolution problem, and let $\hat{V}^\rho(\mathbf{0})$ be its approximation. We report the approximation quality as $|\hat{V}^\rho(\mathbf{0})/V^{256}(\mathbf{0}) - 1|$, using the true value of the greedy policy for the maximum resolution problem as our basis for comparison. This error measure vanishes as $\hat{V}^\rho(\mathbf{0})$ approaches $V^{256}(\mathbf{0})$. In both sets of experiments, it appears that we can decrease the temporal resolution substantially while incurring only a small loss in policy performance. This is significant, since cutting the temporal resolution in half in an n-task problem results in a reduction in the number of states that is exponential in the number of tasks.

It is worth noting that the unstructured problem instances appear to endure changes in temporal resolution more gracefully than the structured instances. The difference in approximation error between the highest and lowest resolution problem instances is an order of magnitude smaller under unstructured problem instances.



(a) random durations

(b) gaussian durations

Figure 3.10: Approximation quality using reduced temporal resolution. Note the order of magnitude difference in scale.

3.3 Automatic State Space Construction

In Section 3.2 we described the bounded state model, an MDP formulation of the task scheduling problem restricted to a cost-bounded subset of states from the wrapped state model of Section 3.1. The transition and reward structure of this model is identical to that of the wrapped state model in any state with sufficiently low cost, while the remaining states are abstractly represented with a single absorbing state. By choosing appropriately large costs for the absorbing state, we can guarantee that the optimal policy for the bounded model never reaches states that exceed the cost threshold. This allows us to guarantee that these solutions are a good solution to the original task scheduling problem.

Our empirical results with the bounded model approach suggest that there may be a finite cost bound that suffices to capture an optimal policy near the initial state. In order to benefit from using the bounded model solution (as opposed to the cheaper-to-compute greedy policy) we need to select the bound carefully: if it is too small, the resulting policy performs worse than the greedy policy, while if the bounds are too large the model includes many irrelevant states.

One straightforward way to address these concerns is to integrate modeling and solving bounded models into a meta-algorithm that iteratively increases the cost bounds.

This meta-algorithm terminates whenever it appears that the optimal policy has stabilized. This approach has several drawbacks: It does not carry information between iterations, many states in the bounded model may be irrelevant, and we can only identify policy stabilization heuristically.

Most of the structure is identical in two bounded models with similar cost bounds. If $\mathbf{x}$ is a state with much lower cost than either models' bound, then the transition system and expected reward at that state is unchanged between the two models. The more transitions it would take to escape these bounds from $\mathbf{x}$, the more similar its value is in either model. This suggests that the optimal policy and value for one model would be a good initial approximation for the other. The meta-algorithm framework described above does not explicitly take this into account; doing so may save a substantial amount of time and effort.

The second drawback is that cost bounded models may include many irrelevant states. Since the wrapped state space is countably infinite, we have restricted the set of states we consider by identifying a distinguished initial state $\mathbf{x} = \mathbf{0}$. Costs increase with distance from $\mathbf{0}$ in the wrapped state model, penalizing policies that visit states that are arbitrarily far away. However, even if we have a policy that never visits states with cost greater than the bound θ, it may be the case that it visits far fewer states than the corresponding bounded model contains. For example, it is often the case that one component of the state ever grows large under good scheduling policies; choosing bounds large enough to accommodate such states forces us to include states that allow any component to grow similarly large, whether or not the policy ever reaches these states. If we can identify these states (or more accurately, if we can enumerate only the necessary states) we may significantly reduce the cost of solving the model.

Figure 3.11 provides an example of this phenomenon. We found a bounded model solution for a two-task problem with utilization target $\mathbf{u} = (13/20, 7/20)$ and discretized, truncated normal durations with worst-case execution times of 17. We then looked at the set of states reachable from the initial state $\mathbf{x} = \mathbf{0}$ under that policy; these states are shown in blue in the figure. The largest cost among these states was slightly more than 20; the sixty-three red states are included in the tightest bounded model that contains all of the blue states. It appears that the number of unreachable

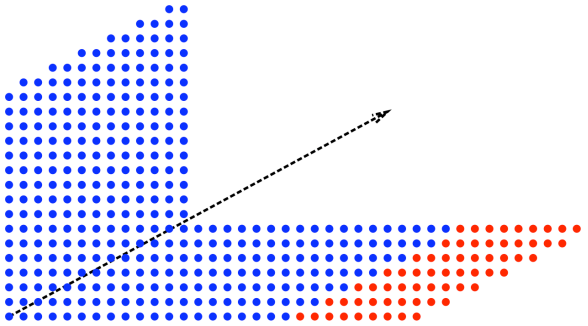


Figure 3.11: An optimal bounded model policy may only visit a small subset of model states. The optimal bounded model may reach any of the blue states from the initial state $\mathbf{x} = \mathbf{0}$, but will never reach the red states.

states in a tight model grows as the utilization target becomes more skewed towards a single task, and also increases with the number of tasks as well.

Finally, we would prefer to address termination in a more principled, less heuristic fashion. Our empirical results (for example, those in Figures 3.5 and 3.6 in Section 3.2.2), suggest we may be justified in terminating whenever the value does not vary between subsequent iterations, since we have seen no evidence that subsequent iterations can improve the policy after that point. However, in general this may not be the case, and $V^*_\theta(\mathbf{0}) = V^*_{\theta+1}(\mathbf{0})$ does not necessarily mean that we have found the optimal value at $\mathbf{0}$; it just means that there is no policy within the bounds $(\theta + 1)$ that is better than the θ-bounded best policy. While it seems unlikely that increasing the bounds further would yield a better policy, given the speed limit on transitions and the fact that we are only adding higher cost states to the model, we do not yet have any guarantee that this is true.

In this section we discuss our Expanding State Policy Iteration (ESPI) algorithm [36], which addresses these concerns by interleaving state space enumeration with policy evaluation and improvement in order to compactly represent the states necessary for finding good policies. ESPI takes as input a collection of initial states and a default policy. It proceedings iteratively, exploring and enumerating the initial state set to include states necessary for evaluating and improving its current policy. We construct an MDP model over the expanded state space, and then perform one or more rounds of policy iteration in order to obtain a new policy. The algorithm terminates when no improvements to its current policy are found. ESPI can be viewed as a way of

incorporating reachability analysis [16] into the bounded model generation, since we explicitly restrict attention to states that might be encountered given an intermediate policy that we want to evaluate.

Policies produced by ESPI will never be worse than the default policy, although it may terminate with a suboptimal policy. We extend ESPI by developing a family of related algorithms that vary the size of the enumerated state spaces and the number of rounds of policy iteration applied to each intermediate model. These extensions are consistent in the sense that as both of these parameters grow large, the solutions converge on the optimal wrapped state solution. First we describe the basic version of this algorithm.

3.3.1 The ESPI Algorithm

While we designed the ESPI algorithm specifically with the task scheduling problem structure in mind, we will first describe the algorithm for general, countable state Markov decision processes. Let $M = (\mathcal{X}, \mathcal{A}, R, P)$ be an MDP. Then ESPI takes a distinguished collection of initial states $X_I \subseteq \mathcal{X}$ and a default policy π_I. We then iterate over three steps: *model expansion*, *policy evaluation*, and *policy improvement*. The model expansion step is the most involved, in that it constructs an MDP over a subset of the wrapped state space. Policy evaluation and improvement are straightforward analogs to their Policy Iteration equivalents (see, for example, Section 6.4 of Puterman's book [72] or our brief description in Section 2.1), except that they are restricted to the models constructed in the model expansion step.

- *Model Expansion* explores a subset of states that are necessary and sufficient to evaluate and improve ESPI's current policy. This is performed by constructing an *evaluation envelope* consisting of the states reachable from X_I under the current policy, then constructing an *improvement envelope* about the closure to accommodate the successor states of actions besides the policy actions. The evaluation envelope is constructed in such a way that we can compute the policy value exactly at each state it contains, while the improvement envelope allows us to accurately evaluate the utility of modifying the policy at states in the evaluation envelope. Finally, we construct an MDP model M_d over the

improvement envelope using techniques similar to those employed in Section 3.2 for constructing cost-bounded models.

- *Policy Evaluation* consists of computing the value of the current policy in the context of the intermediate model M_d. We design this process so that the value of the current policy applied to M_d is identical to its value over the closure in the original problem.

- *Policy Improvement* uses the improvement envelope to accurately assess the utility of actions different from those suggested by the current policy.

Model Expansion At each iteration d, the model expansion step consists of exploring the set of states reachable from the initial state set X_I under the intermediate policy π_d. We begin by constructing the evaluation envelope X_d, which is the *closure* of π_d about the initial states X_I. The closure of an arbitrary policy π on a set of state X, denoted $\mathcal{C}_\pi X$, is just the set of states that can be reached from $X \subseteq \mathcal{X}$ while following π in any number of steps. We will define this more precisely in terms of a policy expansion operator. The expansion of policy π on X, denoted $\mathcal{E}$, is defined as

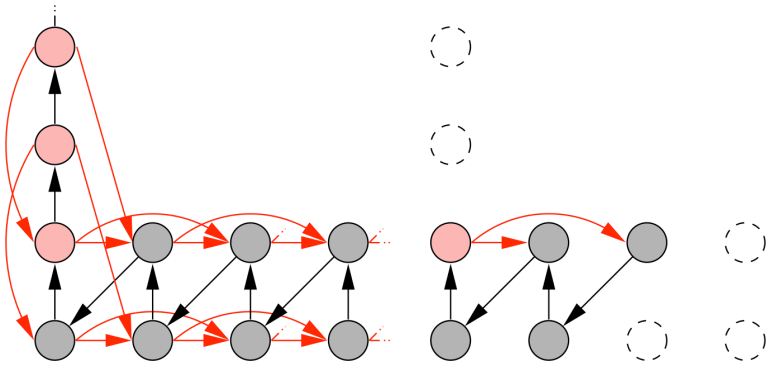
$$\mathcal{E}_\pi X = \{y \in \mathcal{X} \mid \exists x \in X,\ P(y|x, \pi(x)) > 0\}. \tag{3.10}$$

We use this expansion operator to define the k-step closure of a policy. The k-step closure of a policy π on X, denoted $\mathcal{C}^k_\pi X$, is the set of states that can be reached in k or fewer transitions from X. We define this recursively,

$$\begin{aligned} \mathcal{C}^1_\pi X &= X \cup \mathcal{E}_\pi X \\ \mathcal{C}^k_\pi X &= \mathcal{C}^{k-1}_\pi(\mathcal{C}^1_\pi X). \end{aligned} \tag{3.11}$$

The one step closure of X is just X and its expansion, while the k-step closure consists of X and the results of expanding it k or fewer times. If we let $P^{(j)}(y|x, \pi)$ denote the probability of reaching y from x in exactly j transitions under policy π and define $P^{(0)}(x|x, \pi) = 1$, then it is equivalent to define the k-step closure as

$$\mathcal{C}^k_\pi X = \bigcup_{j=0}^{k} \{y \in \mathcal{X} \mid \exists x \in X,\ P^{(j)}(y|x, \pi) > 0\}.$$



(a) Greedy policy π (b) Policy closure $\mathcal{C}_\pi\{\mathbf{0}\}$

Figure 3.12: The greedy policy closure.

We define the closure $\mathcal{C}_\pi X$ of π on X as the limit of the k-step closure as k grows large:

$$\mathcal{C}_\pi X = \lim_{k\to\infty} \mathcal{C}_\pi^k X. \tag{3.12}$$

Figure 3.12 illustrates the closure of the greedy policy at the initial state of our running example from Figure 3.1. The greedy policy is shown in Figure 3.12(a): Action 0 is selected in the grey states and transitions along the black edges, while Action 1 is chosen in the red states and follows the red edges. In Figure 3.12(b) we have removed all of the states that are unreachable from the initial state $\mathbf{x} = \mathbf{0}$ (bottom left) when following the greedy policy.

For convenience, we also define the unqualified expansion operator $\mathcal{E}$ and k-step closure operators $\mathcal{C}^k$. These are equivalent to the corresponding policy operators using policies that assign non-zero probability to taking every action in each state. Thus, $\mathcal{E}X$ is the set of states that can be reached in one transition from X under any action, and $\mathcal{C}^k X$ is the set of states that can be reached in k or fewer transitions from X under any possible policy. The ∞-step closure $\mathcal{C}X$ is the set of all states that are reachable from X.

The policy closure $\mathcal{C}_\pi X$ is exactly the set of states that are both necessary and sufficient to evaluate π at every state in the closure. Consider the policy value recurrence from Equation 2.2,

$$(\Gamma_\pi V^\pi)(x) = \sum_{y\in\mathcal{X}} P(y|x,\pi(x))[R(x,\pi(x),y) + \gamma V^\pi(y)].$$

Evaluation Envelope By construction, $\mathcal{C}_\pi X$ contains all and only the states necessary to evaluate this recurrence at each state it contains: if x is in the closure, it is in X or it is reachable from X under π; if x is not in the closure, then it is unreachable from X. Since $\mathcal{C}_\pi X$ allows us to evaluate the policy backup operator, we refer to it as the *evaluation envelope*.

In order to improve a policy π at a state x, we need to evaluate the action backup operator (Equation 2.13)

$$(\Gamma_a V^\pi)(x) = \sum_{y \in \mathcal{X}} P(y|x,a)[R(x,a,y) + \gamma V^\pi(y)].$$

for each action a. In general, the evaluation envelope is not guaranteed to include the states necessary to perform this backup at every state it contains. For example, in Figure 3.12, we would not be able to evaluate $\Gamma_0 V^\pi$ exactly at the initial state because the successor state $(2, 0)$ is not part of the closure.

Improvement Envelope In order to correctly perform policy improvement, we need to be able to evaluate $(\Gamma_a V)(x)$ at each state x in the evaluation envelope. To accomplish this, we expand the evaluation envelope to accommodate the successors of arbitrary actions by taking the 1-step (unqualified) closure of $\mathcal{C}_\pi X$,

$$\mathcal{C}^1(\mathcal{C}_\pi X) = \mathcal{E}(\mathcal{C}_\pi X) \cup \mathcal{C}_\pi X$$

This set contains all of the successors of states in $\mathcal{C}_\pi X$, so it is possible to perform action backups for these states as long as we already know V^π at each successor state. However, $\mathcal{C}^1(\mathcal{C}_\pi X)$ in general does not contain enough information to evaluate V^π. In order to capture this information, we construct the *improvement envelope* $\mathcal{C}_\pi(\mathcal{C}^1(\mathcal{C}_\pi X))$.

Figure 3.13 shows the improvement envelope for the greedy policy in our running example. We have also included the transitions that necessitate adding each of the states shown.

Policy Evaluation At iteration d of ESPI we construct the evaluation envelope $X_d = \mathcal{C}_{\pi_d} X_I$, then expand it to obtain the improvement envelope $Y_d = \mathcal{C}_{\pi_d}(\mathcal{C}^1 X_d)$. Next, we need to compute the policy value V^{π_d} at each state in the improvement

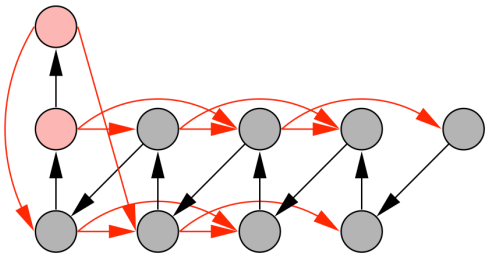


Figure 3.13: The improvement envelope for the greedy policy in Figure 3.12.

envelope. For example, this can be approximated using successive approximations to compute $\Gamma^k_{\pi_d} V$ for suitably large k, or computed exactly by directly solving the system $V = \Gamma_{\pi_d} V$. This step differs from the policy evaluation step of policy iteration only in that we restrict computations to just the states in the improvement envelope.

Policy Improvement Similarly, policy improvement is a direct analog to its policy iteration equivalent. This consists of evaluating the action backup operator at just the states in the evaluation envelope X_d. We only perform policy improvement over the evaluation envelope because we are not guaranteed to have all of the successors of states in Y_d/X_d. (Later we extend ESPI to permit improvements at every state in the improvement envelope as well) This gives us the policy π_{d+1} for the next iteration, with

$$\pi_{d+1}(x) \in \operatorname*{argmax}_{a \in \mathcal{A}} \{(\Gamma_a V^{\pi_d})(x)\}$$

for each state x in X_d. ESPI terminates when π_d and π_{d+1} agree at every state in the evaluation envelope.

When ESPI terminates, we only need to retain the final evaluation envelope; the states in Y_d/X_d are not reachable from X_I under the final policy. In fact, if π_I can be implicitly defined over all of $\mathcal{X}$, then we only need to explicitly store π_d for states in X_d where π_d and π_I differ. For example, we can define the task scheduling greedy or utilization policies without explicitly storing a lookup table mapping states to actions; the greedy policy and our bounded state approximations tend to agree in the majority of states, so only storing states where they differ can result in substantial savings in storage when we store or communicate the policy.

On general Markov decision processes we can not guarantee that the final policy π_d is optimal, since we may never consider a policy that reaches a high reward state

if it can only be reached by moving through states with very low reward. There is no apparent reason to alter the policy if it increases the likelihood of incurring large penalties, so ESPI may never expand its improvement envelope enough to enclose the high reward state. Therefore, ESPI is most useful for applications like our task scheduling problem, where costs increase monotonically away from some desired state or trajectory.

Algorithm 1 combines the steps described above to provide a pseudocode description of the ESPI algorithm.

Algorithm 1 `ESPI`(X_I, π_I)

1: $\pi_0 := \pi_I$
2: **for** $d = 0, 1, 2, \ldots$ **do**
3: $\quad X_d := \mathcal{C}_{\pi_d} X_I$
4: $\quad Y_d := \mathcal{C}_{\pi_d} \mathcal{C}^1 X_d$
5: $\quad$ Solve $V_d = \Gamma_{\pi_d} V_d$ over Y_d.
6: $\quad \forall x \in \mathcal{X}, \quad \pi_{d+1}(x) = \begin{cases} \operatorname{argmax}_{a \in \mathcal{A}} \{(\Gamma_a V_d)(x)\} & x \in X_d \\ \pi_I & otherwise \end{cases}$
7: $\quad$ **if** $\pi_{d+1} = \pi_d$ over X_d, **then return**
8: **end for**

When ESPI is used on any finite MDP, it is guaranteed to terminate. The evaluation and improvement envelopes can at most contain the entire finite state space. Further, we can employ the same argument used to demonstrate the correctness of policy iteration in finite MDPs here: π_{d+1} differs from π_d only if there is some state x in the evaluation envelope where $V^{\pi_{d+1}}(x) \geq V^{\pi_d}(x)$, so the policies we consider are monotonically increasing in value. There are only finitely many policies we can consider on any finite MDP, so ESPI must terminate.

In general countably infinite state spaces ESPI may not terminate. Trivially, if the default policy does not have finite closure, then we can not compute its evaluation or improvement envelopes. Even if the default policy does have finite closure we still can run into trouble. For example, suppose we have an MDP over the positive integers and two deterministic actions, $\{0, 1\}$, and action $a = 0$ is a no-op, *i.e* it leaves the process in the same state, while $a = 1$ increments the state. The default policy $\pi_I = 0$ has finite closure for any finite initial state set X_I, but if $R(x, a, y) = r(y)$

is monotonically increasing in y, then ESPI will not terminate. For example, if $X_I = \{0\}$, then $X_d = \{0, \ldots, d\}$.

This underscores the importance of our choice of initial policy. For any finite subset of states X, $\mathcal{C}_{\pi_I} X$ must be finite. This is sufficient to guarantee that each evaluation envelope X_d and improvement envelope Y_d is finite, since π_d and π_I can differ on at most finitely many states. However, our example above shows that even then ESPI may not terminate; ESPI will terminate only when it is either impossible to leave the current evaluation envelope or when there is no benefit to doing so, so termination depends on the application domain.

3.3.2 Algorithm Analysis

Since ESPI's termination is application-dependent, we now consider its specific application to the wrapped state model of the task scheduling problem. ESPI is intended to exploit the structure of the task scheduling problem cost function in order to find good policies efficiently. In particular, high cost states have high cost successors and predecessors; similarly, low cost states are near other low cost states (see Lemma 2.2.1). Since there are only finitely many low cost states, it seems likely that good finite-state policies should exist. We will demonstrate that there is at least one policy that has finite closure.

Notice that ESPI can only terminate when every intermediate policy π_d has a corresponding finite improvement envelope Y_d. The only way that a task scheduling policy can have an infinite improvement envelope is if it has non-zero probability of reaching states with arbitrarily cost. It seems unlikely that such a policy would be optimal, but a number of factors make obtaining a formal proof of this result difficult. In practice, ESPI terminates on the wrapped state model of task scheduling problems given an appropriate default policy. For the remainder of this section we consider the conditions necessary to select a good default policy, and propose a class of policies with guaranteed finite improvement envelopes.

When the closure $\mathcal{C}_\pi X$ of *any* finite state set X under π is always finite, we say that π has *finite closure*. We have identified a class of policies that have finite closure on

the task scheduling problem. These policies are *eventually non-expansive* in that they are guaranteed to reduce cost in any state with sufficiently high cost. We formalize this concept below.

We say that a policy π is *non-expansive* if and only if it never increases costs; that is, for any state $\mathbf{x}$, $c(\mathbf{y}) \leq c(\mathbf{x})$ whenever $P(\mathbf{y}|\mathbf{x}, \pi(\mathbf{x})) > 0$. Since there are only finitely many states that satisfy any given cost bound in the task scheduling problem, any non-expanding policy has finite closure. However, as long as every task has non-zero probability of running at least one quanta when dispatched, none of these policies exist: every possible successor of the initial state has positive cost.

An *eventually non-expansive* policy π is one that is non-expansive in states with sufficiently high cost – that is, there is some cost bound θ_π such that π will never transition the system into a higher cost state from any state $\mathbf{x}$ with cost $c(\mathbf{x}) \geq \theta_\pi$. These policies have finite closure on the task scheduling problem with L_p costs, which we formalize in the following lemma.

Lemma 3.3.1. *If π is an eventually non-expansive task scheduling policy with cost bound θ_π under cost function c_p, then for any finite set of wrapped states X, $\mathcal{C}_\pi X$ is finite.*

Proof. Let X be a finite collection of wrapped states and π be an eventually non-expansive policy with cost bound θ_π. Let c_{max} be the maximum cost among the states in X. We can use θ_π to partition X into X_- and X_+, so that states in X_- have cost less than θ_π, while states in X_+ have cost at least θ_π.

Notice that $\mathcal{E}_\pi X = (\mathcal{E}_\pi X_-) \cup (\mathcal{E}_\pi X_+)$, which implies that $\mathcal{C}_\pi^k X = (\mathcal{C}_\pi^k X_-) \cup (\mathcal{C}_\pi^k X_+)$ as well. We are not guaranteed that π may increase costs at states in X_-, so the largest cost among states in $\mathcal{E}_\pi X_-$ can exceed c_{max}. According to Lemma 2.2.1, the difference between a state's cost and its successor's cost is at most $2T$, so the largest cost state in $\mathcal{E}_\pi X_-$ is less than $\theta_\pi + 2T$. Since π never increases costs at states in X_+, the largest cost in $\mathcal{E}_\pi X_+$ is at most c_{max}.

It follows that the largest cost in $\mathcal{C}_\pi^1 X$ is less than the larger of c_{max} and $\theta_\pi + 2T$. We can apply this same argument repeatedly to show that $\mathcal{C}_\pi^k X$ has cost bounded by

$\max\{c_{max}, \theta_\pi + 2T\}$ as well, so the closure $\mathcal{C}_\pi X$ has bounded cost. By Lemma 3.2.1 only finite state sets have bounded cost, so π has finite closure. □

We can generalize this result to any MDP in which (1) there are at most finitely many states with cost less than any finite bound, and (2) if y is a successor state of x, then the difference in reward or cost between x and y is bounded by a global constant.

Significantly, if we have an eventually non-expansive policy π and another policy π' that differs from π in only finitely many states, then π' is also eventually non-expansive. If $c_{max} \geq \theta_\pi$ is the largest cost state where π and π' disagree, then π' is eventually non-expansive with cost bound c_{max}; if c_{max} is less than θ_π, then π' is eventually non-expansive with cost bound θ_π. Suppose that the policy π_d produced by ESPI is eventually non-expansive. π_{d+1} and π_d can differ only within the finite evaluation envelope of π_d, so π_{d+1} is also eventually non-expansive. Therefore, if π_I is eventually non-expansive, so is every intermediate policy produced by ESPI, so all of the closures we need to compute are finite.

The cost-greedy policy

$$\pi_g(\mathbf{x}) = \operatorname*{argmin}_{i \in \mathcal{A}} \mathbb{E}_{t \sim P_i}\{c(\mathbf{x} + t\Delta_i)\}$$

(an equivalent definition is provided in Equation 3.8) is not necessarily eventually non-expansive, because in problems with three or more tasks, the greedy policy may dispatch a task that increases the cost with low probability as long as it reduces cost in expectation, even if some action that strictly reduces cost is available. Whether or not the greedy policy has finite closure is an open question; anecdotally, in thousands of problem instances we have not found any examples in which the greedy policy did not have finite closure.

The utilization policy

$$\pi_u(\mathbf{x}) = \operatorname*{argmax}_{i \in \mathcal{A}}\{\tau(\mathbf{x})u_i - x_i\}$$

is eventually non-expansive. For convenience, we will demonstrate that this is true for L_1-costs, but the result can be extended to other order L_p norms.

Lemma 3.3.2. *The scheduling policy that always dispatches the most underutilized task, π_u, is eventually non-expansive.*

Proof. We will show that π_u decreases costs in any state $\mathbf{x}$ with $c_1(\mathbf{x})$ greater than $2T(n-1)$, where n is the number of tasks and T is the worst-case execution time among all tasks. The cost of a state is the sum of component costs for each task; the component cost of a task J_i in L_1-norm is $|\tau(\mathbf{x})u_i - x_i|$. Let $\mathcal{O}(\mathbf{x})$ be the set of overutilized tasks at $\mathbf{x}$,

$$\mathcal{O}(\mathbf{x}) = \{i \in \mathcal{A} \mid x_i > \tau(\mathbf{x})u_i\}.$$

The set of underutilized tasks is the complement of $\mathcal{O}(\mathbf{x})$ in $\mathcal{A}$. The sum of component costs among overutilized tasks equals that among underutilized costs, since

$$\begin{aligned}
\sum_{i\in\mathcal{O}(\mathbf{x})} (x_i - \tau(\mathbf{x})u_i) &= \Big(\sum_{i\in\mathcal{O}(\mathbf{x})} x_i\Big) - \tau(\mathbf{x}) \sum_{i\in\mathcal{O}(\mathbf{x})} u_i \\
&= \tau(\mathbf{x}) - \Big(\sum_{i\in\mathcal{U}(\mathbf{x})} x_i\Big) - \tau(\mathbf{x})\Big(1 - \sum_{i\in\mathcal{U}(\mathbf{x})} u_i\Big) \\
&= \sum_{i\in\mathcal{U}(\mathbf{x})} (\tau(\mathbf{x})u_i - x_i).
\end{aligned}$$

The underutilized and overutilized tasks each account for half of the cost $c_1(\mathbf{x})$, since the underutilized and overutilized tasks partition $\mathcal{A}$. We can write

$$c_1(\mathbf{x}) = 2 \sum_{i\in\mathcal{O}(\mathbf{x})} (x_i - \tau(\mathbf{x})u_i) = 2 \sum_{i\in\mathcal{U}(\mathbf{x})} (\tau(\mathbf{x})u_i - x_i).$$

Let $\mathbf{x}$ be a state with $c_1(\mathbf{x}) \geq 2T(n-1)$, and suppose $\pi_u(\mathbf{x}) = a$. Then we have

$$\sum_{i\in\mathcal{U}(\mathbf{x})} (\tau(\mathbf{x})u_i - x_i) \geq T(n-1),$$

and the most underutilized task J_a has component cost at least T (that is, $\tau(\mathbf{x})u_a - x_a \geq T$) since there are at most $(n-1)$ underutilized tasks and the cost may be distributed evenly among them.

Consider how dispatching the most underutilized task J_a affects the cost. Let $\mathbf{y} = \mathbf{x} + t\Delta_a$ be the successor state after J_a runs for duration t. By construction, J_a

remains underutilized at $\mathbf{y}$, since

$$\begin{aligned}\tau(\mathbf{y})u_a - y_a &= \tau(\mathbf{x})u_a + tu_a - x_a - t \\ &\geq T - t(1 - u_a) \\ &> 0;\end{aligned}$$

this last step follows because $t(1 - u_a) < T$. Then $\mathcal{O}(\mathbf{y}) \subseteq \mathcal{O}(\mathbf{x})$, since the share of every other task is decreasing relative to J_a's share. Meanwhile, if some task J_j is overutilized in both $\mathbf{x}$ and $\mathbf{y}$, then its component cost decreases when J_a is run, since

$$y_j - \tau(\mathbf{y})u_j = x_j - \tau(\mathbf{x})u_j - tu_j > 0$$

so $\mathbf{y}$'s cost can not exceed $\mathbf{x}$'s. Therefore, π_u is eventually non-expansive with cost bound $2T(n-1)$. □

Taken together, Lemmas 3.3.1 and 3.3.2 are sufficient to guarantee that each iteration of ESPI using the utilization policy and initial states $X_I = \{\mathbf{0}\}$ will terminate. We have not been able to prove that ESPI terminates on every task scheduling problem instance using this configuration, but it does seem likely. Loosely speaking, policies that stay within a low cost bound seem likely to be better than those that may reach higher cost states, since any trajectory that reaches high cost states must travel through a sequence of states with similarly high cost. Therefore, we conjecture that there exist optimal policies with finite closure; if so, some finite variant of ESPI must terminate with this policy.

3.3.3 Algorithm Extensions

The formulation of ESPI discussed above is guaranteed to find locally optimal scheduling policies, since π_d is guaranteed to have better value than π_{d-1} whenever they differ. However, the kinds of improvements that can be made are limited by the fact that we only look one step ahead when constructing an improvement envelope for a policy. We also would like to exploit the states enumerated during model expansion better, both by considering policy improvements over the entire improvement envelope and

by finding the best policy that stays within that envelope before repeating model expansion.

This suggests a pair of straightforward and orthogonal extensions to ESPI; we can adjust the amount of lookahead by expanding improvement envelopes that admit longer trajectories that deviate from the policy under consideration, and we can perform multiple rounds of policy evaluation and improvement once we have computed this envelope. This leads to a family of algorithms, ESPI(k,l); the parameter k adjusts the amount of lookahead applied when constructing improvement envelopes, while the parameter l tunes the degree to which each improvement envelope is exploited. ESPI(1,1) is most similar to the basic algorithm described in Algorithm 1, except that ESPI(1,1) attempts to improve the policy over all of the states in Y_d.

Adjustable Lookahead: Basic ESPI may terminate with a locally optimal policy because of its limited lookahead capabilities. Consider the improvement envelope $Y = \mathcal{C}_\pi(\mathcal{C}^1 X)$ of the policy π about its evaluation envelope X. While we can evaluate π exactly at each state in Y, we can only perform exact policy improvements at states in X. If there are high reward states in $\mathcal{X}/Y$ that are successors to states in $\mathcal{C}^1 X$, we may never amend π to reach them if their predecessors have high cost.

A simple solution is to increase the amount of lookahead involved when we compute the improvement envelope. In general, replacing the one-step closure $\mathcal{C}^1 X$ with the k-step closure $\mathcal{C}^k X$ will allow us to consider $(k-1)$-step improvements to π, since then the improvement envelope $Y = \mathcal{C}_\pi(\mathcal{C}^k X)$ contains all of the successor states to each state in $\mathcal{C}^{k-1} X$.

To exploit these larger closures fully, it is appropriate (and often necessary) to perform more than a single round of policy improvement over this envelope, since a single round of improvement still constrains us to relatively myopic improvements that can not break us out of local optima.

Envelope Exploitation: We can further exploit the envelopes constructed by ESPI as well. In principle, we can perform multiple policy improvements and evaluations over the improvement envelope Y_d at iteration d. However, we must make these improvements in a way that avoids policies that can exit Y_d, since otherwise the sequence of policies considered may no longer be of monotonically increasing quality.

At iteration d of ESPI(k,l), the algorithm expands the improvement envelope Y_d. We can define an MDP model over this state space, $M_d = (Y_d \cup \{z_d\}, \mathcal{A}, R_d, P_d)$, by restricting the wrapped state model to just the states in Y_d and aggregating the states in $\mathcal{W}/Y_d$ into a single absorbing state z_d. For states $\mathbf{x}$ and $\mathbf{y}$ in $Y_d \cup \{z_d\}$ and actions i in $\mathcal{A}$, we define

$$R_d(\mathbf{x}, i, \mathbf{y}) = \begin{cases} -c(\mathbf{y}) & \mathbf{y} \in Y_d \\ -c(z_d) & \mathbf{y} = z_d \end{cases}$$

and

$$P_d(\mathbf{y}|\mathbf{x}, i) = \begin{cases} P_w(\mathbf{y}|\mathbf{x}, i) & \mathbf{y} \in Y_d \\ \sum_{\mathbf{z} \in \mathcal{X}/W_d} P_w(\mathbf{z}|\mathbf{x}, i) & \mathbf{y} = z_d \end{cases}$$

where $c(z_d)$ is the absorbing state cost. If we choose $c(z_d)$ appropriately, any policy for M_d that is better than ESPI's current policy π_d will never reach the absorbing state. For example, we can accomplish this by setting $c(z_d)$ greater than $c_{max} = \max\{c(\mathbf{x}) \mid \mathbf{x} \in Y_d\}$: since π_d stays within Y_d by construction, V^{π_d} at any state in Y_d is at least $-c_{max}/(1-\gamma)$, while the value of the absorbing state is exactly $-c(z_d)/(1-\gamma)$. Since policy improvement never decreases the value of the policies it produces, it will never consider a policy that reaches the absorbing state.

Basic ESPI performs a single round of policy evaluation and improvement over a subset of states in each model M_d. The straightforward extension is to perform multiple rounds of evaluation and improvement over the entire model. As the number of rounds grows large, this is equivalent to computing the optimal policy of M_d. By choosing the absorbing state cost appropriately, the optimal solution π_d^* to M_d is the highest value policy with closure $\mathcal{C}_{\pi_d^*} X_I$ contained in Y_d.

Putting these results together we get the family of algorithms ESPI(k,l), which performs l rounds of policy improvement on improvement envelopes that admit k step trajectories from the closure of each intermediate policy. Practically, the most interesting instances of the extended algorithm are ESPI(k,1) and ESPI(k,∞). ESPI(k,1) has the least cost per iteration, but is more likely to either settle on lower quality policies or to take more iterations to find a good policy. ESPI(k,∞) maximally exploits each improvement envelope. In fact, since this corresponds to exactly solving each model M_d, we can discard policy iteration for more efficient solution methods like Modified Policy Iteration [73, 72].

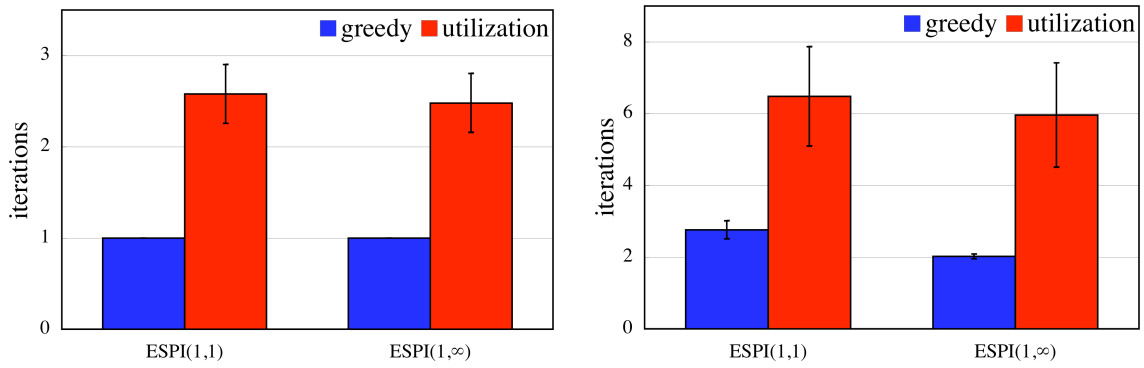
In the next section, we study the behavior of multiple variants of ESPI on simulated problem instances. We focus on the impact of varying the solution depth, looking at the difference between performing a single round of policy improvement at each iteration compared to exactly solving each intermediate model. We also compare the performance of different default policies, and notice that although we were unable to prove that the greedy policy (Equation 3.8) has finite closure, it appears to have finite closure in practice, and ESPI using this policy finds good policies more quickly than the utilization policy (Equation 3.9).

3.3.4 Empirical Results

In this section, we examine ESPI's practical performance on simulated problem instances. We address two considerations: we provide evidence that ESPI does terminate in practice on task scheduling problems, and we demonstrate that the policies found by ESPI are qualitatively identical to the corresponding best bounded model solutions – that is, they have the same value.

We divide our discussion into two parts. First we compare the policy approximation performance for different choices of default policies and solution depths. We consider initializing ESPI with the utilization policy $\pi_u(\mathbf{x}) = \text{argmax}_i\{\tau(\mathbf{x})u_i - x_i\}$, which we demonstrated has finite closure in the proof of Lemma 3.3.2. We also consider using the greedy policy $\pi_g(\mathbf{x}) = \text{argmin}_i\mathbb{E}_{t\sim P_i}\{c(\mathbf{x} + t\Delta_i)\}$ as the default policy. After examining the performance several ESPI variants, we compare this performance to that of the bounded model solutions obtained in Section 3.2.2.

Comparison of ESPI variants: In these experiments, we evaluated the quality of intermediate policies produced at each iteration of ESPI in order to test the rate of convergence. We considered four ESPI variants; these consist of pairing either the utilization policy π_u or the greedy policy π_g as the default policy with either a single round of policy evaluation and improvement or by solving each intermediate model completely; *i.e.*, by using either ESPI(1,1) or ESPI(1,∞). We did not consider varying the amount of lookahead because in practice a single step was sufficient to discover policies that were qualitatively equivalent to those found by solving the largest bounded approximations.

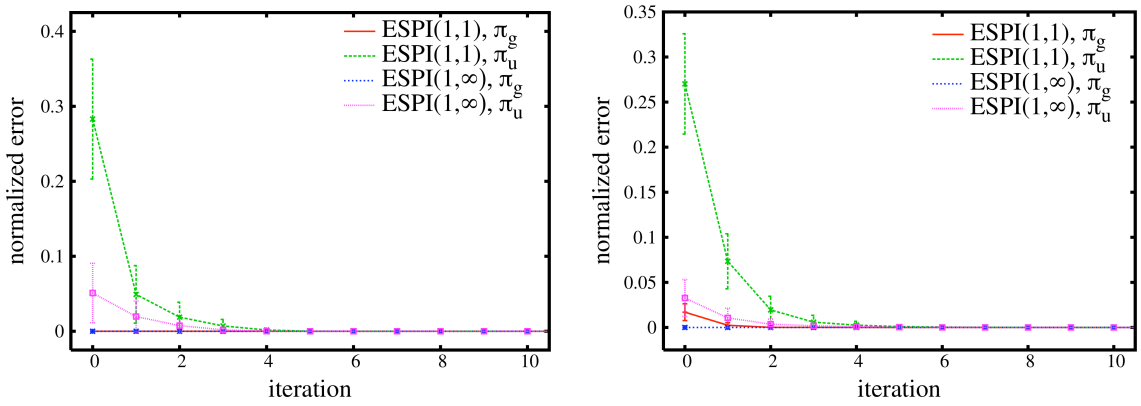


(a) two tasks (b) three tasks

Figure 3.14: Number of iterations for several ESPI variants.

Our evaluation set consisted of the same problem instances used to compare bounded model solution performance to that of heuristic policies. We restate our methodology for generating these problem instances here. We considered problem instances with two or three tasks. Task durations were distributed according to discretized, truncated normal distributions. These distributions had means selected uniformly at random between 1 and 32, variances between 1 and 16, and worst-case execution times between 8 and 32. We chose utilization targets for tasks by selecting $\mathbf{q} \in [1, 16]^n$ uniformly at random, then setting $\mathbf{u} = \mathbf{q} \cdot (\sum_i q_i)^{-1}$. This data set consisted of one hundred problem instances: fifty for the two task case, and fifty for the three task case.

At iteration d, ESPI produces an intermediate policy π_d over the improvement envelope Y_d, which contains the evaluation envelope X_d. ESPI terminates when $\pi_d = \pi_{d-1}$. We ran each variant of ESPI to completion on each problem instance. In Figure 3.14, we compare the number of iterations each variant performed prior to convergence. All variants of ESPI terminated after finitely many iterations on all of the problem instances we considered. Figure 3.14(a) displays the mean number of iterations needed to solve our two-task problem instances; 95% confidence intervals are shown. When defaulting to the greedy policy, ESPI(1,1) and ESPI(1,∞) terminated after a single iteration; the greedy policy is at least locally optimal on all of these problem instances, and seems likely to be the optimal policy. This is consistent with our observations using bounded model approach, as we were unable to find solutions to two-task problems that improved on the greedy policy regardless of the bounds.



(a) two tasks (b) three tasks

Figure 3.15: Value approximation performance over ESPI iterations.

In both two- and three task problem instances we required more iterations to converge under the utilization policy than the greedy policy. Figure 3.14(b) shows our results for the three task setting. As in the two task case, the choice of default policy had much greater impact than the solution depth on the number of iterations needed to find a solution. ESPI(1,∞) has a slight edge over ESPI(1,1) when greedy policies are used.

Each of the four ESPI variants considered reached the same solution, qualitatively, as the best bounded models on every problem instance we considered. Figure 3.15 shows the rate of convergence for each ESPI variant. This is measured using the normalized approximation error $|V_d/V_D - 1|$, where V_d is the value computed at ESPI iteration d, and V_D is the value at termination. If ESPI terminates prior to iteration d, we use the final value V_D to compute performance statistics. In ESPI(1,1), $V_d(\mathbf{0}) = V^{\pi_d}(\mathbf{0})$, while in ESPI(1,$\infty$), $V_d(\mathbf{0})$ is the optimal value of the initial state for the MDP restricted to the improvement envelope Y_d.

In both Figures 3.15(a) and 3.15(b), ESPI(1,1) with the utilization policy converged the most slowly, while pairing ESPI(1,∞) with the greedy policy produces a near-optimal policy after a single iteration. A good choice of default policy appears to be a much more important factor in determining performance than the solution depth, as variants using the greedy policy converged more quickly.

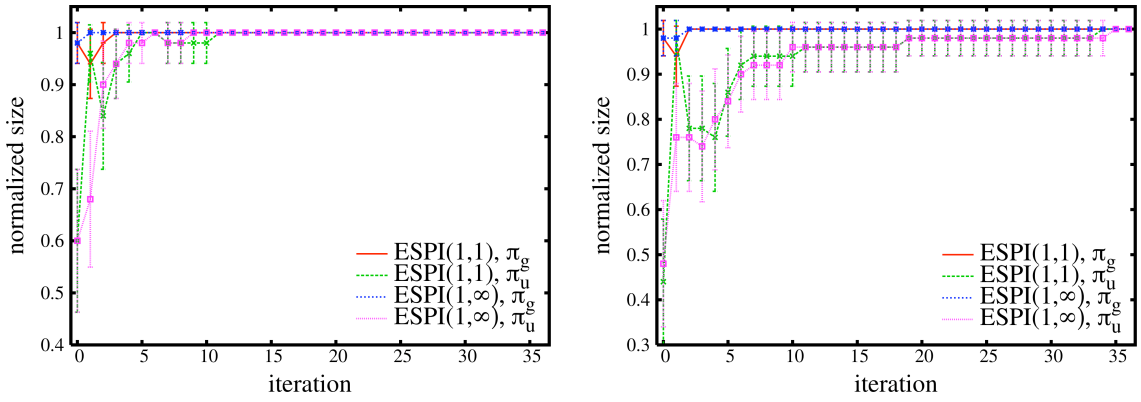
The size of evaluation and improvement envelopes remained constant in each two-task problem instance regardless of the choice of default policy or solution depth.

In these problems, the evaluation envelopes had a mean size (with 95% confidence intervals) of 248($\pm$41.5) states, while the improvement envelopes had a mean size of 496($\pm$83.1) states – each improvement envelope tended to be about twice as large as the corresponding evaluation envelope. This can be attributed to the binary choice between two actions in two-task problems.

In two dimensions, we can specify a class of policies by setting a base state $\mathbf{x} = (x_1, x_2) \in \mathbb{Z}^2$ subject to the constraint that at least one of x_1 or x_2 is zero. We define the decision boundary of the resulting policy as the line through $\mathbf{x}$ parallel to the utilization ray. Use $\pi_{\mathbf{x}}$ to denote such a policy. Then for any such choice of $\mathbf{x}$ and $\mathbf{y}$, $|\mathcal{C}_{\pi_{\mathbf{x}}}\{\mathbf{x}\}| = |\mathcal{C}_{\pi_{\mathbf{y}}}\{\mathbf{y}\}|$; this may be a bit easier to visualize in the utilization state space than in the wrapped model, since each of these policies uses the same strategy to track a different ray through space. The closures have the same size in the wrapped model since these policies are periodic.

The utilization policy falls into this class by setting the initial state as the base state; its decision boundary is exactly the utilization ray. The greedy policy is a member of this class as well, with a base point $\mathbf{x}$ such that $x_1 \leq T_2$ and $x_2 \leq T_1$ – we know that it can not be outside these bounds because in the region between the utilization ray and the decision boundary, the greedy policy must choose an overutilized task. Outside of these bounds, running an overutilized task can only increase cost while running an underutilized one strictly decreases cost, so the greedy policy must choose the underutilized task. The proximity of the greedy base state to the initial state means that the greedy policy closure about its base point necessarily contains the initial state, so the closure about the initial state is identical to its closure about the base point, and therefore the utilization and greedy policy closures about the initial state have the same size. A similar argument leads to the same conclusion regarding the improvement envelopes of these policies in two-task domains.

Envelope sizes for three-task instances do vary between iterations. In Figure 3.16, we report statistics of these envelope sizes. Figure 3.16(a) shows the mean normalized evaluation envelope size as a percentage of the final envelope size, and Figure 3.16(b) similarly shows the improvement envelope size. We report normalized envelope size rather than absolute envelope size because all ESPI variants end up with the same evaluation envelope at termination, since they eventually settle on the same policy.



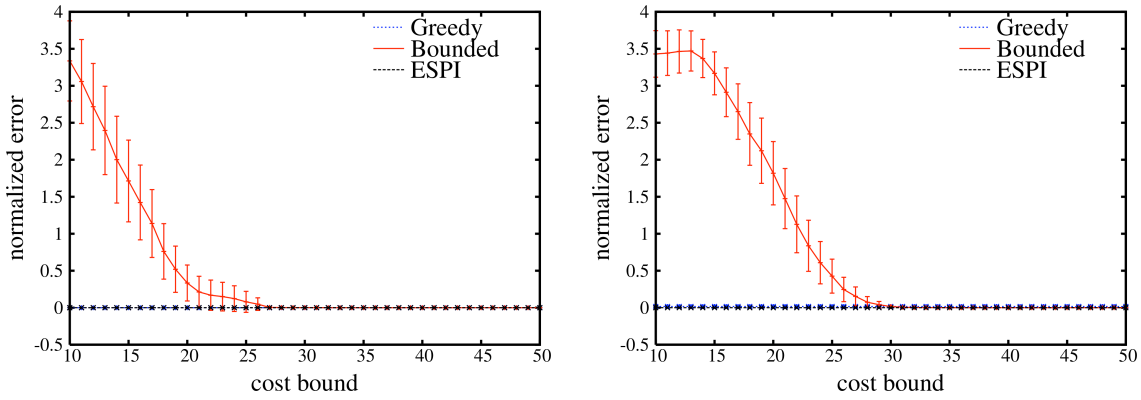
(a) evaluation envelope (b) improvement envelope

Figure 3.16: Normalized envelope sizes for three-task problems over ESPI iterations.

There are some minor differences in the size of improvement envelopes that amounts to about 3% of the mean envelope size due to variations in policy inside the improvement envelope but outside the evaluation envelope of the final policy, and so do not impact the quality of the eventual solution. The mean evaluation envelope size at termination (with 95% confidence intervals) is 7952($\pm$1521.9) states, while the mean improvement envelope size is 25939($\pm$4822) states, averaged across all variants. Here the improvement envelope size is slightly larger than thrice the size of the evaluation envelope.

Notice that the graph of approximation performance at each iteration in Figure 3.15(b) flattens out after less than ten iterations, and the evaluation envelope size stabilizes a few iterations later. This suggests that most of the work of finding a good policy is complete at that point. The improvement envelope size when defaulting to the utilization policy does not stabilize until around iteration 35, suggesting that the remaining work performed by ESPI involves certifying that the policy can not be improved by some combination of modifications beyond the policy closure, then propagating that difference in value back.

Comparison to previous techniques: In the previous set of experiments, we examined the sequence of policies considered by ESPI. We found that the algorithm converged even if we initialized it with the greedy policy; in fact, we saw notably better performance when using the greedy policy instead of the the utilization policy. We also saw that in three-task problems, ESPI(1,∞) converges in fewer iterations



(a) two tasks (b) three tasks

Figure 3.17: Comparison of ESPI to bounded model solutions for varying bounds.

than does ESPI(1,1) (note that ESPI(1,∞) does perform more computation at each iteration, since it solves each intermediate model). With this in mind, we elected to compare the performance of ESPI(1,∞) to the our bounded model approximation strategy from Section 3.2.

Our previous results suggest that ESPI produces solutions that are qualitatively identical to those produced by the best bounded models. We perform a side-by-side comparison of ESPI to the greedy policy and the bounded model solutions for varying bounds in Figure 3.17. These results are based on the same problem set as described in the previous experiments, and show the normalized approximation error $|\hat{V}(\mathbf{0})/V(\mathbf{0}) - 1|$ averaged across all of the problem instances, where V is the final ESPI value and $\hat{V}$ is the value of the greedy policy V^{π_g} (blue dotted line), the optimal bounded model solution V_θ^* (red solid line), or the value found by ESPI at termination V (black dashed line).

Figure 3.17(a) shows the performance on two-task problems as we discussed above, both the greedy and ESPI solutions are identical to the largest bounded model solutions. Figure 3.17(b) shows the performance on three-task problems; the greedy policy is worse than the ESPI solution, while the bounded model solution and ESPI solution eventually agree. This is made clear in the detailed plot shown in Figure 3.18.

Figure 3.19 compares the size of ESPI's improvement envelope to the number of states in bounded models with varying bounds. From Figure 3.17, we can see that the

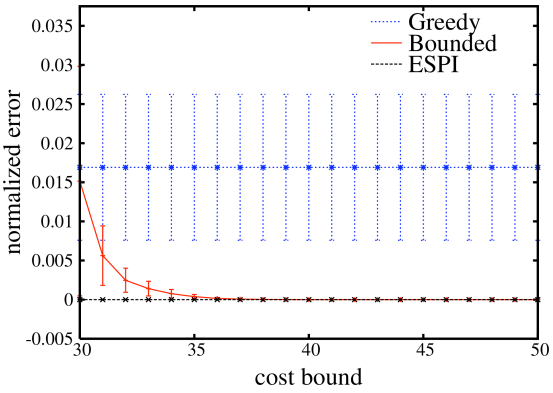
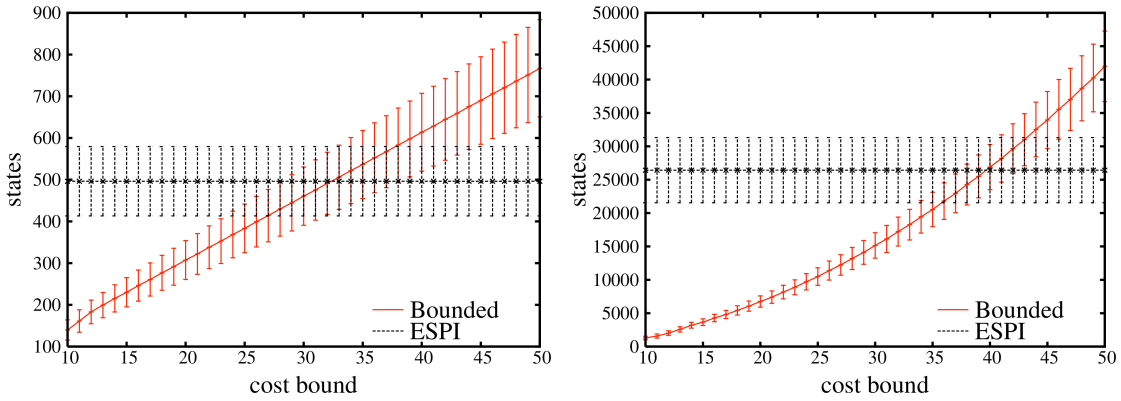


Figure 3.18: Detail of the three task ESPI approximation performance comparison from Figure 3.17(b).



(a) two tasks (b) three tasks

Figure 3.19: Comparison of ESPI improvement envelope size to bounded model state space size.

smallest bounded model sufficient to attain the best policies appear around iteration 27 for two-task problem instances and iteration 32 for instances with three tasks. ESPI's improvement envelopes tend to be larger than the smallest good bounded models. This is not too surprising. The bounded model solutions and ESPI's solutions are each guaranteed to be optimal in a restricted sense: if the bounded model policy has closure that lies inside its bounds, then it is the best such policy; the ESPI solution is the optimal policy over the improvement envelope, which includes one step of lookahead. While some particular, small bound may be sufficient to represent a good policy, we would need a larger model to certify its quality to the extent provided by ESPI, especially if the bounded model bounds tightly contain its policy's closure.

3.4 Comparison to Earliest-Deadline-First Scheduling

Our experiments in Sections 3.2.2 and 3.2.2 evaluated the performance of MDP solution methods relative to other MDP-based techniques, but to this point we have not addressed the performance of our methods in a real-time context. In Chapter 1 Section 1.2 we discussed the earliest-deadline-first (EDF) real-time scheduling algorithm. This algorithm prioritizes available jobs according to the earliest deadline among them. EDF is known to be optimal in the sense that it meets all deadlines whenever possible in preemptive [58] and work-conserving, non-preemptive settings [43]. However, it is generally considered to perform poorly under overload conditions when deadlines are likely to be missed. In these situations, tasks may be denied the shared resource for arbitrary lengths of time without regard to their relative importance or criticality.

In this set of experiments, we compared the performance of ESPI and heuristic MDP scheduling policies to EDF scheduling. Of course, our representation does not include deadlines, so it is necessary to introduce them into our experimental setup in order to compare these policies. Recall from Chapter 1 that our system consists of tasks J_i that produce an infinite sequence of jobs $J_{i,j}$, such that job $J_{i,(j+1)}$ becomes available immediately upon completion of $J_{i,j}$ (Since only one job of any task is available at any time, our discussion thus far has been able to neglect this detail). Job $J_{i,0}$ of each task J_i is available upon system initialization. Deadlines are specified relative to arrival times; each task J_i has an associated deadline d_i, so that if its job $J_{i,j}$ arrives at time τ, then it has deadline $d_{i,j} = \tau + d_i$. In our experiments, we chose task deadlines proportional to their worst-case execution times, with $d_i = \rho T_i$ for some *deadline ratio* $\rho > 0$. As ρ approaches zero, deadlines become progressively harder to meet, while larger deadline ratios result in deadlines that are easier to meet. We vary the deadline ratio in our experiments in order to study its impact on the likelihood of missing deadlines under each scheduling policy.

The *deadline miss rate*, or just *miss rate*, is the number of missed deadlines normalized by the number of jobs released. Lu *et al* have argued that minimizing the miss rate is crucial for achieving good real-time performance [60, 61]. It is important to note that

EDF tries to achieve a miss rate of zero, without regard to fairness, while our MDP-based techniques are concerned with fairness but do not explicitly consider deadlines. However, since fairness is a mechanism for enforcing timeliness [1, 12], we do expect that our methods will not perform poorly under the miss rate metric.

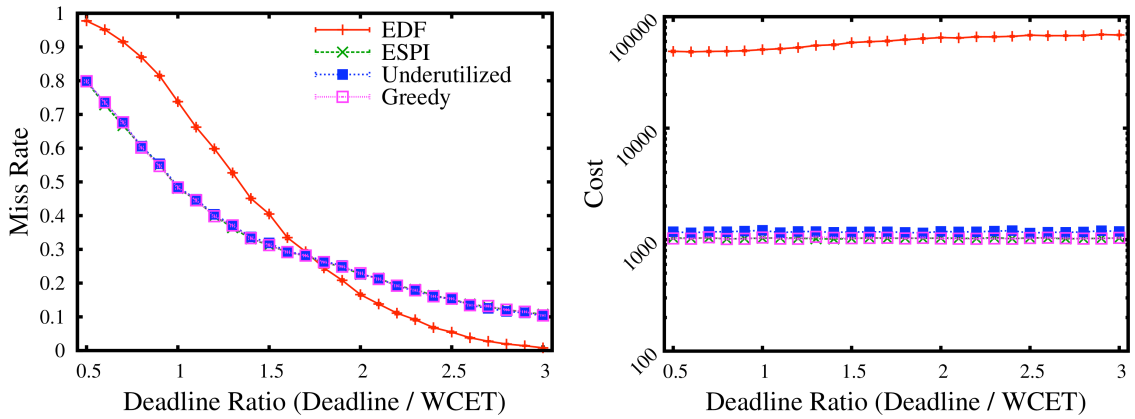
We evaluated scheduling policies on thirty three-task problem instances. Tasks had durations distributed according to uniform random histograms with worst-case execution times selected uniformly at random from the interval [2,32]. Under this scenario, expected task durations tend to be about half of their worst-case execution times. Utilization targets defined according to $\mathbf{u} = \mathbf{q} \cdot (q_1 + q_2 + q_3)^{-1}$ with $\mathbf{q}$ drawn uniformly at random from the box $[1, 32]^3$.

The four scheduling policies considered were (1) an approximation to the optimal policy obtained using ESPI$(1, \infty)$, (2) the cost-greedy policy (Equation 3.8), (3) the policy of running the most underutilized task (Equation 3.9), and (4) EDF. We evaluated the quality of each policy by simulating thirty trajectories of 1,000 decision epochs from the initial state $\mathbf{x} = \mathbf{0}$ on each problem. We report the miss rate and cumulative discounted cost (*i.e.*, negative value) observed along these trajectories.

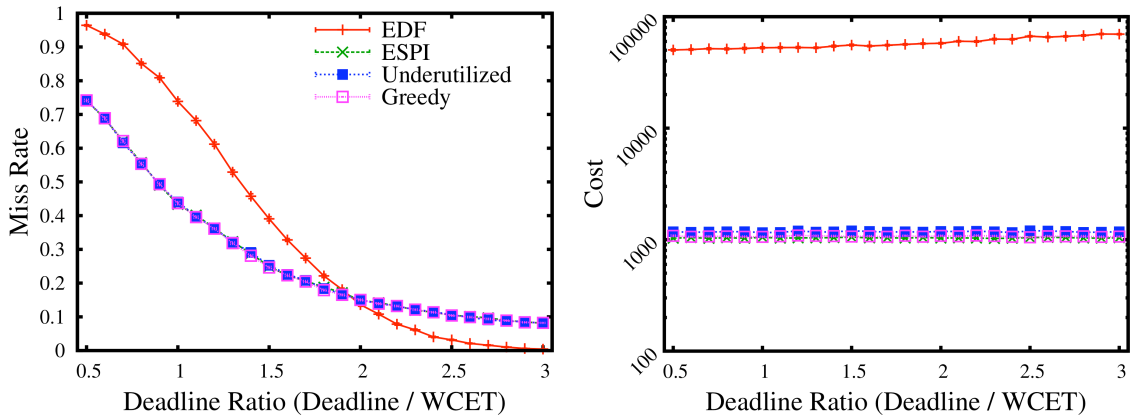
Figures 3.20 and 3.21 illustrate the results of these experiments for four representative problem instances (95% confidence intervals on miss rate and cost are shown, and are quite tight). The vector of (unnormalized) utilization targets $\mathbf{q}$, worst-case execution times $\mathbf{T}$, and mean durations μ are provided. The left-hand column of figures shows the miss rate as a function of the deadline ratio. The right-hand column of figures displays the long-term, discounted cost (negative value) of each scheduling policy. Notice that costs are shown in logarithmic scale, as EDF (which, we emphasize, does not consider these costs in its scheduling decisions) incurs much higher costs than our MDP-based policies.

Figure 3.20 shows two problem instances in which the MDP techniques achieve much better miss rate performance when overload conditions are likely, suggesting that the MDP-based techniques are most appropriate in these situations. Interestingly, there is little difference in the miss rate performance among the MDP-based scheduling policies. This suggests that the computationally cheap heuristic policies may be an appropriate surrogate for the more expensive finite-state approximation methods like

ESPI when reducing the overhead of scheduling is critical, or when there are too many tasks to feasibly enumerate a sufficient subset of states.



(a) $\mathbf{q} = (15, 32, 4)$, $\mathbf{T} = (15, 31, 21)$, $\mu = (6.4, 15.7, 10.3)$.

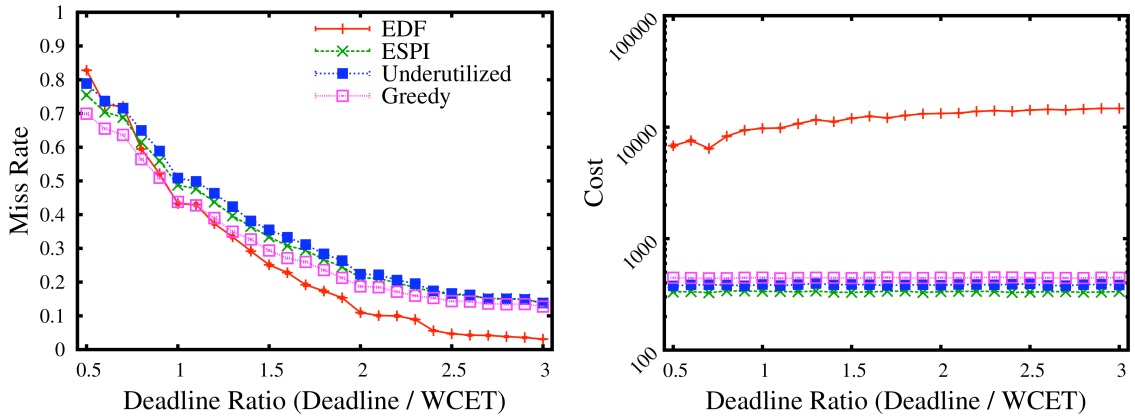


(b) $\mathbf{q} = (27, 3, 14)$, $\mathbf{T} = (20, 15, 24)$, $\mu = (9.3, 6.8, 11.3)$

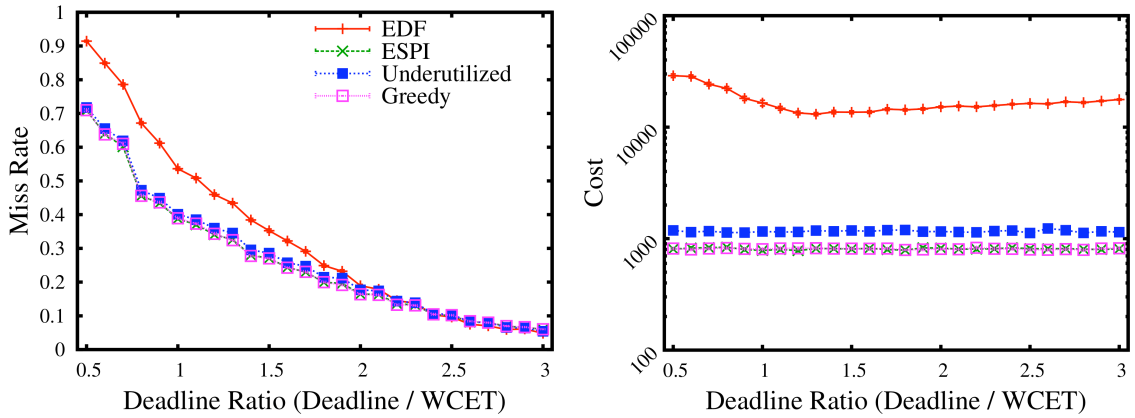
Figure 3.20: Timeliness comparison for two three-task problem instances.

Figure 3.21 presents results from two problem instances in which performance is a bit more skewed over the range of deadline ratios considered. In Figure 3.21(a), the difference in performance among the scheduling policies is less pronounced for small deadline ratios than in the previous examples. In this problem, EDF dominates the MDP-based policies even for relatively tight deadlines. In Figure 3.21(b), the MDP-based techniques outperform EDF until deadlines become fairly loose, at which point all of the policies achieve similar miss rates.

Figure 3.22 aggregates data across the thirty problem instances (including the individual instances above; 95% confidence intervals are shown) and affirms that, in



(a) $\mathbf{q} = (24, 7, 28)$, $\mathbf{T} = (14, 3, 7)$, $\mu = (5.8, 1, 3.6)$.



(b) $\mathbf{q} = (23, 22, 7)$, $\mathbf{T} = (5, 13, 22)$, $\mu = (2.6, 5.5, 10.8)$

Figure 3.21: Timeliness comparison for two additional three-task problem instances.

general, MDP-based techniques are able to achieve a better deadline miss rate than EDF when overload situations are likely. These results also support our claim that each of the MDP-based scheduling techniques achieves similar miss rate performance.

One question left open is how the choice of utilization targets affects miss rate performance, as one may consider assigning these parameters in order to optimize for real-time performance, rather than focusing on using these parameters to enforce a desired share. If we want to minimize the miss rate of one task in particular, it seems clear that pushing its utilization target closer to one will cause it to be dispatched more often, and so decrease its miss rate. We have not performed tests to evaluate this hypothesis.

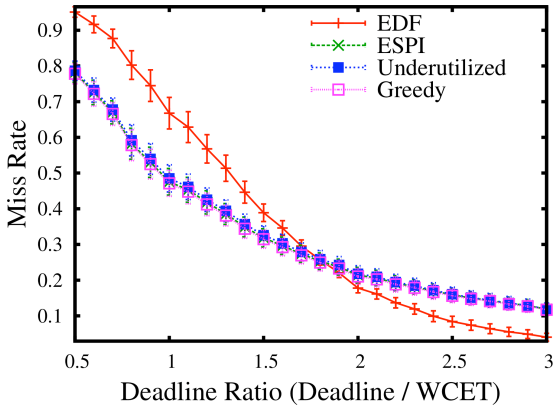


Figure 3.22: Timeliness comparison averaged across thirty three-task problem instances.

3.5 Discussion

Wrapped State Model: We published the periodicity results that lead to the wrapped state model in Glaubius *et al.* [38]. We also introduced a slightly different bounded model formulation in that work that individually restricted the range of state component values.

The wrapped model can be viewed as a state aggregation strategy, as it lumps together all states that have similar futures in terms of both cost and the relative transition distribution over states – that is, two utilization states $\mathbf{x}$ and $\mathbf{y}$ are equivalent if and only if for for each action i and every successor $\mathbf{z}$ of $\mathbf{x}$ with cost $c(\mathbf{z})$ there is a corresponding successor $\mathbf{z}'$ of $\mathbf{y}$ such that $c(\mathbf{z}) = c(\mathbf{z}')$ and $P(\mathbf{z}|\mathbf{x}, i) = P(\mathbf{z}'|\mathbf{y}, i)$. We are able to contract infinitely large sets of states down to a single exemplar using the observation that every state that is related by translation along the utilization ray satisfies this condition.

Dean and Givan [24] use the notion of *stochastic bisimulation* between states in order to perform model minimization through state aggregation. In that work, two states are similar if their relative transition distributions were similar in the sense that each had an identical probability of transitioning to similar states under the same actions. This property was used to automate state aggregation after partitioning the state space so that each cell was homogenous with respect to rewards. Subsequent work [34] by Givan, Dean, and Greig extended the definition of stochastic bisimulation to include reward similarity, and demonstrated that a model obtained by aggregating

similar states retains an optimal policy. Subsequent work has considered approximate stochastic bisimulation [31, 30] to allow aggregation of states when their rewards and transitions to similar states are almost identical.

It is possible to derive the wrapped state model from the utilization state model using this concept, since each equivalence class under periodicity is homogenous with respect to costs. We presented a problem-specific development of the model equivalence in order to highlight the importance of the choice of cost function and state-independent task durations in permitting this aggregation.

Stochastic bisimulation between states has since been generalized to arrive at exact and approximate *homomorphisms* in MDPs and SemiMDPs [74, 75, 92]. Whereas stochastic bisimulation requires that each action behaves essentially the same at similar states, states that are related by homomorphism require that every action at one state has some other action that behaves the same at any similar state. As with stochastic bisimulation, it can be shown that aggregation performed based on exact state homomorphisms retains an optimal policy and its value. This result would be useful in problems involving many identical tasks with equal utilization targets, since this suggests that we could further reduce the size of the problem, effectively exploiting the symmetry between tasks.

Li *et al.* [56] defines several degrees of state abstraction for MDPs; these are classified by the amount of information about the original model that is retained in the abstract models. The amount and type of information retained determines how accurately abstract model solutions mirror solutions to the original problem. In general, larger models retain more information, and so provide better solutions. One particularly important class of abstractions they identify are *model-irrelevance* abstractions, which preserve the transition and reward structure of the problem. The wrapped state model is a model-irrelevance abstraction because these abstractions contain models obtained by aggregating homomorphic states.

Bounded State Model: One early approach to approximating the utilization state model value function involved truncating the state space by introducing a maximum system lifetime [94]. This approach introduces an artificial limit on the number of decision epochs that elapse before termination. Policies obtained using this approach tend to exhibit edge effects near termination, as they angle to put the system as near

target utilization as possible at the last possible moment. While this method is not practical for most real systems, it served to motivate the truncation method used to develop the bounded state model.

We also reported a more complicated version of the bounded state model that used component-wise bounds to restrict the set of states considered [38]. In short, a bounds vector $\mathbf{b} \succeq \mathbf{0}$ is introduced, then a truncated model is defined over the wrapped states $\{\mathbf{x} \in \mathcal{X} : \mathbf{x} \preceq \mathbf{b}\}$. This is more flexible than the cost-bounded model presented in Section 3.2. This approach can be used to address the descrepancy between the number of model states and the states necessary for finding a good policy (see Figure 3.11 and the corresponding discussion). However, it is more complicated to tune this model since each dimension must be bounded individually, and the approximation performance of the model is more difficult to analyze because it is no longer possible to draw a simple distinction between the costs of modeled and unmodeled states.

ESPI: We derived the basic version of the ESPI algorithm described in Section 3.3.1 [36] in the context of the component-wise bounded model described above, but did not provide empirical verification of the algorithm. The ESPI algorithm is motivated by earlier approaches that incorporate reachability analysis into state modeling to solve large stochastic dynamic programming problems. Boutilier *et al.* [16] developed such an approach based on an abstract state space representation. The reachable state space is defined using a set of constraints. The set of reachable states could then be expanded using a constraint propagation strategy rather than explicititly enumerating reachable states; such an approach may be useful in scaling our techniques to larger numbers of threads, assuming that a suitable compact policy representation and constrained state space formulation can be found. Boutilier's work was inspired by the GraphPlan [15] algorithm for planning in deterministic domains, which iteratively enumerates the successors of a state set, starting with the initial state, until a goal state is found.

The ESPI(k,∞) algorithms are similar in spirit to the approach of Dean *et al* for anytime reinforcement learning [25]. They generate and solve progressively larger MDPs over restricted sets of states from the original MDP that converge to the original MDP. Rather than restricting model states based on the number of transitions needed to reach them, as we do in our work, they prune some immediate successor

states that are reachable only on low-probability transitions. This is motivated by a different focus: they are interested in finding coarse models that can be used to provide solutions quickly, then refining those models until the current most accurate solution is needed. On the other hand, we are interested in finding policies for intermediate models that translate exactly from the intermediate model to the original problem domain.

General Discussion: We have detailed two broad approaches to approximating the optimal task scheduling policy based on the wrapped state model of the problem. Our results from experiments looking at problem instances with two tasks suggest that the cost greedy policy may be the optimal policy in this case. In problems with three tasks we were able to find policies that outperform the cost-greedy policy, so the greedy policy is not optimal in general, and we suspect that the difference in value between the greedy and optimal policies grows with the number of tasks. We have not been able to demonstrate that it is necessarily the case that the greedy policy is optimal when there are only two tasks, but it seems likely that this is the case. Proving this claim may also help to demonstrate an *a priori* analytical bound on how well the greedy policy approximates the optimal in arbitrary dimensions.

Our results also suggest that the optimal task scheduling policy only visits finitely many states when we begin execution in the initial state $\mathbf{x} = \mathbf{0}$. If it is the case that every periodic task scheduling problem has finite closure, then it follows that the optimal policy of some corresponding cost bounded model is the optimal task scheduling policy for that problem. Knowing that such a cost bound exists would also be helpful in demonstrating that ESPI converges as well.

Even without formal proof of these guarantees, we have demonstrated that the MDP-based approaches in this section are feasible on a variety of simulated problems. However, these techniques do rely on explicit enumeration of a subset of the wrapped state space. Since these subsets grow exponentially in the number of tasks, these methods are unlikely to scale beyond a small number of tasks. In order to scale to larger problem instances, we will need to develop additional tools or approximation strategies. This includes approximation performance results that describe relate the value of efficiently computable policies, like the cost-greedy policy, approximate the

optimal solution. Another direction that we consider in Chapter 5 involves using optimization techniques to tune a class of compact, parameterized policies.

Chapter 4

Scheduling via Reinforcement Learning

In the previous chapters, we considered the problem of scheduling tasks under the assumption that we had perfect prior knowledge of their parameters. In this setting, we could find a (near-)optimal policy using dynamic programming. However, we often do not know the distribution of task durations in advance. We may have an inaccurate model, if we have any prior knowledge at all. We must then solve the Reinforcement Learning problem of deciding how to behave optimally online while tuning our scheduling policy based on incoming observations of the executing system. Our controller must consider the *exploration/exploitation dilemma*: the controller must balance behaving optimally with respect to available information against choosing apparently suboptimal actions in order to improve that information. The controller may commit to a suboptimal policy if it ceases exploring too soon, while if it continues to explore for too long, it may waste time and effort making poor choices. This dilemma is particularly significant in the real-time domain, as sustained suboptimal behavior translates directly into poor quality of service the system's users.

The results in this chapter were first made available as a technical report [35]; we have since obtained a tighter bound on the sample complexity of learning near-optimal scheduling policies by bounding the L_1-norm model estimation error rather than the l_∞-norm error. We have also corrected a mistake in the earlier analysis.

4.1 Related Work

The exploration/ exploitation dilemma is a well-known and extensively studied issue in Reinforcement Learning, and there is a substantial literature that addresses it. We present here a sample of the relevant literature. Kaelbling *et al.*'s survey [44] and Sutton and Barto's book [87] ground these ideas. Kakade's dissertation [46] provides a framework for formally studying this dilemma from a standpoint of computational efficiency. Szepesvári's recent survey [89] summarizes contemporary approaches.

Heuristic strategies are often employed due to their relative simplicity. These consist of randomizing the exploitive, value-greedy policy (see 2.5), and include the ϵ-greedy and Boltzmann (or softmax) action selection methods [44]. At each decision epoch k, ϵ-greedy exploration chooses to exploit with probability $(1 - \epsilon_k)$, otherwise an action is selected uniformly at random. This strategy will eventually converge to the optimal policy, provided the sequence $(\epsilon_k)_{k=0}^{\infty}$ decays at an appropriate rate [28]. Boltzmann action provides randomization by exponentially weighting each action according to its estimated value; an action is selected at random according to the normalized weights.

A principle that unifies many successful methods for efficient exploration is *optimism in the face of uncertainty* [44, 91]. When presented with a choice between two actions with similar estimated value, methods using this principle tend to select the action that has been tried less frequently. Optimism can take the form of optimistic initialization [29], *i.e.*, bootstrapping initial approximations with large values. For example, R-Max [19] and MBIE [86] assume the value of unvisited states to be as large as possible, although in the case of MBIE this is somewhat incidental.

Interval estimation is a general technique that biases action selection towards exploration computing and leveraging confidence intervals about model estimates. Interval-based techniques allow the learner to to behave optimistically *within reason* by quantifying model uncertainty. For example, actions may be selected according to the upper bounds of confidence intervals on their estimated values. This idea has been employed in solving Bandit problems (for some examples, see [7, 28, 62, 65]), a special case of Reinforcement Learning concerned with single-state with multiple actions having unknown rewards. Even-Dar et al. [28] adapted Bandit strategies to the general

MDP case by treating each state as a distinct Bandit instance with rewards corresponding to the estimated values of successor states. Model-Based Interval Estimation (MBIE) [84, 85, 86], applies interval estimation more naturally by deriving confidence intervals on the model parameters, essentially constructing bounds on reasonably possible models. The exploration policy used during learning is (near-)optimal with respect to an optimistic value estimate consistent with these bounds. Other examples of the interval estimation approach include [9, 91, 8].

We are interested in quantifying the difficulty of finding good scheduling policies in terms of the number of observations necessary to have a good chance to arrive at such a policy – that is, we are interested in the *sample complexity* of discovering good scheduling policies. The PAC, or *Probably Approximately Correct* learning model [95, 51] is a formal framework for investigating questions of this nature. Fiechter [32] and Kearns and Singh [48, 50] were among the first to study the sample complexity of reinforcement learning in the PAC model setting. Kearns and Singh [48, 50] developed the Explicit Explore or Exploit, or E^3, algorithm, a PAC learner that is guaranteed to find an ε-approximation to the optimal value function with probability at least $(1-\delta)$ after seeing a number of samples polynomial in the size of the state space $|\mathcal{X}|$, number of actions $|\mathcal{A}|$, and tolerance parameters $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$.

A pivotal component of E^3 is the specification of *known* and *unknown* states. Known states are ones in which each action has been tried a sufficiently large number of times; any state in which this is not satisfied is unknown. At each decision epoch, E^3 follows an exploration policy that is optimal with respect to a *known state MDP* that approximates the underlying true MDP. The known state MDP consists of the known states with transition and reward estimated from observations. The unknown states are aggregated into a single absorbing state. Whenever an unknown state is encountered, E^3 follows balanced wandering, simply choosing the least-sampled action in that state. Thus unknown states gradually become known until a near-optimal policy is found.

The R-Max algorithm [19] extends E^3 by associating an optimistic reward with the unknown states, encouraging their exploration. Brafman and Tennenholz proved that R-Max is a PAC reinforcement learning algorithm; Kakade [46] derived tighter bounds

on its sample complexity for both the discounted and average reward criteria. PAC guarantees also exist for MBIE.

PAC analysis for these algorithms is restricted to finite-state MDPs with bounded reward. In this chapter, we address this limitation in order to obtain similar results for the MDP formulation of the task scheduling problem. Kakade [46] suggested that the sample complexity ought to scale polynomially with the number of parameters of the transition model. In our scheduling problem the state-independent task duration distributions completely specify the transition system, so there is reason to hope that finite-sample complexity results are attainable in this domain.

4.2 Exploration Approach

The repeated structure of the transition system is crucial for modeling our scheduling problem as an MDP. Specifically, the transition dynamics are portable between states. We focus on indirect, model-based techniques [49] for discovering policies, as these techniques allow us to further exploit the transition structure. While direct methods, such as Q-learning [96, 97] and SARSA [76] require slightly less bookkeeping, since they maintain only a table of estimated values while direct methods require the estimated model parameters in addition to these values, it is not clear that we can adapt them to exploit the problem structure.

Since task duration distributions are discrete probability distributions with finite support, we simply estimate them using the empirical probability measure: we assume a collection of m observations $\{(i_k, t_k) : k = 1, \ldots, m\}$, where task J_{i_k} was dispatched at decision epoch k and ran for duration $t_k \sim P(\cdot|i_k)$ (we denote the duration distribution of task J_i as $P(\cdot|i)$ rather than $P_i(\cdot)$ in this chapter in order to reduce the risk of ambiguity). Let $\omega_m(i)$ be the number of samples of task J_i, and let $\omega_m(i, t)$ be the

number of those observations in which J_i ran for t quanta,

$$\omega_m(i) = \sum_{k=1}^{m} \mathbb{I}\{i_k = i\}, \tag{4.1}$$

$$\omega_m(i,t) = \sum_{k=1}^{m} \mathbb{I}\{i_k = i \wedge t_k = t\}. \tag{4.2}$$

Then the empirical measure $P_m(t|i)$ is just

$$P_m(t|i) = \omega_m(i,t)/\omega_m(i). \tag{4.3}$$

Since costs are deterministic, user-specified functions of state, the transition model is the only source of uncertainty in our problem.

4.3 Online Learning Results

We consider the difficultly of learning good scheduling policies in this section. We approach this question both analytically and empirically. In Section 4.3.1, we derive a PAC bound [95, 51] on a balanced wandering approach to exploration [48, 50, 28, 19]. Our result is novel, as it extends results derived for the finite-state, bounded cost setting to a domain with a countably infinite state space and unbounded costs. These results rely on a specific Lipschitz-like condition that restricts the growth rate of the value function under L_p-cost functions (cf. Equation 2.11 and Lemma 2.2.1 in Section 2.2), and the finite support of the duration distributions, *i.e.*, finite worst-case execution times of tasks. In Section 4.3.2, we present simulation results from experiments comparing several exploration strategies.

4.3.1 Analytical PAC Bound

We consider the sample complexity of estimating a near-optimal policy with high confidence by bounding the number of suboptimal actions taken during exploration. This definition was proposed by Kakade [46]. Our approach follows similar developments by Kearns and Singh [48, 50] and Kakade. We decompose analysis into two

parts. We derive bounds on the value estimation error as a function of the model estimation error. Then we bound the model error with high probability as a function of the number of observations. We consider the state-action value function Q for each task J_i, (cf. Equation 2.16 in Section 2.2) with

$$Q^*(\mathbf{x}, i) = (\Gamma_i V^*)(\mathbf{x}) = \sum_{t=1}^{T_i} P(t|i)[\gamma V^*(\mathbf{x} + t\Delta_i) - c(\mathbf{x} + t\Delta_i)]; \tag{4.4}$$

recall that

$$V^*(\mathbf{x}) = \max_{i \in \mathcal{A}}\{(\Gamma_i V^*)(\mathbf{x})\} = \max_{i \in \mathcal{A}}\{Q^*(\mathbf{x}, i)\}.$$

We define the optimal state value function V_m and state-action value function Q_m of the estimated model with duration distributions P_m similarly. Recall from Equation 2.17 in Section 2.2 that the optimal policy satisfies

$$\pi^*(\mathbf{x}) \in \operatorname*{argmax}_{i \in \mathcal{A}} \{Q^*(\mathbf{x}, i)\}.$$

The optimal solution to the estimated model is denoted π_m and can be defined analogously in terms of Q_m.

Notice that an accurate estimate of the optimal value function yields an accurate estimate of the optimal policy. Singh and Yee [80] demonstrated that the loss in value under the greedy policy with respect to an approximation $\hat{V}$ is at most $2\gamma\|V^* - \hat{V}\|_\infty/(1-\gamma)$. It is straightforward to obtain a similar bound on this loss in terms of $\|Q^* - Q_m\|_\infty$ using Equation 4.4

In order to establish our central result constraining the number of suboptimal actions taken with high probability, we first provide the following theorem. Proof of this theorem is somewhat involved, so we defer it to Section 4.5.

Theorem 4.3.1. *If there is a constant β such that for all tasks J_i,*

$$\sum_{t=1}^{T_i} |P_m(t|i) - P(t|i)| \leq \beta \tag{4.5}$$

and if there is a finite maximum worst-case execution time $T = \max\{T_i : i = 1 \dots n\}$, *then then for every state* $\mathbf{x}$ *and task* J_i,

$$|Q_m(\mathbf{x}, i) - Q^*(\mathbf{x}, i)| \leq \frac{2T\beta}{(1-\gamma)^2}.$$

This result serves an identical role to the Simulation Lemma of Kearns and Singh [48, 50] relating model estimation error to value estimation error. Our bound replaces the quadratic dependence on the number of states in that result with a dependence on T, the maximal worst-case execution time. This is consistent with results from Kakade's dissertation [46] indicating that the sample complexity of obtaining a good approximation should depend polynomially on the number of parameters of the transition model, which is $O(|\mathcal{X}|^2|\mathcal{A}|)$ for general MDPs, but is Tn in the domain considered here. The dependence on $(1-\gamma)^2$ is related to the pointwise bounds on the value function (cf. Equation 2.22). We conjecture that similar results can be obtained in the infinite-horizon discounted reward case for with problems with sparse, highly structured transition systems that admit Lipschitz value functions.

We say that Q_m is an ε-approximation to Q^* whenever $\|Q_m - Q\|_\infty \leq \varepsilon$. The result of Theorem 4.3.1 allows us to quantify the model estimation error necessary to obtain a value approximation that is at least an ε-approximation, since requiring

$$2T\beta/(1-\gamma)^2 \leq \varepsilon,$$

or equivalently,

$$\beta \leq \varepsilon(1-\gamma)^2/(2T), \tag{4.6}$$

is sufficient to guarantee $\|Q^* - Q_m\|_\infty \leq \varepsilon$. Another way to interpret this bound is that, if we can guarantee bounded approximation errors with probability $(1-\delta)$ for some confidence level $0 < \delta < 1$, then with equally high confidence the policy π_m will never choose an action that has value worse than 2ε from the optimal, thus eliminating expensive mistakes (in terms of the error tolerance ε) with high probability. To see this, notice that if Q_m is a ε-approximation, for any two tasks J_i and J_j and state $\mathbf{x}$,

$$Q_m(\mathbf{x}, i) - Q_m(\mathbf{x}, j) \leq Q^*(\mathbf{x}, i) + \varepsilon - Q^*(\mathbf{x}, j) + \varepsilon,$$

so the ordering of J_i and J_j under $Q^*(\mathbf{x}, \cdot)$ is preserved by Q_m as long as their true values differ by at least 2ε.

It remains to relate this result to m, the number of samples of the transition system, to the model estimation error in Equation 4.5. For this, we rely on the following lemma:

Lemma 4.3.1. *(Lemma 8.5.5 of Kakade [46]) Suppose that M i.i.d. samples are obtained from a probability measure p supported on a discrete set of N elements. Let $\hat{p}$ be the corresponding empirical measure. Then for any $\varepsilon > 0$ and $\delta \in (0, 1)$, if $M \geq \left(\frac{8N}{\varepsilon^2}\right) \log\left(\frac{2N}{\delta}\right)$, then with probability at least $(1 - \delta)$,*

$$\sum_{j=1}^{N} |\hat{p}(j) - p(j)| \leq \varepsilon.$$

A union bound argument shows that to guarantee

$$\mathbb{P}\Big\{\bigwedge_{i=1}^{n}\Big(\sum_{t=1}^{T} |P_m(t|i) - P(t|i)| \geq \beta\Big)\Big\} \leq \delta,$$

we require $\mathbb{P}\left\{\sum_{t=1}^{T} |P_m(t|i) - P(t|i)| \geq \beta\right\} \leq \delta/n$ for each task. If we assume balanced wandering, *i.e.* $\omega_m(i) = m/n$ for each task J_i (assuming for convenience that m is a multiple of n), then by Lemma 4.3.1 we can guarantee the desired model accuracy β with probability at least $(1 - \delta)$ for

$$m \geq (8Tn/\beta^2) \log(2Tn/\delta). \tag{4.7}$$

Now we are prepared to relate our value estimation tolerance ε to the number of samples m. Equation 4.6 implies that

$$\frac{1}{\beta^2} \geq (2T)^2/(\varepsilon^2(1-\gamma)^4)$$

substituting this into Equation 4.7 implies that in order to guarantee $\|Q_m - Q^*\|_\infty \leq \varepsilon$ with probability at least $(1 - \delta)$, it is sufficient to require at least

$$m = \left(\frac{16T^3 n}{\varepsilon^2(1-\gamma)^4}\right) \log\left(\frac{2Tn}{\delta}\right)$$

samples. As with previous bounds, the sample complexity scales polynomially in the parameters of interest. One note is that the worst-case execution time T enters into the bound from two sources, hence the cubic dependence. The complexity of learning the duration distributions depends on T, since it limits the number of "bins" that samples can fall into; $2T/(1-\gamma)$ also bounds the growth of the value function between consecutive decision epochs (see Lemma 4.5.1 in Section 4.5 for the details of this result), which we must take into consideration in order to estimate Q^* accurately.

Due to the transition system portability, our result is most similar to those for accurate estimation in a single state. For comparison, Kakade derives a bound on the number of experiences necessary to learn the distribution at a state, finding $m = O\left(\left(\frac{|\mathcal{X}|\log^2\varepsilon}{\varepsilon^2(1-\gamma)^2}\right)\log\left(\frac{|\mathcal{X}||\mathcal{A}|}{\delta}\right)\right)$ (Lemma 8.5.6 of [46]) under the assumption that rewards are restricted to the interval $[0,1]$ and that R-max [19] is used. (The factor $\log^2\varepsilon$ appears to be an artifact of the analysis, which considers a finite horizon). A result for learning the entire model must include an additional factor of at least $|\mathcal{X}||\mathcal{A}|$ in the general case. In our bound, the worst-case execution time T plays a similar role to the size of the state space in this bound, with the value function bound introducing an additional factor $(T/(1-\gamma))^2$.

4.3.2 Empirical Evaluation

The analytical result of the previous section give some sense of the finite-sample performance for learning a good schedule; however, we required several simplifying assumptions, such as balanced wandering, so the bound may not be tight. In practice, alternative exploration strategies may yield better performance than our bound would indicate. We compared the performance of several exploration strategies in the context of the task scheduling problem by conducting experiments comparing ϵ-greedy, balanced wandering, and an interval-based exploration strategy.

In our experiments we consider the number of mistakes made before converging to the *cost-greedy* scheduling policy

$$\pi(\mathbf{x}) = \operatorname*{argmin}_{i\in\mathcal{A}}\left\{\sum_t P(t|i)c(\mathbf{x}+t\Delta_i)\right\},$$

rather than the optimal policy in two-task problem instances. This is motivated by the observation that in every case we have considered, the greedy policy is identical to the optimal for the two task case; the relative efficiency of computing this policy allows us to perform much larger scale experiments than would be possible if, say, we must compute an ESPI approximation to the optimal value function after each sample. In this case, Bandit strategies for determining the best action are appropriate, since the focus is on correctly identifying and acting with respect to the best reward.

For interval-based optimistic exploration, we use the confidence intervals derived for the multi-armed bandit case by Even-Dar *et al* [28] for their Successive Elimination algorithm. That algorithm constructs intervals of the form $\alpha_k = \sqrt{\log(nk^2c)/k}$ about the expected reward of each action at decision epoch k, then eliminates actions that appear worse than the apparent best using an overlap test. This constant c encapsulates a scale factor and the reciprocal of a PAC parameter δ found in the original source. Since there are states in which every action is optimal our focus is not on action elimination *per se*; we instead use the maximum $R_m(\mathbf{x}, i) + \alpha_{k,i}$ among all actions to choose which action to execute, where we have adjusted the confidence intervals according to the potentially different number of samples of each task,

$$\alpha_{k,i} = \sqrt{\log(n\omega_k(i)^2c)/\omega_k(i)}$$

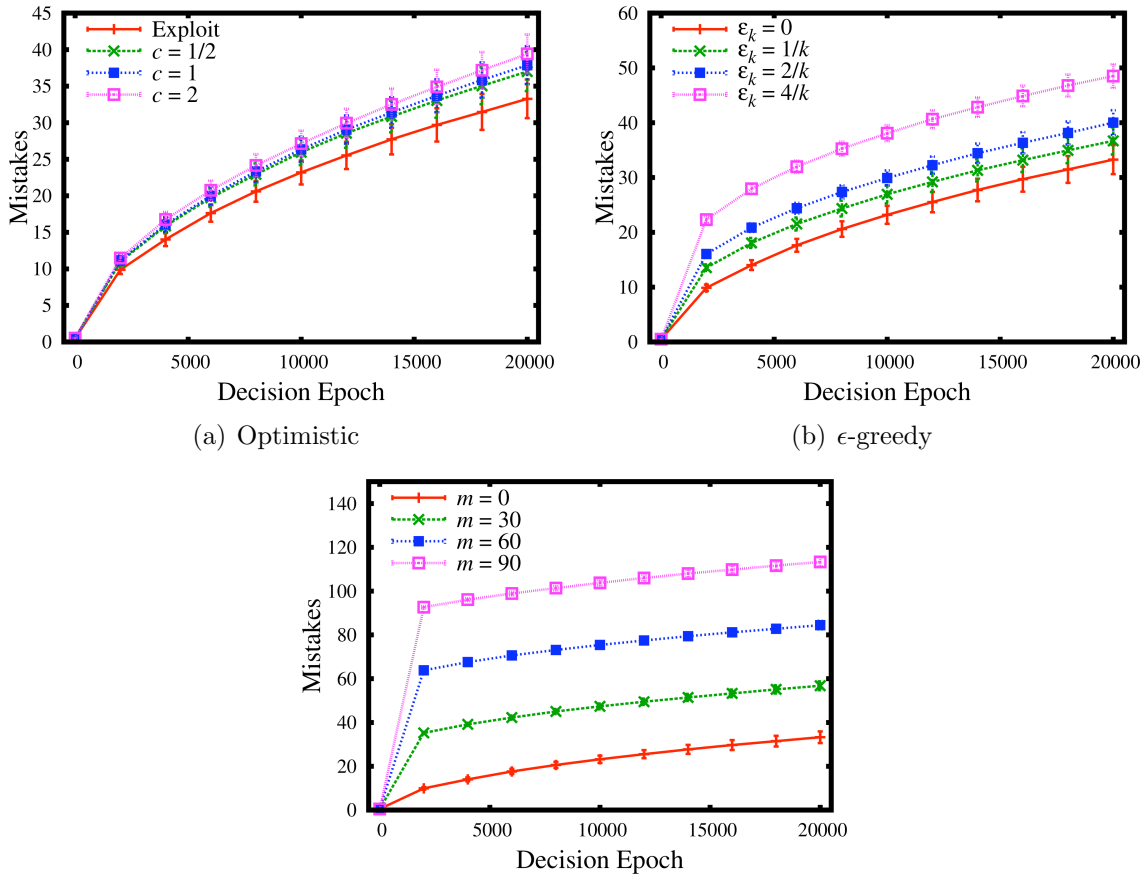
In our experiments, we vary the parameter c to control the chance of taking exploratory actions. As c shrinks, the intervals narrow and this strategy tends to exploit the estimated model.

Balanced wandering simply executes each task a fixed number of times m prior to exploiting. We vary this parameter in order to determine its impact on the learning rate. When $m = 0$, this strategy always exploits its current model knowledge. In our experiments with ϵ-greedy, we set the random selection rate at decision epoch k, $\epsilon_k = \epsilon_0/k$ for varying values of ϵ_0; this strategy always exploits when $\epsilon_0 = 0$.

In order to compare the performance of these exploration strategies, we generated 1000 random problem instances with two tasks. Duration distributions for these tasks consisted of discretized normal distributions supported on the interval [1,32], with means and variances selected uniformly at random from the respective intervals [1,32] and [1,8]. Utilization targets for each task were chosen according to

$\mathbf{u} = (u_1', u_2')/(u_1' + u_2')$, where u_1' and u_2' were integers selected uniformly at random between [1,128].

We conducted experiments by initializing the model in the initial state $\mathbf{x} = (0, 0)$. The controller simulated a single trajectory over 20,000 decision epochs in each problem instance with each exploration strategy. We report the number of mistakes – the number of times the exploration strategy chooses an action that has smaller reward in expectation than the greedy policy. The results of these experiments are shown in Figure 4.1.



(a) Optimistic

(b) ϵ-greedy

(c) Balanced

Figure 4.1: Simulation comparison of exploration techniques. Note the differing scales on the vertical axes.

Comparison Results: In Figure 4.1, we report 95% confidence intervals on the mean number of mistakes each exploration strategy makes, averaged across the problem instances described above. Note that these plots have different scales due to the variation in mistake rates among exploration strategies.

Figure 4.1(a) compares the performance of interval-based optimistic action selection to that of "Exploit", the policy that behaves greedily with respect to the estimated model at each decision epoch. All of the interval-based exploration settings we considered exhibited statistically similar mistake rate performance. Interestingly, the exploitive strategy yields similar performance to the explorative strategies despite its lack of an explicit exploration mechanism.

The benefits of the exploitive policy are clear in comparison to two the other exploration strategies we consider. Figure 4.1(b) illustrates the performance of ϵ-greedy exploration. Notice that the mistake rate decreases along with the likelihood of taking exploratory actions – that is, as ϵ_0 approaches zero. Exploration actually may result not improve performance in this domain.

This is further supported by our results for balanced wandering. The theory behind balanced wandering is that making a few initial mistakes early on will pay off in the long run due to more uniformly accurate models. Figure 4.1(c) shows that this is not the case in in our scheduling domain, as a purely exploitive strategy $m = 0$ outperforms each of the balanced wandering approaches.

These results suggest that the exploitive strategy may be the best available exploration method in the task scheduling problem. One justification is that the environment itself enforces rational exploration: if some task is never dispatched, the controller will find itself in progressively worse states, as is impossible to track the utilization target while ignoring states when each task has non-zero utilization target.

Problem Parameter Interaction: The results in Figure 4.1 were obtained by generating many random problems in order to control for the effects of problem parameters. We are also interested in understanding how the problem parameters interact with the mistake rate. In particular, we wish to understand how the choice of utilization target impacts learning, since this effects the number of model states and

may contribute to a better understanding of how to select these parameters effectively We performed another set of experiments to examine this question. We considered utilization targets of the form $\mathbf{u} = (1/(1+u_2'), u_2'/(1+u_2'))$, with u_2' an integer in [1,32]. This allows us to vary the ratio between the target utilization of each task – that is, we essentially give task J_2 progressively larger resource shares relative to J_1. For each utilization target, we executed a trajectory using the exploitive strategy on problems with the duration distributions from the previous experiments.

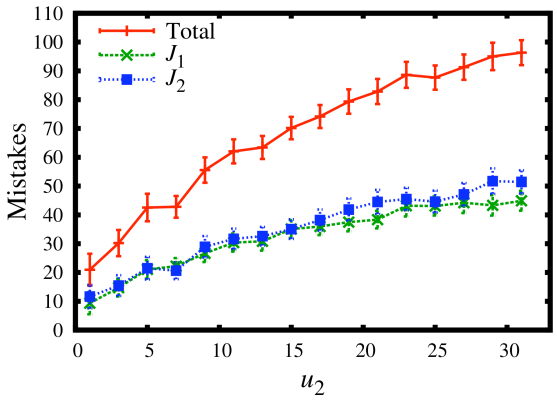


Figure 4.2: Simulation comparison of mistake rates under varying task parameters.

We report the aggregate mistake rate after 20,000 decision epochs in Figure 4.2 (labeled "Total") as well as the rate of dispatching task J_1 when J_2 is the appropriate choice (labeled "J_1") and vice versa (labeled "J_2"). One might expect that as the utilization target skews more towards J_2 that the reward associated with each task would separate, making it easier to identify the correct action. Figure 4.2 illustrates that this is not the case; as the utilization target becomes more skewed, the learner is actually likely to make more mistakes.

Interestingly, while the mistake rate increases as the utilization target becomes more skewed towards J_2, this does not appear to result in a commensurate increase in the likelihood of incorrectly dispatching any particular task more often.

4.4 Conclusions

In this chapter, we have considered the problem of learning near-optimal schedules when the system model is not known in advance. We have presented analytical results that bound the number of suboptimal actions that are taken prior to arriving at a near-optimal policy with high certainty. Interestingly, the transition system's portability results in bounds that are similar to those for estimating the underlying model in a single state.

This naturally leads to a comparison to the multi-armed bandit model (see, for example, [28]), in which there is a single state with several available actions. Each action causes the emission of a reward according to a corresponding unknown, stationary random process. However, a bandit model does not appear to apply directly because while the duration distributions are stationary processes that are invariant between states, the payoff associated with each action is state-dependent.

We have focused on the PAC model of learning rather than deriving bounds on *regret* – the loss in value incurred because of suboptimal behavior while learning [9, 8]. Regret bounds may translate more readily into guarantees about transient real-time performance effects during learning, since guarantees regarding cost, and hence value, translate into guarantees about task timeliness.

We have presented empirical results which suggest that a learner that always exploits its current information outperforms agents that explicitly encourage exploration. This occurs because any policy that consistently ignores some action will get progressively farther from the utilization target, resulting in arbitrarily large costs. Thus the domain itself appears to enforce an appropriate level of exploration, perhaps obviating the need for an explicit exploration mechanism. It is an open question whether we can identify a more general class of MDPs that exhibit this behavior.

4.5 Proof of Theorem 4.3.1

Recall that in Theorem 4.3.1, we claim that the approximation error

$$|Q_m(\mathbf{x}, i) - Q^*(\mathbf{x}, i)|$$

is uniformly at most $2T\beta/(1-\gamma)^2$ given that the model estimation error is at most in the L_1-norm by β. In order to prove this result, we require a pair of lemmas. The first establishes a "speed limit" that controls the growth of the optimal value function of any task scheduling problem instance (more generally, we can show that if costs induce a pseudo-norm over the state space, as do the L_p cost functions, then the value function of our scheduling MDP is Lipschitz in that pseudonorm).

Lemma 4.5.1. *For any state* $\mathbf{x}$*, task* J_i*, and duration* t*,*

$$|V^*(\mathbf{x} + t\Delta_i) - V^*(\mathbf{x})| \leq tc(\Delta_i)/(1-\gamma).$$

Proof. Note that

$$\begin{aligned}|V^*(\mathbf{x} + t\Delta_i) - V^*(\mathbf{x})| &= \left|\max_j\{Q^*(\mathbf{x} + t\Delta_i, j)\} - \max_j\{Q^*(\mathbf{x}, j)\}\right| \\ &\leq \max_j |Q^*(\mathbf{x} + t\Delta_i, j) - Q^*(\mathbf{x}, j)|\,.\end{aligned}$$

By expanding Q^* according to Equation 4.4,

$$\begin{aligned}&|Q^*(\mathbf{x} + t\Delta_i, j) - Q^*(\mathbf{x}, j)| \\ &= \Big|\sum_s P(s|j)[\gamma V^*(\mathbf{x} + t\Delta_i + s\Delta_j) - c(\mathbf{x} + t\Delta_i + s\Delta_j) \\ &\qquad - \gamma V^*(\mathbf{x} + s\Delta_j) + c(\mathbf{x} + s\Delta_j)]\Big| \\ &\leq \sum_s P(s|j)\,|c(\mathbf{x} + s\Delta_j) - c(\mathbf{x} + t\Delta_i + s\Delta_j)| \\ &\qquad + \gamma \sum_s P(s|j)\,|V^*(\mathbf{x} + t\Delta_i + s\Delta_j) - V^*(\mathbf{x} + s\Delta_j)|\,.\end{aligned}$$

By Lemma 2.2.1 from Section 2.2, we have the bound

$$|c(\mathbf{x} + s\Delta_j) - c(\mathbf{x} + t\Delta_i + s\Delta_j)| \leq tc(\Delta_i),$$

so the first term in this sum can be bounded thus. $|V^*(\mathbf{x} + t\Delta_i + s\Delta_j) - V^*(\mathbf{x} + s\Delta_j)|$ has the same form as our initial condition, so we can iteratively apply this argument to obtain the bound

$$|V^*(\mathbf{x} + t\Delta_i) - V^*(\mathbf{x})| \leq \sum_{k=0}^{\infty} \gamma^k tc(\Delta_i) = tc(\Delta_i)/(1-\gamma).$$

□

The next result allows us to bound the error when estimating the expectation of the cost and optimal value functions, given that the model estimation error is bounded. We use this result to bound the computational complexity due to the magnitude of costs.

Lemma 4.5.2. *If task J_i satisfies $\sum_{t=1}^{T} |P_m(t|i) - P(t|i)| \leq \beta$ and if there is a constant $\lambda \geq 0$ such that for every state $\mathbf{x}$ and duration t, $f : \mathcal{X} \to \mathbb{R}$ satisfies $|f(\mathbf{x} + t\Delta_i) - f(\mathbf{x})| \leq \lambda t$, then*

$$\left| \sum_t [P_m(t|i) - P(t|i)] f(\mathbf{x} + t\Delta_i) \right| \leq T\lambda\beta$$

Proof. Let $\sigma_t = \text{sign}(P_m(t|i) - P(t|i))$. Then

$$\begin{aligned}
&\left| \sum_{t=1}^{T} [P_m(t|i) - P(t|i)] f(\mathbf{x} + t\Delta_i) \right| \\
&\quad \leq \left| \sum_{t=1}^{T} [P_m(t|i) - P(t|i)] (f(\mathbf{x}) + t\lambda\sigma_t) \right| \\
&\quad = \left| \sum_{t=1}^{T} [P_m(t|i) - P(t|i)] f(\mathbf{x}) + \sum_{t=1}^{T} [P_m(t|i) - P(t|i)] t\lambda\sigma_t \right| \\
&\quad = \lambda \sum_{t=1}^{T} |P_m(t|i) - P(t|i)|\, t \\
&\quad \leq T\lambda\beta.
\end{aligned}$$

Since $\sum_t P_m(t|i) - P(t|i) = 0$, the term involving $f(\mathbf{x})$ vanishes, leading to the desired result. □

Recall that R is the expected reward,

$$R(\mathbf{x}, i) = -\sum_t P(t|i)c(\mathbf{x} + t\Delta_i); \tag{4.8}$$

The expected value of the successor $\mathbf{y}$ of $\mathbf{x}$ is

$$\mathbb{E}_P\left\{V^*(\mathbf{y})\right\} = \sum_t P(t|i)V^*(\mathbf{x} + t\Delta_i). \tag{4.9}$$

We denote R_m and $\mathbb{E}_{P_m}\{V_m(\mathbf{y})\}$ analogously. Equations 4.8 and 4.9 allow us to write Q^* more concisely as

$$Q^*(\mathbf{x}, i) = R(\mathbf{x}, i) + \gamma\mathbb{E}_P\{V^*(\mathbf{y})\}$$

Proof. (Theorem 4.3.1) Let $(\mathbf{x}, i)$ be a state-task pair. Then by the triangle inequality,

$$|Q_m(\mathbf{x}, i) - Q^*(\mathbf{x}, i)| \leq |R_m(\mathbf{x}, i) - R(\mathbf{x}, i)| + \gamma\,|\mathbb{E}_{P_m}\{V_m(\mathbf{y})\} - \mathbb{E}_P\{V^*(\mathbf{y})\}| \tag{4.10}$$

We can simplify the error of estimating the expected reward using Lemmas 2.2.1 and 4.5.2,

$$|R_m(\mathbf{x}, i) - R(\mathbf{x}, i)| \leq c(\Delta_i)\sum_{t=1}^{T}|P_m(t|i) - P(t|i)|\,t < 2T\beta;$$

the final inequality holds because $c(\Delta_i) < 2$. The difference in expected future state values can be decomposed into two terms. The first term corresponds to the expected error under P_m due to misestimation of the value, while the second term consists of the value estimation error due to the inaccurate duration distribution estimate:

$$\begin{aligned}
&|\mathbb{E}_{P_m}\{V_m(\mathbf{y})\} - \mathbb{E}_P\{V^*(\mathbf{y})\}| \\
&= \left|\sum_t P_m(t|i)V_m(\mathbf{x} + t\Delta_i) - P(t|i)V^*(\mathbf{x} + t\Delta_i)\right| \\
&= \left|\sum_t P_m(t|i)V_m(\mathbf{x} + t\Delta_i) - [P(t|i) + P_m(t|i) - P_m(t|i)]V^*(\mathbf{x} + t\Delta_i)\right| \\
&= \left|\sum_t P_m(t|i)[V_m(\mathbf{x} + t\Delta_i) - V^*(\mathbf{x} + t\Delta_i)] + \sum_t[P_m(t|i) - P(t|i)]V^*(\mathbf{x} + t\Delta_i)\right| \\
&\leq \sum_t P_m(t|i)\,|V_m(\mathbf{x} + t\Delta_i) - V^*(\mathbf{x} + t\Delta_i)| + \left|\sum_t[P_m(t|i) - P(t|i)]V^*(\mathbf{x} + t\Delta_i)\right|.
\end{aligned}$$

The second term here can be simplified by applying Lemmas 4.5.1 and 4.5.2,

$$\left|\sum_t [P_m(t|i) - P(t|i)]V^*(\mathbf{x} + t\Delta_i)\right| \leq 2T\beta/(1-\gamma)$$

Substitute the bounds on $|R_m(\mathbf{x}, i) - R(\mathbf{x}, i)|$ and $|\mathbb{E}_{P_m}\{V_m(\mathbf{y})\} - \mathbb{E}_P\{V^*(\mathbf{y})\}|$ into Equation 4.10,

$$\begin{aligned}
&|Q_m(\mathbf{x}, i) - Q^*(\mathbf{x}, i)| \\
&\quad \leq 2T\beta + \gamma 2T\beta/(1-\gamma) + \sum_t P_m(t|i)\,|V_m(\mathbf{x} + t\Delta_i) - V^*(\mathbf{x} + t\Delta_i)| \\
&\quad = 2T\beta/(1-\gamma) + \sum_t P_m(t|i)\,|V_m(\mathbf{x} + t\Delta_i) - V^*(\mathbf{x} + t\Delta_i)|
\end{aligned}$$

Since

$$|V_m(\mathbf{x} + t\Delta_i) - V^*(\mathbf{x} + t\Delta_i)| \leq \max_j |Q_m(\mathbf{x} + t\Delta_i, j) - Q^*(\mathbf{x} + t\Delta_i, j)|$$

we may iteratively substitute according to Equation 4.10 to obtain the stated bound

$$\begin{aligned}
|Q_m(\mathbf{x}, i) - Q^*(\mathbf{x}, i)| &\leq \sum_{k=0}^{\infty} \gamma^k 2T\beta/(1-\gamma) \\
&\leq 2T\beta/(1-\gamma)^2.
\end{aligned}$$

□

Chapter 5

Conic Scheduling Policies

The scheduling algorithms described in Chapter 3 are able to find good approximations to the optimal task scheduling policy. Those methods operate by enumerating a set of "good" system states which, due to the state space periodicity (Chapter 2), are guaranteed to be finite. Those methods enable formal verification techniques [38], as it is possible to explore the full set of states reachable under the derived policy from the initial state $\mathbf{x} = \mathbf{0}$.

However, those *finite-state approximation* techniques have a major drawback. The utilization state spaces of these models and their corresponding wrapped state spaces have dimension equal to the number of tasks. Thus those methods are only practical for small problems involving two or three tasks.

One popular strategy for addressing these concerns is to restrict search to a small class of compactly parameterizable policies [88, 71]. In special cases, the optimal scheduling policy may be a member of this family, and can be solved for directly [72]. This is not the case in general, so function optimization techniques must be employed for tuning the policy parameters until a "good" policy is found [52].

We propose a class of parameterized *conic* scheduling policies based on the geometry of those found using finite-state approximation methods [37]. Any scheduling policy partitions the utilization state space into n distinct regions, one for each task. We observed that under the best available scheduling policies, these partitions almost divide the space into n cones. We exploit this regularity to derive our class of policies with decision boundaries that closely resemble those observed under the finite-state approximations.

In Section 5.1 we describe this class of conic scheduling policies in greater detail. In Section 5.2, we provide a sufficient condition for policy stability – *i.e.*, the system state converges to a region with bounded cost – and demonstrate the existence of such a conic policy. We provide empirical evidence for the effectiveness of these policies in Section 5.3, and conclude with a discussion of related work and directions of further investigation in Section 5.4.

5.1 Conic Scheduling Policies

Choosing an appropriate policy class is admittedly more art than science, and requires understanding the application domain and the properties of good policies. With this in mind, we first illustrate examples of scheduling policy behavior observed using the finite-state approximation techniques described in Chapter 3. We use these observations to select an efficient parameterization that requires only $\Theta(n^2)$ parameters, where n is the number of tasks.

Figure 5.1 shows an approximation to the optimal scheduling policy for a problem instance with two tasks, restricted to states near the initial state $\mathbf{x} = \mathbf{0}$ (lower left corner). Each point denotes a state in the wrapped state space. The target utilization ray is shown as a dashed ray labeled $\lambda\mathbf{u}$, and corresponds to a target utilization of $\mathbf{u} = (7, 5)^\top/12$. Each task has a different duration distribution supported on the interval $(0, 8]$. Task J_1 advances along the horizontal axis, and task J_2 advances along the vertical axis. The scheduling policy selects task J_1 in states denoted by closed red circles, and task J_2 in those denoted by closed blue squares. Notice that the *decision boundary* – the surface separating regions of state space where the policy is homogeneous – can be described using a ray parallel to the utilization ray. This is illustrated by the dashed ray labeled $\lambda\mathbf{u} + \mathbf{d}$. We will describe the offset $\mathbf{d}$ below.

It is more difficult to illustrate the policy in three-task problems, since the state space is three dimensional. To establish an intuition for what is occurring in three-space, consider the set of utilization states that the system may reach after exactly t time quanta have elapsed:

$$H_t = \{\mathbf{x} \in \mathcal{X} : \tau(\mathbf{x}) = t\}.$$

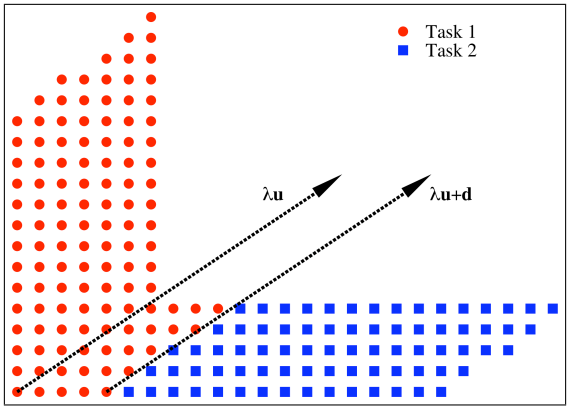
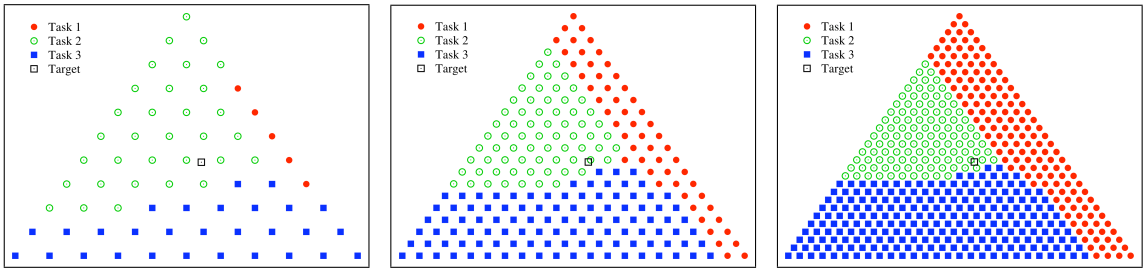


Figure 5.1: The scheduling policy for this two-task problem can be defined by partitioning the state space along a ray parallel to the utilization ray $\lambda\mathbf{u}$.



(a) H_{10} (b) H_{20} (c) H_{30}

Figure 5.2: Near-optimal scheduling policies for a three task problem, shown at time horizons H_{10}, H_{20}, and H_{30}. The leftmost state in each is $(t, 0, 0)$, the rightmost is $(0, t, 0)$, and the topmost is $(0, 0, t)$.

We call H_t a *time horizon*, and it consists of all of the utilization states (*i.e.*, integral points) in a $(n-1)$-simplex with vertices $\{t\Delta_i : i = 1, \ldots, n\}$. In two dimensions, H_t is the set of states that lie on the line segment joining $(t, 0)$ and $(0, t)$. In three dimensions, H_t is the set of states contained in the equilateral triangle with vertices $(t, 0, 0)$, $(0, t, 0)$, and $(0, 0, t)$. The target utilization at time t is $t\mathbf{u}$, and is the *ideal point* that the system state should be near in H_t.

Figure 5.2 illustrates an approximation to the optimal scheduling policy for a problem instance with three tasks. The policy is illustrated by plotting it at three different time horizons, H_{10}, H_{20}, and H_{30}. The target utilization is $\mathbf{u} = (6, 8, 9)^\top/23$, and corresponds to a point in each horizon, shown as an open box. Tasks are non-identical, but each has duration supported on the interval $(0, 8]$. The policy executes J_1 in closed red circle states, J_2 in open green circle states, and J_3 in closed blue square states.

As in the two task case, the policy partitions each time horizon into one region for each task. This appears to be representative behavior. Together with the observation of periodicity from Chapter 2, this gives us two criteria for designing a parametric class of scheduling policies: (1) the policy should be periodic, so that it chooses the same action at $\mathbf{x}$ and every utilization state along $\mathbf{x} + \lambda\mathbf{u}$; and (2) the policy should divide each time horizon into n homogeneous regions. This now leads to our formulation of conic policies.

We define two types of parameters for conic policies. The first n parameters describe a *decision offset* vector $\mathbf{d}$; this is a vector perpendicular to $\mathbf{1}$, the vector of all ones, (*i.e.*, $\mathbf{d}$ is parallel to each time horizon) that roots the partition relative to the utilization target.

The remaining n^2 parameters define a collection of *action vectors* $\mathbf{A} = [\mathbf{a}_1 \ \ldots \ \mathbf{a}_n]$ that are used to determine the regions where each action is taken. For each task J_i, the action vector $\mathbf{a}_i$ is an n-vector perpendicular[5] to $\mathbf{1}$. We use the decision offset and action vectors to partition each time horizon into homogeneous regions as follows.

Informally, a conic policy selects a task to dispatch in state $\mathbf{x}$ by determining which action vector most points towards $\mathbf{x}$ from the decision offset. More formally, $\tau(\mathbf{x})\mathbf{u}$ is the ideal utilization point at time $\tau(\mathbf{x})$. We root the policy decision boundaries on the *decision ray* $\lambda\mathbf{u} + \mathbf{d}$ at that time, $\tau(\mathbf{x})\mathbf{u} + \mathbf{d}$. Let

$$\mathbf{z}(\mathbf{x}) = \mathbf{x} - \tau(\mathbf{x})\mathbf{u} - \mathbf{d} \tag{5.1}$$

be the displacement vector from the offset of the ideal utilization point to $\mathbf{x}$. Then we choose to run the task J_i if its action vector is well-aligned with $\mathbf{z}(\mathbf{x})$ – that is, if the *response* $\mathbf{a}_i^\top \mathbf{z}(\mathbf{x})$ is maximal among all action vectors. We state this formally in the following definition.

[5]Since we constrain the decision offset $\mathbf{d}$ and each action vector $\mathbf{a}_i$ to be parallel to the time horizons, they actually lie in an $(n-1)$-dimensional space and could be parameterized using $(n-1)$-vectors instead of n-vectors. We adopt the higher-dimensional representation here for expository purposes, since it can be communicated more concisely.

Definition 5.1.1. *The conic policy* $\pi(\cdot; \mathbf{d}, \mathbf{A})$ *with decision offset* $\mathbf{d}$ *and a matrix of action vectors* $\mathbf{A}$ *chooses actions in each utilization state* $\mathbf{x}$ *according to*

$$\pi(\mathbf{x}; \mathbf{d}, \mathbf{A}) \in \underset{i=1,\ldots,n}{\operatorname{argmax}} \{\mathbf{a}_i^\top \mathbf{z}(\mathbf{x})\}$$

Notice that it is possible for multiple action vectors to have the same response at $\mathbf{x}$. We recommend breaking ties uniformly at random in this case. Under this tie breaking convention, the conic policy degenerates to a uniform random scheduling policy whenever all of the action vectors are equal.

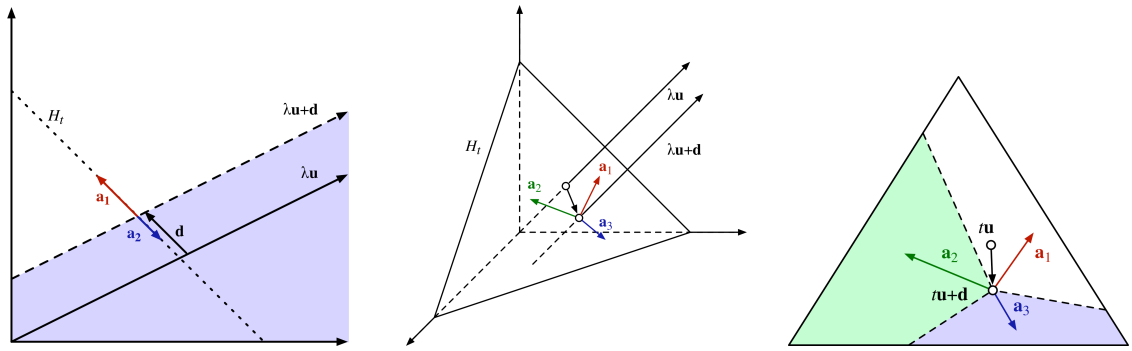
Figure 5.3 illustrates this policy in the state space of two- and three-task problems. Figure 5.3(a) shows an example policy for a two-task problem instance. The decision offset $\mathbf{d}$ shifts the decision boundary parallel to the target utilization ray, while the action vectors determine the policy action on either side of the decision boundary. In two dimensions, the action vectors are not strictly necessary, since any policy that runs task J_2 above the decision boundary and J_1 below it will diverge, reaching states with arbitrarily negative costs, since this policy eventually dispatches the same task repeatedly.

The action vectors are necessary when there are three or more tasks. This is illustrated in Figures 5.3(b) and 5.3(c). In Figure 5.3(b), we show the time horizon H_t situated in three-space. The action vectors are shown in the plane. Figure 5.3(c) shows the perpendicular projection of the time horizon onto the plane; the decision boundaries for each action are shown, and consist of the region where the offset between a state and $t\mathbf{u}+\mathbf{d}$ is well-aligned with one of the action vectors. Notice that the policy breaks this simplex into three conic regions emanating from $t\mathbf{u} + \mathbf{d}$.

For any number of tasks n, the decision offset and action vectors act to partition each time horizon H_t into n distinct cones whenever the action vectors are distinct. We demonstrate this formally in the proof of Lemma 5.1.1.

Lemma 5.1.1. *For any decision offset* $\mathbf{d}$*, action vectors* $\mathbf{A}$*, time horizon* H_t*, and task* J_i*, the set of states*

$$\Lambda_{t,i} = \{\mathbf{x} \in H_t : i \in \operatorname{argmax}_j \{\mathbf{a}_j^\top \mathbf{z}(\mathbf{x})\}\}$$



(a) two tasks (b) three tasks (c) three task horizon H_t

Figure 5.3: Illustration of conic policies in two dimensions (Figure 5.3(a)) and three dimensions (Figures 5.3(b) and 5.3(c)). Figure 5.3(c) shows the time horizon H_t from Figure 5.3(b) with decision boundaries between regions where the policy is homogeneous.

is a cone with apex $t\mathbf{u} + \mathbf{d}$.

Proof. A set $\mathcal{Y}$ is a cone with apex $\mathbf{v}$ if and only if $\mathcal{Y}$ is convex and for any $\mathbf{y} \in \mathcal{Y}$, any $\mathbf{y}'$ on the ray from $\mathbf{v}$ through $\mathbf{y}$ is also in $\mathcal{Y}$. We demonstrate below that these two properties hold for $\Lambda_{t,i}$ relative to $t\mathbf{u} + \mathbf{d}$. We make use of the fact that for any real-valued n-vector $\mathbf{v}$, $\tau(\mathbf{v}) \equiv \mathbf{v}^\top \mathbf{1}$ is a linear map.

Convexity: Suppose that $\mathbf{x}$ and $\mathbf{x}'$ are states in $\Lambda_{t,i}$ and that $\mathbf{y} = \alpha\mathbf{x} + \beta\mathbf{x}'$ is a utilization state for some $\alpha \in [0, 1]$ and $\beta = 1 - \alpha$. $\mathbf{z}(\cdot)$ is convex, *i.e.*,

$$\begin{aligned}\mathbf{z}(\alpha\mathbf{x} + \beta\mathbf{x}') &= \alpha\mathbf{x} + \beta\mathbf{x}' - \tau(\alpha\mathbf{x} + \beta\mathbf{x}')\mathbf{u} - \mathbf{d} \\ &= \alpha\mathbf{x} + \beta\mathbf{x}' - \alpha\tau(\mathbf{x})\mathbf{u} - \beta\tau(\mathbf{x}')\mathbf{u} - \alpha\mathbf{d} - \beta\mathbf{d} \\ &= \alpha\mathbf{z}(\mathbf{x}) + \beta\mathbf{z}(\mathbf{x}'),\end{aligned}$$

therefore

$$\mathbf{A}^\top \mathbf{z}(\mathbf{y}) = \alpha\mathbf{A}^\top \mathbf{z}(\mathbf{x}) + \beta\mathbf{A}^\top \mathbf{z}(\mathbf{x}').$$

Since $\mathbf{a}_i$ maximizes each term in the right-hand side, it also maximizes the left-hand side, implying that $\mathbf{y}$ is in $\Lambda_{t,i}$.

Homogeniety: For any $\mathbf{x}$ in $\Lambda_{t,i}$, a state $\mathbf{y}$ in H_t is along the ray through $\mathbf{x}$ from $t\mathbf{u}+\mathbf{d}$ if and only if $\mathbf{y} = \lambda\mathbf{z}(\mathbf{x}) + t\mathbf{u} + \mathbf{d}$ for some $\lambda > 0$. Since $\mathbf{z}(\mathbf{x})$ and $\mathbf{d}$ are perpendicular

to $\mathbf{1}$, $\tau(\mathbf{z}(\mathbf{x})) = \tau(\mathbf{d}) = 0$. Therefore,

$$\begin{aligned}\mathbf{z}(\mathbf{y}) &= \mathbf{z}(\lambda\mathbf{z}(\mathbf{x}) + t\mathbf{u} + \mathbf{d}) \\ &= \lambda\mathbf{z}(\mathbf{x}) + t\mathbf{u} + \mathbf{d} - \tau(\lambda\mathbf{z}(\mathbf{x}) + t\mathbf{u} + \mathbf{d})\mathbf{u} - \mathbf{d} \\ &= \lambda\mathbf{z}(\mathbf{x}) + t\mathbf{u} + \mathbf{d} - \lambda\tau(\mathbf{z}(\mathbf{x}))\mathbf{u} - t\mathbf{u} - \tau(\mathbf{d})\mathbf{u} - \mathbf{d} \\ &= \lambda\mathbf{z}(\mathbf{x}),\end{aligned}$$

and so

$$\mathbf{A}^\top\mathbf{z}(\mathbf{y}) = \lambda\mathbf{A}^\top\mathbf{z}(\mathbf{x}).$$

It follows that if $\mathbf{a}_i$ maximizes $\mathbf{A}^\top\mathbf{z}(\mathbf{x})$, it also maximizes $\mathbf{A}^\top\mathbf{z}(\mathbf{y})$, and so $\mathbf{y}$ is in $\Lambda_{t,i}$. □

Above, in the statement of Lemma 5.1.1 we define the cone $\Lambda_{t,i}$ as the set of states in the horizon where $\mathbf{a}_i$ has maximal response. It might appear more natural to define $\Lambda_{t,i}$ as the set of states in horizon H_t where the parameterized policy dispatches task J_i. However, this latter definition breaks down at the decision boundaries, since if we use a nondeterministic tie breaking procedure, the policy may not be homogeneous along these boundaries.

Conic policies select among the tasks whose action vectors yield the greatest response in any given state. Our proof of Lemma 5.1.2 formalizes periodicity of the conic policy by showing that the set of action vectors that gives the greatest response $\mathbf{x}$ also yields the greatest response at any utilization state along the ray $\{\mathbf{x} + \lambda\mathbf{u} : \lambda \geq 0\}$.

Lemma 5.1.2. *For any decision offset* $\mathbf{d}$*, action vectors* $\mathbf{A}$*, and utilization state* $\mathbf{x}$*, if* J_i *maximizes* $\mathbf{a}_i^\top\mathbf{z}(\mathbf{x})$ *among all tasks, then* J_i *maximizes* $\mathbf{a}_i^\top\mathbf{z}(\mathbf{x} + \lambda\mathbf{u})$ *for any real scalar* λ*.*

Proof. Since

$$\begin{aligned}\mathbf{z}(\mathbf{x} + \lambda\mathbf{u}) &= \mathbf{x} + \lambda\mathbf{u} - \tau(\mathbf{x} + \lambda\mathbf{u})\mathbf{u} - \mathbf{d} \\ &= \mathbf{x} + \lambda\mathbf{u} - \tau(\mathbf{x})\mathbf{u} - \lambda\mathbf{u} - \mathbf{d} \\ &= \mathbf{z}(\mathbf{x}),\end{aligned}$$

so for any task J_i,

$$\mathbf{a}_i^\top \mathbf{z}(\mathbf{x}) = \mathbf{a}_i^\top \mathbf{z}(\mathbf{x} + \lambda \mathbf{u}),$$

and so if $\mathbf{a}_i^\top \mathbf{z}(\mathbf{x})$ is maximal among all tasks, then it also maximizes $\mathbf{a}_i^\top \mathbf{z}(\mathbf{x}+\lambda\mathbf{u})$. □

The example policy shown in Figure 5.2 is periodic but *not* conic: in order for the partition induced by the policy to be conic, we would need to be able to represent the decision boundary by exactly three rays emanating from a common point. Thus, the conic partition of the parameterized policy is not optimal in general. In Section 5.2, we show that while the best conic policy may not be optimal among all possible scheduling policies, it contains stable policies that maintain the system near target utilization. In Section 5.3 we demonstrate that we can find conic policies that perform well; particularly, there are conic policies that consistently outperform the cost-greedy (Equation 3.8) and utilization-based (Equation 3.9) heuristic policies of Section 3.2.2 on problems that are too large to solve using finite-state approximations.

5.2 Conic Policy Stability

Above, we described a case in which the system might diverge to states with arbitrarily negative costs. In Figure 5.1, if the policy instead dispatched task J_1 in every state below the decision boundary, upon reaching any of these states, the policy would then repeatedly run that task forever. We say that such a policy is unstable, since it never settles into a region with bounded cost. We now derive a sufficient condition under which a conic policy is guaranteed to be stable. Informally, a stable policy is one under which the system state is guaranteed to converge to a region with finite, bounded costs.

A *trajectory* generated under policy π is a sequence of utilization states $(\mathbf{x}_k)_{k=0}^{\infty}$ such that $\mathbf{x}_{k+1}$ is distributed according to $P(\cdot|\mathbf{x}_k, \pi(\mathbf{x}_k))$. We say that a scheduling policy is *stable* if and only if we can guarantee that any trajectory generated under that policy is eventually within some bounded neighborhood of the utilization ray. The cost function is defined in terms of the L_p distance between the state $\mathbf{x}$ and the ideal

point $\tau(\mathbf{x})\mathbf{u}$ on the utilization ray (first defined in Equation 2.11)

$$c_p(\mathbf{x}) = \|\mathbf{x} - \tau(\mathbf{x})\mathbf{u}\|_p$$

This means that a stable policy maintains the system in states with costs relatively near zero, which in turn ensures that every task makes progress relative to one another.

Theorem 5.2.1 provides a sufficient condition on the decision offset and action vectors to guarantee that the corresponding conic policy is stable. The proof is stated in terms of the Euclidean distance rather than the general L_p distance; since these norms are topologically equivalent, stability in Euclidean distance also implies stability in the rest.

Theorem 5.2.1. *If $\pi = \pi(\cdot; \mathbf{d}, \mathbf{A})$ is a conic policy, and there is some $\varepsilon > 0$ such that for every utilization state $\mathbf{x}$,*

$$(\Delta_{\pi(\mathbf{x})} - \mathbf{u})^\top \mathbf{z}(\mathbf{x}) \leq -\varepsilon \|\mathbf{z}(\mathbf{x})\| \tag{5.2}$$

where $\|\cdot\|$ is the Euclidean norm, then π is stable.

We defer a formal proof of Theorem 5.2.1 to supplemental material in Section 5.5 at the end of this chapter, and provide only a brief sketch here. The vector $(\Delta_{\pi(\mathbf{x})} - \mathbf{u})$ is the instantaneous change in state at $\mathbf{x}$ when following π. The precondition of the theorem requires that the angle between this state derivative and the displacement $\mathbf{z}(\mathbf{x})$ between $\mathbf{x}$ and the decision ray is negative. This guarantees that there is always an instantaneous reduction in distance from the decision ray under π.

However, since the state changes in discrete jumps according to the task durations, it is possible for the policy to move the system from a state that is close to the decision ray to one that is farther away. Since tasks have bounded worst-case durations, the policy will always move the system closer to the decision ray from a state that is far enough away. Closer to the decision ray, the policy may throw the state farther from the decision ray, but because of the duration bounds, the distance of this successor state from the decision ray is bounded. Therefore, we can draw a cylinder with finite radius centered around the decision ray such that any trajectory starting from

a state inside the cylinder must stay inside, while trajectories originating outside of the cylinder are eventually pulled inside, and then stay there.

We can conclude that the system state must always enter into a bounded neighborhood of the decision ray. Since the utilization ray is at a fixed distance from the decision ray, this result also implies that the policy converges to a bounded region about the target utilization.

While Theorem 5.2.1 provides a sufficient condition to guarantee stability, it does not supply us with a stable parameterization. Corollary 5.2.1 provides an example of a stable policy.

Corollary 5.2.1. *A conic policy* $\pi(\cdot; \mathbf{d}, \mathbf{A})$ *with action vectors*

$$\mathbf{a}_i = (\mathbf{u} - \Delta_i) / \|\mathbf{u} - \Delta_i\| \tag{5.3}$$

is stable for any choice of decision offset $\mathbf{d}$.

The proof of this corollary is surprisingly involved, and so we defer this to the supplemental material in Section 5.5 as well. The intuition behind the proof is relatively straightforward, however. By construction, each action vector $\mathbf{a}_i$ defined by Equation 5.3 points exactly opposite the direction of travel relative to the utilization target, $(\Delta_i - \mathbf{u})$; that is, the angle between these two vectors is 180°. Because of this relationship, we can conclude that if the angle between an action vector $\mathbf{a}_i$ and the displacement $\mathbf{z}(\mathbf{x})$ is "comfortably" smaller than 90° (expressed as a constraint on their dot product in terms of ε), then the angle between $\mathbf{z}(\mathbf{x})$ and the $(\Delta_i - \mathbf{u})$ must be more than 90°, satisfying the precondition of Theorem 5.2.1. Figure 5.4 illustrates these action vectors for a three-task problem instance.

The choice of decision offset is unnecessary when determining stability. If a policy is stable with offset $\mathbf{d}$, it will also be stable with offset $\mathbf{d}'$. This is because the state derivative when dispatching a particular task is independent of the decision offset. A stable choice of action vectors causes the system state to enter a stochastic orbit around the decision ray; moving the decision offset just moves that orbit through the state space.

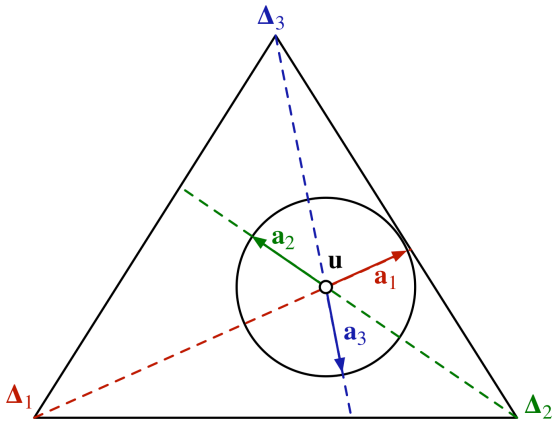


Figure 5.4: Scaled action vectors $\mathbf{a}_i = (\mathbf{u} - \Delta_i)/\left\|\Delta_i - \mathbf{u}\right\|$.

This line of reasoning seems to suggest that we should always choose $\mathbf{d} = \mathbf{0}$ as the decision offset, so that the system orbits around the utilization ray. This is not the case, however. In practice, the state stochastically orbits the decision ray, but because of the difference in durations between tasks and because there is a non-smooth change in the direction of travel when changing actions, the average location tends to differ from the decision ray. Selecting a good conic policy parameterization appears to consist of establishing a decision offset and action vectors so that this average state is as close as possible to the utilization target.

5.3 Conic Policy Performance

Finding the optimal conic policy analytically appears to be a difficult problem. Rather than approaching the problem from this direction, we instead employ stochastic optimization techniques to select good parameterizations of the conic policy. We discuss these methods and evaluate them empirically in this section.

We ran tests comparing the performance of selected conic policies to two heuristic scheduling policies. The, cost-greedy policy π_g (Equation 3.8) is defined

$$\pi_g(\mathbf{x}) = \underset{i \in \mathcal{A}}{\operatorname{argmin}}\ \mathbb{E}_{t \sim P_i}\{c(\mathbf{x} + t\Delta_i)\};$$

it always executes the task that leads to the least immediate cost in expectation. The second heuristic policy is the utilization-based policy π_u (Equation 3.9), defined as

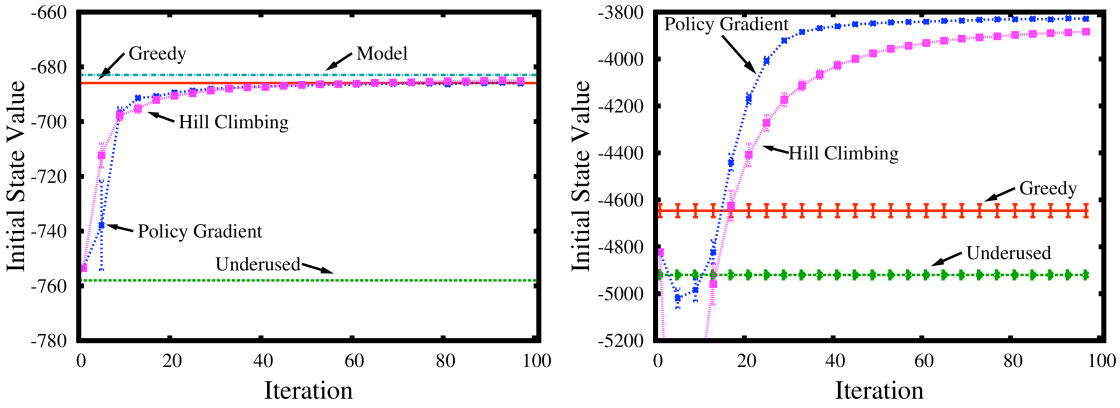
$$\pi_u(\mathbf{x}) \in \operatorname*{argmax}_{i \in \mathcal{A}} \{\tau(\mathbf{x}) u_i - x_i\};$$

this always runs the most underutilized task. Where possible, we also compare conic policy performance to finite-state approximations.

To find good conic policies we implemented the hill climbing and policy gradient search methods described by Kohl and Stone [52]. Both of these search methods follow a similar outline. We begin with an initial policy parameterization; in each case, the stable conic policy in Equation 5.3 with a decision offset of $\mathbf{d} = \mathbf{0}$. At each iteration, we generate a population of nearby policies by adding small random perturbations to each parameter. In our experiments, we found that a population size of $3n(n+1)$ works well for either search strategy, where n is the number of tasks. These policies are evaluated by performing Monte Carlo evaluation [87] – *i.e.*, by repeatedly simulating the policy from the initial state $\mathbf{x} = \mathbf{0}$ to estimate the policy value and averaging the sum of discounted, observed rewards (recall that reward equals negative cost). This population is then used to determine a policy for the next iteration.

Hill climbing search chooses the policy at each iteration by selecting the policy with greatest estimated value among this population. As in Kohl and Stone's work [52], we select the best policy among these perturbed parameter settings regardless of whether that policy is worse than the current policy. This affords the algorithm some limited ability to escape local optima. In our experiments, parameter perturbations were drawn uniformly at random from the interval $\pm(1/m + 0.98^m)$ at iteration m.

Policy gradient search instead uses these perturbations to estimate the gradient of the value with respect to the policy parameters. The policy at the next iteration is determined by stepping a fixed distance along the gradient. In our experiments, parameters were perturbed by adding a value at random from among $\{0, \pm m^{-1/6}\}$, while the step size along the gradient was $1/m + 0.98^m$. We decay the step size and the perturbation size in the hill climbing experiments so that we eventually settle on some policy. The specific decay rates were chosen, loosely speaking, to keep this from



(a) 4 Task Problem Instance

(b) 10 Task Problem Instance

Figure 5.5: Experimental results showing the performance of policy search strategies as a function of the number of iterations performed. Values labeled "Hill climbing" and "Policy Gradient" show the eponymous search performance, "Greedy" and "Underused show the heuristic policy performance, and "Model" the finite-state approximation of optimal.

happening too quickly. A formal discussion of appropriate decay strategies can be found, for example, in [33].

We performed two experiments comparing the performance of conic policies, heuristic policies, as well as finite-state approximation policies when possible. The first experiment examines how the value of each conic policy evolves with each iteration of the policy search method. The second experiment examines the performance of the different policies for problem instances as a function of the number of tasks.

In each problem instance the task duration distributions were random histograms with worst-case execution time T_i in the interval $[2, 32]$. The utilization targets for each problem instance were selected by choosing integers $\mathbf{q} \in [2, 32]^n$ uniformly at random, so that the utilization target is $\mathbf{u} = \mathbf{q} \cdot (\sum_i q_i)^{-1}$. For both the heuristic and conic policies, evaluating the value function at the initial state was carried out using Monte Carlo evaluation.

In the first experiment, we generated a single problem instance with 4 tasks. We performed 100 iterations of policy search using both hill climbing and policy gradient search and estimated the value at each iteration. We compared their performance to the utilization policy π_u, the greedy policy π_g, and a finite-state approximation to the

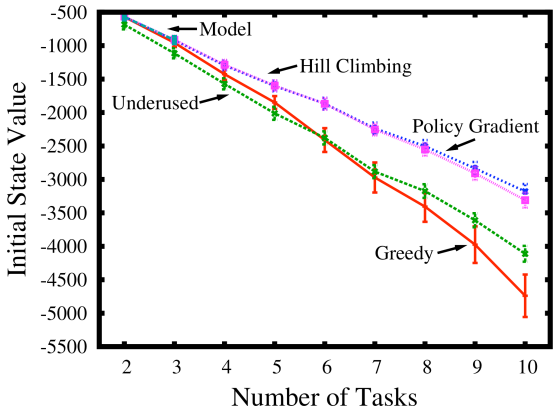


Figure 5.6: Comparison of policy performance for varying numbers of tasks. finite-state approximation of the optimal policy is shown only for two and three tasks.

optimal policy. These values are shown in Figure 5.5(a) as "Hill Climbing," "Policy Gradient," "Underused," "Greedy," and "Model," respectively. Since rewards equal negative costs, policy values closer to zero indicate better performance. 95% confidence intervals were obtained by performing 30 repetitions of each search strategy. Conic policy evaluation was performed by averaging across 1000 simulated trajectories of 1000 decision epochs. A discount factor of $\gamma = 0.99$ was used.

Figure 5.5(a) shows that explicitly solving the MDP model gives the best results. However, Hill Climbing, Policy Gradient and Greedy produce comparable results. Utilization performs relatively poorly. We have observed that π_u has value near the default stable policy with decision offset $\mathbf{d} = \mathbf{0}$.

Figure 5.5(b) shows the results of repeating this experiment using a problem instance with 10 tasks. In this case, finite-state approximation of the optimal policy is intractable. Exact evaluation of the heuristic policies also requires enumerating their set of reachable states, and so is intractable as well. Instead, Monte Carlo simulation is used to estimate their values, which are shown with 95% confidence intervals. In this case Hill Climbing outperforms either heuristic policy. The additional structure used by Policy Gradient search allows it to outperform Hill Climbing.

In the second experiment we compared the performance of these policies across problem instances with varying number of tasks. For each number of tasks we generated 100 independent problem instances with the same method described above. Average values with 95% confidence intervals are shown in Figure 5.6. We report the policy

found after 100 iterations of search. Results are shown for the finite-state approximation for two and three task problem instances. Monte Carlo evaluation was used to evaluate the heuristic policies and conic policies in each case, consisting of 100 simulated trajectories of 1000 decision epochs each.

In two or three task problem instances, Greedy, Model, Hill Climbing and Policy Gradient all perform similarly. With more tasks the conic policy clearly outperforms either heuristic policy. This supports the use of conic policies for scaling to larger problem instances.

5.4 Discussion

Related Work: The work presented in this chapter demonstrates the ability of the parameterized conic policy to approximate the optimal task scheduling policy and to scale our approach to larger problems. We selected to tune the policy parameters using hill climbing search and sparse stochastic policy gradient estimation methods primarily for their simplicity.

The study of policy gradient estimation in reinforcement learning is a broad area of study with numerous applications. Peters and Schaal [71] a survey of these methods in the context of robotics, where the emphasis is on scalability to high-dimensional policy parameterization. We provide a brief overview of topics in this area.

Most of the work on direct policy search appears to be focused on robust estimation of policy gradients. Broadly speaking, these methods fall into two categories: finite difference methods [52] like those used in this work, and what Peters and Schaal refer to as "likelihood ratio" methods [98, 69, 88].

The focus in finite difference methods is on estimating gradients by examining the value of parameterizations in the neighborhood of the current policy. This is performed by perturbing the policy parameters and performing Monte Carlo estimation of the parameter values. The downside of this approach is that it assumes experience

is cheap; for example, that we have an accurate generative model of the system. The benefit is that these methods tend to achieve low-variance gradient estimates.

Likelihood ratio methods follow in the footsteps of Williams's REINFORCE methods [98]. These methods are sample efficient: they are provided a fixed collection of trajectories observed under some exploration policy, or equivalently, that a fixed sequence of random numbers used to generate policies is provided [69]. Policies are assumed to assign non-zero probability to every action in every state, so that the likelihood of each trajectory under the policy can be computed. This is used to provide an estimate of the policy value, and together with an analytical gradient of the policy with respect to its parameters, can be used to estimate the policy gradient.

Using these methods to tune the conic policy could allow us to perform policy search online without a prior model of the system. However, extending to this case requires addressing a couple of issues. The straightforward gradient estimate is not invariant to reparameterization, so we may see different performance if we define the decision offsets and action vectors in the n-dimensional space of this paper rather than in the $(n-1)$-dimensional space perpendicular to $\mathbf{1}$. Perturbations to different parameters also affect the conic policy differently; a small change to the decision offset changes the policy more than a commensurate change to the action vector parameters. Natural policy gradient methods [45] sidestep this issue by deriving a gradient estimate that is invariant to change of parameters.

Conclusions: In this chapter we have introduced a scalable conic scheduling policy design technique that compactly approximates the geometric structure of policies obtained using direct solution techniques. This technique allows us to derive good scheduling policies for open soft real-time systems with large numbers of tasks. Our results indicate that direct solution techniques are most appropriate when tractable, while conic policies provide strong scalable performance where direct solution methods fail.

Our experiments demonstrate that search is able to find a good conic policy when initialized with the stable policy from Equation 5.3. However, it is unclear whether these methods are likely to converge to a global optimum among conic policies when restricted to this initial parameterization. Typically, this would be addressed by using

randomized restarts in order to sample many local optima. However, that approach fails for our task scheduling problem, as most random conic policies are unstable and policies in their neighborhood also tend to be unstable. Since unstable policies reach states with large magnitude cost, these policies have almost uniformly low value, so there is no clear direction that search can follow to reach a good parameterization.

We suggest two promising approaches for addressing this problem. One is to use a richer policy representation; for example, choosing to dispatch tasks at random, with probability proportional to to the action vector responses, may provide a more informative value gradient [88]. A second method is to derive a more comprehensive characterization of stable conic policies, which would allow us to sample safely from a wider variety of initial conditions.

5.5 Supplemental Material

Proof of Theorem 5.2.1:

Proof. Let $(\mathbf{x}_k)_{k=0}^{\infty}$ be a trajectory, with $\mathbf{x}_0$ arbitrary and $\mathbf{x}_{k+1}$ determined by executing $\pi(\mathbf{x}_k)$ in $\mathbf{x}_k$. We need to prove that this (arbitrary) trajectory converges to a bounded neighborhood of the utilization ray. Our proof consists of two parts: first we show that the trajectory eventually enters this neighborhood, and then we show that the trajectory can not escape this neighborhood.

Let $\mathbf{x}$ be an arbitrary state and let $\pi(\mathbf{x}) = i$. Suppose that $\mathbf{y} = \mathbf{x} + t\Delta_i$ is a successor of $\mathbf{x}$ under π. We can write the displacement $\mathbf{z}(\mathbf{y})$ between $\mathbf{y}$ and the decision ray in terms of the displacement at $\mathbf{x}$,

$$\begin{aligned}\mathbf{z}(\mathbf{y}) &= \mathbf{x} + t\Delta_i - \tau(\mathbf{x})\mathbf{u} - t\mathbf{u} - \mathbf{d} \\ &= \mathbf{z}(\mathbf{x}) + t(\Delta_i - \mathbf{u}).\end{aligned}$$

This allows us to derive an upper bound on the squared magnitude of $\mathbf{z}(\mathbf{y})$, as

$$\begin{aligned}\|\mathbf{z}(\mathbf{y})\|^2 &= (\mathbf{z}(\mathbf{x}) - t(\Delta_i - \mathbf{u}))^\top (\mathbf{z}(\mathbf{x}) - t(\Delta_i - \mathbf{u})) \\ &= \mathbf{z}(\mathbf{x})^\top \mathbf{z}(\mathbf{x}) + t^2 (\Delta_i - \mathbf{u})^\top (\Delta_i - \mathbf{u}) \\ &\quad + 2t(\Delta_i - \mathbf{u})^\top \mathbf{z}(\mathbf{x}) \\ &\leq \|\mathbf{z}(\mathbf{x})\|^2 + t^2 \|\Delta_i - \mathbf{u}\|^2 - 2t\varepsilon \|\mathbf{z}(\mathbf{x})\|.\end{aligned}$$

Defining

$$\eta(\mathbf{x}, t) \equiv t^2 \left\|\Delta_{\pi(\mathbf{x})} - \mathbf{u}\right\|^2 - 2t\varepsilon \|\mathbf{z}(\mathbf{x})\|$$

allows us to write the inequality above more concisely as

$$\|\mathbf{z}(\mathbf{y})\|^2 \leq \|\mathbf{z}(\mathbf{x})\|^2 + \eta(\mathbf{x}, t). \tag{5.4}$$

We define $M = \max_i \{T_i^2 \|\Delta_i - \mathbf{u}\|^2\}$ to bound the first term of $\eta(\mathbf{x}, t)$ above (recall that T_i is the worst-case execution time of Task J_i). For any $\alpha > 0$ let $\rho_\alpha = (M + \alpha)/(2\varepsilon)$ be the radius of a cylinder centered on the decision ray. Then if $\|\mathbf{z}(\mathbf{x})\| \geq \rho_\alpha$,

$$\eta(\mathbf{x}, t) \leq M - 2\varepsilon(M + \alpha)/(2\varepsilon) = -\alpha.$$

We can substitute this inequality into Equation 5.4 to get $\|\mathbf{z}(\mathbf{y})\|^2 \leq \|\mathbf{z}(\mathbf{x})\|^2 - \alpha$. In other words, executing the policy action always reduces the distance between $\mathbf{x}$ and the decision ray if $\mathbf{x}$ is far enough away.

This result guarantees that the trajectory is eventually within ρ_α of the decision ray for any $\alpha > 0$. If this were not the case, we would be able to find some K such that any $k \geq K$ has $\|\mathbf{z}(\mathbf{x}_k)\| \geq \rho_\alpha$, but for any $m > 0$, we have

$$\begin{aligned}\|\mathbf{z}(\mathbf{x}_{K+m})\|^2 &\leq \|\mathbf{z}(\mathbf{x}_{K+m-1})\|^2 - \alpha \\ &\vdots \\ &\leq \|\mathbf{z}(\mathbf{x}_K)\|^2 - m\alpha,\end{aligned}$$

so $\mathbf{x}_{K+m}$ is within ρ_α of the decision ray for large m.

By the triangle inequality, we have

$$\begin{aligned}\|\mathbf{z}(\mathbf{y})\| &= \|\mathbf{z}(\mathbf{x}) + t(\Delta_i - \mathbf{u})\| \\ &\leq \|\mathbf{z}(\mathbf{x})\| + t\,\|\Delta_i - \mathbf{u}\| \\ &\leq \|\mathbf{z}(\mathbf{x})\| + M^{1/2},\end{aligned}$$

so if $\|\mathbf{z}(\mathbf{x})\| \leq \rho_\alpha$, $\|\mathbf{z}(\mathbf{y})\| \leq \rho_\alpha + M^{1/2}$. Since the state gets closer to the decision ray when $\|\mathbf{z}(\mathbf{x})\|$ is greater than ρ_α, and cannot get farther than $\rho_\alpha + M^{1/2}$ when the state is inside this neighborhood, the trajectory must eventually enter and stay within distance $\rho_\alpha + M^{1/2}$ of the decision ray for any $\alpha > 0$. Since the trajectory is arbitrary, this must hold for any trajectory generated while following π. □

Proof of Corollary 5.2.1:

Proof. To simplify notation, let $\mathbf{z}$ denote $\mathbf{z}(\mathbf{x})$ for some arbitrary state $\mathbf{x}$. Under Equation 5.3,

$$\mathbf{a}_i^\top \mathbf{z} = -(\Delta_i - \mathbf{u})^\top \mathbf{z} / \|\Delta_i - \mathbf{u}\|.$$

Therefore, we just need to show that for some $\alpha > 0$,

$$\max_i \{\mathbf{a}_i^\top \mathbf{z}\} \geq \alpha \|\mathbf{z}\|$$

for every $\mathbf{z}$, since then $(\Delta_{\pi(\mathbf{x})} - \mathbf{u})^\top \mathbf{z} \leq -\alpha \|\mathbf{z}\| \|\Delta_i - \mathbf{u}\|$, satisfying Theorem 5.2.1's precondition. To simplify the discussion, we assume that $\min_i\{\|\Delta_i - \mathbf{u}\|\}$ is a factor of α. This term is guaranteed to be positive because no single task is assigned the entire processor. Demonstrating the claim therefore reduces to demonstrating that in every state there is a task J_i such that $(\mathbf{u} - \Delta_i)^\top \mathbf{z} \geq \alpha \|\mathbf{z}\|$.

For the sake of contradiction, suppose that for all $\alpha > 0$, there is a state $\mathbf{x}$ (and corresponding displacement $\mathbf{z}$) such that

$$\max_i \{(\mathbf{u} - \Delta_i)^\top \mathbf{z}\} < \alpha \|\mathbf{z}\|. \tag{5.5}$$

We can rewrite the left-hand side of Equation 5.5 according to

$$\max_i \{(\mathbf{u} - \Delta_i)^\top \mathbf{z}\} = \mathbf{u}^\top \mathbf{z} - \min_i \{z_i\}$$

(recall that $\Delta_i^\top \mathbf{z} = z_i$). We will proceed to show that satisfying Equation 5.5 for arbitrarily small α requires an invalid utilization target. To achieve this, we first need an upper bound on the left-hand side of the equation.

To obtain this bound, recall that $\mathbf{z}$ is the displacement between $\mathbf{x}$ and the decision ray (see Equation 5.1). This lies in a plane perpendicular to $\mathbf{1}$, and so $\sum_{i=1}^{n} z_i = 0$. Let ζ be the sum of positive components of $\mathbf{z}$, then $-\zeta$ is the sum of its negative components.

Without loss of generality, we may assume that the components of the utilization target are ordered with $u_1 \geq u_2 \geq \ldots \geq u_n$. This implies that u_n lies in the interval $(0, 1/n]$ and that $u_1 \leq 1 - u_n(n-1)$, as otherwise $\mathbf{u}$'s components would not sum to one.

Using these observations, it is straightforward to verify that for a fixed utilization target, the left-hand side of Equation 5.5 is maximized by fixing $z_1 = \zeta$ and $z_n = -\zeta$, and setting the other components to zero. Then

$$\begin{aligned} \mathbf{u}^\top \mathbf{z} - \min_i \{z_i\} &\leq \zeta u_1 - \zeta u_n + \zeta \\ &\leq \zeta(2 - n \cdot u_n) \end{aligned}$$

Substituting this bound into Equation 5.5 yields the inequality $\zeta(2 - n \cdot u_n) < \alpha \|\mathbf{z}\|$, or equivalently,

$$u_n > 2/n - \alpha \|\mathbf{z}\| / (\zeta n).$$

Thus as α approaches zero, the smallest utilization target u_n must exceed $1/n$, a contradiction. □

Chapter 6

Conclusions

In this dissertation we have proposed a system for scheduling tasks with timing constraints subject to a novel combination of operating conditions. We consider scheduling multiple tasks that require mutually exclusive access to a non-preemptive, shared resource. Each task is composed of an infinite sequence of identical jobs; a new job of a task arrives immediately upon completion of the previous job. Job durations are stochastic, but are assumed independent.

We have developed problem models and solution algorithms for scheduling policy design in this setting, with the objective of maximizing proportional fairness according to user-specified resource utilization targets u_i for each task J_i in $(J_i)_{i=1}^n$. We enforce timeliness of task execution by minimizing long-term cost, defined as an increasing function of the task lag $|tu_i - x_i(t)|$, where $x_i(t)$ is the cumulative resource usage of task J_i at time t. Thus, our algorithms find scheduling policies that maintain each task's resource usage near the specified share target under non-preemptive execution constraints and in spite of uncertainty in job execution time.

We have modeled this scheduling problem as a Markov Decision Process (MDP) with an infinite state space, unbounded costs, and with decision epochs synchronous with task completion. We have proven that despite the size of the state space and unbounded costs, this process has nontrivial optimal[6], deterministic control policies in the infinite-horizon discounted reward setting. We have shown that for a family of periodic cost functions, including the sum of task lags (*i.e.* the L_1 cost function $c_1(\mathbf{x}(t)) = \sum_{i=1}^n |x_i(t) - tu_i|$), we can eliminate infinitely many MDP states from

[6]Optimality in that these policies maximize long-term discounted rewards, or equivalently, minimize long-term discounted costs.

consideration without sacrificing optimality, provided that the vector of utilization targets $\mathbf{u}$ has only rational components.

The wrapped model induced by periodicity remains infinite with unbounded costs, thus further techniques were needed in order to compute or approximate optimal scheduling policies. We presented two approximation methods that exploit the observation that only finitely many states with have cost less than any chosen, finite bound in the wrapped state model. Our bounded state model restricts attention to states with cost less than a given bound. As desired, the bounded model solution converges to the true optimal scheduling policy as the cost bound, and thus the size of the approximate model, increases. Further, by penalizing boundary conditions appropriately, we are guaranteed to find the best policies that stay within the specified bounds, which under L_1 costs corresponds directly to bounding the maximum lag among all tasks at all times.

Using the bounded state model to design scheduling policies potentially requires solving many related MDPs until a satisfactory policy is found that respects suitably small cost bounds. We have developed an algorithm, Expanding State Policy Iteration (ESPI), that automates this process by iteratively constructing minimal subsets of the state space that are necessary to perform policy evaluation and improvement about the initial MDP state. While we have not been able to prove that this method terminates in general, we are able to guarantee that it strictly improves its policy at each iteration. We generalized this algorithm to allow varying degrees of lookahead in order to ensure better coverage of the state space.

Both of these methods rely on enumeration of a finite subset of the MDP state space. Even with state space compression due to periodicity, this space grows exponentially with the number of tasks. Thus, we developed a class of conic scheduling policies that approximate the geometry of the state space partitions induced by finite-state approximation methods. These policies can be efficiently parameterized with $\Theta(n^2)$ parameters. We have verified the efficacy of this approach in simulation.

Additionally, we have provided a PAC bound on the sample complexity of reinforcement learning effective scheduling policies when task duration distributions are not provided in advance; this result is novel, as existing results are confined to learning MDPs with finitely many states and bounded rewards, and so do not cover our task

scheduling MDP. Our empirical results in this case indicate that there is no need to incorporate an explicit exploration mechanism, as the structure of the task scheduling MDP enforces rational exploration.

Open Questions: Two open questions remain in our analysis of finite-state approximation methods. First, if we identify a fixed initial state or, more generally, a finite set of initial states, how many wrapped model states may an optimal policy reach? If some optimal policy has finite closure, *i.e.*, there are only finitely many states reachable when starting from a finite set of states, then it follows that some cost bound captures the optimal exactly. This result may also be useful in determining the convergence properties of ESPI. The second open question is whether or not the greedy policy is optimal in the two task setting. This hypothesis is consistent with our empirical results. If true, it seems reasonable that an n steps of lookahead may suffice to attain optimal performance in the n task setting, which could lead to more efficient solution methods.

One of the most optimistic assumptions we have made with our system model is that jobs are independent of one another. While this is a common assumption in real-time systems with repeating tasks [58, 59], it is unlikely to hold in practice. For example, repeated uses of a robot actuator are not independent, since the duration of a job depends on the state of the actuator upon acquisition. Although we could enforce independence by requiring that each job must return the actuator to a fixed reference state prior to completion, this would unnecessarily increase job durations. We have performed some preliminary research on identifying and exploiting system modes [35] that provides an interesting avenue to address this concern.

We have omitted deadlines from our discussion of task timeliness. We have observed that if we associate deadlines with jobs as a function of worst-case execution time, we achieve better deadline miss rates than non-preemptive EDF when overload situations are likely. Explicitly incorporating deadlines into our model may allow us to further improve performance under these conditions. However, this would require encoding additional information into our state representation; it seems likely that state reduction techniques similar to those used in this work could then be employed to reduce the size of the resulting model.

References

[1] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *RTSS'98: Proceedings of the 1998 Real-Time Systems Symposium*, pages 4–13, Washington DC, USA, 1998. IEEE Computer Society.

[2] Luca Abeni and Giorgio Buttazzo. Resource reservations in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, 2004.

[3] James H. Anderson and Anand Srinivasan. Pfair scheduling: Beyond periodic task systems. In *RTCSA'00: Proceedings of the 7th International Conference on Real-Time Systems and Applications*, pages 297–306, Washington DC, USA, 2000. IEEE Computer Society.

[4] James H. Anderson and Anand Srinivasan. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. *Journal of Computer and System Sciences*, 68(1):157–204, 2004.

[5] Alia Atlas and Azer Bestavros. Statistical rate monotonic scheduling. In *RTSS'98: Proceedings of the 1998 Real-Time Systems Symposium*, pages 123–132, Washington DC, USA, 1998. IEEE Computer Society.

[6] Alia Atlas and Azer Bestavros. Statistical rate monotonic scheduling. Technical Report 1998-010, Boston University, Boston, MA, USA, 1998.

[7] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[8] Peter Auer, Thomas Jaksch, and Ronald Ortner. Near-optimal regret bounds for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 21, pages 89–96, 2009.

[9] Peter Auer and Ronald Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 19, pages 49–56. MIT Press, 2007.

[10] Sanjoy K. Baruah. The non-preemptive scheduling of periodic tasks upon multiprocessors. *Real-Time Systems*, 32(1–2), 2006.

[11] Sanjoy K. Baruah and Samarjit Chakraborty. Schedulability analysis of non-preemptive recurring real-time tasks. In *International Parallel and Distributed*

Processing Symposium, pages 149–156, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[12] Sanjoy K. Baruah, N. K. Cohen, C. Greg Plaxton, and Donald A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[13] Sanjoy K. Baruah, Johannes Gehrke, and C. Greg Plaxton. Fast scheduling of periodic tasks on multiple resources. In *IPPS'95: Proceedings of the 9th International Symposium on Parallel Computing*, pages 280–288, Washington DC, USA, 1995. IEEE Computer Society.

[14] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[15] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *IJCAI '95: Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1636–1642, 1995.

[16] Craig Boutilier, Ronen I. Brafman, and Christopher Geib. Structured reachability analysis for markov decision processes. In *Proceedings of the 14th International Conference on Uncertainty in Artificial Intelligence*, pages 24–32, San Francisco, California, USA, 1998. Morgan Kaufmann.

[17] Craig Boutilier, Richard Dearden, and Moisés Goldzsmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.

[18] Justin Boyan and Andrew Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems*, volume 7, pages 369–376, 1995.

[19] Ronen I. Brafman and Moshe Tennenholtz. R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2003.

[20] Giorgio Buttazzo and Enrico Bini. Optimal dimensioning of a constant bandwidth server. In *RTSS'06: Proceedings of the 2006 Real-Time Systems Symposium*, pages 169–177, Washington DC, USA, 2006. IEEE Computer Society.

[21] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems*. Springer, 2004.

[22] Hyeonjoong Cho, Binoy Ravindran, and Douglas E. Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *RTSS'06: Proceedings of the 2006 Real-Time Systems Symposium*, pages 101–110, Washington DC, USA, 2006. IEEE Computer Society.

[23] Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 8, pages 1017–1023. MIT Press, 1996.

[24] Thomas Dean and Robert Givan. Model minimization in markov decision processes. In *AAAI '97: Proceedings of the 14th National Conference on Artificial Intelligence*, pages 1006–1111. AAAI, 1997.

[25] Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann E. Nicholson. Planning with deadlines in stochastic domains. In *AAAI '93: Proceedings of the 11th National Conference on Artificial Intelligence*, pages 574–579, 1993.

[26] UmaMaheswari C. Devi and James H. Anderson. Improved conditions for bounded tardiness under EPDF fair multiprocessor scheduling. *Journal of Computer and System Sciences*, 75(7):388–420, 2009.

[27] Bogdin Doytchinov, John Lehoczky, and Steven Shreve. Real-time queues in heavy traffic with earliest-deadline-first queue discipline. *The Annals of Applied Probability*, 11(2):332–378, 2001.

[28] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Pac bounds for multi-armed bandit and markov decision processes. In *COLT '02: Proceedings of the 15th Annual Conference on Computational Learning Theory*, pages 255–270, London, UK, 2002. Springer-Verlag.

[29] Eyal Even-Dar and Yishay Mansour. Convergence of optimistic and incremental Q-learning. In *Advances in Neural Information Processing Systems*, volume 13, pages 1499–1506, 2001.

[30] Norm Ferns, Pablo Samuel Castro, Doina Precup, and Prakash Panangaden. Methods for computing state similarity in markov decision processes. In *UAI'06, Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence*, 2006.

[31] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In *UAI'04, Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 162–169, 2004.

[32] Claude-Nicolas Fiechter. Expected mistake bound model for on-line reinforcement learning. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 116–124, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[33] Michael C. Fu. Stochastic gradient estimation. In S. G. Henderson and B. L. Nelson, editors, *Handbooks in Operations Research and Management Science: Simulation*, pages 575–616. Elsevier Science Publishers, Ltd., 2006.

[34] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, 2003.

[35] Robert Glaubius, Terry Tidwell, Christopher Gill, and William D. Smart. Scheduling design with unknown execution time distributions or modes. Technical Report WUCSE-2009-15, Washington University in St. Louis, 2009.

[36] Robert Glaubius, Terry Tidwell, Christopher Gill, and William D. Smart. Scheduling policy design for autonomic systems. *International Journal on Autonomous and Adaptive Communications Systems*, 2(3):276–296, 2009.

[37] Robert Glaubius, Terry Tidwell, Braden Sidoti, David Pilla, Justin Meden, Christopher Gill, and William D. Smart. Scalable scheduling policy design for open soft real-time systems. Technical Report WUCSE-2009-71, Washington University in St. Louis, 2009.

[38] Robert Glaubius, Terry Tidwell, William D. Smart, and Christopher Gill. Scheduling design and verification for open soft real-time systems. In *RTSS'08: Proceedings of the 2008 Real-Time Systems Symposium*, pages 505–514, Washington DC, USA, 2008. IEEE Computer Society.

[39] Geoffrey J. Gordon. Stable function approximation in dynamic programming. In *ICML '95: Proceedings of the 12th International Conference on Machine Learning*, pages 261–268, 1995.

[40] Nan Guan, Wang Yi, Zonghua Gu, Qingxu Deng, and Ge Yu. New schedulability test conditions for non-preemptive scheduling on multiprocessor platforms. In *RTSS'08: Proceedings of the 2008 Real-Time Systems Symposium*, pages 137–146, Washington DC, USA, 2008. IEEE Computer Society.

[41] Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, Massachusetts, USA, 1960.

[42] Kevin Jeffay and Steve Goddard. A theory of rate-based execution. In *RTSS'99: Proceedings of the 1999 Real-Time Systems Symposium*, pages 304–314, Washington DC, USA, 1999. IEEE Computer Society.

[43] Kevin Jeffay, D. Stanat, and C. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *RTSS'91: Proceedings of the 1991 Real-Time Systems Symposium*, pages 129–139, 1991.

[44] Leslie Pack Kaelbling, Michael Littman, and Andrew Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[45] Sham M. Kakade. A natural policy gradient. In *In Advances in Neural Information Processing Systems*, volume 14, 2002.

[46] Sham M. Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, London, UK, 2003.

[47] Mehdi Kargahi and Ali Movaghar. A method for performance analysis of earliest-deadline-first scheduling policy. *Journal of Supercomputing*, 37(2):197–222, 2006.

[48] Michael J. Kearns and Satinder P. Singh. Near-optimal reinforcement learning in polynomial time. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 260–268. Morgan Kaufmann Publishers Inc., 1998.

[49] Michael J. Kearns and Satinder P. Singh. Finite-sample convergence rates for Q-learning and indirect algorithms. In *Advances in Neural Information Processing Systems*, volume 11, pages 996–1002, Cambridge, Massachusetts, USA, 1999. MIT Press.

[50] Michael J. Kearns and Satinder P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 2-3(49):209–232, 2002.

[51] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, Massachusetts, USA, 1994.

[52] Nate Kohl and Peter Stone. Machine learning for fast quadrupedal locomotion. In *AAAI '04: Proceedings of the 19th National Conference on Artificial Intelligence*, pages 611–616, 2004.

[53] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[54] Terran Lane and Andrew Wilson. Toward a topological theory of relational reinforcement learning for navigational tasks. In *Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2005)*, 2005.

[55] J. P. Lehoczky. Real-time queueing theory. In *RTSS'96: Proceedings of the 1996 Real-Time Systems Symposium*, pages 186–195, Washington DC, USA, 1996. IEEE Computer Society.

[56] Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for MDPs. In *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, pages 531–539, 2006.

[57] Peng Li, Hyeonjoong Cho, Binoy Ravindran, and Douglas E. Jensen. Stochastic, utility accrual real-time scheduling with task-level and system-level timeliness assurances. In *ISORC'05: Proceedings of the 8th IEEE International Symposium*

on Object-Oriented Real-Time Distributed Computing, pages 216–223, Washington DC, USA, 2005. IEEE Computer Society.

[58] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[59] Jane W. S. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[60] Chenyang Lu, John A. Stankovic, Gang Tao, and Sang H. Son. Design and evaluation of a feedback control EDF scheduling algorithm. In *RTSS'99: Proceedings of the 1999 Real-Time Systems Symposium*, pages 56–67, Washington DC, USA, 1999. IEEE Computer Society.

[61] Chenyang Lu, John A. Stankovic, Gang Tao, and Sang H. Son. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23(1/2):85–126, 2002.

[62] Shie Mannor and John N. Tsitsiklis. The sample complexity of exploration in the multi-armed bandit problem. *Journal of Machine Learning Research*, 5:623–648, 2004.

[63] Sorin Manolache, Petru Eles, and Zebo Peng. Schedulability analysis of applications with stochastic task execution times. *ACM Transactions on Embedded Computing Systems*, 3(4):706–735, 2004.

[64] Andrew McCallum. Overcoming incomplete perception with utile distinction memory. In *ICML '93: Proceedings of the 10th International Conference on Machine Learning*, pages 190–196, 1993.

[65] Volodymyr Mnih, Csaba Szepesvári, and Jean-Yves Audibert. Empirical bernstein stopping. In *ICML '08: Proceedings of the 25th International Conference on Machine Learning*, pages 672–679, New York, NY, USA, 2008. ACM.

[66] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. PhD thesis, Cambridge, MA, USA, 1983.

[67] Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9:815–857, 2008.

[68] Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*, 2004.

[69] Andrew Y. Ng and Michael I. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *UAI'00, Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 406–415, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[70] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 16, pages 799–806. MIT Press, 2004.

[71] Jan Peters and Stefan Schaal. Policy gradient methods for robots. In *IEEE/RSJ Conference on Intelligent Robots and Systems*, pages 2219–2225, 2006.

[72] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994.

[73] Martin L. Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11):1127–1137, July 1978.

[74] Balaraman Ravindran and Andrew G. Barto. Model minimization in hierarchical reinforcement learning. In *Proceedings of the 5th International Symposium on Abstraction, Reformulation, and Approximation (SARA 2002)*, pages 196–211, 2002.

[75] Balaraman Ravindran and Andrew G. Barto. SMDP homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes. In *IJCAI '03: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1011–1018, 2003.

[76] G.A. Rummery and N. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.

[77] John Rust. Numerical Dynamic Programming in Economics. In H. M. Amman, D. A. Kendrick, and J. Rust, editors, *Handbook of Computational Economics*, volume 1, chapter 14, pages 619–729. Elsevier Science Publishers Ltd., 1996.

[78] Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28(2-3):101–155, 2004.

[79] David Silver, Richard Sutton, and Martin Müller. Reinforcement learning of local shape in the game of Go. In *IJCAI '07: Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1053–1058, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[80] Satinder P. Singh and Richard C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994.

[81] William D. Smart and Leslie Pack Kaelbling. Practical reinforcement learing in continuous spaces. In *ICML '00: Proceedings of the 17th International Conference on Machine Learning*, pages 903–910, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[82] Anand Srinivasan and James H. Anderson. Efficient scheduling of soft real-time applications on multiprocessors. *Journal of Embedded Computing*, 1(2):285–302, 2005.

[83] Anand Srinivasan and James H. Anderson. Optimal rate-based scheduling on multiprocessors. *Journal of Computer and Systems Science*, 72(6):1094–1117, 2006.

[84] Alexander L. Strehl and Michael L. Littman. An empirical evaluation of interval estimation for markov decision processes. In *ICTAI '04: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 128–135, Washington, DC, USA, 2004. IEEE Computer Society.

[85] Alexander L. Strehl and Michael L. Littman. A theoretical analysis of model-based interval estimation. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 856–863, New York, NY, USA, 2005. ACM.

[86] Alexander L. Strehl and Michael L. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

[87] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, USA, 1998.

[88] Richard S. Sutton, David McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12, pages 1057–1063, 2000.

[89] Csaba Szepesvári. Reinforcement learning algorithms for MDPs. Technical Report TR09-13, University of Alberta, Edmonton, 2009.

[90] Csaba Szepesvári and Rémi Munos. Finite time bounds for sampling based fitted value iteration. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 880–887, 2005.

[91] István Szita and András Lőrincz. The many faces optimism: a unifying approach. In *ICML '08: Proceedings of the 25th International Conference on Machine Learning*, pages 1048–1055. ACM, 2008.

[92] Jonathan Taylor, Doina Precup, and Prakash Panangaden. Bounding performance loss in approximate MDP homomorphisms. In *Advances in Neural Information Processing Systems*, pages 1649–1656, 2008.

[93] Gerald Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[94] Terry Tidwell, Robert Glaubius, Christopher Gill, and William D. Smart. Scheduling for reliable execution in autonomic systems. In *Proceedings of the 5th International Conference on Autonomic and Trusted Computing (ATC 2008)*, volume 5060 of *Lecture Notes in Computer Science*, pages 149–161, 2008.

[95] L. G. Valiant. A theory of the learnable. In *STOC '84: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 436–445. ACM, 1984.

[96] Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[97] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[98] Ronald J. Williams. Simple statistical gradient-following algorithms for connectioninst reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

[99] Wei Zhang and Thomas G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI '95: Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1114–1120. Morgan Kaufmann, 1995.