DETERMINISTIC AND STOCHASTIC MODELS FOR PRACTICAL SCHEDULING PROBLEMS

by

Elvin Coban

Submitted to the Tepper School of Business
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in
Operations Management & Manufacturing

CARNEGIE MELLON UNIVERSITY

December 2012

Dissertation Committee:

Professor Alan Scheller-Wolf, (Chair)
Dr. Aliza R. Heching
Professor John N. Hooker
Assistant Professor Willem-Jan van Hoeve

Abstract

This dissertation analyzes three scheduling problems motivated by real life situations. In many manufacturing and service industries, scheduling is an important decision-making process. Unfortunately, scheduling problems are often computationally challenging to solve, and even modeling a scheduling problem can be difficult.

Chapter 2 considers single-facility non-preemptive scheduling problems with long time horizons having jobs with time windows (i.e., release times and due dates). I combine constraint programming (CP) and mixed integer linear programming (MILP) using a hybrid method: logic-based Benders decomposition. I first divide the long time horizon into segments to make the problem tractable. This gives rise to two versions of the single-facility scheduling problem: *segmented* and *unsegmented*. In the *segmented* problem, each job must be completed within one time segment. In the *unsegmented* problem, jobs can overlap two or more segments. I analyze different objective functions, and introduce relevant Benders cuts. I find that for the segmented problem, logic-based Benders decomposition is always superior and should be used from the start. For the unsegmented problem, I find that logic-based Benders decomposition is not necessarily the fastest method, but clearly the most robust. Hence, I suggest a strategy of applying CP first, and if it fails to solve the problem within a few seconds, switching to logic-based Benders decomposition.

Chapter 3 addresses the problem of staffing in a service center with cross-trained agents, heterogeneous customers, and quality guarantees, inspired by a large IT services delivery organization. Agents are either high or low skilled, and serve customer requests that are also heterogeneous - with respect to both their complexity and their priority: (i) Higher priority customer requests preempt lower priority customer requests in the queue; and (ii) Less skilled agents can only service low complexity requests, while highly skilled agents can service all types of requests. I capture this service center's operations using a multi-server queue under a preemptive-resume priority service discipline, and model it as a Markov chain. I then apply approximation and bounding techniques to evaluate different control policies: I consider the class of threshold-based priority policies that prioritize the requests according to the number of each class of requests in the system. Four different types of customer requests can be successfully analyzed by our method: our method is accurate and fast when compared to simulation. I also provide managerial insights about the capacity provisioning problem at the service center, such as how to set thresholds, how to negotiate costs with customers, and what mix of agents to hire. I demonstrate that threshold-based policies can be simple but effective, and that they might decrease the total cost without any changes in the service center.

In Chapter 4, I analyze a project scheduling problem with disruptions, cross-trained agents, heterogeneous projects and quality guarantees where the durations of a project's tasks/disruptions and the arrivals of the projects are uncertain. This problem is inspired by a real problem at a large, global SDO's service center. Analysis of real data shows that considering delays in the release times and processing times are sufficient to capture the uncertainty in the system. The goal is to find a robust and effective assignment and schedule of tasks for each agent. I develop a robust scheduling model based on logic-based Benders decomposition, in which uncertainty is captured by an uncertainty set which might not include the least likely outcomes. Then, I simplify this robust scheduling model by the convexity theorem I prove.

Acknowledgements

I would like to express my deepest gratitude to my advisor Dr. Alan Scheller-Wolf for his wonderful guidance over the years, his patience, and his generous help with my professional development. I would like to express my sincere gratitude to Dr. Aliza R. Heching and Dr. John N. Hooker for their invaluable guidance, inspiration, and encouragement. This thesis would not be possible without Dr. Scheller-Wolf, Dr. Heching, and Dr. Hooker's endless support. I also would like to thank Dr. Willem-Jan van Hoeve for his support, insightful suggestions, his time and effort in evaluating my work. It has been an honor working with such prominent figures in the Operations Management and the Operations Research communities.

I am very thankful to Çiğdem Baybars and Dr. İlker Baybars for their hospitality and morale support throughout the years. I am also indebted to several friends whose insightful discussions contributed to my research. Many friends have made my stay at Carnegie Mellon University enjoyable. These include Alp Akcay, Ishani Aggarwal, Negar Soheili Azad, David Bergman, Andre A. Cire, Xin Fang, E. Begum Gulsoy, Canan Gunes, Gizem Korpeoglu, Ersin Korpeoglu, Musab Kurnaz, Carla Michini, Marco Molinaro, Emre Nadar, Selvaprabu Nadarajah, Charudatta Phatak, Nandana Sengupta, Nazli Turan, and Yangfang (Helen) Zhou. The thoughts we exchanged, and the laughter we shared made these Ph.D. years fun and memorable. I would like to express my sincere thanks to my dear friends Merve Peyic and Ece Erkol for their endless friendship and support.

I also would like to thank our Ph.D. coordinator Lawrence Rapp for making life easier for me at Carnegie Mellon University.

I owe everything to the unconditional love and support from my family. I am very grateful to my mother Mehbüp Çoban, my father Ali Çoban and my sister Güler Çoban Çicek for the concern, caring, endless love and support they provided me throughout my life. My nephews Kaan Çicek and Berk Çicek bring joy to my life and I would like to thank them as they always remind me that life is beautiful and amazing. I am especially thankful to M. Sarper Göktürk. I probably would not survive the Ph.D. without his love, encouragement, inspiration and support, which cheered me up during the most difficult times. He has also been a great colleague with whom I had the privilege to discuss research and I feel very lucky to have him in my life.

TABLE OF CONTENTS

List of Tables

List of Figures

1 INTRODUCTION

Three scheduling problems motivated by real life situations are analyzed in this dissertation. In many manufacturing and service industries, scheduling is an important decision-making process: how to optimize the allocation of resources to tasks over given time periods. Different objective functions can be optimized, such as finding a feasible solution, minimizing makespan (finishing time of the last scheduled task), and minimizing total cost. If a task cannot be finished on time, a penalty can be charged both in the form of loss of goodwill and proportional to the tardiness of the delivery. However, solving scheduling problems is nontrivial; in fact, since scheduling problems are computationally hard to solve, even modeling a scheduling problem can be difficult.

Inspired by the importance of scheduling in real life, I study three practical scheduling problems. In Chapter 2, I solve single-facility non-preemptive scheduling problems over a long time horizon, with tasks with deterministic durations but different release times and deadlines. Chapter 2 contributes to the scheduling literature by being the first to model the single-facility scheduling model with long time horizons using a hybrid method - logic-based Benders decomposition- and by providing insights by comparing the performance of logic-based Benders decomposition with state-of-art models. In Chapter 3, I analyze a service center at a large, global, IT services delivery organization (SDO) with cross-trained agents, heterogeneous customer requests, and service level agreements. I model the system as a multi-server queueing system and apply approximation and bounding techniques to evaluate different control policies. Chapter 3 contributes to the queueing and service science literature by demonstrating a simple but effective request-assignment policy, and providing managerial insights about the potential benefits of a such policy. In Chapter 4, I study a similar problem as Chapter 3: a project scheduling problem at a large IT SDO with cross-trained agents, heterogeneous projects, and service quality guarantees. Durations of projects' tasks and arrival times are uncertain. In addition, projects are subject to random disruptions. Chapter 4 contributes to the scheduling literature by developing a novel robust scheduling model using uncertainty sets, and by characterizing the robust model under polyhedral uncertainty sets.

Chapter 2 considers single-facility non-preemptive scheduling problems with long time horizon having jobs with time windows (i.e., release times and due dates). I combine constraint programming (CP) and mixed integer linear programming (MILP) using a hybrid method: logic-based Benders decomposition. I first divide the long time horizon into segments, for example, an annual production plan is separated into weekly plans. Thus, rather than break the problem into facility assignment and job scheduling, I decompose it into smaller scheduling subproblems on segments of the time horizon. The master problem assigns jobs to time segments, and subproblems schedule jobs within each segment. This allows us to deal with long time horizons that would otherwise make the problem intractable.

I consider two versions of the single-facility scheduling problem: *segmented* and *unsegmented*. In the *segmented* problem, each job must be completed within one time segment. The boundaries between segments might therefore be regarded as weekends or shutdown times during which jobs cannot be processed. In the *unsegmented* problem, jobs can overlap two or more segments. I analyze different objective functions: finding a feasible solution, minimizing makespan, or minimizing total tardiness, and introduce relevant Benders cuts.

I find that for these problems, logic-based Benders scales up more effectively than state-of-the-art CP and MILP solvers, especially for the segmented problem. The logic-based Benders decomposition solves much larger instances of the feasibility and makespan problems, and its speed advantage increases rapidly as the problem size increases. For the unsegmented problem, logic-based Benders decomposition continues to dominate MILP while being much slower than CP on most of the smaller instances. However, CP begins to lose its ability to solve instances as they scale up, whereas logic-based Benders decomposition continues to solve them. Logic-based Benders decomposition is therefore not necessarily the fastest method, but clearly the most robust. Because CP solves unsegmented instances quickly if it solves them at all, I suggest a strategy of applying CP first, and if it fails to solve the problem within a few seconds, switching to logic-based Benders decomposition. For segmented instances, logic-based Benders decomposition is always superior and should be used from the start. To our knowledge, logic-based Benders decomposition has not previously been applied to a single-facility scheduling with a long time horizon. I show that using logic-based Benders decomposition is particularly useful for these problems.

Chapter 3 addresses the problem of staffing in a service center with cross-trained agents, heterogeneous customers, and quality guarantees. I model a system, inspired by a large IT SDO, with agents who are either high or low skilled. These agents serve customer requests that are also heterogeneous - with respect to both the complexity and their priority: (i) Higher priority customer requests preempt lower priority customer requests in the queue; and (ii) Low skilled agents can only service low complexity requests, while high skilled agents can service all types of requests. I capture this service center's operations using a multi-server queue under a preemptive-resume priority service discipline, and model this system as a Markov chain. However, as an exact solution of such a system is numerically intractable, I apply approximation and bounding techniques to evaluate different control policies.

I consider the class of threshold-based priority policies that prioritize the requests according to the number of each class of requests in the system. By approximating the interval of time during which the agent is busy without interruption, I turn the intractable Markov chain into a tractable one, allowing application of standard Matrix Analytic Methods. I show that four different types of customer requests can be successfully analyzed by our method: our method is accurate and fast when compared to simulation. I also provide managerial insights about the capacity provisioning problem at the service center. To the best of our knowledge this chapter is the first to consider the following combination of factors within a service center model: (i) agents are differentiated according to their level of expertise, which may limit the specific classes of requests they can serve; (ii) requests require different skill levels and also may belong to different priority classes; higher priority class requests can preempt lower priority class requests; (iii) threshold policies are utilized; and (iv) the system does not operate under a heavy traffic regime. I contribute to the queueing and service science literature by demonstrating a simple but effective request-assignment policy, and providing managerial insights, such as, how to set thresholds, how to negotiate costs with customers, and what mix of agents to hire.

In Chapter 4, I analyze a project scheduling problem with disruptions, cross-trained agents, heterogeneous projects and quality guarantees where the durations of a project's tasks and the arrivals of the projects are uncertain. This problem is inspired by a real problem at a large, global SDO's service center. Analysis of real data shows that considering delays in the release times and processing times are sufficient to capture the uncertainty: Since the number of projects in the service center is very large, ignoring uncertain arrivals of the projects and solving the scheduling problem with the already existing projects in the service center on a rolling basis represent the service center's operations well. The goal is to find a robust and effective assignment and schedule of tasks for each agent. We do this using a robust scheduling model, in which uncertainty is captured by an uncertainty set which might not include the least likely outcomes.

I develop my robust scheduling model based on logic-based Benders decomposition. This hybrid model has three stages: The first stage is the master problem that finds the assignment of tasks to agents. The second and the third stages are the subproblems scheduling these assigned tasks for each agent considering the uncertainty set. For the sake of illustration, I consider only the case in which delays in the release times are uncertain, as disruptions are typically much more variable than the processing times. I prove that the objective function is convex in terms of these delays when the uncertainty set is a polyhedron. Interestingly, the three-stage decomposition above is simplified into a two-stage decomposition by using the convexity analysis I prove. Chapter 4 contributes to the scheduling literature by introducing a novel robust scheduling model in which the uncertainty is represented by an uncertainty set. I also simplify logic-based Benders decomposition under polyhedral uncertainty sets. To the best of our knowledge, this is the first work solving a project scheduling problem with uncertainty using robust optimization with uncertainty sets.

My dissertation widens our knowledge of deterministic and stochastic models for three practical scheduling problems. To our knowledge, the second chapter is the first attempt to solve a pure scheduling problem (including any side constraints) with logic-based Benders decomposition. In the third chapter, I contribute an approximation method to provide managerial insights for a service center's operations with cross-trained agents, heterogeneous customers and quality guarantees. Current numerical experiments show that the approximation method is accurate and fast. In Chapter 4, I contribute a novel robust project scheduling model to capture an SDO's service center's operations with disruptions, cross-trained agents, heterogeneous projects and quality guarantees. There is much future work to be done in each of these problem domains but I am confident that the findings of this dissertation will guide future studies on similar, or even more complex problems.

2 SINGLE-FACILITY SCHEDULING BY LOGIC-BASED BENDERS DECOMPOSITION

Logic-based Benders decomposition can combine mixed integer programming and constraint programming to solve planning and scheduling problems much faster than either method alone. We find that a similar technique can be beneficial for solving pure scheduling problems as the problem size scales up. We solve single-facility non-preemptive scheduling problems with time windows and long time horizons. The Benders master problem assigns jobs to predefined segments of the time horizon, where the subproblem schedules them. In one version of the problem, jobs may not overlap the segment boundaries (which represent shutdown times, such as weekends), and in another version, there is no such restriction. The objective is to find feasible solutions, minimize makespan, or minimize total tardiness.

2.1 Introduction

Logic-based Benders decomposition (Hooker and Ottosson, 2003) is a generalization of Benders decomposition that accommodates a much wider range of problems. In contrast with the classical Benders method, the subproblem can in principle be any combinatorial problem, not necessarily a linear or nonlinear programming problem. For example, it can be a scheduling problem solved by constraint programming (CP), a method well suited to scheduling.

This flexibility has led to the application of logic-based Benders decomposition to planning and scheduling problems that naturally decompose into an assignment and a scheduling portion. The Benders master problem assigns jobs to facilities using

This chapter is joint work with John N. Hooker.

mixed integer programming (MILP), and the subproblem uses CP to schedule jobs on each facility. This approach can reduce solution time by several orders of magnitude relative to methods that use MILP or CP alone (Hooker, 2004, 2005b,a, 2006, 2007a; Jain and Grossmann, 2001; Thorsteinsson, 2001).

In this paper, we investigate whether a similar technique can solve pure scheduling problems, which lack the obvious decomposition that one finds in planning and scheduling. Rather than break the problem into facility assignment and job scheduling, we decompose it into smaller scheduling subproblems on segments of the time horizon. The master problem assigns jobs to time segments, and subproblems schedule jobs within each segment. This allows us to deal with long time horizons that would otherwise make the problem intractable.

In particular, we solve single-facility non-preemptive scheduling problems with time windows in which the objective is to find a feasible solution, minimize makespan, or minimize total tardiness. In one version of the problem, which we call the *segmented* problem, each job must be completed within one time segment. The boundaries between segments might therefore be regarded as weekends or shutdown times during which jobs cannot be processed. In a second version of the problem, which we refer to as *unsegmented*, jobs can overlap two or more segments. We address both variants.

Obvious cases of single-facility scheduling include machine scheduling in a manufacturing plant or task scheduling in a computer with one processor. Less obvious cases occur when a complex plant is scheduled as one facility, as for example when a paint manufacturing plant must be committed to produce one color at a time (French, 1982). A multistage process with a single bottleneck may also result in a single-facility model (Pinedo, 1995).

The paper is organized as follows. After a survey of previous work, we present a brief introduction to logic-based Benders decomposition, and state pure MILP and

CP models for the single-facility scheduling problem. We then describe a logic-based Benders approach to the segmented feasibility, makespan, and tardiness problems. We do the same for the unsegmented feasibility and makespan problems, for which the Benders cuts are considerably more complex. We then present computational results. We conclude that the relative advantage of the Benders approach increases rapidly as the time horizon and number of jobs grow larger, particularly for segmented feasibility and makespan problems. The Benders method is not necessarily faster on unsegmented instances, but it is more robust and the only method to solve them all.

2.2 Previous Work

Logic-based Benders decomposition was introduced in (Hooker, 1995; Hooker and Yan, 1995), and a general theory was presented in (Hooker, 2000; Hooker and Ottosson, 2003). Application to planning and scheduling was proposed in (Hooker, 2000) and first implemented in (Jain and Grossmann, 2001).

Classical Benders decomposition derives Benders cuts from dual or Lagrange multipliers in the subproblem (Benders, 1962; Geoffrion, 1972). However, this presupposes that the subproblem is a linear or nonlinear programming problem. Logic-based Benders decomposition has the advantage that Benders cuts can, at least in principle, be obtained from a subproblem of any form by solving its *inference dual* (Hooker, 1996). The solution of the dual is a *proof* of optimality for fixed values of the master problem variables (whence the name "logic-based"). The core idea of Benders decomposition is that *this same proof* may establish a bound on the optimal value when the master problem variables take other values. The corresponding Benders cut enforces this bound in the master problem.

Logic-based Benders cuts must be designed specifically for each class of problems, but this provides an opportunity to exploit problem structure. The Benders framework is also natural for combining MILP and CP, because one method can be used to solve the master problem and the other the subproblem. This is particularly advantageous when the subproblem is a scheduling problem, for which CP methods are well suited (Baptiste et al., 2001; Hooker, 2007b). The combinatorial nature of the scheduling problem is no longer a barrier to generating Benders cuts.

Logic-based Benders cuts have some resemblance to cuts generated in oracle-based optimization (e.g., (Babonneau et al., 2007)) but differ in several respects. The Benders subproblem contains a different set of variables than the master problem and cuts, while in oracle-based optimization, the subproblem and cuts contain the same variables. The solution of the master problem, rather than a query point, defines the Benders subproblem, and Benders cuts can in principle be derived from any proof of optimality for the subproblem.

Additional applications of logic-based Benders include logic circuit testing (Hooker and Yan, 1995), propositional satisfiability (Hooker and Ottosson, 2003), multi-stage facility scheduling (Harjunkoski and Grossmann, 2002), dispatching of automated guided vehicles (Corréa et al., 2004), steel production scheduling (Harjunkoski and Grossmann, 2001), real-time scheduling of computer processors (Cambazard et al., 2004), traffic diversion (Chu and Xia, 2004), batch scheduling in a chemical plant (Maravelias and Grossmann, 2004a,b) (and, in particular, polypropylene batch scheduling (Timpe, 2002)), stochastic constraint programming (Tarim and Miguel, 2006), customer service with queuing (Terekhov et al., 2005), and scheduling of distributed processors for computation (Benini et al., 2005; Cambazard et al., 2004).

In all of these applications, the subproblem is a feasibility problem rather than an optimization problem, which simplifies the task of designing Benders cuts. Effective Benders cuts can nonetheless be developed for optimization subproblems, based on somewhat deeper analysis of the inference dual. This is accomplished for planning and scheduling problems in (Hooker, 2004, 2005b,a, 2006, 2007a), where the objective is to minimize makespan, the number of late jobs, or total tardiness. The subproblem is a cumulative scheduling problem, in which several jobs may be processed simultaneously subject to resource constraints.

Other applications of logic-based Benders to optimization include 0-1 programming (Hooker and Ottosson, 2003; Chu and Xia, 2004), mixed integer/linear programming (Codato and Fischetti, 2006), tournament scheduling (Rasmussen, 2008; Rasmussen and Trick, 2007), location/allocation problems (Fazel-Zarandi and Beck, 2009), shift selection with task sequencing (Barlatta et al., 2010), single- and multi-stage batch chemical processes (Maravelias, 2006), multimachine assignment scheduling with a branch-and-cut approach (Sadykov and Wolsey, 2006), and single-facility scheduling in the present paper. Temporal decomposition similar to that employed here is applied in (Bent and Hentenryck, 2010) to large-scale vehicle routing problems with time windows. However, unlike a Benders method, the algorithm is heuristic and does not obtain provably optimal solutions. In addition, no cuts or nogoods are generated.

To our knowledge, logic-based Benders decomposition has not previously been applied to single-facility scheduling. A broad survey of single-facility scheduling methods for minimizing tardiness can be found in (Koulamas, 2010), which assumes that all release dates are equal. Four MILP formulations of single-facility scheduling are analyzed in (Keha et al., 2009), where it is observed that the discrete-time model is most widely used and yields the tightest bounds. We use this model for our comparisons with pure MILP.

2.3 Logic-Based Benders Decomposition

Logic-based Benders decomposition is based on the concept of an *inference dual.* Consider an optimization problem

$$\begin{aligned}&\min\ f(x)\\&C(x)\\&x \in D\end{aligned} \tag{2.1}$$

where $C(x)$ represents a constraint set containing variables x, and D is the domain of x (such as $\mathbb{R}^n$ or $\mathbb{Z}^n$). The inference dual is the problem of finding the tightest lower bound on the objective function that can be deduced from the constraints:

$$\begin{aligned}&\max\ v\\&C(x) \overset{P}{\vdash} (f(x) \geq v)\\&v \in \mathbb{R},\ P \in \mathcal{P}\end{aligned} \tag{2.2}$$

Here $C(x) \overset{P}{\vdash} (f(x) \geq v)$ indicates that proof P deduces $f(x) \geq v$ from $C(x)$. The domain of variable P is a family $\mathcal{P}$ of proofs, and the dual solution is a pair (v, P). When the primal problem (2.1) is a feasibility problem with no objective function, the dual can be viewed as the problem finding a proof P of infeasibility.

If (2.1) is a linear programming (LP) problem $\min\{cx \mid Ax \geq b,\ x \geq 0\}$, the inference dual becomes the classical LP dual (assuming feasibility) for an appropriate proof family $\mathcal{P}$. Namely, each proof P corresponds to a tuple $u \geq 0$ of multipliers, and P deduces the bound $cx \geq v$ when the surrogate $uAx \geq ub$ dominates $cx \geq v$; that is, $uA \leq c$ and $ub \geq v$. The dual therefore maximizes v subject to $uA \leq c$, $ub \geq v$, and $u \geq 0$. Equivalently, it maximizes ub subject to $uA \leq c$ and $u \geq 0$, which is the classical LP dual.

Logic-based Benders decomposition applies to problems of the form

$$\begin{aligned} &\min\ f(x,y) \\ &C(x,y) \\ &x \in D_x,\ y \in D_y \end{aligned} \tag{2.3}$$

Fixing x to $\bar{x}$ defines the subproblem

$$\begin{aligned} &\min\ f(\bar{x},y) \\ &C(\bar{x},y) \\ &y \in D_y \end{aligned} \tag{2.4}$$

Let proof P solve the inference dual of the subproblem by deducing the bound $f(\bar{x},y) \geq v^*$. A Benders cut $v \geq B_{\bar{x}}(x)$ is derived by identifying a bound $B_{\bar{x}}(x)$ that the same proof P deduces for any given x. Thus, in particular, $B_{\bar{x}}(\bar{x}) = v^*$. The kth master problem is

$$\begin{aligned} &\min\ v \\ &v \geq B_{x^i}(x),\ i = 1,\ldots,k-1 \\ &x \in D_x \end{aligned} \tag{2.5}$$

where $x^1,\ldots,x^{k-1}$ are the solutions of the first $k-1$ master problems. The optimal value v_k of the master problem is a lower bound on the optimal value of (2.3), and each $B_{x^i}(x^i)$ is an upper bound. The algorithm terminates when v_k is equal to the minimum of $B_{x^i}(x^i)$ over $i = 1,\ldots,k$.

Classical Benders decomposition is the result of applying logic-based Benders decomposition to a problem of the form

$$\begin{aligned} &\min\ f(x) + cy \\ &g(x) + Ay \geq b \\ &x \in D_x,\ y \geq 0 \end{aligned} \tag{2.6}$$

The subproblem is an LP:

$$\begin{aligned} &\min\ f(\bar{x}) + cy \\ &Ay \geq b - g(\bar{x}) \\ &y \geq 0 \end{aligned} \tag{2.7}$$

whose inference dual is the LP dual. Its solution u defines a surrogate $uAy \geq u(b - g(\bar{x}))$ that dominates $cy \geq v^*$ and therefore deduces that $f(\bar{x})+cy \geq f(\bar{x})+u(b-g(\bar{x}))$. The same u deduces $f(x)+cy \geq f(x)+u(b-g(x))$ for any x, and we have the classical Benders cut $v \geq f(x)+u(b-g(x))$. When the subproblem is infeasible, the dual has an extreme ray solution u that proves infeasibility because $uA \leq 0$ and $u(b-g(x)) > 0$. The Benders cut is therefore $u(b-g(x)) \leq 0$.

In practice, the solution of the subproblem inference dual is the proof of optimality obtained while solving the subproblem. The simplest type of Benders cut is a *nogood cut*, which states that the solution of the subproblem cannot be improved unless certain x_j's are fixed to different values. For example, we might observe that only part of the master problem solution appears as premises in the optimality proof, perhaps $x_j = \bar{x}_j$ for $j \in J$. Then the optimal value of the subproblem is at least v^* so long as $x_j = \bar{x}_j$ for $j \in J$. This yields a nogood cut $v \geq B_{\bar{x}}(x)$ with

$$B_{\bar{x}}(x) = \begin{cases} v^* & \text{if } x_j = \bar{x}_j \text{ for } j \in J \\ -\infty & \text{otherwise} \end{cases}$$

If the subproblem is infeasible, then perhaps $x_j = \bar{x}_j$ for $j \in J$ appear as premises in the proof of infeasibility. A nogood cut states simply that $x_j \neq \bar{x}_j$ for some $j \in J$.

Further analysis of the optimality proof may yield *analytic* Benders cuts that provide useful bounds when $x_j \neq \bar{x}_j$ for some $j \in J$. We may also be able to infer valid cuts by re-solving the subproblem when some of the premises $x_j = \bar{x}_j$ are dropped. All of these techniques are illustrated below.

2.4 The Problem

In the unsegmented problem, there are n jobs to be processed, and each job j has release time r_j, deadline (or due date) d_j, and processing time p_j. If we let $J = \{1, \ldots, n\}$, the problem is to assign each job $j \in J$ a start time s_j so that time windows are observed:

$$r_j \le s_j \le d_j - p_j, \ \ j \in J$$

and jobs run consecutively:

$$s_j + p_j \le s_k \text{ or } s_k + p_k \le s_j, \ \ \text{all } j,k \in J \text{ with } j \ne k$$

We minimize makespan by minimizing $\max_{j \in J}\{s_j + p_j\}$. To minimize tardiness, we drop the deadline constraints $s_j \le d_j - p_j$ and minimize

$$\sum_{j \in J} \max\{s_j + p_j - d_j, \ 0\} \tag{2.8}$$

The segmented problem is the same except for the additional constraint that each job must be completed within one segment $[a_i, a_{i+1}]$ of the time horizon:

$$a_i \le s_j \le a_{i+1} - p_j \text{ for some } i \in I, \quad \text{all } j \in J$$

Notation is summarized in Table 4.2.

2.4.1 MILP Formulation

In the discrete-time MILP formulation of the problem, binary variable $z_{jt} = 1$ when job j starts at time t. Let T be the set of discrete times, and assume $r_j, d_j \in T$ for each j. The unsegmented problem is written

$$\begin{array}{ll}
\min \ \ [\text{objective function}] & (a) \\
\displaystyle\sum_{t \in T} z_{jt} = 1, \ \ j \in J & (b) \\
\displaystyle\sum_{j \in J} \sum_{\bar{t} \in T_{jt}} z_{j\bar{t}} \le 1, \ \ t \in T & (c) \\
z_{jt} = 0, \quad \text{all } t \in T \text{ with } t < r_j, \ j \in J & (d) \\
z_{jt} = 0, \quad \text{all } t \in T \text{ with } t > d_j - p_j, \ j \in J & (e) \\
z_{jt} \in \{0,1\}, \ \ j \in J, t \in T & (f)
\end{array}$$

Table 2.1: Nomenclature for the segmented problem.

Indices	
$j \in J$	job
$i \in I$	segment of the time horizon
$t \in T$	time
Sets	
J	set of jobs ($\{1, 2, ..., n\}$)
I	set of segments
T	set of discrete times
T_{jt}	set of discrete times at which job j running at time t can start
J_i	set of jobs assigned to segment i by the master problem
$J(t_1, t_2)$	set of jobs whose time windows fall within $[t_1, t_2]$
Parameters	
r_j	release time of job j
d_j	deadline (or due date) of job j
p_j	processing time of job j
a_i	start time of segment i
$\bar{y}_{ij}$	solution value of y_{ij} in the master problem
$\tilde{r}_{ij}$	starting time of the effective time window of job j on segment i
$\tilde{d}_{ij}$	ending time of the effective time window of job j on segment i
ϵ_{ij}	slack of job j on segment i
$\bar{r}$	tuple of distinct release times
$\bar{d}$	tuple of distinct deadlines
M_i^*	minimum makespan of segment i
Variables	
s_j	start time of job j
z_{jt}	$= 1$ if job j starts at time t
M	makespan over all segments
M_i	makespan of segment i
T_j	tardiness of job j
v_j	segment in which job j is processed
y_{ij}	$= 1$ if job j is processed in segment i

where $T_{jt} = \{\bar{t} \mid t - p_j + 1 \leq \bar{t} \leq t\}$. Constraint (b) requires that every job be assigned a start time. The clique inequality (Fulkerson, 1971; Padberg, 1973) in (c) ensures that jobs do not overlap. Constraint (d) prevents a job from starting before its release time, and (e) prevents it from ending after its deadline.

The feasibility problem seeks a feasible solution and consists of the constraints (b)–(f). The makespan problem minimizes the makespan M subject to (b)–(f) and

$$M \geq (t + p_j)z_{jt}, \quad j \in J, \ t \in T$$

The tardiness problem minimizes $\sum_{j \in J} T_j$ subject to (b)–(d), (f), and

$$\begin{aligned} & T_j \geq \sum_{t \in T} (t + p_j)z_{jt} - d_j, \quad j \in J \\ & T_j \geq 0, \quad j \in J \end{aligned} \tag{2.9}$$

where T_j is the tardiness of job j.

The segmented problem consists of the above and the additional constraints

$$z_{jt} = 0, \quad \text{for } t = a_{i+1} - p_j + 1, \ldots, a_{i+1} - 1, \text{ all } i \in I, \ j \in J \tag{2.10}$$

2.4.2 CP Formulation

In principle, a CP model of the unsegmented problem is quite simple. Again letting s_j be the start time of job j, a model is

$$\begin{array}{ll} \min \ [\text{objective function}] & (a) \\ r_j \leq s_j \leq d_j - p_j, \quad j \in J & (b) \\ \text{noOverlap}(s, p) & (c) \end{array} \tag{2.11}$$

where $s = (s_1, \ldots, s_n)$ and $p = (p_1, \ldots, p_n)$ in constraint (c), and where noOverlap is a global constraint that requires the jobs to run sequentially. The makespan problem minimizes $\max_{j \in J}\{s_j + p_j\}$ subject to (b)–(c). The tardiness problem minimizes (2.8) subject to (b)–(c) without the upper bound in (b).

The segmented problem can be formulated by introducing a variable v_j with domain I that indicates which segment job j is assigned. We then add to (2.11) the constraints

$$a_{v_j} \leq s_j \leq a_{v_j+1} - p_j, \;\; j \in J$$

2.5 Segmented Feasibility Problem

We now apply logic-based Benders decomposition to the problem of finding a feasible schedule. The master problem assigns jobs to time segments, and the subproblem decouples into a scheduling problem for each segment. We first address the segmented problem, for which the time horizon is already divided into segments $[a_i, a_{i+1}]$ for $i \in I$.

2.5.1 Master Problem

The master problem is an MILP formulation in which binary variable $y_{ij} = 1$ when job j is assigned to segment i.

$$\begin{array}{l} \displaystyle\sum_{i \in I} y_{ij} = 1, \;\; j \in J \\ \text{Benders cuts} \\ \text{Relaxation} \\ y_{ij} \in \{0,1\}, \;\; j \in J, \; i \in I \end{array}$$

Benders cuts are developed below. The master problem also contains a relaxation of the problem, expressed in terms of the master problem variables. This results in more reasonable assignments in the early stages of the Benders algorithm and therefore reduces the number of iterations. The relaxation is described in Section 2.5.3 below.

Given a solution $\bar{y}_{ij}$ of the master problem, let $J_i = \{j \mid \bar{y}_{ij} = 1\}$ be the set of jobs it assigns to segment i. The subproblem decomposes into a scheduling problem for each segment:

$$\begin{array}{l}\left.\begin{array}{l} r_j \leq s_j \leq d_j - p_j \\ a_i \leq s_j \leq a_{i+1} - p_j \end{array}\right\}, \quad j \in J_i \\ \text{noOverlap}(s(i, \bar{y}), p(i, \bar{y}))\end{array}$$

where $s(i, \bar{y})$ is the tuple of variables s_j for $j \in J_i$, and similarly for $p(i, \bar{y})$. The subproblems are solved by CP.

In each iteration of the Benders algorithm, the master problem is solved for $\bar{y}$. If the resulting subproblem is feasible on every segment, the algorithm stops with a feasible solution. Otherwise, one or more Benders cuts are generated for each segment on which the scheduling subproblem is infeasible. These cuts are added to the master problem, and the process repeats. If at some point the master problem is infeasible, so is the original problem.

2.5.2 Benders Cuts

The simplest Benders cut for an infeasible segment i is a nogood cut that prevents assigning the same jobs (perhaps among others) to that segment in subsequent iterations:

$$\sum_{j \in J_i} (1 - y_{ij}) \geq 1$$

This cut is quite weak, however, because it can be satisfied by omitting just one job in J_i from the future assignments to segment i. The cut can be strengthened by identifying a smaller set $\bar{J}_i \subset J_i$ of jobs that appear as premises in the proof of infeasibility for segment i. This yields the cut

$$\sum_{j \in \bar{J}_i} (1 - y_{ij}) \geq 1$$

Unfortunately, standard CP solvers do not provide this kind of dual information. We therefore seek to identify heuristically a smaller set $\bar{J}_i$ that creates infeasibility.

A simple heuristic is to remove jobs from J_i one at a time, checking each time if the scheduling problem on segment i is still infeasible. If it becomes feasible, the job is restored to J_i (Algorithm 1). This requires repeated solution of each subproblem, but the time required to do so tends to be small in practice.

Algorithm 1: Strengthening nogood cuts for the feasibility problem.

```
Let $\bar{J}_i = J_i = \{j_1, \dots, j_k\}$;
for $\ell = 1, \dots, k$ do
    if the jobs in $\bar{J}_i \setminus \{j_\ell\}$ cannot be scheduled on segment $i$ then
        Remove $j_\ell$ from $\bar{J}_i$
```

This heuristic may be more effective if jobs less likely to lead to feasibility are removed first. Let the *effective time window* $[\tilde{r}_{ij}, \tilde{d}_{ij}]$ of job j on segment i be its time window adjusted to reflect the segment boundaries. Thus

$$\begin{aligned}\tilde{r}_{ij} &= \max\{\min\{r_j, a_{i+1}\}, a_i\}\\ \tilde{d}_{ij} &= \min\{\max\{d_j, a_i\}, a_{i+1}\}\end{aligned}$$

Let the *slack* of job j on segment i be

$$\epsilon_{ij} = (\tilde{d}_{ij} - \tilde{r}_{ij}) - p_j$$

We can remove the jobs in order of decreasing slack, which means they are indexed so that

$$\epsilon_{i1} \geq \cdots \geq \epsilon_{ik}$$

2.5.3 Relaxation

The convergence of a Benders method can often be accelerated by augmenting the master problem with some valid inequalities in the master problem variables, resulting in a relaxation of the original problem.

A simple relaxation in the present case requires that jobs be scheduled so that for every time interval $[t_1, t_2]$, the set $J(t_1, t_2)$ of jobs whose time windows fall within

$[t_1, t_2]$ has total processing time at most $t_2 - t_1$. Thus

$$J(t_1, t_2) = \{j \in J \mid t_1 \le r_j,\ d_j \le t_2\}$$

It suffices to enumerate, for each segment, distinct intervals of the form $[r_j, d_k]$ for all j, k with $r_j \le d_k$. So we have the relaxation

$$\sum_{\ell \in J(r_j, d_k)} p_\ell y_{i\ell} \le \tilde{d}_{ik} - \tilde{r}_{ij}, \quad \text{all } i \in I \text{ and all distinct } [r_j, d_k]$$

Algorithm 2 generates a relaxation with fewer redundant constraints. Note that it fixes $y_{ij} = 0$ when job j's time window does not overlap segment i.

Algorithm 2: Generating a relaxation for the segmented feasibility problem.

```
Let $\bar{r}_1, \ldots, \bar{r}_t$ be the distinct elements of $\{r_1, \ldots, r_n\}$;
Let $\bar{d}_1, \ldots, \bar{d}_u$ be the distinct elements of $\{d_1, \ldots, d_n\}$;
for all $i$ do
    for all $j = 1, \ldots, t$ with $a_i < \bar{r}_j < a_{i+1}$ do
        for all $k = 1, \ldots, u$ with $\bar{d}_k < a_{i+1}$ do
            Generate the inequality $\sum_{\ell \in J(\bar{r}_j, \bar{d}_k)} p_\ell y_{i\ell} \le \bar{d}_k - \bar{r}_j$
        Generate the inequality $\sum_{\ell \in J(\bar{r}_j, \infty)} p_\ell y_{i\ell} \le a_{i+1} - \bar{r}_j$;
    for all $k = 1, \ldots, u$ with $a_i < \bar{d}_k < a_{i+1}$ do
        Generate the inequality $\sum_{\ell \in J(0, \bar{d}_k)} p_\ell y_{i\ell} \le \bar{d}_k - a_i$
    Generate the inequality $\sum_\ell p_\ell y_{i\ell} \le a_{i+1} - a_i$;
    Set $y_{i\ell} = 0$ for all $\ell \in J(0, a_i)$;
    Set $y_{i\ell} = 0$ for all $\ell \in J(a_{i+1}, \infty)$;
```

2.6 Unsegmented Feasibility Problem

We now suppose that the jobs are not required to fit inside segments of the time horizon. We create segments $[a_i, a_{i+1}]$ solely for purposes of decomposition, which means that a job can overlap two or more segments. This leads to several cases that complicate the master problem and the Benders cuts.

Table 2.2: Additional nomenclature for the unsegmented problem.

Sets	
J_{i0}	set of jobs that are fully processed in segment i
Parameters	
$\bar{x}_{ij}$	solution value of x_{ij} in the master problem
$\bar{y}_{ij}$	solution value of y_{ij} in the master problem
$\bar{y}_{ijk}$	solution value of y_{ijk} in the master problem, for $k \in \{0,1,2,3\}$
$\bar{a}_i$	updated start time of the segment i
Variables	
x_{ij}	amount of time job j is processed during segment i
y_{ij}	$= 1$ if at least a portion of job j is processed in segment i
y_{ij0}	$= 1$ if all of job j is processed in segment i
y_{ij1}	$= 1$ if a portion of job j is processed at the start of segment i
y_{ij2}	$= 1$ if a portion of job j is processed at the end of segment i
y_{ij3}	$= 1$ if a portion of job j is processed throughout segment i
α_i	$= 1$ if $x_{ij} \le \bar{x}_{ij}$ and segment i begins while job j is in process
β_i	$= 1$ if $x_{ij} \le \bar{x}_{ij}$ and segment i ends while job j is in process
γ_i	$= 1$ if $x_{ij_1} + x_{ij_2} \le x_i^*$ and segment i begins while job j_1 is in process and ends while job j_2 is in process

2.6.1 Master Problem

Let continuous variable x_{ij} indicate the amount of time that job j processes during segment i. Then

$$\begin{aligned} &\sum_{i \in I} x_{ij} = p_j, \quad j \in J \\ &x_{ij} \ge 0, \quad i \in I, j \in J \end{aligned} \tag{2.12}$$

The assignment of jobs (or portions of jobs) to segments is governed by several binary variables as introduced in Table 2.2.

We have the constraints

$$\begin{array}{ll}
\sum_{i\in I} y_{ij} \geq 1, \;\; j \in J & (a) \\
y_{ij} = y_{ij0} + y_{ij1} + y_{ij2} + y_{ij3}, \;\; i \in I, j \in J & (b) \\
\sum_{j\in J} y_{ij1} \leq 1, \;\; \sum_{j\in J} y_{ij2} \leq 1, \;\; \sum_{j\in J} y_{ij3} \leq 1, \;\; i \in I & (c) \\
y_{ij1} \leq y_{i-1,j,2} + y_{i-1,j,3}, \;\; i \in I, i > 1, \; j \in J & (d) \\
y_{ij2} \leq y_{i+1,j,1} + y_{i+1,j,3}, \;\; i \in I, i < n, \; j \in J & (e) \\
y_{ij3} \leq y_{i-1,j,3} + y_{i-1,j,2}, \;\; i \in I, i > 1, \; j \in J & (f) \\
y_{ij3} \leq y_{i+1,j,3} + y_{i+1,j,1}, \;\; i \in I, i < n, \; j \in J & (g) \\
\sum_{i\in I} y_{ij0} \leq 1, \;\; \sum_{i\in I} y_{ij1} \leq 1, \;\; \sum_{i\in I} y_{ij2} \leq 1, \;\; j \in J & (h) \\
y_{1j1} = y_{1j3} = y_{nj2} = y_{nj3} = 0, \;\; j \in J & (i) \\
\sum_{i\in I} y_{ij3} \leq \left\lfloor \dfrac{p_j}{a_{i+1} - a_i} \right\rfloor, \;\; j \in J & (j) \\
y_{ij}, y_{ij0}, y_{ij1}, y_{ij2}, y_{ij3} \in \{0, 1\}, \;\; i \in I, j \in J & (k)
\end{array} \tag{2.13}$$

Constraint (a) ensures that every job is assigned to at least one segment. Constraints (b) define y_{ij}. Constraints (c) ensure that at most one partial job is processed first, last, or throughout a segment. Constraints (d)–(g) require contiguity for the portions of a job. Constraints (h) say that a job can start, finish, or execute completely in at most one segment. Constraints (i) give boundary conditions. Constraints (j) are redundant but tighten the continuous relaxation of the MILP formulation.

To connect x_{ij} with the binary variables, note that we have the following disjunction for each i, j:

$$\begin{pmatrix} y_{ij} = 0 \\ x_{ij} = 0 \end{pmatrix} \vee \begin{pmatrix} y_{ij0} = 1 \\ x_{ij} = p_j \end{pmatrix} \vee \begin{pmatrix} y_{ij1} = 1 \\ x_{ij} \leq p_j \end{pmatrix} \vee \begin{pmatrix} y_{ij2} = 1 \\ x_{ij} \leq p_j \end{pmatrix} \vee \begin{pmatrix} y_{ij3} = 1 \\ x_{ij} = a_{i+1} - a_i \end{pmatrix}$$

Using the standard convex hull formulation of a disjunction of linear systems (Jeroslow, 1987), this becomes:

$$\begin{array}{l}
x_{ij1} \leq p_j y_{ij1}, \;\; x_{ij2} \leq p_j y_{ij2} \\
x_{ij} = p_j y_{ij0} + x_{ij1} + x_{ij2} + (a_{i+1} - a_i) y_{ij3} \\
x_{ij1}, x_{ij2} \geq 0
\end{array} \tag{2.14}$$

The master problem consists of (2.12)–(2.13), (2.14) for all $i \in I$ and $j \in J$, Benders cuts, and a relaxation. The Benders cuts are described in the next section.

To formulate the subproblem, suppose that the solution of the current master problem is $\bar{y}_{ij}, \bar{y}_{ij0}, \bar{y}_{ij1}, \bar{y}_{ij2}, \bar{x}_{ij}$ for all i, j. Define

$$\begin{aligned} J_i &= \{j \mid \bar{y}_{ij} = 1\} \\ J_{i0} &= \{j \mid \bar{y}_{ij0} = 1\} \end{aligned}$$

The subproblem must take into account whether the master problem assigned certain jobs to begin or end a segment. A general formulation of the subproblem on segment i is

$$\begin{array}{l} \left.\begin{array}{l} r_j \le s_j \le d_j - p_j \\ \bar{a}_i \le s_j \le \bar{a}_{i+1} - p_j \end{array}\right\}, \quad j \in J_{i0} \\ \text{noOverlap}(s(i, \bar{y}_0), p(i, \bar{y}_0)) \end{array} \tag{2.15}$$

where

$$\bar{a}_i = \begin{cases} a_i + \bar{x}_{ij_1}, & \text{if } \bar{y}_{ij_1 1} = 1 \text{ for some } j_1 \in J \\ a_i, & \text{otherwise} \end{cases}$$
$$\bar{a}_{i+1} = \begin{cases} a_{i+1} - \bar{x}_{ij_2}, & \text{if } \bar{y}_{ij_2 2} = 1 \text{ for some } j_2 \in J \\ a_{i+1}, & \text{otherwise} \end{cases}$$

Also $s(i, \bar{y}_0)$ is the tuple of variables s_j for $j \in J_{i0}$, and similarly for $p(i, \bar{y}_0)$. Note that if a portion of some job j_3 spans the entire segment ($\bar{y}_{ij_3 3} = 1$), the constraint set is empty.

The relaxation is similar to that for the segmented problem but must account for jobs that run in two or more segments. The relaxation is generated by Algorithm 3.

2.6.2 Benders Cuts

The Benders cuts generated for an infeasible segment i depend on whether there are partial jobs assigned to the beginning or end of the segment.

Algorithm 3: Generating a relaxation for the unsegmented feasibility problem.

```
Let $\bar{r}_1, \ldots, \bar{r}_t$ be the distinct elements of $\{r_1, \ldots, r_n\}$;
Let $\bar{d}_1, \ldots, \bar{d}_u$ be the distinct elements of $\{d_1, \ldots, d_n\}$;
for all $i$ do
    for all $j = 1, \ldots, t$ with $a_i < \bar{r}_j < a_{i+1}$ do
        for all $k = 1, \ldots, u$ with $\bar{d}_k < a_{i+1}$ do
            Generate the inequality $\sum_{\ell \in J(\bar{r}_j, \bar{d}_k)} p_\ell y_{i\ell 0} \le \bar{d}_k - \bar{r}_j$
        Generate the inequality $\sum_{\ell \in J(\bar{r}_j, \infty)} x_{i\ell} \le a_{i+1} - \bar{r}_j$;
    for all $k = 1, \ldots, u$ with $a_i < \bar{d}_k < a_{i+1}$ do
        Generate the inequality $\sum_{\ell \in J(0, \bar{d}_k)} x_{i\ell} \le \bar{d}_k - a_i$
    Generate the inequality $\sum_\ell x_{i\ell} \le a_{i+1} - a_i$;
    Set $y_{i\ell} = 0$ for all $\ell \in J(0, a_i)$;
    Set $y_{i\ell} = 0$ for all $\ell \in J(a_{i+1}, \infty)$;
```

- *Case 1.* There are no partial jobs in segment i. Then we can use the simple nogood cut

$$\sum_{j \in J_{i0}} (1 - y_{ij0}) \ge 1 \tag{2.16}$$

This can be strengthened as in the segmented problem by removing some jobs that are not necessary for infeasibility (Algorithm 1 with J_{i0} replacing J_i).

- *Case 2.* There is a partial job only at the start of the segment (say, job j_1). We solve a modified subproblem by maximizing x_{ij_1} rather than checking for feasibility with $x_{ij_1} = \bar{x}_{ij_1}$. That is, we maximize x_{ij_1} subject to (2.15) with $\bar{a}_i = a_i$ and $\bar{a}_{i+1} = a_{i+1}$.

  - *Case 2a.* The modified subproblem is infeasible. Because feasibility is not restored by reducing x_{ij_1} to zero, we again use the cut (2.16) and strengthen it by removing some jobs that are not necessary for feasibility.

Suppose now that the modified subproblem is feasible, and let $x^*_{ij_1}$ be the maximum value of x_{ij_1}. We know $x^*_{ij_1} < \bar{x}_{ij_1}$, because otherwise segment i would be feasible.

- *Case 2b.* $x^*_{ij_1} = 0$. Here, job j_1 must be removed as the first job if the other jobs in the segment are completely processed. We have the Benders cut

$$(1 - y_{ij_1 1}) + \sum_{j \in J_{i0}} (1 - y_{ij0}) \geq 1$$

- *Case 2c.* $x^*_{ij_1} > 0$. Now the Benders cut can say that either $x_{ij_1} \leq \bar{x}_{ij_1}$ or one of the other jobs must be dropped. We introduce 0-1 variable α_i that is 1 when $x_{ij_1} \leq \bar{x}_{ij_1}$. Then we have the cut

$$\begin{aligned} &\alpha_i + \sum_{j \in J_{i0}} (1 - y_{ij0}) \geq 1 \\ &x_{ij_1} \leq \bar{x}_{ij_1} + p_{j_1}(1 - \alpha_i) \\ &\alpha_i \in \{0, 1\} \end{aligned} \tag{2.17}$$

  The cut can be strengthened by removing jobs in J_{i0} until $x^*_{ij_1} \geq \bar{x}_{ij_1}$ (Algorithm 4).

Algorithm 4: Strengthening nogood cuts for the unsegmented feasibility problem.

```
Let $\bar{J}_{i0} = J_{i0} = \{j_2, \ldots, j_k\}$;
for $\ell = 2, \ldots, k$ do
    Let $x^*_{ij_1}$ be the maximum of $x_{ij_1}$ subject to (2.15) with $\bar{a}_i = a_i$ and
    $J_{i0} = \bar{J}_{i0} \setminus \{j_\ell\}$
    if $x^*_{ij_1} < \bar{x}_{ij_1}$ then
        Remove $j_\ell$ from $\bar{J}_{i0}$
```

- *Case 3.* There is a partial job only at the end of the segment (say, job j_2). The cuts are similar to Case 2. The modified subproblem finds the maximum value $x^*_{ij_2}$ of x_{ij_2} subject to (2.15) with $\bar{a}_i = a_i$ and $\bar{a}_{i+1} = a_{i+1}$.

  - *Case 3a.* The modified subproblem is infeasible. We use cut (2.16) and strengthen it as before.

– *Case 3b.* $x^*_{ij_2} = 0$. We have the Benders cut

$$(1 - y_{ij_2 2}) + \sum_{j \in J_{i0}} (1 - y_{ij0}) \geq 1$$

– *Case 3c.* $x^*_{ij_2} > 0$. We have the cut

$$\begin{aligned} &\beta_i + \sum_{j \in J_{i0}} (1 - y_{ij0}) \geq 1 \\ &x_{ij_2} \leq \bar{x}_{ij_2} + p_{j_2}(1 - \beta_i) \\ &\beta_i \in \{0, 1\} \end{aligned} \tag{2.18}$$

The cut can be strengthened by removing jobs in J_{i0} until $x^*_{ij_2} \geq \bar{x}_{ij_2}$.

- *Case 4.* There are two partial jobs in the segment, namely j_1 at the start and j_2 at the end. We solve a modified subproblem by finding the maximum value x^*_i of $x_{ij_1} + x_{ij_2}$ subject to (2.15) with $\bar{a}_i = a_i$ and $\bar{a}_{i+1} = a_{i+1}$.

  – *Case 4a.* The modified subproblem is infeasible. We use cut (2.16) and strengthen it as before.

  – *Case 4b.* The modified subproblem is feasible but $x^*_i = 0$. Then j_1 and j_2 must be removed as the first and last jobs if the other jobs are completely processed, and we have the Benders cuts

$$\begin{aligned} (1 - y_{ij_1 1}) + \sum_{j \in J_{i0}} (1 - y_{ij0}) \geq 1 \\ (1 - y_{ij_2 2}) + \sum_{j \in J_{i0}} (1 - y_{ij0}) \geq 1 \end{aligned}$$

  – *Case 4c.* The modified subproblem is feasible and $0 < x^*_i < \bar{x}_{ij_1} + \bar{x}_{ij_2}$. In this case the Benders cuts says that either (a) $x_{ij_1} + x_{ij_2} \leq x^*_i$, or (b) j_1 or j_2 must be removed as the first or last job if the other jobs are completely processed:

$$\begin{aligned}&\gamma_i + (1 - y_{ij_1 1}) + (1 - y_{ij_2 2}) + \sum_{j \in J_{i0}} (1 - y_{ij0}) \geq 1\\&x_{ij_1} + x_{ij_2} \leq x_i^* + (p_{j_1} + p_{j_2})(1 - \gamma_i)\\&\gamma_i \in \{0, 1\}\end{aligned}$$

This cut can be strengthened by removing jobs from J_{i0} until $x_i^* \geq \bar{x}_{ij_1} + \bar{x}_{ij_2}$.

– *Case 4d.* The modified subproblem is feasible and $x_i^* \geq \bar{x}_{ij_1} + \bar{x}_{ij_2}$. Then if $x_{ij_1}^*, x_{ij_2}^*$ are defined as before, we must have either $x_{ij_1}^* < \bar{x}_{ij_1}$ as in Case 2, or $x_{ij_2}^* < \bar{x}_{ij_2}$ as in Case 3. This is because the feasible set for (x_{ij_1}, x_{ij_2}) is a box with the upper right corner possibly cut off with a 45$^\text{o}$ line. If $x_{ij_1}^* < \bar{x}_{ij_1}$, we add cut (2.17) as in Case 2c. If $x_{ij_2}^* < \bar{x}_{ij_2}$, we add cut (2.18) as in Case 3c. Either cut can be strengthened as before.

2.7 Segmented Makespan Problem

We now address the problem of minimizing makespan when jobs must complete processing within a time segment. The subproblem is itself a minimization problem and therefore generates two types of Benders cuts: strengthened nogood cuts similar to those developed above, and cuts that bound the makespan. The bounding cuts can themselves be based on a nogood principle or a deeper analysis of the inference dual (analytic Benders cuts).

2.7.1 Master Problem

The master problem for segmented makespan problem is

$$\begin{aligned}
&\min\ M\\
&\sum_i y_{ij} = 1,\ \ j \in J\\
&\text{Benders cuts}\\
&\text{Relaxation}\\
&y_{ij} \in \{0,1\},\ \ \text{all } i, j
\end{aligned}$$

Given a solution $\bar{y}_{ij}$ of the master problem, the subproblem decomposes into a minimum makespan problem for each segment i. It minimizes M_i subject to (2.5.1) and

$$M_i \geq s_j + p_j,\ \ \text{all } j \in J \text{ with } \bar{y}_{ij} = 1$$

Thus the makespan on a segment is understood to be the completion time of the last job on the segment, not the completion time minus a_i. If M_i^* is the minimum makespan on segment i, the optimal value of the original problem is $\max_{i \in I} M_i^*$. Note that if no jobs are assigned to segment i, the constraint set is empty, and $M_i^* = -\infty$. Obviously, the latest segment that contains one or more jobs controls the overall makespan.

The relaxation described for the feasibility problem is also valid for the makespan problem. In addition, we can give the following bound on the makespan for each distinct r_j:

$$M \geq \tilde{r}_{ij} + \sum_{\ell \in J(r_j,\infty)} x_{i\ell},\ \ i \in I,\ j \in J$$

Algorithm 5 generates a relaxation with fewer redundant constraints.

2.7.2 Strengthened Nogood Cuts

If the scheduling problem on segment i is infeasible, we use the same strengthened nogood cuts as in the segmented feasibility problem. If segment i has a feasible

Algorithm 5: Generating a relaxation for the makespan problem.

```
Let $\bar{r}_1, \ldots, \bar{r}_t$ be the distinct elements of $\{r_1, \ldots, r_n\}$;
for all $i$ do
    for all $j = 1, \ldots, t$ with $a_i < \bar{r}_j$ do
        Generate the inequality $M \geq \bar{r}_j + \sum_{\ell \in J(\bar{r}_j, \infty)} x_{i\ell}$
    Generate the inequality $M \geq w_i + \sum_{\ell \in J} x_{i\ell}$;
    for $j = 1, \ldots, n$ do
        Generate the inequality $w_i \geq a_i y_{ij}$
```

schedule with minimum makespan M_i^*, the simplest nogood cut is

$$M \geq M_i^* \left(1 - \sum_{j \in J_i} (1 - y_{ij}) \right)$$

This says that the makespan for subproblem i cannot be less than M_i^* unless at least one job is removed from J_i. The cut can be strengthened to

$$M \geq M_i^* \left(1 - \sum_{j \in \bar{J}_i} (1 - y_{ij}) \right) \tag{2.19}$$

where $\bar{J}_i \subset J_i$ is a smaller set of jobs that results in the same minimum makespan M_i^*. A simple heuristic for computing $\bar{J}_i$ appears as Algorithm 6. The jobs can be removed in order of increasing $\tilde{r}_{ij} + p_j$, and we therefore index the jobs in J_i so that

$$\tilde{r}_{i1} + p_1 \leq \cdots \leq \tilde{r}_{ik} + p_k$$

Algorithm 6: Strengthening nogood cuts for the makespan problem.

```
Let $\bar{J}_i = J_i = \{j_1, \ldots, j_k\}$;
for $\ell = 1, \ldots, k$ do
    if the minimum makespan is $M_i^*$ when jobs in $\bar{J}_i \setminus \{j_\ell\}$ are assigned to segment $i$
    then
        Remove $j_\ell$ from $\bar{J}_i$
```

Another way to obtain additional cuts is to use a two-tiered bound. Let $M_i(J)$ be the minimum makespan that results when jobs in J are assigned to segment i, so that in particular $M_i(J_i) = M_i^*$. Let Z_i be the set of jobs that can be removed, one at a time, without affecting makespan, so that

$$Z_i = \{j \in J_i \mid M_i(J_i \setminus \{j\}) = M_i^*\}$$

Then for each i we have the cut

$$M \geq M_i(J_i \setminus Z_i) \left(1 - \sum_{j \in J_i \setminus Z_i} (1 - y_{ij}) \right)$$

in addition to (2.19). This cut is redundant and should be deleted when $M_i(J_i \setminus Z_i) = M_i^*$.

2.7.3 Analytic Benders Cuts

The reasoning we use to obtain analytic Benders cuts is similar to that used in (Hooker, 2007a). Let P_i be the minimum makespan problem on segment i, with minimum makespan M_i^*. Let $J_i' = \{j \in J_i \mid r_j \leq a_i\}$ be the set of jobs in J_i with release times before segment i, and let $J_i'' = J_i \setminus J_i'$. Suppose we remove the jobs in $S \subset J_i'$ from segment i and let $\hat{M}_i$ be the minimum makespan of the problem $\hat{P}_i$ that remains. We first show that

$$M_i^* - \hat{M}_i \leq p_S + \max_{j \in J_i'}\{\tilde{d}_{ij}\} - \min_{j \in J_i'}\{\tilde{d}_{ij}\} \tag{2.20}$$

where $p_S = \sum_{j \in S} p_j$.

Consider any optimal solution of $\hat{P}_i$ and extend it to a solution s of P_i by scheduling the tasks in S sequentially after $\hat{M}_i$. Because $S \subset J_i'$, these jobs start after their release time. The makespan of s is $\hat{M}_i + p_S$. If $\hat{M}_i + p_S \leq \min_{j \in J_i'}\{\tilde{d}_{ij}\}$, then s is clearly feasible for P_i, which means $M_i^* \leq \hat{M}_i + p_S$ and (2.20) follows. On the other hand, if $\hat{M}_i + p_S > \min_{j \in J_i'}\{\tilde{d}_{ij}\}$, we have

$$\hat{M}_i + p_S + \max_{j \in J_i'}\{d_j\} - \min_{j \in J_i'}\{d_j\} > \max_{j \in J_i'}\{d_j\} \tag{2.21}$$

Because $M_i^* \leq \max_{j \in J_i'}\{d_j\}$, (2.21) implies (2.20).

Thus if the jobs in $S \subset J_i'$ are removed from segment i, we have from (2.20) a lower bound on the resulting optimal makespan $\hat{M}_i$. If one or more jobs in J_i'' are removed, this bound is no longer valid. We have the following Benders cut

$$M \geq M_i^* - \left(\sum_{j \in J_i'} p_j(1 - y_{ij}) + \max_{j \in J_i'}\{d_j\} - \min_{j \in J_i'}\{d_j\} \right) - M_i^* \sum_{j \in J_i''} (1 - y_{ij}) \qquad (2.22)$$

when one or more jobs are removed from segment i, and $M \geq M_i^*$ otherwise, provided $y_{ij} = 1$ for at least one $j \in J_i$. The second summation in (2.22) takes care of the case where one or more jobs in J_i'' are removed. If $y_{ij} = 0$ for all $j \in J_i$, the cut is simply $M \geq 0$.

We can linearize this cut by writing the following for each i:

$$\begin{aligned}
& M \geq M_i^* - \sum_{j \in J_i'} p_j(1 - y_{ij}) - w_i - M_i^* \sum_{j \in J_i''} (1 - y_{ij}) - M_i^* q_i \\
& q_i \leq 1 - y_{ij}, \ \ j \in J_i \\
& w_i \leq \left(\max_{j \in J_i'}\{d_j\} - \min_{j \in J_i'}\{d_j\} \right) \sum_{j \in J_i'} (1 - y_{ij}) \\
& w_i \leq \max_{j \in J_i'}\{d_j\} - \min_{j \in J_i'}\{d_j\}
\end{aligned}$$

where binary variable $q_i = 1$ when $y_{ij} = 0$ for all $j \in J_i$. The cut can be strengthened by replacing J_i, J_i' and J_i'' with $\bar{J}_i$, $\bar{J}_i'$, and $\bar{J}_i''$, where $\bar{J}_i$ is computed as in Algorithm 6, $\bar{J}_i' = \{j \in \bar{J}_i \mid r_j \leq a_i\}$, and $\bar{J}_i'' = \bar{J}_i \setminus \bar{J}_i$.

2.8 Unsegmented Makespan Problem

The master problem for the unsegmented makespan problem minimizes makespan M subject to (2.12)–(2.14), Benders cuts, and a relaxation. The subproblem on each segment i minimizes makespan M_i subject to (2.15) and

$$\begin{aligned}
&M_i \geq s_j + p_j, \text{ all } j \in J \text{ with } \bar{y}_{ij0} = 1\\
&M_i \geq a_i + \bar{x}_{ij}, \text{ all } j \in J \text{ with } \bar{y}_{ij1} = 1\\
&M_i \geq a_{i+1}, \text{ if } \bar{y}_{ij2} = 1 \text{ or } \bar{y}_{ij3} = 1 \text{ for some } j \in J
\end{aligned}$$

The relaxation is the same as for the segmented makespan problem.

2.8.1 Strengthened Nogood Cuts

If the scheduling problem on segment i is infeasible, we generate the same strengthened nogood cuts as in the unsegmented feasibility problem. Suppose, then, that segment i has a feasible schedule with minimum makespan M_i^*. As before, we let $J_i = \{j \mid \bar{y}_{ij} = 1\}$ and $J_{i0} = \{j \mid \bar{y}_{ij0} = 1\}$.

- *Case 1.* There are no partial jobs in segment i. Then we have the nogood cut

  $$M \geq M_i^* \left(1 - \sum_{j \in J_i} (1 - y_{ij0})\right)$$

  This cut can be strengthened as in the segmented problem. We also have the cut

  $$M \geq M_i(J_i \setminus Z_i) \left(1 - \sum_{j \in J_i \setminus Z_i} (1 - y_{ij0})\right)$$

  where Z_i is computed as before.

- *Case 2.* A partial job j_2 is assigned to the end of the segment. In this case the minimum makespan on the segment is $M_i^* = a_{i+1}$ unless j_2 is removed from the end. The Benders cut is simply

  $$M \geq M_i^* y_{ij_2 2} \tag{2.23}$$

- *Case 3.* There is no partial job at the end of the segment but a partial job j_1 at the start. The nogood cut is

$$M \geq M_i^* \left(1 - (1 - y_{ij_1 1}) - \sum_{j \in J_{i0}} (1 - y_{ij0}) \right)$$

  which can be strengthened by heuristically removing jobs from J_{i0}.

2.8.2 Analytic Benders Cuts

To analyze the subproblem more deeply we partition J_{i0} as follows:

$$\begin{aligned} J'_{i0} &= \{ j \in J_{i0} \mid r_j \leq a_i \} \\ J''_{i0} &= \{ j \in J_{i0} \mid r_j > a_i \} \end{aligned}$$

We consider the same three cases as above.

- *Case 1.* There are no partial jobs in segment i. The nogood cut is very similar to that obtained for the segmented case:

$$\begin{aligned} & M \geq M_i^* - \sum_{j \in J'_{i0}} p_j (1 - y_{ij0}) - w_i - M_i^* \sum_{j \in J''_{i0}} (1 - y_{ij0}) - M_i^* q_i \\ & q_i \leq 1 - y_{ij0}, \ \ j \in J_{i0} \\ & w_i \leq \left(\max_{j \in J'_{i0}} \{d_j\} - \min_{j \in J'_{i0}} \{d_j\} \right) \sum_{j \in J'_{i0}} (1 - y_{ij0}) \\ & w_i \leq \max_{j \in J'_{i0}} \{d_j\} - \min_{j \in J'_{i0}} \{d_j\} \end{aligned}$$

  The cut can be strengthened as before.

- *Case 2.* This yields only the nogood cut (2.23).

- *Case 3.* There is no partial job at the end of the segment but there is a partial job j_1 at the start of the segment. We will investigate the effect on makespan of reducing job j_1's processing time in segment i below its current assignment

$\bar{x}_{ij_1}$. Let $\delta = \bar{x}_{ij_1} - x_{ij_1}$ be the amount of the reduction. The Benders cuts generated depend on whether one or more jobs start at their release times in the minimum makespan schedule.

– *Case 3a.* Some job $j \in J_i$ starts at its release time in the optimal solution s^* of the subproblem on segment i. That is, $s_j^* = r_j$ for some $j \in J_{i0}$. Let k be the first such job. We may assume that j_1 and all jobs between j_1 and k are scheduled contiguously in s^* (i.e., there is no idle time between them). Now suppose the jobs between j_1 and k are scheduled δ earlier, and δ is increased to the value δ^* at which one of these jobs hits its release time. Thus

$$\delta^* = \min_{j \in J_{i0}} \left\{ s_j^* - \tilde{r}_{ij} \;\middle|\; s_j^* < s_k \right\}$$

This increases by δ^* the gap Δ_i between the job k' immediately before k and job k, where $\Delta_i = r_k - s_{k'}^* - p_{k'}$. Suppose further that for a given $\delta \leq \delta^*$, no job after k is short enough to be moved into the gap $\Delta_i + \delta$ while observing its release time. Then we know that the minimum makespan remains at M_i^* when job j_1's processing time is reduced by δ, assuming no jobs are removed from segment i.

To write the corresponding Benders cut, let $p_{\min}$ be the processing time of the shortest job that can be moved into the gap $\Delta_i + \delta^*$. Thus

$$p_{\min} = \min_{j \in J_{i0}} \left\{ p_j \;\middle|\; s_j^* > s_k^*,\ \tilde{r}_{ij} + p_j \leq r_k,\ p_j \leq \Delta_i + \delta^* \right\}$$

Then the minimum makespan remains M_i^* if $p_{\min} > \Delta_i + \delta$ (again assuming no jobs are removed from segment i). We introduce a binary variable η_i that is 0 when this inequality is satisfied. This yields the Benders cut

$$
\begin{aligned}
&M \geq {M_i}^*(1-\eta_i) - {M_i}^* \sum_{j \in J_i}(1-y_{ij}) \\
&\bar{x}_{ij_1} - x_{ij_1} + \Delta_i \leq p_{\min} + (\bar{x}_{ij_1} + \Delta_i - p_{\min})\eta_i - \epsilon \\
&\bar{x}_{ij_1} - x_{ij_1} + \Delta_i \geq p_{\min} - (p_{j_1} - \bar{x}_{ij_1} + p_{\min} - \Delta_i)(1-\eta_i)
\end{aligned}
$$

where $\epsilon > 0$ is a small number. When $\eta_i = 1$, this cut is of no value, but a simpler cut becomes useful:

$$
M \geq \left(a_i + \sum_{j \in J_{i0}} p_j \right) \eta_i + x_{ij_1} - {M_i}^* \sum_{j \in J_i}(1-y_{ij})
$$

It says that the jobs after j_1 can at best be scheduled contiguously.

– *Case 3b.* No job starts at its release time in the optimal solution of the subproblem. That is, $s_j^* > r_j$ for all $j \in J_{i0}$. Thus all jobs are scheduled contiguously. As $\delta = \bar{x}_{ij_1} - x_{ij_1}$ increases, minimum makespan decreases by an equal amount, at least until a job in J_{i0} hits its release time. At this point we can increase δ by another ϵ and check whether minimum makespan continues to decrease by ϵ. If so, we can further increase δ until another job hits its release time, and so forth until makespan stops decreasing at the margin, at which point we revert to Case 3a. We now write a Benders cut that allows makespan to decrease at the same rate δ increases up to $\delta \leq \delta_{k^*}$.

To make this more precise, let $s^*(\delta)$ be the minimum makespan solution on segment i when x_{ij_1} is fixed to $\bar{x}_{ij_1} - \delta$ rather than $\bar{x}_{ij_1}$, and let $M_i^*(\delta)$ be the corresponding minimum makespan. Let δ_0 be the value of δ at which the first job hits its release time, so that

$$
\delta_0 = \min_{j \in J_{i0}} \{s^*(0) - \tilde{r}_{ij}\}
$$

Now define

$$\delta_k = \delta_{k-1} + \min_{i \in J_{i0}} \{s^*(\delta_{k-1}) - \tilde{r}_{ij}\}$$

and let k^* be the smallest k for which $M_i^*(\delta_k + \epsilon) = M_i^*(\delta)$. Then we generate the Benders cut

$$\begin{aligned}
&\bar{x}_{ij_1} - x_{ij_1} \le \delta_{k^*} + (\bar{x}_{ij_1} - \delta_{k^*})\lambda_i \\
&M \ge {M_i}^*(1-\lambda_i) - (\bar{x}_{ij_1} - x_{ij_1}) - {M_i}^* \sum_{j \in J_i} (1 - y_{ij})
\end{aligned}$$

where $\lambda_i \in \{0,1\}$. This cut is useless when $\lambda_i = 1$, but the following cut is helpful in this case:

$$M \ge a_i\lambda_i + x_{ij_1} + \sum_{j \in J_{i0}} p_j\lambda_i - {M_i}^* \sum_{j \in J_i} (1 - y_{ij})$$

After generating these cuts, we move to Case 3a.

2.9 Segmented Tardiness Problem

The objective in the tardiness problem is to minimize total tardiness. We study the segmented version of the problem. Analysis of the unsegmented tardiness problem is more complex and is left to future research.

2.9.1 Master Problem

The master problem for the segmented tardiness problem is

$$\begin{aligned}
&\min \sum_{i \in I} T_i \\
&T_i \ge 0, \ \ i \in I \\
&\sum_i y_{ij} = 1, \ \ j \in J \\
&\text{Benders cuts} \\
&\text{Relaxation} \\
&y_{ij} \in \{0,1\}, \ \ \text{all } i, j
\end{aligned}$$

Given a solution $\bar{y}_{ij}$ of the master problem, the subproblem for each segment i is:

$$\begin{aligned}
&\min \sum_{j \in J_i} \max\{s_j + p_j - d_j, 0\} \\
&\left.\begin{aligned} &r_j \le s_j \\ &a_i \le s_j \le a_{i+1} - p_j \end{aligned}\right\} \; j \in J_i \\
&\text{noOverlap}(s(i,\bar{y}), p(i,\bar{y}))
\end{aligned} \tag{2.24}$$

The relaxation for the feasibility problem must be modified to make it valid for the tardiness problem, because deadlines are now due dates. The only hard deadlines are the upper bounds a_{i+1} of the segments. We have the relaxation

$$\sum_{\ell \in J(r_j,\infty)} p_\ell y_{i\ell} \le a_{i+1} - \tilde{r}_{ij}, \;\; i \in I, \; j \in J$$

Algorithm 7 generates a relaxation with fewer redundant constraints.

Algorithm 7: Generating a relaxation for the tardiness problem.

```
Let $\bar{r}_1, \ldots, \bar{r}_t$ be the distinct elements of $\{r_1, \ldots, r_n\}$;
for all $i$ do
    for all $j = 1, \ldots, t$ with $a_i < \bar{r}_j$ do
        Generate the inequality $\sum_{\ell \in J(\bar{r}_j,\infty)} p_\ell y_{i\ell} \le a_{i+1} - \bar{r}_j$
    Generate the inequality $\sum_\ell p_\ell y_{i\ell} \le a_{i+1} - a_i$
```

We also use a bound on tardiness that is developed in (Hooker, 2007a).

$$\begin{aligned}
&T \ge \sum_{i \in I} \underline{T}_i \\
&\underline{T}_i \ge \sum_{j \in J} T'_{ij}, \;\; i \in I \\
&T'_{ij} \ge a_i + \sum_{\ell=1}^{j} p_{\pi(\ell)} y_{ij} - d_j - (1 - y_{ij}) \left(a_i + \sum_{\ell \in H_{ij}} p_{\pi(\ell)} - d_j \right), \;\; i \in I, \; j \in J \\
&T'_{ij} \ge 0, \;\; \text{all } i, j
\end{aligned}$$

Here $H_{ij} = \{1, \ldots, j\} \setminus (J(0, a_i) \cup J(a_{i+1}, \infty))$ is the set of jobs with time windows that overlap segment i. The jobs are indexed so that $d_1 \le \cdots \le d_n$, and π is a permutation of the indices for which $p_{\pi(1)} \le \cdots \le p_{\pi(n)}$.

2.9.2 Strengthened Nogood Cuts

If the tardiness problem on segment i is infeasible, we use the same strengthened nogood cuts as in the feasibility problem. Otherwise, let T_i^* be the minimum tardiness. If $T_i^* > 0$, we use the strengthened nogood cut

$$T_i \geq T_i^* \left(1 - \sum_{j \in \bar{J}_i} (1 - y_{ij})\right) \tag{2.25}$$

where $\bar{J}_i \subset J_i$ is a smaller set of jobs that result in the same minimum tardiness T_i^*. A simple heuristic for computing $\bar{J}_i$ appears as Algorithm 8. The jobs can be removed in order of decreasing tightness ϵ_{ij}, but note that ϵ_{ij} can be negative.

Algorithm 8: Strengthening nogood cuts for the tardiness problem.

```
Let $\bar{J}_i = J_i = \{j_1, \ldots, j_k\}$;
for $\ell = 1, \ldots, k$ do
    if the minimum tardiness is $T_i^*$ when jobs in $\bar{J}_i \setminus \{j_\ell\}$ are assigned to segment $i$ then
        Remove $j_\ell$ from $\bar{J}_i$
```

Another way to strengthen the cuts is to use a two-tiered bound similar to the makespan case. Let $T_i(J)$ be the minimum makespan that results when jobs in J are assigned to segment i. If we define

$$Z_i = \{j \in J_i \mid T_i(J_i \setminus \{j\}) = T_i^*\}$$

then we have the cut

$$T_i \geq T_i(J_i \setminus Z_i) \left(1 - \sum_{j \in J_i \setminus Z_i} (1 - y_{ij})\right)$$

in addition to (2.25). This cut is redundant and should be deleted when $T_i(J_i \setminus Z_i) = T_i^*$.

2.9.3 Analytic Benders Cuts

Let P_i be the subproblem on a segment i with a feasible schedule, and let T_i^* be the minimum tardiness of P_i. When $T_i^* > 0$, we generate an analytic Benders cut as follows. Let $\hat{P}_i$ be the minimum tardiness problem that results when the jobs in $S \subset J_i$ are removed from P_i. Let $\hat{T}_i$ be the minimum tardiness of $\hat{P}_i$, and $\hat{F}_i$ a solution of $\hat{P}_i$ that achieves tardiness $\hat{T}_i$. Let

$$r_i^{\max} = \max\left\{\max\{r_j \mid j \in J_i\}, a_i\right\}$$

be the last release time of the jobs in J_i, or a_i, whichever is larger. Because P_i is feasible, we know $r_i^{\max} \leq a_{i+1}$.

Let $\hat{M}_i$ be the makespan of an optimal solution of $\hat{P}_i$. We construct a solution F_i of P_i by adding the jobs in S to $\hat{F}_i$. In particular, we schedule the jobs in S contiguously after $\max\{r_i^{\max}, \hat{M}\}$, in arbitrary order. The makespan of F_i is at most

$$r_i^{\max} + \sum_{\ell \in J_i} p_\ell$$

because in the worst case, all the jobs in the optimal solution of $\hat{P}_i$ are scheduled after $r_i^{\max}$. The tardiness incurred in F_i by each job $j \in S$ is therefore at most

$$\left(r_i^{\max} + \sum_{\ell \in J_i} p_\ell - d_j\right)^+$$

where $(\alpha)^+ = \max\{\alpha, 0\}$. Thus the total tardiness of F_i is at most

$$\hat{T}_i + \sum_{j \in S}\left(r_i^{\max} + \sum_{\ell \in J_i} p_\ell - d_j\right)^+ \tag{2.26}$$

F_i is feasible if all the jobs finish before a_{i+1}. So we know F_i is feasible if

$$r_i^{\max} + \sum_{\ell \in J_i} p_\ell \leq a_{i+1} \tag{2.27}$$

In this case, the tardiness (2.26) is an upper bound on T_i^*, and we have

$$\hat{T}_i \geq T_i^* - \sum_{j \in S} \left(r_i^{\max} + \sum_{\ell \in J_i} p_\ell - d_j \right)^+$$

This leads to the Benders cut

$$\hat{T}_i \geq \begin{cases} T_i^* - \displaystyle\sum_{j \in J_i} \left(r_i^{\max} + \sum_{\ell \in J_i} p_\ell - d_j \right)^+ (1 - y_{ij}), & \text{if (2.27) holds} \\ T_i^* \left(1 - \displaystyle\sum_{j \in J_i} (1 - y_{ij}) \right), & \text{otherwise} \end{cases}$$

2.10 Problem Generation

We generated random instances by selecting parameters uniformly from intervals as follows:

$$\begin{array}{l} r_j \in [0, \alpha R] \\ d_j - r_j \in [\gamma_1 \alpha R, \gamma_2 \alpha R] \\ p_j \in [0, \beta(d_j - r_j)] \end{array}$$

where R is the length of the time horizon, measured by the number of segments. Thus γ_1 and γ_2 control the width of the time windows, and β controls the processing time relative to the window width.

We set parameters as indicated in Table 2.3. We distinguished tight from wide time windows for segmented problems, because wider windows could result in less effective propagation and/or relaxations. The remaining parameters were chosen to obtain instances that are (a) nontrivial to solve and (b) usually feasible for the optimization problems (less often feasible for the feasibility problems). To accomplish this, we adjusted β and γ_2 empirically to sample instances near a phase transition where average problem difficulty peaks and there is a mix of feasible and infeasible instances. This required adjusting β to different values as the problem scaled up.

Table 2.3: Parameters for generation of problem instances.

	Segmented Problems			*Unsegmented*
	Feasibility	*Makespan*	*Tardiness*	*Feas. & Makespan*
α	0.5			
γ_1	0.5 for tight windows 0.25 for wide windows			0.25
γ_2	1.0			
β	Tight windows: 1/20 for 6–8 segs. 1/24 for 10 segs. 1/28 for 12 segs. 1/32 for 14 segs. Wide windows: 0.035	0.025 for ≤ 13 segs. 0.032 for > 13 segs.	0.05	1/15 for 5-8 segs. 1/20 for 9-12 segs. 1/30 for 12-16 segs. 1/35 for >16 segs.

2.11 Computational Results

We formulated and solved the instances with IBM's OPL Studio 6.1, which invokes the ILOG CP Optimizer for CP models and CPLEX for MILP models. We used OPL's script language to implement the Benders method. We generated 10 instances for each problem size and type, for a total of 580 instances.

Tables 2.4 and 2.5 display computational results for the segmented problems. The advantage of logic-based Benders increases rapidly as the problem scales up, relative to both CP and MILP. The Benders method failed to solve only 4 of 420 instances within ten minutes, while CP failed to solve 247 and MILP failed to solve 113.

Table 2.6 displays computational results for the unsegmented problems. The Benders method continues to have a substantial advantage over MILP, but it is considerably slower than CP on the easier problems. However, examination of the individual instances reveals that the Benders method is more robust. The Benders method solved all 160 unsegmented instances, while CP failed to solve 20 instances within ten minutes. CP was very fast for the instances it solved (average of 0.79 seconds), but Benders solved the remaining instances in an average of only 5.94

Table 2.4: Computation times in seconds for the segmented problem with tight time windows. The number of segments is 10% the number of jobs. Ten instances of each size are solved.

	Feasibility			Makespan			Tardiness		
Jobs	CP	MILP	Bndrs	CP	MILP	Bndrs	CP	MILP	Bndrs
60	0.1	14	1.9	60	7.7	6.4	0.1	16	3.0
80	181*	45	2.7	420*	147	11	63*	471*	20
100	199*	58	4.3	600*	600	17	547*	177*	11
120	272*	137	4.8	600*	600	39	600*	217*	2.9
140	306*	260*	6.8	600*	432*†	33	600*	373*	5.0
160	314*	301*	8.0	600*	359*	14			
180	600*	350*†	4.8	600*	557*†	5.3			
200	600*	†	5.8	600*	600*†	6.6			

*Solution terminated at 600 seconds for some or all instances.
†MILP solver ran out of memory for some or all instances, which are omitted from the average solution time.

seconds. This suggests that one ought to try CP first, and if it fails to solve the problem in a few seconds, switch to Benders.

The volatility of CP may be due to the fact that filtering and bounds propagation can be effective on a long time horizon when time windows interact in a certain way, but when this does not occur, a huge search tree is generated. This phenomenon may not affect the Benders method because the scheduling segments are small enough to result in limited search trees even when propagation is ineffective.

We also investigated the effectiveness of analytic Benders cuts. They incur greater overhead than nogood cuts, because each analytic cut requires multiple inequalities in the master problem. This could offset faster convergence. To test this hypothesis, we re-solved all the instances with nogood cuts but without analytic cuts (Table 2.7). We found that for segmented problems, the analytic cuts make little difference on the average when time windows are narrow. However, they bring significant and occasionally dramatic reductions in computation time for wide time windows. Because these cuts do no harm (on the average) in either case and are advantageous for wider time windows, it is advisable to use them. As for unsegmented problems, the analytic

Table 2.5: Average computation times in seconds for the segmented problem with wide time windows. The number of segments is 10% the number of jobs. Ten instances of each size are solved.

	Feasibility			Makespan			Tardiness		
Jobs	CP	MILP	Bndrs	CP	MILP	Bndrs	CP	MILP	Bndrs
60	0.05	12	1.9	0.2	16	5.8	0.2	8.0	2.3
80	0.28	22	2.5	180*	59	9.0	1.5	94	3.7
100	0.14	37	3.8	360*	403*	14	79*	594*	85*
120	0.13	61	5.0	540*	600*	25	600*	251*	183*
140	61*	175	7.0	600*	600*	107	600*	160*	4.3
160	540*	216*	4.8	600*	562*	157			
180	600*	375*†	4.5	600*	535*	10			
200	600*	†	5.5	600*	560*	6.9			

*Solution terminated at 600 seconds for some or all instances.
†MILP solver ran out of memory for some or all instances, which are omitted from the average solution time.

cuts are clearly beneficial and should be used.

2.12 Conclusion

We adapted logic-based Benders decomposition to a pure scheduling problem that lacks the natural decomposability of the planning and scheduling problems to which the method has been previously applied. The master problem assigns jobs to segments of the time horizon rather than to machines or other resources.

We generate instances that are (a) nontrivial to solve and (b) usually feasible for the optimization problems. We compute the computation time for these instances with varying time horizons. If an instance has a longer time horizon than the other instances, the number of jobs of that instance is also larger than the others. Thus, solving an instance may become hard with longer time horizons due to the increases in the number of jobs and the length of the time horizon.

We find that for single-facility scheduling, logic-based Benders scales up more effectively than state-of-the-art CP and MILP solvers. This is especially true for the segmented problem, in which jobs are not permitted to overlap segment boundaries.

Table 2.6: Average computation times in seconds for the unsegmented problem. The number of segments is 10% the number of jobs. Ten instances of each size are solved,

	Feasibility			Makespan		
Jobs	CP	MILP	Bndrs	CP	MILP	Bndrs
60	0.10	11	2.8	0.2	24	5.1
80	0.14	21	3.7	0.7	376*	8.7
100	0.25	35	7.0	1.1	600*	21
120	0.43	57	23	0.4	600*	93
140	0.72	97	65	1.2	600*	115
160	420*	188	9.0	241*	549*	67
180	123*	307*	79	61*	600*	168
200	180*	410*	29	180*	587*	21

*Solution terminated at 600 seconds for some or all instances.

Table 2.7: Effect of analytic Benders cuts on computation time. The last three columns show the percent of instances in which analytic cuts reduced computation time by more than the stated amount.

Problem class	% reduction		% of instances with reduction		
	Average	Maximum	$> 0\%$	$> 20\%$	$> 50\%$
Segmented makespan:					
tight windows	0	45	46	14	0
wide windows	12	85	79	46	11
Segmented tardiness:					
tight windows	−4*	37	60	6	0
wide windows	7	99	62	36	8
Unsegmented:					
makespan	12	64	76	59	8

*Reflects three very negative outliers.

The Benders method solves much larger instances of the feasibility and makespan problems, and its speed advantage increases rapidly as the problem size increases. It is somewhat faster on the tardiness problem.

The Benders master problem becomes more complex for the unsegmented problem, in which jobs may overlap segment boundaries. Benders decomposition continues to dominate MILP while being much slower than CP on most of the smaller instances. However, CP begins to lose its ability to solve instances as they scale up, whereas Benders continues to solve them, usually in a few seconds. Benders is therefore not necessarily the fastest method but clearly the most robust.

CP solves unsegmented instances quickly if it solves them at all. This suggests a strategy of applying CP first, and if it fails to solve the problem with a few seconds, switch to Benders. For segmented instances, Benders is always superior and should be used from the start.

Possible future research includes the development of Benders cuts for the unsegmented tardiness problem. In addition, convergence might be accelerated with the generation of multiple cuts, or by a "warm start" that adds a collection of Benders cuts to the initial master problem, as in (Aggoun and Vazacopoulos, 2004; Maravelias, 2006). The length of time segments might be adjusted dynamically for better performance. Finally, other forms of decomposition can be explored.

3 STAFFING OF A SERVICE CENTER WITH CROSS-TRAINED AGENTS, HETEROGENEOUS CUSTOMERS, AND QUALITY GUARANTEES

We model a service center at a large, global IT services delivery organization with cross-trained agents and multiple request classes. Agents may be classified as either high or low skilled and customer requests may be classified as simple or complex. Highly skilled agents can serve both simple and complex requests while low skilled agents can only serve simple requests. In addition, customer requests are tagged with a priority; higher priority requests preempt lower priority requests.

We model this system as a multi-server queueing system. Exact solution of the system is numerically intractable. We therefore apply approximation and bounding techniques to evaluate different control policies. Our goals are twofold: (i) to create a new method to determine an appropriate static agent base, and (ii) to determine an effective request-assignment policy. We introduce and analyze different approximations, capturing the operations of the service center with increasing fidelity. Our work demonstrates that a simple but effective request-assignment policy can meet the service level goals for different prioritized requests utilizing an appropriately determined static agent base.

3.1 Introduction

Customer service centers play a vital role in today's business world, offering remote or on site service to customers. These centers are expected to achieve low operating

This chapter is joint work with Aliza R. Heching, and Alan Scheller-Wolf.

costs while maintaining high service quality. These conflicting objectives make the management of service centers very challenging.

We are motivated by a service center at a large, global, IT services delivery organization; we describe a multi-server queueing model that captures this service center's operations. Service center agents are differentiated according to their level of expertise (high skilled and low skilled denoted by H and L, respectively). Customer requests belong to different priority classes ($\{1, ..., m\}$, where 1 denotes the highest priority class) and are of varying complexity (high or low denoted by H and L, respectively). Thus, each customer request is tagged with two labels: the priority class to which it belongs, $\{1, ..., m\}$, and the complexity level of the request, H or L. H customer requests can only be served by high skilled agents, whereas L customer requests can be served by either low skilled agents or high skilled agents.

The task of the service center manager is to determine an effective request-assignment policy given a fixed static high and low skilled agent base, so as to most efficiently meet service level goals for the different customer classes. For example, finishing at least a high percentage of customer requests on time is an important goal to achieve for a service center. The aim of this paper is to create an algorithm to analyze this multi-server queueing system under a preemptive-resume assumption and to determine an appropriate static agent base and a simple but effective request-assignment policy. We also evaluate the benefit of this policy relative to other benchmark policies.

The remainder of the chapter is organized as follows. In Section 3.2 we survey recent work on service center operations. Our queueing models capturing the service centers' operations, and solution strategies for these queueing models are introduced in Section 3.3. Computational results and insights are explored in Section 3.4. Extensions, future work, and concluding comments are provided in Section 3.5.

3.2 Literature Survey

Since the 1970s, service centers have been a fertile area for research with the aim of understanding their performance characteristics and managing their operations (Segal, 1974; Baskett et al., 1975; Henderson and Berry, 1976; Reiser and Lavenberg, 1980; Whitt, 1999; Garnett et al., 2002; Brown et al., 2005; Pot et al., 2008). Operational complexity increases in service centers that serve different types of requests requiring different agent skills. Here, it is important to consider multi-skilled agents as mentioned by Aksin et al. (2007), rather than a single pool of homogeneous agents. Stochastic service systems with multi-skilled agents were first described by Schwartz (1974). In this setting, the agent skill mix becomes another decision variable in the capacity planning problem, i.e., the problem of determining the number of agents required to satisfy the service goals associated with customer requests.

In general, to solve capacity planning problems with a fixed agent base, the performance of the system must first be calculated. Fundamental outputs, such as response times and the long-run fraction of time that agents are busy, are used to illustrate the performance of the service center (Gans et al., 2003). The two most common alternatives for evaluating service center performance are simulation models and analytic queueing models (Aksin et al., 2007).

Agnihothri et al. (2003) used simulation to quantify the impacts of several features of a service center, such as server utilization or the coefficient of variation of the service time, on the characteristics of the optimal multi-skilled workforce at a center with two types of jobs. Wallace and Whitt (2005) used simulation to propose a staffing algorithm for a fixed number of multi-skilled agents with a fixed number of extra waiting spaces and different call types. Cezik and L'Ecuyer (2008) used simulation to model a contact center with multi-skilled agents and abandonment.

With respect to the use of analytic queueing models to analyze multi-skilled service center performance, different methods have been used, depending on the arrival rate and the number of agents. Within the Halfin-Whitt heavy traffic regime, Harrison and López (1999) and Harrison and Zeevi (2004) studied dynamic scheduling of a multi-class queue with abandonment. Whitt (2002) considered an M/M/s queue with a single customer class and customer abandonment, deriving the sensitivity of performance to changes in model parameters. Garnett et al. (2002) proposed rules of thumb for the design of a large call center with impatient customers. Armony (2005) analyzed a large scale system with multiple server pools and a single customer class. The authors proposed an asymptotically optimal routing policy. In addition, they show that a heterogeneous server system outperformed its homogeneous server counterpart.

In cases where the system is not in a heavy traffic regime, the system performance has been analyzed by formulating the system as a Markov chain and applying the Matrix Analytic Method. Green (1985) formulated a service center model with multi-skilled agents, and showed that the underlying Markov chain has a matrix analytic form. Combined with the solution method developed by Neuts (1981), the steady-state distribution of the queueing system is calculated. A matrix analytic solution was also applied by Stanford and Grassmann (1993) to analyze a similar model with both specialized and flexible workers. However, the state space of the queueing model gets extremely large with multi-skilled agents. In this case the application of the Matrix Analytic Method (Neuts, 1981) is theoretically possible, but typically practically infeasible. Thus, approximation methods such as state space truncation and busy period approximation have been developed to tackle this problem.

Kao and Narayanan (1990, 1991) approximated their two priority class queueing system by truncating the Markov chain, separately limiting the number of jobs from each priority class. Using busy period approximations (Harchol-Balter et al., 2003b, 2005; Osogami et al., 2005a,b), intervals of time during which the agent is busy without interruption (so-called "busy periods") can be represented within a Markov chain framework using a phase-type distribution, by deriving moment approximations for the length of these busy periods. This approximation can reduce the size of the chain significantly, enabling the application of standard Matrix Analytic Methods. Harchol-Balter et al. (2005) found that matching three moments of these busy periods is usually possible using a phase-type distribution and provides sufficient accuracy - within a couple percent of simulation - for all of their experiments.

Busy periods approximation can also be used to evaluate the performance of a "Beneficiary-Donor" model (Harchol-Balter et al., 2003b; Osogami et al., 2005a; Enders et al., 2009) where one server (the *donor*) can help the other server (the *beneficiary*) with his jobs. This model is common in service facilities such as call centers and repair facilities. For example, in a repair facility, a technician who can handle jobs of any difficulty may be a donor server, and a technician who can only handle jobs that require limited expertise may be a beneficiary server (Green, 1985). Another example is call centers: a bilingual operator may be a donor server and a monolingual operator may be a beneficiary server (Stanford and Grassmann, 1993, 2000; Shumsky, 2004), or the donor server may be a cross-trained operator who can handle all types of calls, whereas a beneficiary server may be a specialized operator who can handle only a specific type of call (Shumsky, 2004). Often the beneficiary is considered to be *stealing idle cycles* from the donor, and the service discipline is referred to as "cycle stealing" (Osogami et al., 2005b).

Related to the classic Beneficiary-Donor model, threshold type priority policies in which the donor server helps the beneficiary server, conditional on the number of requests existing in each queue, have been shown to perform well in diverse settings, such as inventory systems facing demand from customers with different levels of patience (Enders et al., 2009), and for task assignment and capacity setting in systems with prioritized jobs and heterogeneous servers (Harchol-Balter et al., 2003a,b, 2005;

Osogami et al., 2005b). Threshold policies prevent the build up of a long queue of lower priority requests when there is a small number of high priority requests. Harchol-Balter et al. (2005) analyzed M/PH/k queues with $m > 2$ preemptive-resume priority classes where there is no distinction between the levels of expertise a job will require. They introduced a new technique, Recursive Dimensionality Reduction (RDR) by which the dimensionality of the infinite Markov chains can be iteratively reduced by using busy period transitions. They used Matrix Analytic Methods to find the steady-state distribution and calculated the mean response time by Little's Law.

We have provided only an overview of service center literature; for a detailed review of recent literature on service centers, we refer to the works of Aksin et al. (2007), Mandelbaum (2004) and Gans et al. (2003).

As mentioned in Section 3.1, we are inspired by a service center at a global IT services delivery organization. We model a multi-server queueing system with cross-trained agents, heterogeneous customers, and service quality guarantees. The aim of this paper is to provide a method to solve queueing models so as to perform capacity planning for this service center (i.e., to determine an appropriate static agent base and an effective request-assignment policy). Some particulars of the operations of the service center require us to consider models that are different than those that appear in the existing literature: In particular, to our knowledge our paper is the first to consider the following combination of factors within a service center model: (i) agents are differentiated according to their level of expertise, which may limit the specific classes of requests they can serve; (ii) requests of different complexity levels also belong to different priority classes; higher priority class requests can preempt lower priority class requests; (iii) cycle stealing type threshold policies are utilized; and (iv) the system does not operate under a heavy traffic regime.

3.3 The Models

We describe a multi-server queueing system that captures the operations at a global IT services delivery organization. We first discuss the business setting and then describe the model. The agents differ according to their levels of expertise: high skilled and low skilled. In addition, customer requests belong to different priority classes. Thus, each request is tagged with two labels: the priority class to which it belongs, and its complexity. Customer requests that require a high level of expertise can only be served by high skilled agents, whereas requests requiring a low level of expertise can be served by both low skilled agents and high skilled agents.

There are two important service rules: (i) requests belonging to a higher priority class always have service priority over requests belonging to lower priority classes; and (ii) within a priority class, high complexity requests are given service priority, unless the service policy specifies otherwise. For example, the service policy may specify that within a priority level, for a sufficiently large volume of low complexity requests, these requests receive highest service priority until their volume drops below a prespecified level.

We model this system using a Markov chain to track the number of requests in the system at any point in time. High and low complexity requests are denoted by H and L, respectively. The priority classes are denoted by the set $\{1, 2, ..., m\}$ where 1 represents the highest priority class. The corresponding Markov chain has 2m dimensions - one for each (priority, complexity) pair. This large state space renders exact analysis numerically intractable for all but the smallest number of priority and complexity levels. We therefore apply approximation and bounding techniques to evaluate different control policies.

To model how the global IT services delivery organization schedules requests, we consider the class of "threshold-based priority policies." These policies prioritize requests according to the number of each request type in the system, such as: for any priority level, if the number of low complexity requests exceeds a pre-specified threshold, the high skilled agent stops serving high complexity requests and serves low complexity requests until the number of low complexity requests drops below a pre-specified level. For example, consider a service center to which only 1L (i.e., customer requests belonging to priority class 1 with low complexity) and 1H (i.e., customer requests belonging to priority class 1 with high complexity) requests arrive. Under a threshold-based priority policy 1L requests have priority over 1H requests only when the number of 1L requests equals or exceeds t_{1L}, the threshold at which high skilled agents will serve requests of type 1L until there are $(t_{1L} - 1)$ 1L requests in the system. (If 1H requests always have priority over 1L requests, $t_{1L} = \infty$.) This threshold policy is applied within all priority classes (i.e., $\{1, 2, ..., m\}$) possibly with different parameters t_{jL}; the service rule that requests belonging to a higher priority class always have priority over requests belonging to lower priority classes is always valid.

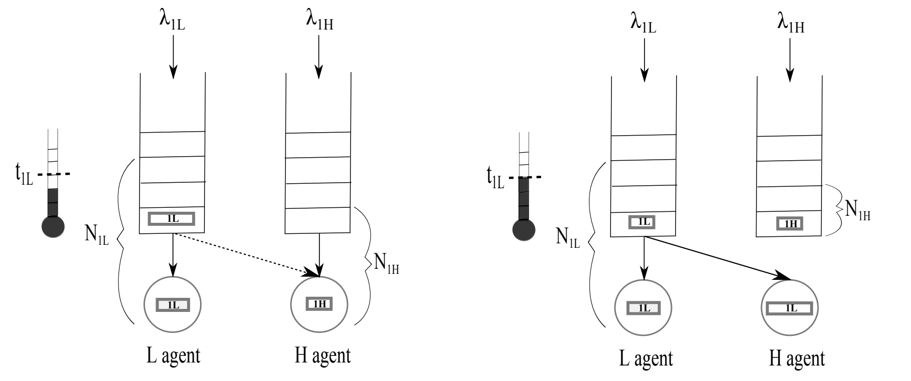
A busy period in the Markov chain is defined as an interval of time during which an agent is busy without interruption. We use so-called *busy period approximations* with our threshold-based priority policy to reduce the size of our (numerically intractable) Markov chain. For example, a high skilled agent stops serving 1H requests and serves 1L requests as soon as the number of 1L requests is higher than $(t_{1L} - 1)$; the high skilled agent serves 1L requests without interruption until the number of 1L requests is $(t_{1L} - 1)$. The period of time during which the high skilled agent serves low complexity requests (i.e, 1L in the example) until she returns back to serving high complexity requests (i.e., 1H in the example) is referred to as a "busy period." We derive three-moment approximations for the length of the busy periods we consider, and represent them in our Markov chain by a phase-type distribution with the same moments.

Busy period approximations combined with threshold-based priority policies convert our intractable Markov chain into a tractable one, and enable application of standard Matrix Analytic Method techniques to the service center problem. We solve the problem in stages. In Section 3.3.1, we first consider a system with only two agents - one high skilled and one low skilled - and a single priority class. In Section 3.3.2 we consider a more complex system where there are two priority classes. In Section 3.3.3, we generalize the model, first by allowing for $n > 2$ agents with each of the two skill levels, and then discussing, in addition, $m > 2$ priority classes.

3.3.1 System 1: 1H and 1L requests

In this section we consider a system where there is a single priority class and only two agents: one high skilled agent and one low skilled agent. We denote the high skilled agent by H and the low skilled agent by L. The high skilled agent serves requests at rate μ_H; the low skilled agent serves requests at rate μ_L. Consistent with what is observed in our motivating industry context, the service rate depends only on the agent type and not on the request type. In this system, only two types of requests arrive to the service center: 1H and 1L. 1H requests can only be served by the high skilled agent while 1L requests can be served by both the high and low skilled agents. We use t_{1L} to denote the threshold at which the high skilled agent gives priority to 1L requests until the number of 1L requests drops below t_{1L}. This system is similar to that described by Harchol-Balter et al. (2005) and Osogami et al. (2005a,b).

Figure 3.1a depicts the system. We observe two agents: one high skilled and one low skilled. Customer requests arriving to the system join an agent queue based upon the complexity of the requests; high complexity requests join the high skilled agent queue while low complexity requests join the low skilled agent queue. Arcs from the queues to the agents represent possible service paths a request may follow. For example, the dashed arc from the low skilled agent's queue to the high skilled agent represents the situation where there are no 1H requests in the system so a 1L request may be served by the idle high skilled agent. There are two 1L requests in the system illustrated in Figure 3.1a: one being served by the low skilled agent and one waiting in the low skilled agent's queue, while, the high skilled agent serves a 1H request.



(a) When the number of 1L requests is less than t_{1L}

(b) When the number of 1L requests equals t_{1L}

Figure 3.1: Representation of the service center for System 1 when threshold for 1L requests, t_{1L}, is equal to 3. The high skilled agent serves the low skilled agent if the number of 1L requests, N_{1L}, is above $(t_{1L} - 1)$ or the number of 1H requests, N_{1H}, is zero. A dashed arc illustrates a possible service path to a high skilled agent that a 1L request might follow when the number of 1L request is less than t_{1L}. Solid arcs illustrate the service paths a request can always receive, given the system state.

Under a threshold-based policy, the high skilled agent stops serving 1H requests and starts serving 1L requests when the number of 1L requests in the system, N_{1L}, meets or exceeds a threshold value denoted by t_{1L}. Figure 3.1b illustrates this phenomenon. Figure 3.1b depicts one high skilled and one low skilled agent, each with his own queue. We observe one 1H and three 1L requests in the system. Since the number of 1L requests is equal to the threshold specified for this system, the high skilled agent stops serving the 1H request and begins serving 1L requests. The high skilled agent continues to serve 1L requests until the number of 1L requests in the system drops below t_{1L}.

3.3.1.1 Markov Chain of Priority Class 1 Requests

We use a Markov chain to model the system in Figure 3.1. 1L and 1H requests arrive at rates λ_{1L} and λ_{1H}, respectively. The Markov chain representing this system tracks the number of 1L requests in the system, N_{1L}, and the number of 1H requests in the system, N_{1H}, and thus is infinite in two dimensions. However, the threshold-based policy allows for a simplification of the Markov chain by using busy period approximation. The simplified Markov chain is illustrated in Figure 3.2 for the case where $t_{1L} = 3$.

The simplified Markov chain depicted in Figure 3.2 is infinite in only one dimension. The numbers of 1L and 1H in the system requests are tracked on the vertical and horizontal axis, respectively. A state in the Markov chain is defined differently above and below the threshold value: Until the number of 1L requests reaches t_{1L}, a state is a pair consisting of the number of 1L and 1H requests in the system. After that, the pair is represented by t_{1L}^{+} or $t_{1L_D}^{+}$ 1L requests and the number of 1H requests in the system. As depicted in Figure 3.2, we track the exact number of 1H requests on the horizontal axis; on the vertical axis, we track the number of 1L requests up to the threshold value, t_{1L}. States denoted by t_{1L}^{+} and $t_{1L_D}^{+}$ signify that there are *at least* t_{1L} 1L requests present. States at which the number of 1L requests is illustrated as $t_{1L_D}^{+}$ are "dummy states" used in the phase-type approximation during a busy period. Note if we are in state $(j_{1L}, 0_{1H})$ where $j \geq 2$, the service rate is $\mu_L + \mu_H$, i.e., the high skilled agent serves 1L requests.

A busy period begins when the number of 1L requests equals $(t_{1L} - 1)$ and a 1L request arrives. Let's call this busy period "a type 1L" busy period. During a type 1L busy period, the high skilled agent serves 1L requests, as illustrated in Figure 3.1; and preempts any 1H requests currently in service. The busy period ends when the number of 1L requests drops below t_{1L}.

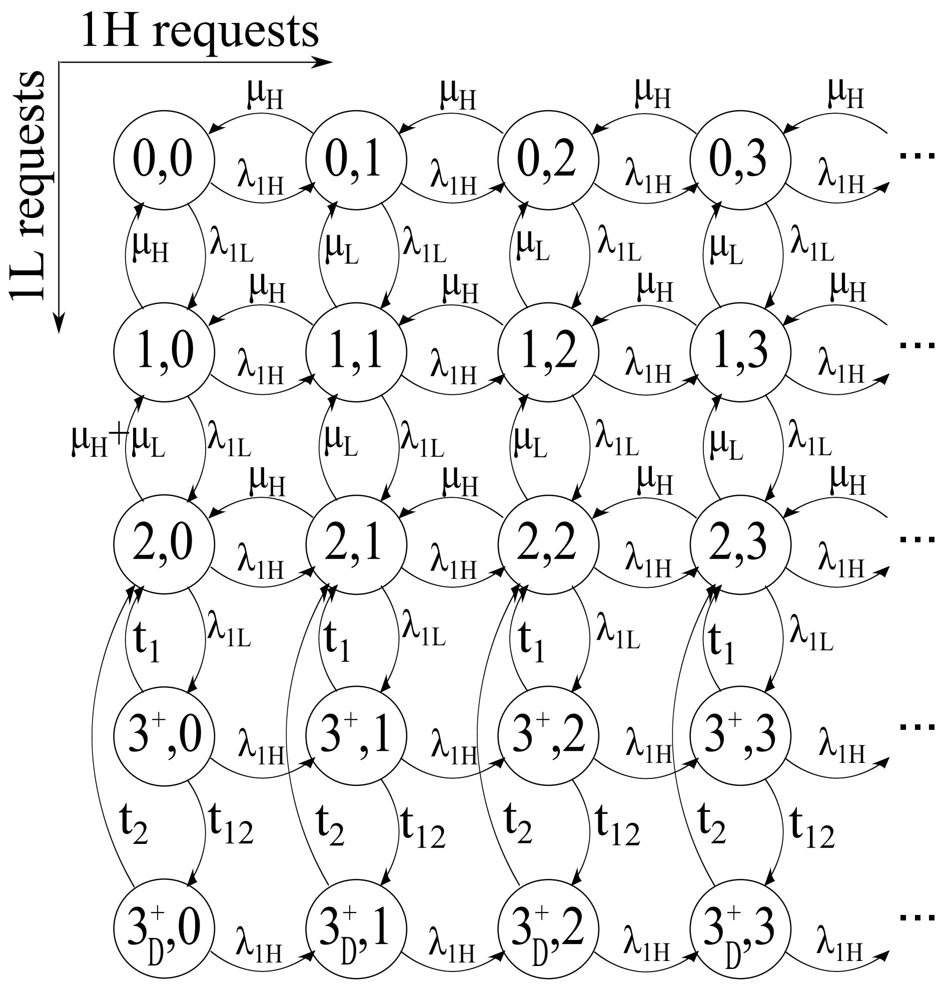


Figure 3.2: Markov chain of 1L and 1H requests. The threshold for 1L requests, t_{1L}, is 3. The busy period is represented by a two phase PH distribution via rates t_1, t_2, and t_{12} (with Coxian representation).

3.3.1.2 Moments of a Type 1L Busy Period

We use a phase type distribution (specifically, a Coxian distribution) to match the first three moments of the distribution of this busy period as in the work of Harchol-Balter et al. (2005). Harchol-Balter et al. (2005) showed that matching the first three moments of busy periods is sufficiently accurate.

To calculate the first three moments of a type 1L busy period, we can use the

Markov chain in Figure 3.3, which tracks 1L requests during a type 1L busy period. We will use a similar chain in later sections; however, when only 1L and 1H requests are in the system, we can model the amount of time the high skilled agent serves 1L requests during a busy period as a simple M/M/1 system. Thus (without jumping to the Markov chain in Figure 3.3), the closed form of the first three moments and the expected number of 1L requests during the busy period are known (Adan and Resing, 2002). Given these moments, the parameters of in the PH distribution (i.e., t_1, t_2, and t_{12}) are calculated via the closed form solution provided by Osogami and Harchol-Balter (2003).

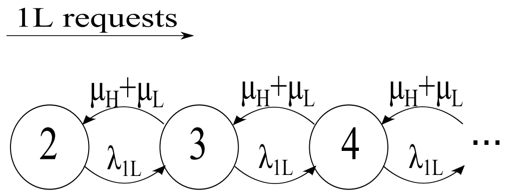


Figure 3.3: Markov chain of 1L requests that can be used to calculate the first three moments of a type 1L busy period that starts with an arrival of a 1L request when there are $(t_{1L}-1)$ (in this case 2) 1L requests in the system. The busy period is the first passage time from state 3 (t_{1L}) to state 2 $(t_{1L}-1)$.

3.3.1.3 Expected Response Times of Priority Class 1 Requests

We use the Matrix Analytic Method (MAM) to analyze the stationary probabilities of the now 1-dimensionally infinite chain. (We describe this in detail in Appendix A.) We use $E\left[N_{1L}^{BP}\right]$ to represent the expected number of 1L requests during a type 1L busy period, π_m^{BP} to denote the stationary probability of state m in the Markov chain represented in Figure 3.3, where m is the number of 1L requests in the system during a busy period. The expected response time of 1H requests, $E\left[T_{1H}\right]$, and 1L requests, $E\left[T_{1L}\right]$, are computed as shown in Equation 3.1 by using the stationary probability of state (i_{1L}, j_{1H}), $\pi_{(i_{1L},j_{1H})}$, in Figure 3.2 calculated via MAM:

$$\begin{aligned}
E\left[T_{1H}\right] &= \frac{\sum_{j} j(\sum_{i} \pi_{(i_{1L},j_{1H})})}{\lambda_{1H}} \\
E\left[N_{1L}^{BP}\right] &= \frac{\sum_{m \geq t_{1L}} (m-(t_{1L}-1))\pi_m^{BP}}{1-\pi_{(t_{1L}-1)}^{BP}} + (t_{1L}-1) \\
E\left[T_{1L}\right] &= \frac{\sum_{i \leq t_{1L}-1} i(\sum_{j} \pi_{i_{1L},j_{1H}}) + E\left[N_{1L}^{BP}\right] \sum_{j}(\pi_{(t_{1L}^{+})_{1L},j_{1H}} + \pi_{(t_{1LD}^{+})_{1L},j_{1H}})}{\lambda_{1L}}
\end{aligned} \tag{3.1}$$

where $j \geq 0$, and $i \in \left\{0, .., t_{1L}-1, t_{1L}^{+}, t_{1LD}^{+}\right\}$.

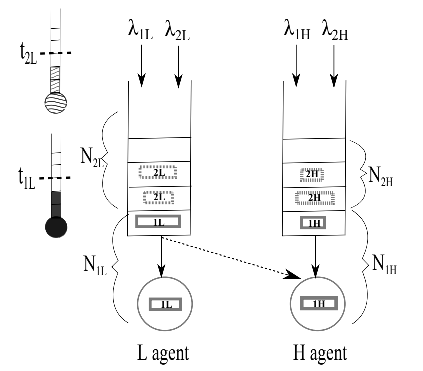
The Markov chain depicted in Figure 3.2, representing the case where there is a single priority class, serves as a building block for more complex systems that we explore next.

3.3.2 System 2: 1H, 1L, 2H and 2L requests

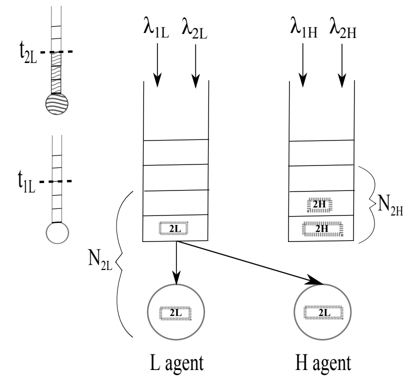
System 2 builds upon the basic system described in System 1 by considering a second priority class. In System 2 we continue to model a system with two agents - one high skilled and one low skilled (denoted by H and L, respectively).

Thus, there are four possible combinations of priority and complexity with which a request may be tagged: 1H, 1L, 2H, and 2L. 1H and 2H requests can only be served by the high skilled agent, while 1L and 2L requests can be served by both low skilled and high skilled agents. There are two threshold values in this system: t_{1L} and t_{2L}: t_{1L} is the threshold value of 1L requests at which the high skilled agent begins to serve 1L requests until the number of 1L requests drops below t_{1L} as introduced in Section 3.3.1; and t_{2L} is the threshold value of 2L requests in the system at which the high skilled agent begins to serve 2L requests until the number of 2L requests drops below t_{2L}. However, as described by the service rules in Section 3.3, priority class 1 requests always receive service priority over priority class 2 requests. Accordingly, the high skilled agent will only begin to serve priority class 2 requests (high or low complexity) if there are no priority class 1 requests in the system that the agent can serve.

Figure 3.4 depicts the queueing system for System 2. Figure 3.4a depicts the system with two agents: one high skilled and one low skilled. Each agent maintains its own queue, where high complexity requests (for both priority classes) are queued in the high skilled agent queue and low complexity requests (for both priority classes) are queued in the low skilled agent queue. There is one 1H and two 2H requests in the high skilled agent's queue, and one 1L and two 2L requests in the low skilled agent's queue and the high skilled agent is serving a 1H request. Figure 3.4b depicts a different scenario where there are only priority class 2 requests in the system: there are three 2L and two 2H requests. Since the number of 2L requests in the system equals the threshold value t_{2L} and there are no requests of higher priority in the system, the high skilled agent begins to serve 2L requests - preempting 2H requests - until the number of 2L requests drops below t_{2L}.



(a) When the numbers of 1L and 2L requests are below t_{2L} and t_{2L}, and there are type 1 requests in the service center



(b) When the number of 2L requests equals t_{2L} and there are not any type 1 requests in the service center

Figure 3.4: Representation of the service center for System 2 when thresholds for 1L and 2L requests, t_{1L} and t_{2L}, are equal to 3. The high skilled agent serves 2L requests if $N_{2L} \geq t_{2L}$ and $N_{1L} + N_{1H} = 0$; or $N_{1H} + N_{1L} + N_{2H} = 0$.

3.3.2.1 Markov Chain of Priority Class 2 requests

To solve this queueing system, we again model the system as a Markov chain. 1L, 1H, 2L, and 2H requests arrive with rates λ_{1L}, λ_{1H}, λ_{2L}, and λ_{2H}, respectively. If we model the exact system as a Markov chain, it will be infinite in four dimensions (i.e., the number of priority, complexity pairs). Using busy period transitions we reduce the Markov chain to a 1D infinite chain. Figure 3.5 depicts the simplified Markov chain.

Response times for priority class 1 requests can be calculated using the Markov chain depicted in Figure 3.2, since under our service rule, these requests are always served before priority class 2 requests. Thus, 2H and 2L requests do not interfere with priority class 1 requests. The aim of this section is to calculate the response times of 2H and 2L requests in Figure 3.5.

We track the exact number of 2H requests on the horizontal axis, and on the vertical axis, we track the number of 2L requests up to the threshold value, t_{2L} (i.e., 3) in Figure 3.5. Unlike the Markov chain of System 1 (Figure 3.2), there are three distinct state spaces illustrated in Figure 3.5: 0 Priority Class 1, 1^+ Priority Class 1, and 1^+_D Priority Class 1.

The three state spaces track the number of priority class 1 requests up to 1, the number of high skilled agents. (This will continue to hold in cases when there are more high skilled agents.) Depending on the number of 1L or 1H request in the system, the service rates at which priority class 2 requests can be served may change: When the system is in a state in 0 Priority Class 1 state space, both agents can serve existing 2H and 2L requests. The service rates of the high and the low skilled agent are thus μ_H and μ_L, respectively. Therefore, the Markov chain of 0 Priority Class 1 state space for 2H and 2L requests is identical the Markov chain of System 1 illustrated in Figure 3.2 except the arrival rates, which are λ_{2H} and λ_{2L} in Figure

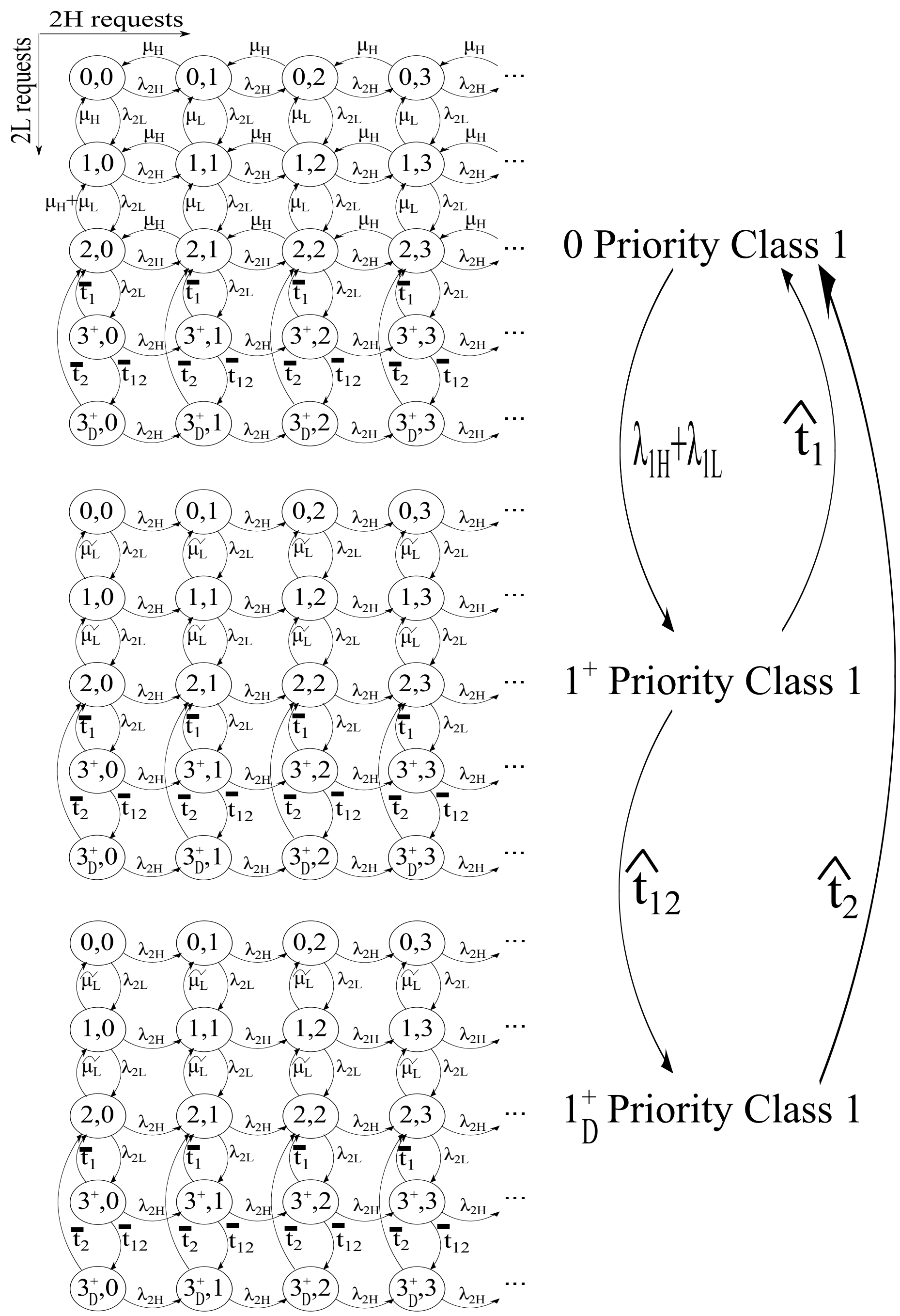


Figure 3.5: Markov chain of 1H, 1L, 2H, and 2L requests. The threshold for 2L requests is 3. The busy periods are represented by two phase PH distributions (with Coxian representations).

3.5. There are multiple types of busy periods in this system. We denote these busy periods based on the request types that started the busy period. The first type we discuss is a priority class 1 busy period. When a priority class 1 request arrives (1L or 1H), the system transfers into 1^+ Priority Class 1 state space and a busy period begins. Let's call this busy period "a priority class 1" busy period.

The priority class 1 busy period corresponds to the time the high agent cannot serve priority class 2 requests, as he is busy with priority class 1 requests, until the system returns to the 0 Priority Class 1 state space. For example, assume the system is in state (2 2L,1 2H) in 0 Priority Class 1 state space, then a 1L (or a 1H) request arrives. (As $\mu_H > \mu_L$ typically, a 1L request gets routed to the high skilled agent instead of the low skilled agent.) The system transitions into (2 2L,1 2H) in 1^+ Priority Class 1 state space. The priority class 1 request (1L or 1H) has higher priority than all the 2H requests, thus, the high skilled agent starts serving the priority class 1 request. Hence, the first three moments of the hitting times from states (1_{1L}, 0_{1H}) and (0_{1L}, 1_{1H}) to state (0_{1L}, 0_{1H}) in Figure 3.2 are needed to approximate the priority class 1 busy period between the 1^+ Priority Class 1 state space and the 0 Priority Class 1 state space. Conditional on the state in which a type 1 busy period starts (state (1_{1L}, 0_{1H}) or (0_{1L} 1_{1H})), the first three moments from each state to the state (0_{1L}, 0_{1H}) are aggregated. These three moments of the hitting times are calculated from the Markov chain illustrated in Figure 3.2 (Osogami and Harchol-Balter, 2003).

Depending upon the state of the system during the priority class 1 busy period, the low skilled agent may continue serving 2L requests. For example, suppose a 1L request arrives before the departure of the existing priority class 1 request. There are now 2 priority class 1 requests in the system, one of which is a 1L request. The low skilled agent begins serving the new 1L request. However, during the priority class 1 busy period, the low skilled agent may serve 2L requests, if there is only 1 priority class 1 request or when there are only 1H requests in the system. We approximate the amount of time that the low skilled agent is available to serve 2L requests during the priority class 1 busy period.

We utilize this approximation because if we wanted to track the exact service rate, this would require transitions between busy period approximations with different rates. For example, suppose there are more than t_{2L} 2L requests in 1^+ Priority Class 1 state space (during a priority class 1 busy period). Then, the 2L requests can be served with service rate at most μ_L. However, if the priority class 1 busy period is over, the service rate becomes $\mu_L + \mu_H$. Thus, the service rate changes conditional on the priority class 1 busy period, which is also approximated by a two phase PH distribution. Since transitioning between different approximations leads to significant inaccuracies, we avoid it through the use of a single approximate service rate, $\tilde{\mu}_L$.

3.3.2.2 Approximate Service Rate $\tilde{\mu}_L$

The approximate rate, $\tilde{\mu}_L$, is calculated adjusting the low skilled agent's service rate μ_L by two key factors: the percentage of the time that the low skilled agent can serve 2L requests during the priority class 1 busy period, and the priority queueing effect on the low skilled agent. To calculate these two key factors, the stationary probabilities of the Markov chain illustrated in Figure 3.6 are used to approximate $\tilde{\mu}_L$.

In Figure 3.6, the states drawn by dotted lines are the non-busy period states (i.e., the high skilled agent can serve priority class 2 requests). The states represented by double circled nodes are the states at which 2L requests can be served during a priority class 1 busy period. 2L requests cannot be served in any of the remaining states in Figure 3.6, since the low skilled agent is busy with 1L requests. Hence, the percentage of the time that the low skilled agent can serve 2L requests during the priority class 1 busy period, β, is calculated from the Markov chain illustrated in

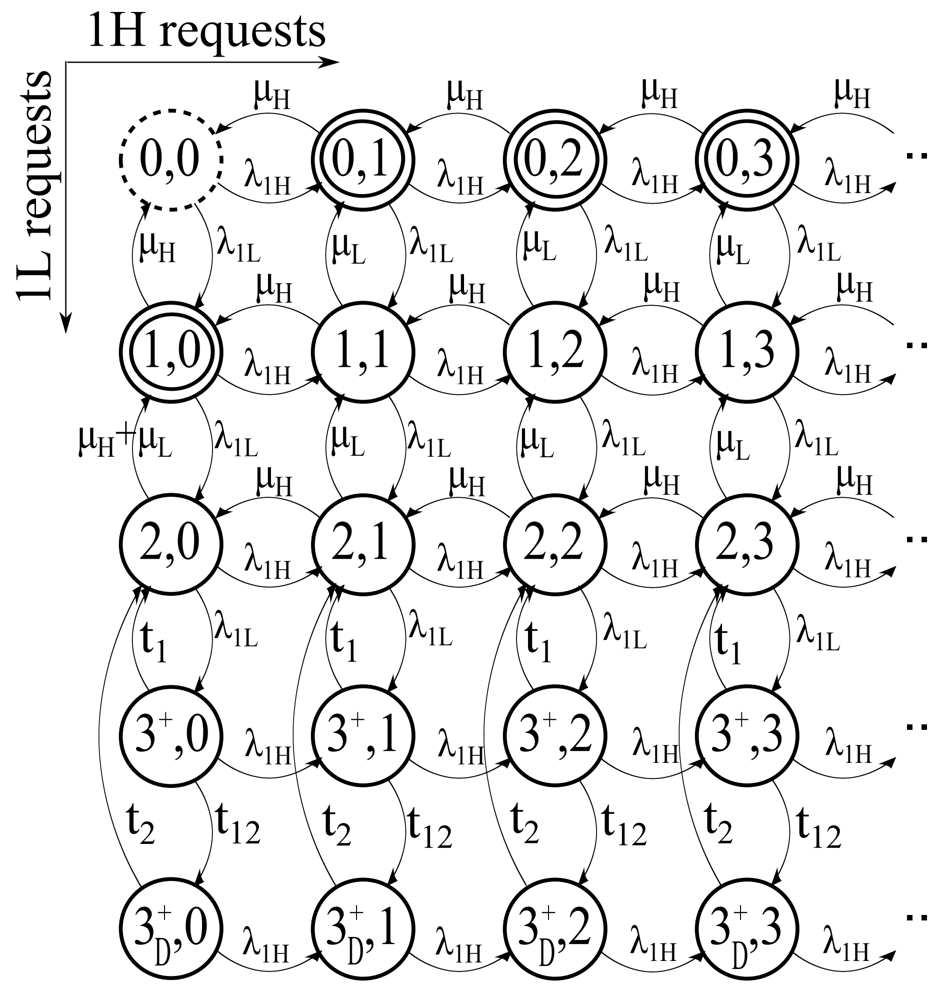


Figure 3.6: Markov chain of 1L and 1H requests used to calculate the approximate service rate for 2L requests during a priority class 1 busy period.

Figure 3.6 where $\pi_{(i_{1L},j_{1H})}$ is the stationary probability of state (i_{1L}, j_{1H}):

$$\beta = \frac{\pi_{(1_{1L},0_{1H})} + \sum_{j\geq 1} \pi_{(0_{1L},j_{1H})}}{1 - \pi_{(0_{1L},0_{1H})}} \tag{3.2}$$

In addition to the percentage of time the low skilled agent does not serve 2L requests during a priority class 1 busy period, the priority queueing effect on the low skilled agent service rate for 2L requests is important, i.e., the fact that 1L requests preempt 2L requests. We approximate this priority queueing effect via γ; γ approximates the difference in performance for 2L requests during a priority class 1 busy period using preemptive-priority queueing discipline and queueing with an approximate, constant service rate (i.e., $\tilde{\mu}_L$). We take γ equal to the sum of the stationary probabilities of the states (in Figure 3.6) where 2L requests are not preempted by priority class 1 requests at the low skilled agent's system.

$$\gamma = \pi_{(0_{1L},0_{1H})} + \pi_{(1_{1L},0_{1H})} + \sum_{j\geq 1} \pi_{(0_{1L},j_{1H})} \tag{3.3}$$

Then, we calculate the approximate low skilled agent's service rate for 2L requests during a priority class 1 busy period, $\tilde{\mu}_L$, as shown in Equation 3.4.

$$\tilde{\mu}_L = \gamma * \beta * \mu_L \tag{3.4}$$

This simple approximation proved to work surprisingly well in our numerical experiments.

3.3.2.3 Moments of a Type 2L Busy Period

The second type of busy period is the busy period of 2L requests started with an arrival of a 2L request when there are $(t_{2L}-1)$ 2L requests in the system as illustrated in Figure 3.5. Let's call this busy period "a type 2L" busy period. We use the Markov chain in Figure 3.7 to calculate the first three moments of this busy period. The states in this Markov chain represent the number of 2L requests in the system. When there are zero priority class 1 requests in the system, the service rate is $\mu_L + \mu_H$ since both high and low skilled agents serve 2L requests; otherwise the service rate is $\tilde{\mu}_L$ as the low skilled agent may serve 2L requests depending the number of priority class 1 requests.

After approximating a two phase PH distribution by using the first three moments of the hitting times from states 3 (t_{2L}) to state 2 $((t_{2L}-1))$ calculated from the Markov chain in Figure 3.7, producing rates $\bar{t}_1$, $\bar{t}_2$, and $\bar{t}_{12}$ used in Figure 3.5, we reduce the size of the Markov chain significantly.

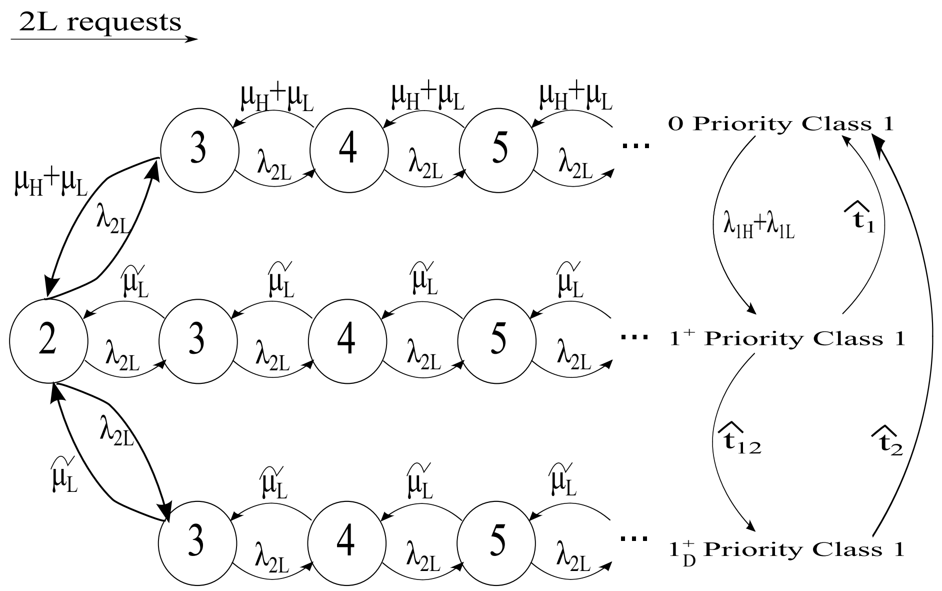


Figure 3.7: Markov chain of 2L requests used to calculate the first three moments of a type 2L busy period that starts with an arrival of a 2L request when there are 2 (i.e., $(t_{2L}-1)$) 2L requests in the system.

3.3.2.4 Expected Response Times of Priority Class 2 Requests

We compute the expected response time of 2H and 2L requests by calculating the limiting probabilities of the Markov chain in Figure 3.5. We use MAM, similar to System 1. Let $\pi^k_{(i,j)}$ denote the stationary probability of state (i_{2L}, j_{2H}) at state space k. The expected response time of 2H request is calculated as follows.

$$E\left[T_{2H}\right] = \frac{\sum\limits_{j} j(\sum\limits_{k,i} \pi^k_{(i_{2L},j_{2H})})}{\lambda_{2H}} \tag{3.5}$$

where $k \in \left\{0 \text{ Priority Class } 1, 1^+ \text{ Priority Class } 1, 1^+_D \text{ Priority Class } 1\right\}$, $j \geq 0$, and $i \in \left\{0, .., t_{2L}-1, t^+_{2L}, t^+_{2LD}\right\}$.

We now calculate the expected response time of 2L requests. We first need to compute the expected number of 2L requests during a busy period, since we do not keep track of 2L requests in the Markov chain (Figure 3.5).

We use the Markov chain in Figure 3.8 to estimate the expected number of 2L

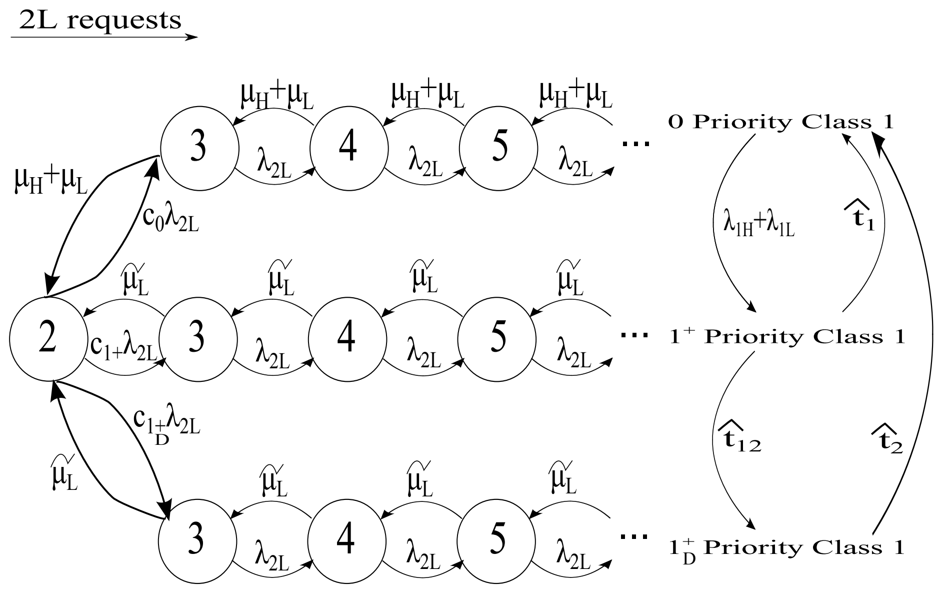


Figure 3.8: Markov chain of 2L requests used to calculate the expected number of 2L requests during a busy period started with an arrival of a 2L request when there are 2 (i.e., $(t_{2L}-1)$) 2L requests in the system.

requests during a busy period. Figure 3.8 is the same as Figure 3.7 except for the transition rates from state 2 (i.e., $(t_{2L}-1)$) to state 3 (i.e., t_{2L}). Since the probabilities of starting a busy period in different state spaces are not the same, the transition rates from state 2 to states 3 are weighted in Figure 3.8 with coefficients c_0, c_{1^+}, and $c_{1^+_D}$. These coefficients represent the probability of starting a type 2L busy period in the different state spaces, and are determined by the stationary probabilities of the Markov chain illustrated in Figure 3.5. Thus, we first solve the Markov chain in Figure 3.5 by MAM to calculate the stationary probabilities. Then, the coefficients, c_0, c_{1^+}, and $c_{1^+_D}$, are calculated by Equation 3.6.

$$\begin{aligned} c_0 &= \sum_j \pi^{\text{0 Priority Class 1}}_{((t_{2L}-1)_{2L},j_{2H})} \\ c_{1^+} &= \sum_j \pi^{1^+\text{ Priority Class 1}}_{((t_{2L}-1)_{2L},j_{2H})} \\ c_{1^+_D} &= \sum_j \pi^{1^+_D\text{ Priority Class 1}}_{((t_{2L}-1)_{2L},j_{2H})} \end{aligned} \tag{3.6}$$

We compute the expected number of 2L requests *during a busy period* as shown in Figure 3.8, $E\left[N_{2L}^{BP}\right]$, and the expected response time of 2L requests,$E\left[T_{2L}\right]$ in Equation 3.7 where $\breve{\pi}_i^k$ is the stationary probability of state i at state space k in Figure 3.8.

$$
\begin{aligned}
E\left[N_{2L}^{BP}\right] &= \frac{\sum\limits_{i \geq t_{2L}} (i-(t_{2L}-1))(\sum\limits_{k} \breve{\pi}_i^k)}{1-\breve{\pi}_{(t_{2L}-1)}} + (t_{2L}-1) \\
E\left[T_{2L}\right] &= \frac{\sum\limits_{i \leq t_{2L}-1} i(\sum\limits_{k,j} \pi^k_{i_{2L},j_{2H}}) + E\left[N_{2L}^{BP}\right] \sum\limits_{j,k}(\pi^k_{(t_{2L}^+)_{2L},j_{2H}} + \pi^k_{(t^+_{2L\,D})_{2L},j_{2H}})}{\lambda_{2L}}
\end{aligned} \tag{3.7}
$$

3.3.3 System 3: $n > 2$ Agents and $m > 2$ Priority Classes

We now build upon the system described in System 2, increasing the number of high and low skilled agents, and adding more priority classes. First, we consider a system with two priority classes and more than two agents in Section 3.3.3.1. Then, in Section 3.3.3.2, we discuss generalizing the number of priority classes.

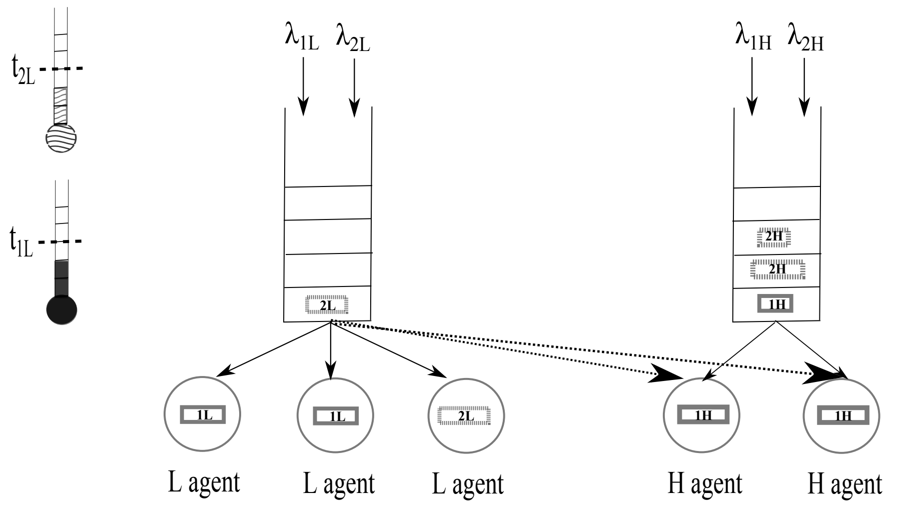
3.3.3.1 System 3(i): $n > 2$ Agents

System 3(i) is based upon System 2 (1L, 1H, 2L, and 2H requests): we increase the number of high and low skilled agents. The number of high skilled agents is denoted by n_H, whereas, the number of low skilled agents is denoted n_L. Each high skilled agent serves requests with rate μ_H, and each low skilled agent serves requests with rate μ_L. Four types of requests, 1H, 1L, 2H, and 2L, arrive to the service center: 1H and 2H requests can only be served by high skilled agents; 1L and 2L requests can be served by high or low skilled agents depending on the threshold policies. There are two threshold values in this system: t_{1L} and t_{2L}. We denote t_{1L} by the threshold at which high skilled agents serve 1L requests until the number of 1L requests drops below t_{1L} as introduced in Sections 3.3.1 and 3.3.2. Similarly, t_{2L} is the threshold at which high skilled agents serve 2L requests until the number of 2L requests drops below t_{2L} if the number of priority class 1 requests is less than the number of high skilled agents, as priority class 1 requests are always given higher priority in the queue relative to priority class 2 requests. Note that if the number of 1L requests is above t_{1L} but below n_H, there are idle high skilled agents that can serve 1H requests (if there are any 1H requests in the system). If there are still idle high skilled agents, they start serving priority class 2 requests.

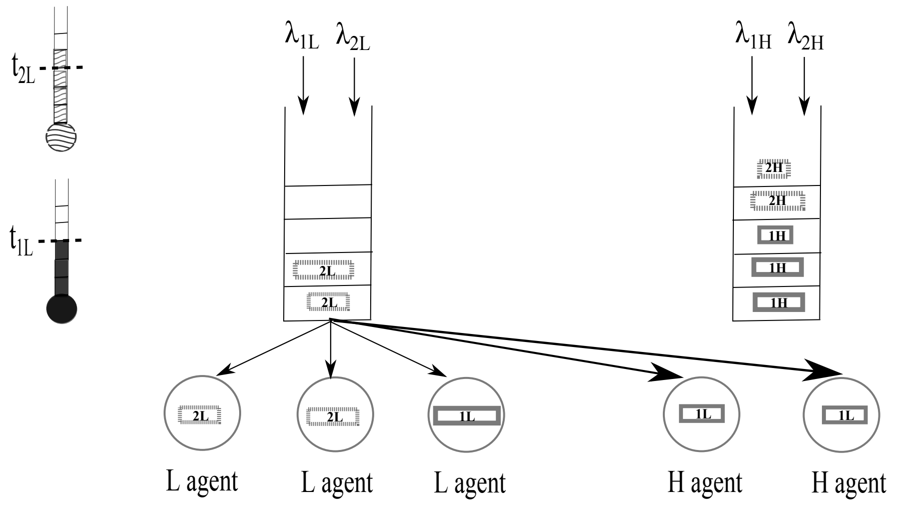
Figure 3.9a depicts this system. We observe three low skilled agents ($n_L = 3$) and two high skilled agents ($n_H = 2$). Customer requests arriving to the system join an agent queue based upon the complexity of the requests; high complexity requests join the high skilled agents' queue while low complexity requests join the low skilled agents' queue. There are one 1H and two 2H requests in the high skilled agents' queue and one 2L request in the low skilled agents' queue. Two low skilled agents serve 1L requests and one low skilled agent serves a 2L request. As the number of 1L and 2L requests are below t_{1L} and t_{2L}, respectively, high skilled agents serve 1H requests as illustrated in Figure 3.9a.

There are three 1H, three 1L, two 2H, and four 2L requests in the system as depicted in Figure 3.9b. The number of 1L and 2L requests is above threshold values, t_{1L} and t_{2L}, respectively. Thus, high skilled agents serve two 1L requests preempting 1H requests until the number of 1L requests drops below t_{1L}. The remaining 1L request is served by a low skilled agent. There are two available low skilled agents that serve 2L requests. Note that even though the number of 2L request is above t_{2L}, priority class 1 requests are given higher priority in the queue relative to priority class 2 requests.

We solve the queueing system by modeling it again as a Markov chain. If we model the exact system as a Markov chain, it will be infinite in four dimensions (i.e., the number of priority, skill level pairs). Using busy period transitions we again reduce the Markov chain to a 1D infinite chain. Since priority class 1 requests are given higher priority in the queue relative to priority class 2 requests, we first solve



(a) When the numbers of 1L and 2L requests are below t_{1L} and t_{2L}, respectively, in the service center



(b) When the number of 1L requests equals t_{1L} in the service center. Even tough the number of 2L requests is above t_{2L}, 1L requests have higher priority than 2L requests and are served by the high skilled agents.

Figure 3.9: Representation of the service center for System 3 when thresholds for 1L and 2L requests, t_{1L} and t_{2L}, equal 3. The high skilled agents serve 1L requests if $N_{1L} \geq t_{1L}$ or $N_{1H} < n_H$ and $N_{1L} \geq 0$. The high skilled agents serve 2L requests if $N_{2L} \geq t_{2L}$ and $N_{1H} + N_{1L} \leq n_H$ or $N_{1H} + N_{1L} + N_{2H} < n_H$ and $N_{2L} \geq 0$

the system for priority class 1 requests.

3.3.3.1.1 Priority Class 1 Requests

There are n_H high skilled and n_L low skilled agents in the system. If the number of 1L requests is equal to the threshold value, t_{1L}, with an arrival of 1L request, a busy period starts and high skilled agents preempt 1H requests until the number of 1L requests drops below t_{1L}. If the number of 1L requests in the system, N_{1L}, is above t_{1L} but below n_H, and if $t_{1L} < n_H$, only N_{1L} high skilled agents preempt 1H requests and start serving 1L requests. The remaining high skilled agents continue serving their own requests until other 1L requests arrive.

Markov Chain of Priority Class 1 Requests

Figure 3.10 depicts the simplified Markov chain for priority class 1 customers. There are three low skilled and two high skilled agents in the system as illustrated in Figure 3.10. A state in the Markov chain represents a pair consisting of the number of 1L and 1H requests until a busy period, and j^+ or ${j_D}^+$ 1L requests and the number of 1H requests during a busy period that starts when j 1L requests are in the system. We track the exact number of 1H requests on the horizontal axis; on the vertical axis, we track the number of 1L requests up to $\max\{t_{1L}, n_H\}$. In case, $t_{1L} < n_H$, we track 1L requests up to n_H in the Markov chain as 1H requests can be served by the remaining idle high skilled agents during the type 1 busy period. States at which the number of 1L requests is illustrated as ${\max\{t_{1L}, n_H\}_D}^+$ are dummy states used in the Coxian representation representing states of a busy period. Note in our example if we are in state $(j_{1L},\ 0_{1H})$ where $j \geq 2$, the service rate is $2\mu_H + \max\{0, \min\{j-2, n_L\}\}\,\mu_L$, i.e., two high skilled agents serve 1L requests and if $j > 2$, the remaining 1L requests are served by $\min\{j-2, n_L\}$ low skilled agents.

Moments of a Type 1L Busy Period

A busy period that starts with an arrival of 1L request when the number of 1L requests equals $(\max\{t_{1L}, n_H\} - 1)$, and ends when when the number of 1L requests drops below $\max\{t_{1L}, n_H\}$ is defined as "a type 1L" busy period again. As in Sections 3.3.1 and 3.3.2, we use a PH distribution (specifically a Coxian distribution) to match

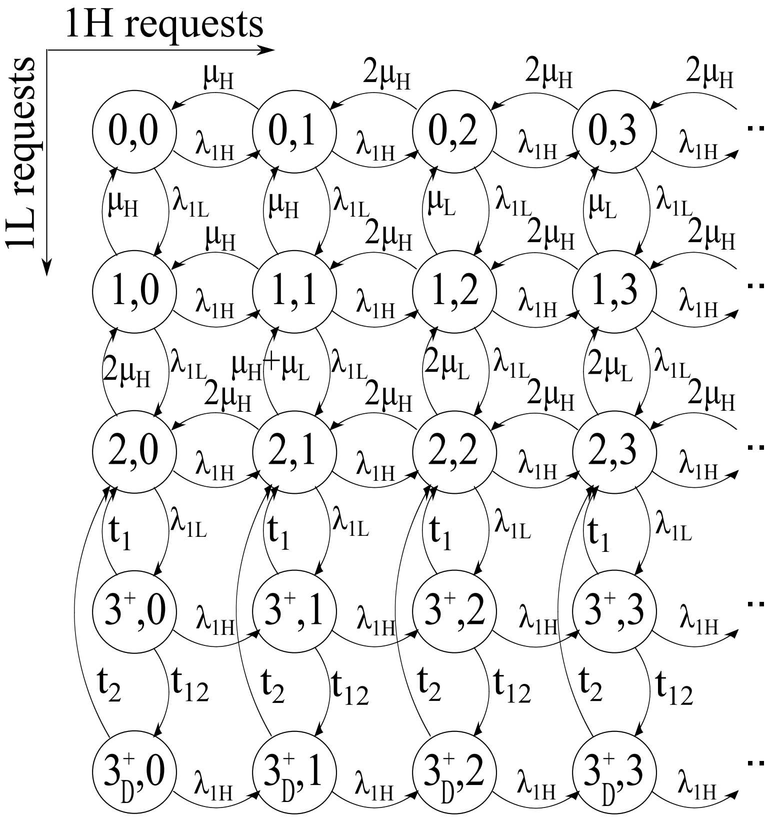


Figure 3.10: Markov chain of 1H and 1L requests. The number of high skilled agent, n_H, is 2 and the number of low skilled agent, n_L, is 3. The threshold for 1L requests is 3. The busy periods are represented by two phase PH distributions (with Coxian representations).

the first three moments of the distribution of a type 1L busy period. Once the busy period starts, we jump to the Markov chain in Figure 3.11 where a state represents the number of 1L requests in the system. Note that the service rates of 1L requests during the busy period differ conditional on the number of low skilled agents and requests in the service center. The busy period is the time to return to state 2 ($t_{2L} - 1$) from state 3 (t_{2L}).

Expected Response Times of Priority Class 1 Requests

The stationary probabilities of the now 1-dimensionally infinite chain illustrated in

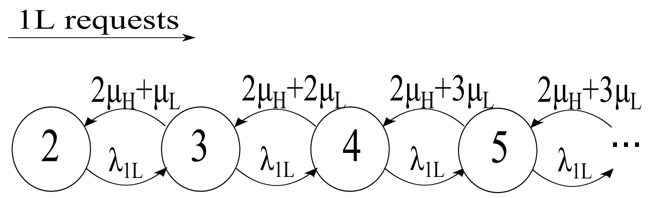


Figure 3.11: Markov chain of 1L requests used to calculate the first three moments of a type 1L busy period started with an arrival of a 1L request when there are 2 (i.e., $(t_{1L}-1)$) 1L requests in the system. The number of high skilled agent, n_H, is 2 and the number of low skilled agent, n_L, is 3.

Figure 3.10 are analyzed by the Matrix Analytic Method (MAM). The expected number of 1L requests during a busy period in Figure 3.11, $E\left[N_{1L}^{BP}\right]$, expected response time of 1H requests, $E\left[T_{1H}\right]$, and the expected response time of 1L requests, $E\left[T_{1L}\right]$, are computed as shown in Equation 3.1 in Section 3.3.1 by using the stationary probabilities calculated via MAM.

3.3.3.1.2 Priority Class 2 Requests

Priority class 1 requests are given higher priority in the queue relative to priority class 2 requests. Thus, we compute the expected response times of priority class 2 requests via using another Markov chain tracking both priority class 1 and 2 customer requests.

There are n_H high skilled and n_L low skilled agents in the system. If the number of 2L requests is brought to the threshold value, t_{2L}, with an arrival of a 2L request, a busy period starts: high skilled agents that are not serving priority class 1 requests preempt 2H requests until the number of 2L requests drops below t_{2L}. If the number of 2L requests in the system, N_{2L}, is at least t_{2L} but below $(n_H-N_{1L}-N_{1H})$, only N_{2L} high skilled agents preempt 2H requests and start serving 2L requests. The remaining high skilled agents continue serving their own requests until other 2L requests arrive.

Markov Chain of Priority Class 2 Requests

We again use busy period transitions to reduce the Markov chain tracking the system exactly to a 1D infinite chain. Figure 3.12 depicts the simplified Markov chain for priority class 2 customers. We track the exact number of 2H requests on the horizontal axis, and on the vertical axis, we track the number of 2L requests up to $\max\{t_{2L}, n_H\}$. A state in the Markov chain represents a pair consisting of the number of 2L and 2H requests until a busy period started with an arrival of 2L request, and $\max\{t_{2L}, n_H\}^+$ or $\max\{t_{2L}, n_H\}^+_D$ 2L requests and the number of 2H requests during a busy period.

There are four distinct state spaces illustrated in Figure 3.12: 0 Priority Class 1, 1 Priority Class 1, 2^+ Priority Class 1, and 2^+_D Priority Class 1. These four state spaces track the number of priority class 1 requests up to 2, the number of high skilled agents. Depending on the number of 1L or 1H requests in the system, the service rates at which priority class 2 requests can be served may change: When the system is in a state in 0 Priority Class 1 state space, both agents can serve existing 2H and 2L requests. The service rates of a high and a low skilled agent are thus μ_H and μ_L, respectively. Therefore, the Markov chain of 0 Priority Class 1 state space for 2H and 2L requests is identical the Markov chain of System 3(i) illustrated in Figure 3.10 except the arrival rates which are λ_{2H} and λ_{2L} in Figure 3.12. Similar to the system in Section 3.3.2, there are multiple types of busy periods in the system illustrated in Figure 3.12. We denote these busy periods based on the request types that started the busy period. The first type we discuss is the priority class 1 busy period. When a priority class 1 request arrives (1L or 1H) when there are already $(n_H - 1)$ priority class 1 requests, the system transfers into n^+_H Priority Class 1 state space and a busy period begins. Note that this busy period is also defined as "a priority class 1" busy period in Section 3.3.2.

The priority class 1 busy period corresponds to the time that there are not any high skilled agents that can serve priority class 2 requests, as they are all busy with

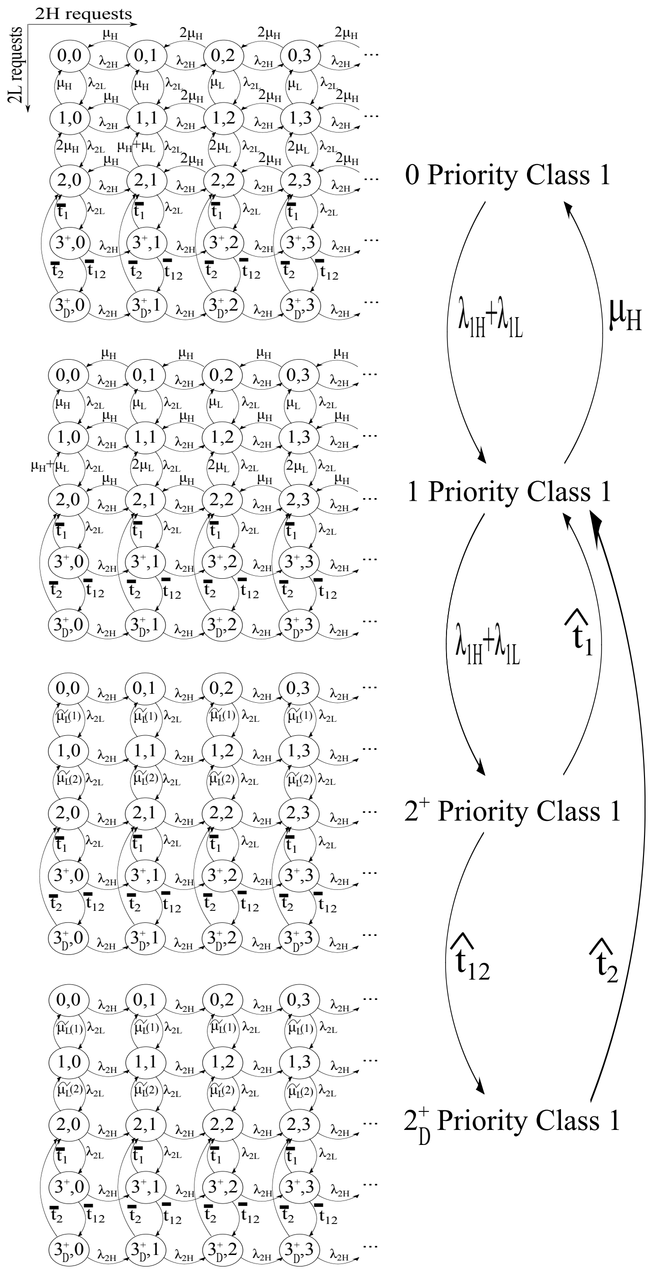


Figure 3.12: Markov chain of 1H, 1L, 2H, and 2L requests. The number of high skilled agents, n_H, is 2 and the number of low skilled agents, n_L, is 3. The threshold for 2L requests is 3. The busy periods are represented by two phase PH distributions (with Coxian representations).

priority class 1 requests, until the system returns to the $(n_H - 1)$ Priority Class 1 state space. Hence, the first three moments of the hitting times from states in which the number of priority class 1 requests equals n_H to states in which the number of priority class 1 requests equals $(n_H - 1)$ in Figure 3.10 are needed to approximate the priority class 1 busy period between the n_H^+ Priority Class 1 state space and the $(n_H - 1)$ Priority Class 1 state space. Conditional on in which state a type 1 busy period starts, the first three moments are aggregated by considering the stationary probabilities of the states where we start the busy period (i.e., states in which the number of priority class 1 requests equals n_H). For example, suppose $n_H = 2$. Referring back to Figure 3.10, possible starting states for a priority class 1 busy period are states $(2_{1L}, 0_{1H})$, $(1_{1L}, 1_{1H})$, and $(0_{1L}, 2_{1H})$. A busy period ends in state $(1_{1L}, 0_{1H})$ or state $(0_{1L}, 1_{1H})$. We first calculate the first three moments of the hitting times from each starting state to each ending state, using the method in the works of Osogami and Harchol-Balter (2003). Then, we aggregate the moments by conditioning on in which state the busy period starts. Then, the priority class 1 busy period is represented by a two phase PH distribution (via rates $\hat{t}_1$, $\hat{t}_2$, and $\hat{t}_{12}$) via matching the first three moments of the hitting times.

Low skilled agents may continue serving 2L requests depending upon the state of the system during the priority class 1 busy period. For example, suppose there are three low skilled and two high skilled agents in the system, and 2 1H and 2 1L requests. If a 1L request arrives, there are now 5 priority class 1 requests in the system, three of which are 1L requests. A low skilled agent begins serving the new 1L request. Suppose $t_{1L} > 3$. Then, all low skilled agents serve 1L requests and cannot serve 2L requests. However, during the priority class 1 busy period, the low skilled agent may serve 2L requests, if there are 4 or fewer priority class 1 requests or when there are only 1H requests in the system. We approximate the amount of time that the low skilled agent is available to serve 2L requests.

Once again,we cannot track the exact service rate since this would require transitions between busy period approximations with different rates. We avoid this through the use of a single approximate service rate, $\tilde{\mu}_L$ as in Section 3.3.2.

Approximate Service Rate $\tilde{\mu_L}$

The approximate rate, $\tilde{\mu}_L$, is again calculated adjusting the low skilled agent's service rate μ_L by two key factors: the percentage of the time that there are low skilled agents that can serve 2L requests during the priority class 1 busy period, and the priority queueing effect on the low skilled agent. The key factors are similar to the ones in Section 3.3.2, but in this new system, the number of low skilled agents is more than one. Hence, we need to extend the two key factors based on the ones introduced before. To calculate these factors, the stationary probabilities of the Markov chain illustrated in Figure 3.13 are used to approximate $\tilde{\mu}_L$.

Figure 3.13 is the same figure as Figure 3.10: the Markov chain of priority class 1 requests. In Figure 3.13, the states drawn by only dotted lines are the non-busy period states (i.e., the high skilled agent can serve priority class 2 requests). There are three and two available low skilled agents to serve 2L requests during a priority class 1 busy period at the states represented by triple and double circled nodes with solid lines, respectively. The states illustrated with double circled nodes where the second circle is represented with dashed line are the states where only one low skilled agent is available to serve 2L requests during a priority class 1 busy period. All states in which the number of 1L requests is represented by 3^+ or 3^+_D, represent cases where the number of available low skilled agents to serve 2L requests is at most two. We jump to Markov chain in Figure 3.14 to calculate the number of available low skilled agents during the busy period states represented by 3^+ or 3^+_D 1L requests in Figure 3.13.

In Figure 3.14, we track 1L requests during a busy period started with an arrival of 1L request when there are $(t_{1L} - 1)$ 1L requests in the system. Thus, the states represent the number of 1L requests during this busy period. Recall that these 1L

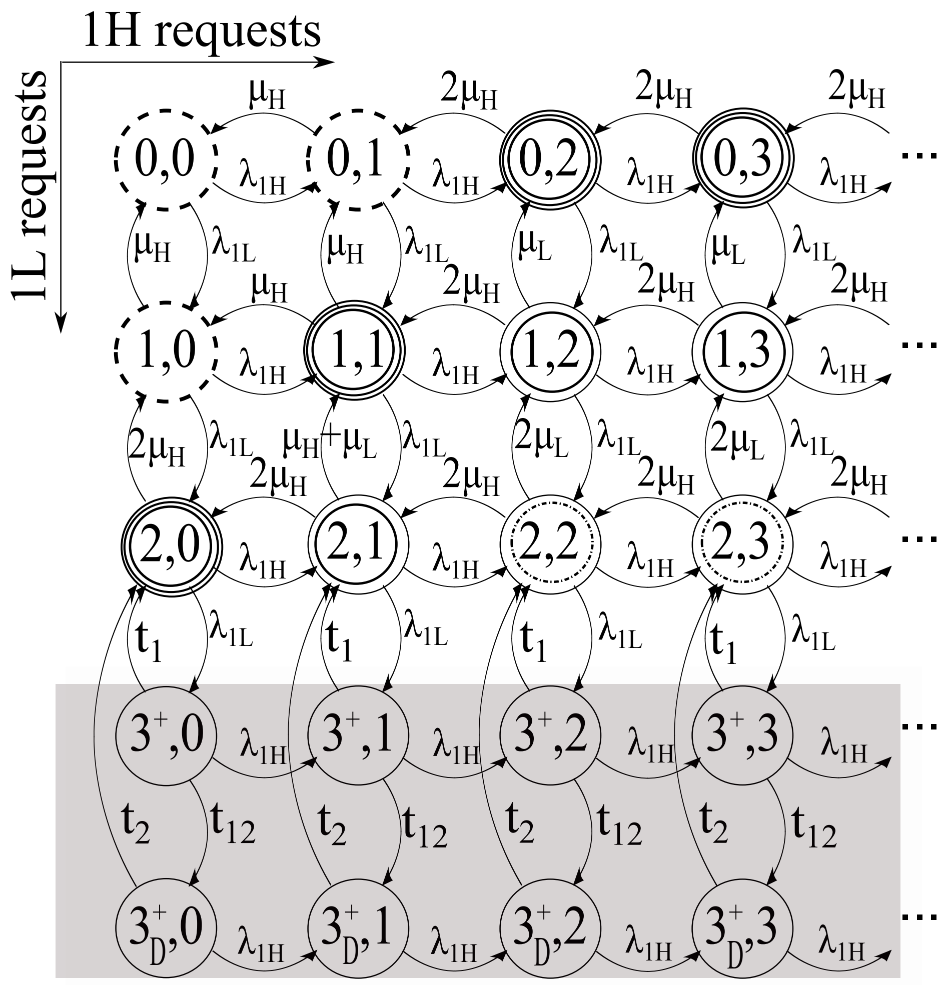


Figure 3.13: Markov chain of 1L and 1H requests used to calculate the approximate service rate for 2L requests during a priority class 1 busy period when the number of high skilled agents is two and the number of low skilled agent is three.

requests get routed to the high skilled agent. Double circled nodes with solid lines illustrate that two available low skilled agents in the system can serve 2L requests, whereas, double circled nodes with dashed line illustrate that only one low skilled agent is available.

2L requests cannot be served in any remaining states in Figure 3.14 (when the number of 1L requests is above five) since the low skilled agent is busy with 1L requests. Conditional on being in a type 1L busy period, $\beta(s)$, the average realized

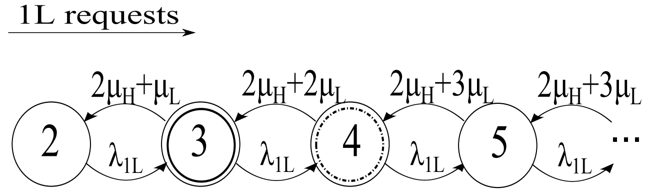


Figure 3.14: Markov chain of 1L and 1H requests during a busy period started with an arrival of 1L request when there are $(t_{1L} - 1)$ 1L requests in the system, used to calculate the approximate service rate for 2L requests during a priority class 1 busy period when the number of high skilled agents is two and the number of low skilled agent is three.

service rate considering only the percentage of the time that there are low skilled agents that can serve 2L requests during the priority class 1 busy period when there are s 2L requests in the system ($s \leq n_L$), is calculated from the Markov chains illustrated in Figures 3.13 and 3.14. We use $\pi_{(i_{1L},j_{1H})}$ for the stationary probability of state (i_{1L}, j_{1H}) in Figure 3.13, and $\pi^{BP}_{m_{1L}}$ is the stationary probability of state m 1L in Figure 3.14. Note that, unlike in Equation 3.2, in Equation 3.8, $\beta(s)$ is calculated for s 2L requests where $s \leq n_L$ to capture the actual service rate 2L requests realize, not the available service rate in the system. For example, suppose there are three available low skilled agents during a priority class 1 busy period and only one 2L request in the system. The service rate equals μ_L, not 3 μ_L. (In Section 3.3.2, we do not need to consider the realized service rate since $n_L = 1$: there is either an available agent or not.)

$$\beta(s) = \frac{\sum_{k=0}^{n_L-1} \min\{s, n_L - k\} \, (\sum_{\substack{i+j=n_H+k \\ k\leq i\leq t_{1L}-1}} \pi_{(i_{1L},j_{1H})} + \sum_{j\geq n_H+1} \pi_{(k_{1L},j_{1H})}) + (\sum_{t=1}^{n_L-1} \min\{s, n_L - t\} \, \pi^{BP}_{(n_H+t)_{1L}})}{(1 - \sum_{\substack{i,j \\ i+j<n_H}} \pi_{(i_{1L},j_{1H})})} \tag{3.8}$$

To calculate the approximate service rate, we also need to compute the priority queueing effect on the low skilled agent service rate for 2L requests, i.e., the fact that 1L requests preempt 2L requests. We approximate this priority queueing effect via

$\gamma(s)$ where $s \in \{1, ..., n_L\}$; $\gamma(s)$ captures the difference between solving the queueing system during a priority class 1 busy period via preemptive-priority queueing and queueing with an approximate, constant service rate (i.e., $\tilde{\mu}_L$).

$$\gamma(s) = 1 - \sum_{\substack{i,j \\ i+j<n_H}} \pi_{(i_{1L}, j_{1H})} + \sum_{k=0}^{n_L-1} \frac{\min\{s, n_L - k\}}{s} (\sum_{\substack{i+j=n_H+k \\ k \le i \le t_{1L}-1}} \pi_{(i_{1L}, j_{1H})} + \sum_{j \ge n_H+1} \pi_{(k_{1L}, j_{1H})}) + (\sum_{t=1}^{n_L-1} \frac{\min\{s, n_L - t\}}{s} \pi^{BP}_{(n_H+t)_{1L}}) \tag{3.9}$$

Finally, we calculate the approximate low skilled agent's service rate for 2L requests during a priority class 1 busy period when there are s 2L requests in the system, $\tilde{\mu}_L(s)$, as shown in Equation 3.4 where $s \le n_L$.

$$\tilde{\mu}_L(s) = (\gamma(s). * \beta(s)) * \mu_L \tag{3.10}$$

Moments of a Type 2L Busy Period

The second type of busy period is the busy period of 2L requests started with an arrival of a 2L request when there are $(t_{2L}-1)$ 2L requests in the system as illustrated in Figure 3.12. Let's call this busy period "a type 2L" busy period. We use the Markov chain in Figure 3.15 to calculate the first three moments of this busy period. The states in this Markov chain represent the number of 2L requests in the system. When the number of priority class 1 request is zero or one in the system, both high and low skilled agents can serve 2L requests. During a priority class 1 busy period, high skilled agents cannot serve priority class 2 requests and the service rate is $\tilde{\mu}_L$ that changes with respect to the number of 2L requests in the system. For example, the service rate from state 3 to state 2 in 2^+ Priority Class 1 state space in Figure 3.15 is $\tilde{\mu}_L(3)$ as there are three 2L requests in the system.

After approximating a two phase PH distribution by using the first three moments of the hitting times from states 3 (t_{2L}) to state 2 $((t_{2L}-1))$ calculated from the Markov

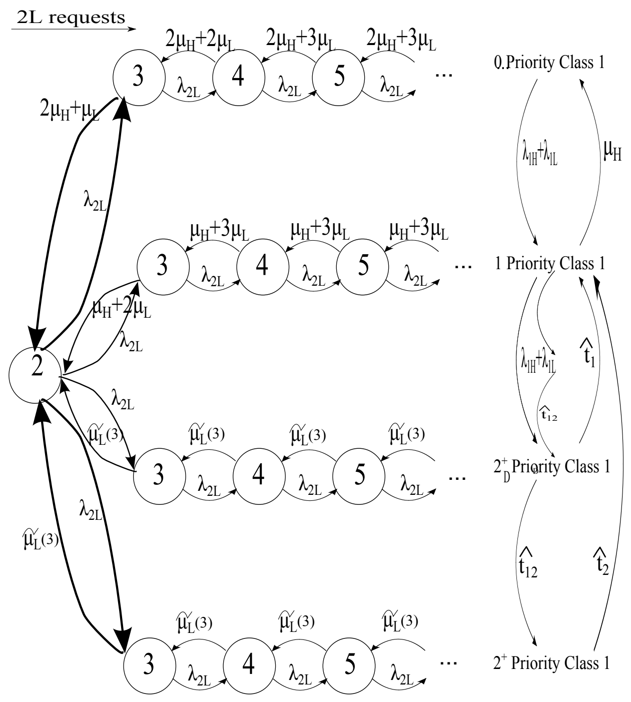


Figure 3.15: Markov chain of 2L requests used to calculate the first three moments of a type 2L busy period started with an arrival of a 2L request when there are 2 (i.e., $(t_{2L}-1)$) 2L requests in the system. The number of high skilled agents is two and and low skilled agents is three.

chain in Figure 3.15, shown with rates $\bar{t}_1$, $\bar{t}_2$, and $\bar{t}_{12}$, we reduce the size of the Markov chain illustrated in Figure 3.12 significantly.

Expected Response Times of Priority Class 2 Requests

We compute the expected response time of 2H and 2L requests by calculating the limiting probabilities of the Markov chain in Figure 3.12, similar to the calculations in Section 3.3.2. Note that since we have more agents than the system in Section 3.3.2, we need to increase the number of state spaces we track in Figure 3.12 and modify the computation of the approximate rate for each low skilled agent. We again use MAM. Let $\pi^k_{(i,j)}$ denote the stationary probability of state (i_{2L}, j_{2H}) at state space

k. The expected response time of 2H request is calculated as follows.

$$E\left[T_{2H}\right] = \frac{\sum_{j} j(\sum_{k,i} \pi^{k}_{(i_{2L},j_{2H})})}{\lambda_{2H}} \tag{3.11}$$

where $k \in \left\{0 \text{ Priority Class } 1, 1 \text{ Priority Class } 1, 2^{+} \text{ Priority Class } 1, 2^{+}_{D} \text{ Priority Class } 1\right\}$, $j \geq 0$, and $i \in \left\{0, .., \max\left\{t_{2L}, n_H\right\} - 1, \max\left\{t_{2L}, n_H\right\}^{+}, \max\left\{t_{2L}, n_H\right\}^{+}{}_{D}\right\}$ ($\max\left\{t_{2L}, n_H\right\}$ $= t_{2L}$ for the example illustrated in Figure 3.12).

We now calculate the expected response time of 2L requests. Here, we need to compute the expected number of 2L requests during a busy period, since we do not keep exact track of 2L requests in the Markov chain (Figure 3.12). We use the Markov chain in Figure 3.16 to estimate the expected number of 2L requests during a busy period. The states represent the number of 2L requests in the system. The service rate for 2L requests changes depending both on the state space and the number of 2L requests in the system.

Figure 3.16 is the same as Figure 3.15 except the transition rates from state 2 (i.e., $(t_{2L} - 1)$) to state 3 (i.e., t_{2L}). Since the probabilities of starting a busy period in different state spaces are not the same, the transition rates from state 2 to states 3 are weighted in Figure 3.8 with coefficients c_0, c_1, c_{2^+}, and $c_{2^+_D}$ that are determined by the stationary probabilities of the Markov chain illustrated in Figure 3.12. The coefficients are calculated by Equation 3.12.

$$\begin{aligned} c_0 &= \sum_{j} \pi^{0 \text{ Priority Class } 1}_{((t_{2L}-1)_{2L},j_{2H})} \\ c_1 &= \sum_{j} \pi^{1 \text{ Priority Class } 1}_{((t_{2L}-1)_{2L},j_{2H})} \\ c_{2^+} &= \sum_{j} \pi^{2^+ \text{ Priority Class } 1}_{((t_{2L}-1)_{2L},j_{2H})} \\ c_{2^+_D} &= \sum_{j} \pi^{2^+_D \text{ Priority Class } 1}_{((t_{2L}-1)_{2L},j_{2H})} \end{aligned} \tag{3.12}$$

We compute the expected number of 2L requests during a busy period as shown in Figure 3.16, $E\left[N^{BP}_{2L}\right]$, and the expected response time of 2L requests,$E\left[T_{2L}\right]$ in

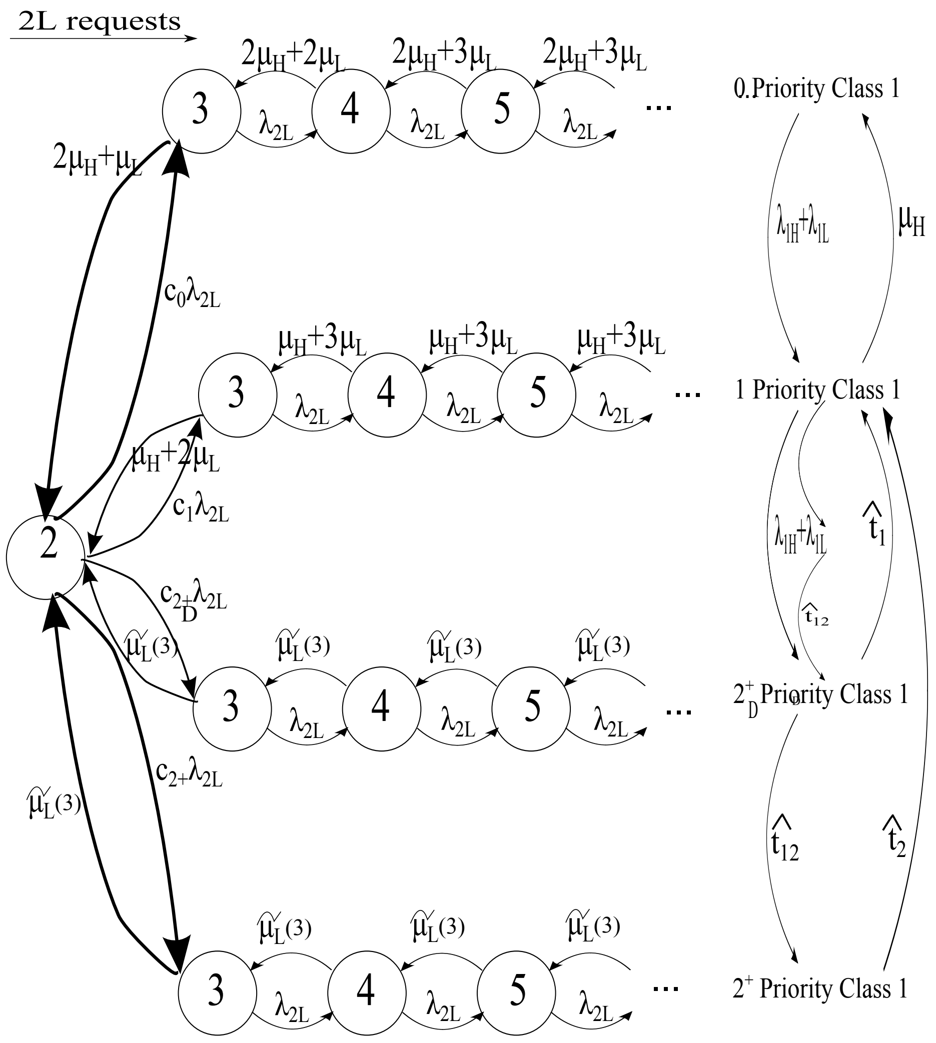


Figure 3.16: Markov chain of 2L requests used to calculate the expected number of 2L requests during a busy period started with an arrival of a 2L request when there are 2 (i.e., $(\max\{t_{2L}, n_H\} - 1)$) 2L requests in the system.

Equation 3.13 where $\breve{\pi}_i^k$ is the stationary probability of state i at state space k in Figure 3.16.

$$E\left[N_{2L}^{BP}\right] = \frac{\sum\limits_{i \geq t_{2L}} (i - (t_{2L} - 1))(\sum\limits_{k} \breve{\pi}_i^k)}{1 - \breve{\pi}_{(t_{2L}-1)}} + (t_{2L} - 1)$$
$$E\left[T_{2L}\right] = \frac{\sum\limits_{i \leq t_{2L}-1} i(\sum\limits_{k,j} \pi^k_{i_{2L}, j_{2H}}) + E\left[N_{2L}^{BP}\right] \sum\limits_{j,k} (\pi^k_{(t^+_{2L})_{2L}, j_{2H}} + \pi^k_{(t^+_{2L\,D})_{2L}, j_{2H}})}{\lambda_{2L}} \tag{3.13}$$

3.3.3.2 System 3(ii): Extension to $m > 2$ Priority Classes

System 3(ii) is based upon System 3(i) (1L, 1H, 2L, and 2H requests, and more than one high skilled and one low skilled agents). The number of high skilled agents is denoted by n_H, whereas, the number of low skilled agents is denoted n_L. The number of priority classes is denoted by m where $m > 2$. There are m threshold values in this system: t_{1L}, t_{2L}, ..., t_{mL}; t_{iL} is the threshold at which high skilled agents serve iL requests until the number of iL requests drops below t_{iL} if there are any high skilled agents that do not serve any priority class k requests where $k < i$ (as priority class k requests are given higher priority in the queue relative to priority class i requests if $k < i$). Note that if the number of iL requests is above t_{iL} but below n_H, there are idle high skilled agents that can serve iH requests (if there are any iH requests in the system). If there are still idle high skilled agents, they start serving priority class $(i + 1)$ requests.

To solve this system, the solution approach introduced in Section 3.3.3.1 can be extended. We use Markov chains to model the system for each priority class. However, the original Markov chain modeling the actual system is infinite in $2 * m$ dimensions. Thus, we use busy period approximations and threshold policies again to have a 1D infinite Markov chain. First, we solve the system for priority class 1 requests. Second, we solve the system for priority class 2 requests capturing the preemptions due to priority class 1 requests by priority class 1 busy periods as introduced in Sections 3.3.2 and 3.3.3.1. Then, the system for priority class 3 requests is solved capturing the preemptions due to priority classes 1 and 2 requests since they are given higher priority in the queue relative to priority class 3 requests. Thus, in general, the system with priority class i requests is solved by capturing preemptions of priority classes $\{1,\ ..,\ i - 1\}$. This iterative way of solving systems is repeated for all priority classes. An important property of this iterative solution method is that for priority class i customers, considering the Markov chain of priority class $(i-1)$ requests is enough since the Markov chain of priority class $(i-1)$ requests has already been affected by higher priority classes.

3.4 Computations

We consider System 3(i) in Section 3.3.3.1: $n > 2$ agents and 2 priority classes. 1L, 1H, 2L, and 2H requests arrive with rates λ_{1L}, λ_{1H}, λ_{2L}, and λ_{2H}, respectively. The low skilled agents serve only 1L and 2L requests with rate μ_L. Depending on the threshold policy, the high skilled agents can serve any requests with rate μ_H. Load of $m\,$C requests (m^{th} priority class request with C complexity) is represented by $\rho_{m\,C}$ that equals $\frac{\lambda_{m\,C}}{\mu_C}$ over all agents where $m = \{1, 2\}$, and $C = \{L, H\}$. As before t_{1L} and t_{2L} denote the thresholds of 1L and 2L requests, respectively.

3.4.1 Data Generation

Typical traffic at the motivating IT SDO's service centers is high for low complexity requests (≈ 0.70) for each low skilled agent, but moderate for high complexity requests (≈ 0.60) for each high skilled agent. In addition, the total load of priority class 1 requests is lower than the total load of priority class 2 requests. Finally the number of high skilled agents is typically less than the number of low skilled agents since the salary of a high skilled agent is more than a low skilled agent.

We generate instances capturing these features. We first analyze the expected number of busy high skilled agents numerically, to provide insights about stability of the system in Section 3.4.1.1. We then discuss performance in Section 3.4.2. More computational results for different parameters are illustrated in Appendix B.

3.4.1.1 Stability of the System

Stability conditions of a system with only priority class 1 requests under a similar threshold policy are analyzed in the works of Osogami et al. (2005a). However, deriving stability conditions of priority class 2 requests has not been studied and is nontrivial, especially for the 2H requests, since the 2H requests are preempted not only by the priority class 1 requests, but also by the 2L requests depending on the threshold policy. To our knowledge, deriving closed form stability conditions is not possible for the priority class 2 requests. The number of high skilled agents is less than the number of low skilled agents, so stability of the high skilled agents' queue is often hard to maintain. Thus, we quantify the expected number of high skilled agents busy with each request type and compare it with the number of high skilled agents to check the stability of the system.

To calculate the expected number of high skilled agents busy with 1L requests preempting requests with high complexity (of priority class 1 or 2), we use the stationary probabilities of the Markov chain illustrated in Figure 3.10. There are two events where 1L requests can preempt high complexity requests: (i) the number of 1L requests is above $(t_{1L} - 1)$, and (ii) the numbers of 1L and priority class 1 requests are below t_{1L} and n_H, respectively, and the number of 1H requests is less than n_H. In event (ii) the number of priority class 1 requests is below n_H, and if the number of 1L requests is at least 1, high skilled agents first serve 1L requests before serving any priority class 2 requests, so 2H requests may be preempted by the 1L requests. Thus, we sum the number of high skilled agents busy with 1L requests after multiplying with the respective state's stationary probability in which event (i) or (ii) occurs. Similarly, the expected number of high skilled agents busy with 2L requests preempting 2H requests is calculated by using the stationary probabilities of the Markov chain illustrated in Figure 3.12. Note that 1H and 2H requests can only be served by the high skilled agents, so their loads are known without utilizing the Markov chains in Figures 3.10 and 3.12.

We generated several instances and investigate their stability. The following parameter set is the base set for the remaining part of the computational results in this section: $\rho_{1L} = 0.7$, $\rho_{1H} = 0.5$, $\rho_{2L} = 2$, and $\rho_{2H} = 0.6$. In the base parameter set, the numbers of high and low skilled agents are 2 and 4, respectively. Different parameter sets are analyzed in Appendix B.

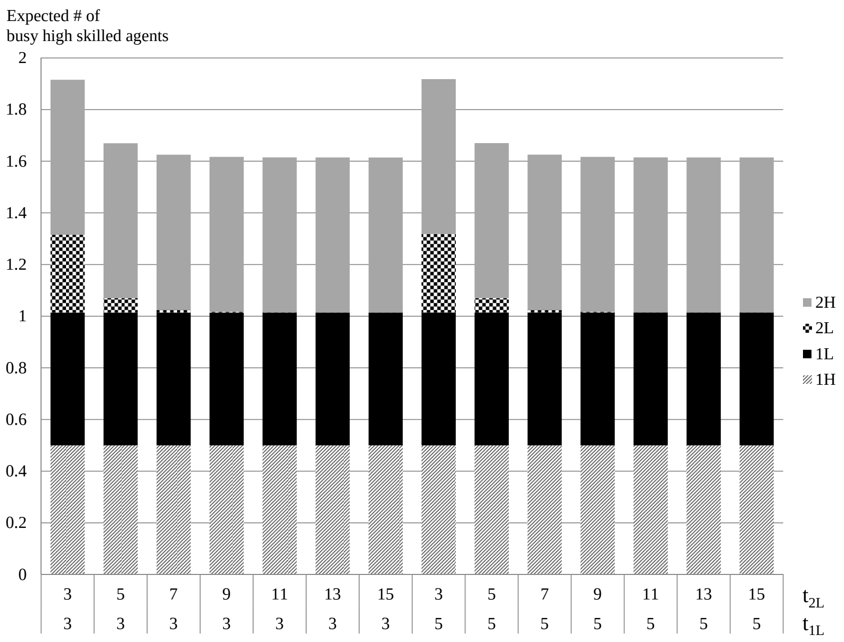


Figure 3.17: Expected number of busy high skilled agents as a function of t_{1L} and t_{2L} when $\rho_{1L} = 0.7$, $\rho_{1H} = 0.5$, $\rho_{2L} = 2$, $\rho_{2H} = 0.6$, $n_L = 4$, and $n_H = 2$.

Figure 3.17 depicts the expected number of high skilled agents busy with 1L, 1H, 2L, and 2H requests as a function of t_{1L} and t_{2L}. When the thresholds are low, the expected number of busy high skilled agents increases as low complexity requests are also served by the high skilled agents preempting high complexity requests. For example, the expected number of high skilled agents when $t_{2L} = 3$ is higher than the case when $t_{2L} = 5$ since 2L requests are more often served by the high skilled agents since they preempt 2H requests more frequently.

3.4.2 Accuracy of MAM

To validate our analysis, we simulate the original service center's operations (i.e., without any approximation). For each (t_{1L}, t_{2L}) threshold pair, the number of replications is 20. Each replication converges at least two-orders of magnitude slower than MAM. We compare the expected response times of requests calculated by simulation and MAM, and define the gap as follows.

$$\frac{{}_{E[T]}\text{MAM}_{-E[T]}\text{Simulation}}{{}_{E[T]}\text{Simulation}}; \tag{3.14}$$

for example $E\left[T_{2H}\right]^k$ is the expected response time of 2H requests calculated by method k.

The gaps of priority class 1 and 2L requests' expected response times are always very small (less than 1%). Thus, Figure 3.18 illustrates the gap of 2H requests' expected response time between simulation and MAM as a function of t_{1L} and t_{2L}.

The gap decreases as t_{2L} increases, and is very small (less than 2%) when $t_{2L} \geq 5$ as depicted in Figure 3.18. The gap of 2H request is approximately 10% when $t_{2L} = 3$. The reason behind this apparently poor performance is seen numerically from Figure 3.17: The expected number of high skilled agents busy with all requests is approximately 1.92, and the simulation's replications do not converge in two hours when $t_{2L} = 3$. Hence, the queue for 2H requests becomes highly loaded. However, when $t_{2L} = 5$, the expected number of busy high skilled agent drops below 1.68 in Figure 3.17, and the gap is small as shown in Figure 3.18. Thus, our approximation method may not work well for the highly loaded queues. Incorporating another coefficient in the approximate service rate $\tilde{\mu_L}$ is a future direction which might help to improve the gap under highly loaded queues. Note though that the simulated

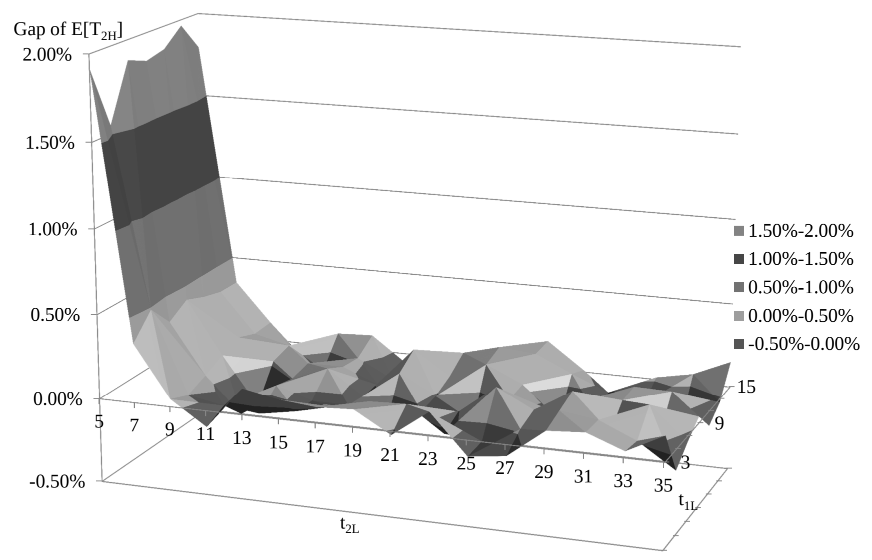


Figure 3.18: Gap of 2H requests' expected response time between simulation and MAM as a function of t_{1L} and t_{2L} when $\rho_{1L} = 0.7$, $\rho_{1H} = 0.5$, $\rho_{2L} = 2$, $\rho_{2H} = 0.6$, $n_H = 2$, and $n_L = 4$.

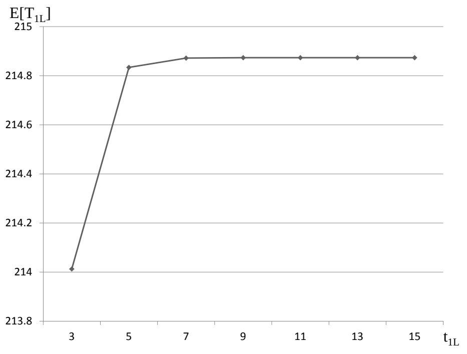
expected response time of 2H requests is above 1840 when $t_{2L} = 3$, whereas, it is below 790 as t_{2L} increases. It is very unlikely that a service center manager would prefer 2H requests to suffer by setting $t_{2L} = 3$ unless keeping 2H requests longer in the system is free. Thus, this inaccuracy region is likely to be unimportant in practical application.

3.4.3 Effect of Threshold

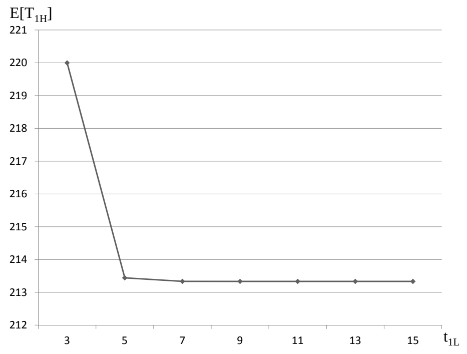
We describe two main effects of thresholds. First, we analyze the effect of the threshold policy on the expected response times. Second, we evaluate the threshold-based policy by comparing it with two naïve policies.

3.4.3.1 Expected Response Times

The expected response time of 1H requests decreases and 1L requests increases as t_{1L} increases, until the expected response times converge to the case of separate queues at some point after which increasing t_{1L} further does not have any effect as illustrated in Figure 3.19. When we increase t_{1L}, 1L requests steal fewer cycles from the high skilled agent which decreases the expected response time of 1H requests. When t_{1L} is 0, 1H requests are never served so they experience infinite expected response time. Note that since priority class 1 requests have higher priority than priority class 2 requests, the expected response times of priority class 1 requests do not change if t_{2L} changes. Note also the effect of t_{1L} on 1H requests' response time is more dramatic than its effect on 1L requests' response time.



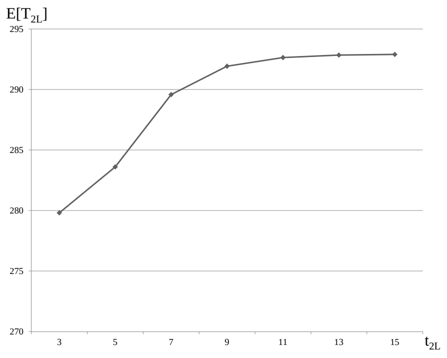
(a) Expected response time of 1L requests as a function of t_{1L}



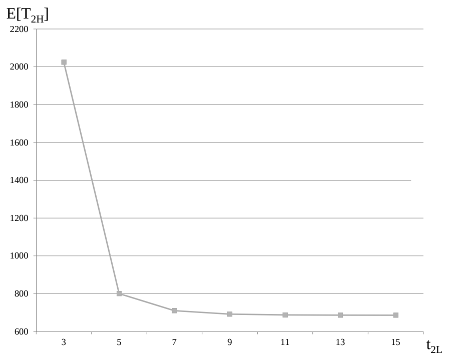
(b) Expected response time of 1H requests as a function of t_{1L}

Figure 3.19: Expected response times of priority class 1 requests as a function of t_{1L} when $\rho_{1L} = 0.7$, $\rho_{1H} = 0.5$, $\rho_{2L} = 2$, $\rho_{2H} = 0.6$, $n_H = 2$, and $n_L = 4$.

Similarly, expected response times of 2L and 2H requests are illustrated in Figures 3.20a and 3.20b, respectively, as a function of t_{2L} when $t_{1L} = 3$. The impact of t_{2L} on 2H requests is higher than 2L requests because when t_{2L} is small, the high skilled agents' queue might become highly loaded for the 2H requests as illustrated in Figure 3.17.



(a) Expected response time of 2L requests as a function of t_{2L}

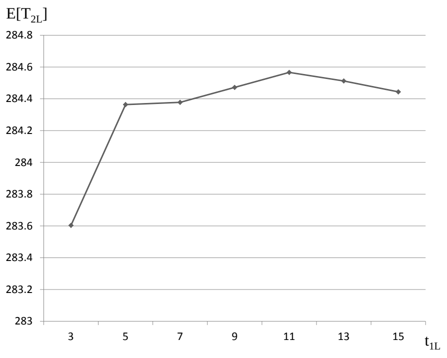


(b) Expected response time of 2H requests as a function of t_{2L}

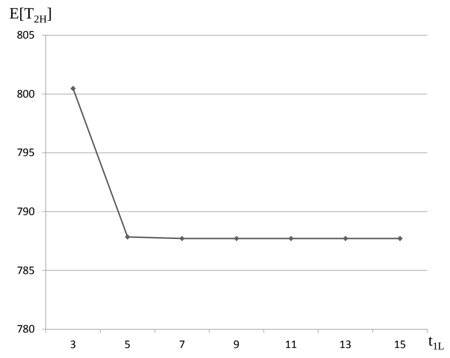
Figure 3.20: Expected response times of priority class 2 requests as a function of t_{2L} when $t_{1L} = 3$, $\rho_{1L} = 0.7$, $\rho_{1H} = 0.5$, $\rho_{2L} = 2$, $\rho_{2H} = 0.6$, $n_H = 2$, and $n_L = 4$.

Finally, in Figure 3.21, the expected response time of 2L and 2H requests are represented as a function of t_{1L} when $t_{2L} = 5$. As t_{1L} increases, the expected response time of 2H requests decreases as the number of 1L requests served by the high skilled agents decreases, which pushes 1H and 2H requests back less. Eventually, the expected response time of 2H requests converges to a value, because priority class 1 requests converge to the case of separate queues at some point after which increasing t_{1L} further does not have any effect. Thus, 2H requests are preempted by the same load of 1L requests after this convergence. In Figure 3.21a, the expected response time of 2L requests first increases as t_{1L} increases, because 1L requests are served less frequently by the high skilled agent than the other cases with smaller t_{1L} values. But when t_{1L} is above 11, the expected response time of 2L starts to decrease.

There are two possible reasons to explain this behavior: variability in the service rates and the gap of the 2L requests' expected response time. Since the high and low skilled agents serve with different service rates, increasing t_{1L} might cause variability in the service rate of 2L requests: As t_{1L} increases, 1L requests stay longer in the system and are served more frequently by the low skilled agents. Thus, when it is 2L requests' turn to be served, they might be served by the high skilled agent more



(a) Expected response time of 2L requests as a function of t_{1L}



(b) Expected response time of 2H requests as a function of t_{1L}

Figure 3.21: Expected response times of priority class 2 requests as a function of t_{1L} when $t_{2L} = 5$, $\rho_{1L} = 0.7$, $\rho_{1H} = 0.5$, $\rho_{2L} = 2$, $\rho_{2H} = 0.6$, $n_H = 2$, and $n_L = 4$.

often (the number of 2L requests in the low skilled agents' queue increases more if 2L requests are preempted by the 1L requests more often) and receive a faster service rate resulting in the decrease illustrated in Figure 3.21a. The other reason might be due to the gap of 2L requests' expected response times as the decrease in the expected response time depicted in Figure 3.21a is less than 0.1%.

3.4.3.2 Total Costs

There are three naïve policies potentially representing service centers' operations with increasing cooperation: (i) Separate queue: high skilled agents serve only high complexity (1H and 2H) requests, and low skilled agents only serve low complexity (1L and 2L) requests, (ii) Cycle stealing: high skilled agents can only serve low complexity requests if they are idle, and (iii) Prioritized cycle stealing: high skilled agents serve high complexity requests and low skilled agents serve low complexity requests, priority class 1 customers have higher priority than priority class 2 customer requests, and if high skilled agents are idle, they can serve low complexity requests. The cycle stealing policy does not support the IT SDO's service centers as priority class 1 requests are more important then priority class 2 requests.

We model each policy by MAM. The objective is to minimize the total cost which is a combination of cost of agents and convex cost of expected response times. For example, the contribution of state i 1L requests to the total cost is the product of $C_{1L}\ i^2$ and the total stationary probability of states in which there are i 1L requests. (Convex costs capture customers becoming increasingly impatient as delays increase.) To calculate the total salary of the agents, the numbers of low and high skilled agents are multiplied with C_L and C_H, respectively.

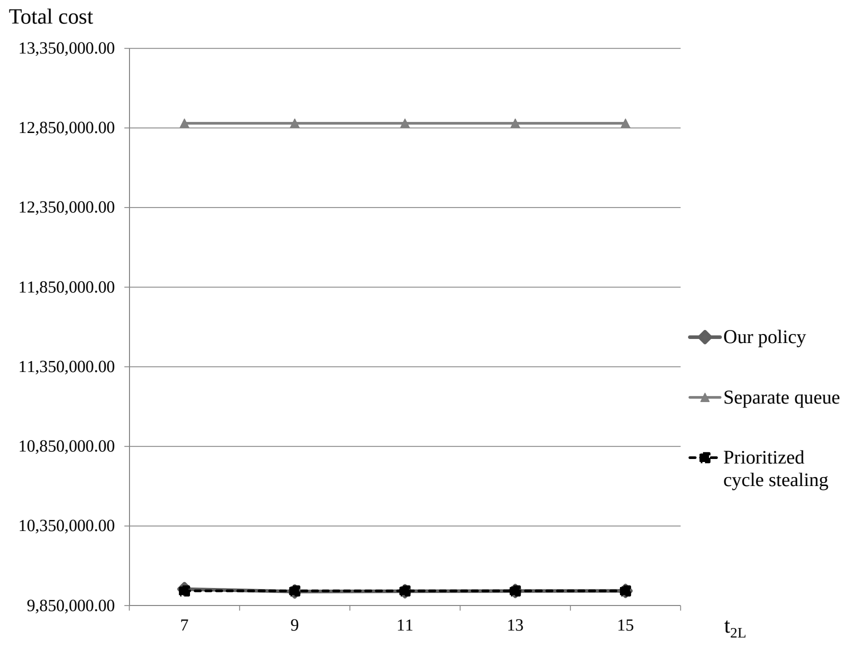


Figure 3.22: Comparison of our policy and naïve policies when $\rho_{1L} = 0.7$, $\rho_{1H} = 0.5$, $\rho_{2L} = 2$, $\rho_{2H} = 0.6$, $C_{1L} = C_{1H} = 250$k, $C_{2L} = 200$k, $C_{2H} = 100$k, $n_H = 2$, and $n_L = 4$.

We compare the total cost calculated by our threshold policy with the separate queue and prioritized cycle stealing policies in Figure 3.22. This example allows us to address the importance of cooperation as the separate queue policy performs the worst. Thus, hurting the high complexity requests by preempting them and helping the low complexity requests by routing them to the high skilled agents decrease the total cost. To analyze our approximation and the prioritized cycle stealing policy in detail, Figure 3.23 is illustrated.

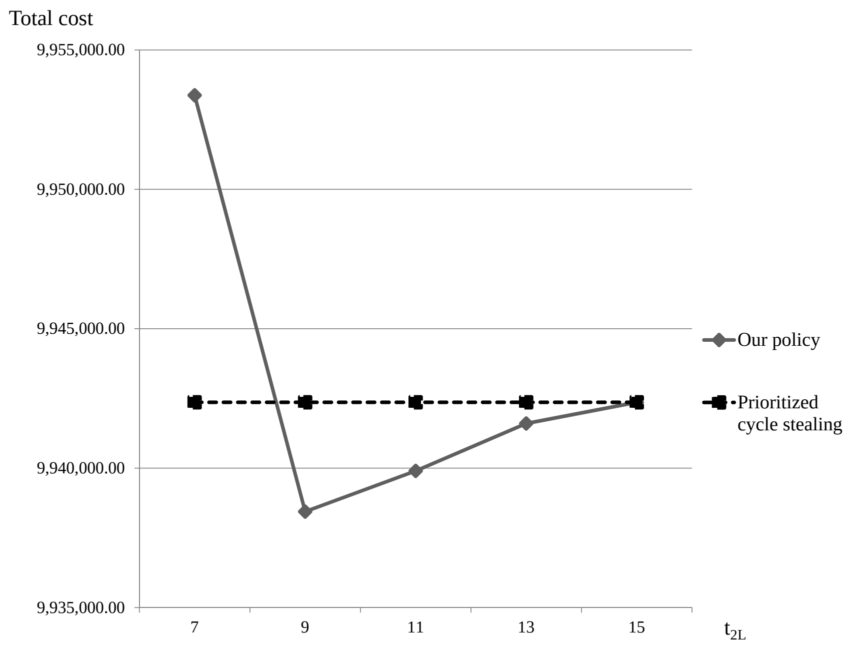


Figure 3.23: Comparison of our policy and the prioritized cycle stealing policy when $\rho_{1L} = 0.7$, $\rho_{1H} = 0.5$, $\rho_{2L} = 2$, $\rho_{2H} = 0.6$, $C_{1L} = C_{1H} = 250$k, $C_{2L} = 200$k, $C_{2H} = 100$k, $n_H = 2$, and $n_L = 4$.

The total cost of the prioritized cycle stealing policy is the same as t_{2L} changes as it is independent on the threshold, whereas the total cost of our policy varies because setting a low t_{2L} helps 2L requests but hurts 2H requests. Depending on the unit cost values of 2L and 2H requests, the total cost may increase. Even tough $C_{2H} < C_{2L}$, the total cost of our policy is higher than the prioritized cycle stealing policy when t_{2L} is below 9, because this hurts 2H performance significantly, but when $t_{2L} = 9$, the total cost of our policy is lower than the prioritized cycle stealing policy. As t_{2L} increases, the total costs of the prioritized cycle stealing policy and our policy converge, because the expected response time converges at some point after which increasing t_{2L} further does not make any difference in the expected response times of priority class 2 requests as illustrated in Figures 3.20a and 3.20b. Thus, the importance of our threshold-based policy is validated by achieving a lower optimal cost than the naïve policies.

3.4.4 What Mix of Agents to Hire

One of our goals is to determine an appropriate static agent base. The objective is again to minimize the total cost. There is a trade-off between the number of agents and the target times of requests. If the cost of an agent is high, even if hiring that agent improves expected response times of the requests, it may increase the total cost.

At the motivating IT SDO's service centers, the cost of expected response times of priority class 1 requests is higher than the priority class 2 requests. In addition, the cost of a high skilled agent is higher than a low skilled agent's cost. We generate representative unit costs for the base parameter set and analyze whether hiring another high skilled agent is good as a function of t_{2L}, illustrated in Figure 3.24.

In Figure 3.24, we compare the service centers with two different agent base: (i) 4 low and 2 high skilled agents, and (ii) 4 low and 3 high skilled agents as a function of t_{2L} when $t_{1L} = 3$. The dashed circles represent optimal t_{2L} levels for agent bases (i) and (ii). For example, the optimal threshold value of 2L requests is 7 (or 11) when the numbers of high and low skilled agents are 3 (or 2) and 4, respectively. Since the objective is to minimize the total cost, the service center manager prefers to hire a high skilled agent if t_{2L} is below 7; otherwise an agent base with 4 low and 2 high skilled agents determines the minimum cost. Thus, given the costs, one can set the optimal threshold values. In addition, if the threshold values are fixed, it may be possible to negotiate costs with customers.

Figure 3.25 represents a service center in which the cost of priority class 2 requests are not the same as in Figure 3.24: C_{2L} decreases, and C_{2H} increases. The optimal

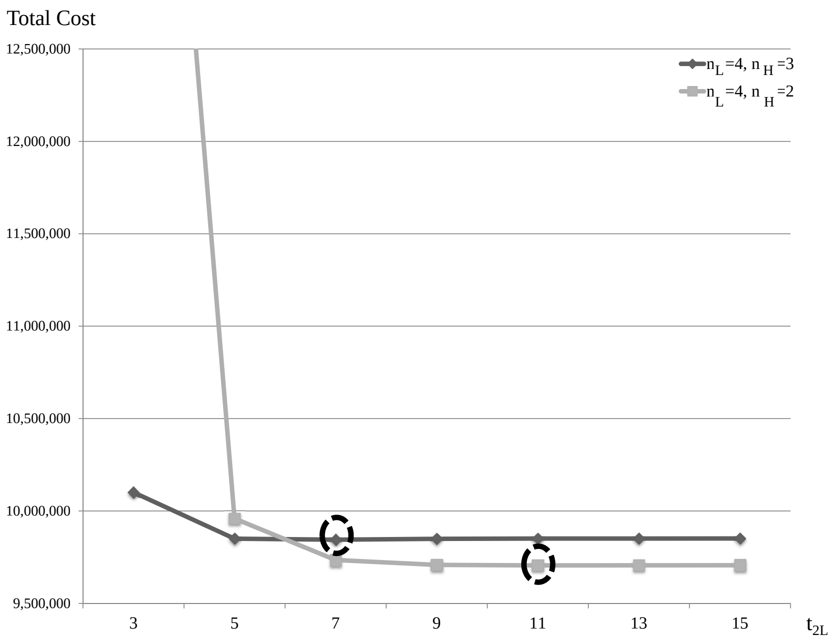


Figure 3.24: Analysis of hiring a high skilled agent as a function of t_{2L} when $t_{1L} = 3$, $\rho_{1L} = 0.7$, $\rho_{1H} = 0.5$, $\rho_{2L} = 2$, $\rho_{2H} = 0.6$, $C_{1L} = C_{1H} = 300$k, $C_{2L} = 150$k, $C_{2H} = 100$k, $C_L = 1,000$k, and $C_H = 1,500$k.

total cost is lower in Figure 3.25 than in Figure 3.24.

3.5 Conclusion and Future Work

We model a service center at a large, global, IT services delivery organization with agents who are either high or low skilled. These agents serve customer requests that are also heterogeneous - with respect to both the complexity and their priority: (i) Higher priority customer requests preempt lower priority customer requests in the queue; and (ii) Low skilled agents can only serve low complexity requests, while high skilled agents can serve all types of requests. Our goals are to create an algorithm that can determine an appropriate static agent base and to determine an effective request-assignment policy.

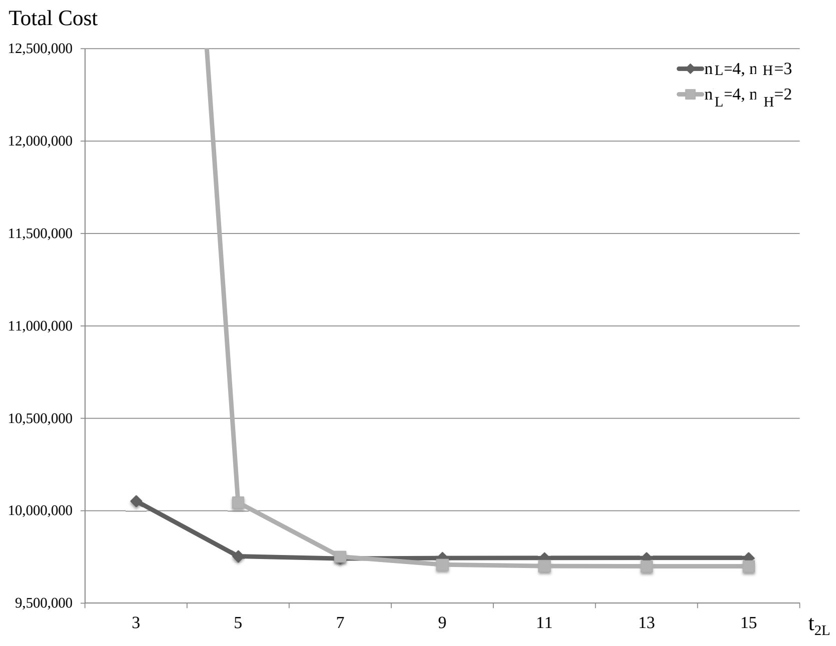


Figure 3.25: Analysis of hiring a high skilled agent as a function of t_{2L} when $t_{1L} = 3$, $\rho_{1L} = 0.7$, $\rho_{1H} = 0.5$, $\rho_{2L} = 2$, $\rho_{2H} = 0.6$, $C_{1L} = C_{1H} = 300$k, $C_{2L} = 120$k, $C_{2H} = 120$k, $C_L = 1,000$k, and $C_H = 1,500$k.

A multi-server queueing system under preemptive-resume priority classes is used to capture this service center's operations; however, since exact solution of such a system is numerically intractable, we apply approximation and bounding techniques to evaluate different control policies. We model the system as a Markov chain by approximating the interval of time during which the agent is busy without interruption. Hence, we turn the intractable Markov chain into a tractable one and apply standard Matrix Analytic Methods. Our work demonstrates that a simple but effective request-assignment policy can meet the service level goals for different prioritized customers utilizing an appropriately determined static agent base.

We show that four different types of customer requests can be successfully analyzed by our method: our method is accurate and fast when compared to simulation. We also provide initial insights to the essential managerial questions about the capacity provisioning problem at the service center, such as, how to set optimal threshold values, how to negotiate costs with customers, and what mix of agents to hire. We also evaluate the threshold-based policy and two naïve policies. The minimum optimal cost is achieved by our threshold-based policy in the numerical experiment, further revealing that the threshold values may improve the total cost without any major changes in the service center, such as, agent base or loads.

One possible future direction is updating the approximate service rate $\tilde{\mu_L}$ for highly loaded high skilled agents' queue as it may improve the performance of MAM for 2H requests. Incorporating time varying arrival and service rates is another potential future work. One solution strategy to handle time varying arrival rates and service rates is to divide the system in phases: underloaded, overloaded, and critically loaded (Mandelbaum and Massey, 1995). Another strategy is to increase the safety staffing above the required level for the expected arrival rate (Gans et al., 2003).

4 ROBUST PROJECT SCHEDULING AT A LARGE IT SERVICES DELIVERY ORGANIZATION

We consider a project scheduling problem at a large IT services delivery organization (SDO) with cross-trained agents, heterogeneous projects, schedule disruptions, and service quality guarantees. Durations of projects' tasks, disruptions during task service, and project arrival times are uncertain. System parameters are estimated using data from the IT SDO. Our goal is to identify an effective schedule of tasks and assignment of tasks to agents, capturing uncertainties using uncertainty sets in a robust scheduling model.

We apply logic-based Benders decomposition to solve the scheduling problem with uncertain parameters. First, a three-stage decomposition is proposed. The master problem assigns projects' tasks to agents. Then, Subproblem 1 finds an optimal sequence without considering uncertainty. In Subproblem 2, uncertainty is captured by computing the worst-case result given an uncertainty set. If uncertainty sets are modeled polyhedrally, we prove that the three-stage decomposition is simplified to a two-stage decomposition combining Subproblems 1 and 2. The master problem assigns tasks to agents, and the subproblem schedules these assigned tasks considering the uncertainty set. The objective is to minimize the weighted tardiness.

4.1 Introduction

Project assignment and scheduling is challenging especially in a services delivery organization where the numbers of agents and projects are large, even in a deterministic

This chapter is joint work with Aliza R. Heching, John N. Hooker, and Alan Scheller-Wolf.

setting. With the prevalence of uncertainty in real-life, we consider this problem in a stochastic setting: Ignoring uncertainty may lead to underestimation of projects' finishing times and cost, leading to dissatisfied customers.

We analyze a project scheduling problem with disruptions, cross-trained agents, heterogeneous projects and service quality guarantees, where the durations of a project's tasks, occurrence of disruptions, and the arrivals of the projects are uncertain. This problem is motivated by a real problem at a large, global, IT services delivery organization (SDO). We investigate whether robust optimization, a recent approach to optimization under uncertainty, can be used when the uncertainties in the IT SDO are represented by uncertainty sets. Our goal is to find an effective and robust assignment and schedule of the tasks for each agent.

The remainder of the paper is organized as follows. In Section 4.2 we survey recent work on project scheduling with uncertainty and alternative solution methods. The deterministic scheduling and general robust models are introduced in Section 4.3. In Section 4.4, we introduce logic-based Benders decomposition and apply it to the robust model from Section 4.3. We propose a three-stage decomposition for our robust project scheduling problem, and simplify it to a two-stage decomposition in Section 4.5. Concluding comments, extensions, and future work are provided in Section 4.6.

4.2 Literature Survey

Projects are usually managed under high levels of uncertainty stemming from sources such as resource availability, team competence, commitment of upper management, and project data (processing times, release dates and due dates). Uncertainty is very common, thus few projects are completed without time or cost overruns (Pich et al., 2002). Especially in service centers at large IT services delivery organizations, assigning and scheduling projects for each agent become quite difficult, due to both uncertainties as well as the large numbers of agents and projects.

The project scheduling literature proposes different strategies to cope with uncertainty. A common method is *proactive/reactive scheduling*: First, a robust baseline schedule is generated given the distribution of the uncertainty; then, as uncertainties occur, the baseline schedule is updated. Note that it is also possible to fully reschedule projects with the *proactive/reactive scheduling* method (Demeulemeester and Herroelen, 2009). *Event-based dynamic programming* is another method to solve the project scheduling problem as studied in (Koole, 2000). This method deals with event operators (building blocks of the value function) where an operator is associated with each basic event in the system, such as an arrival at a queue or a service completion. The uncertainty is captured by the transition probabilities between events, which are assumed to be known.

An alternative method is scenario-based modeling, which proceeds as follows. Given the distribution of the uncertainty in the problem, one can generate all scenarios (or a modest sample of scenarios). Using the generated scenarios, the problem can be solved as a deterministic problem. A major drawback is that a large number of scenarios (even when limiting to a sample of scenarios) makes the problem difficult to solve (Dembo, 1991). Another method is to define the problem as a multi-stage stochastic program with recourse. For example, Yen and Birge (2006) modeled an airline crew scheduling problem with uncertain disruptions as a two-stage stochastic integer program with recourse. Hence, given the *accurate probabilistic description of uncertainty*, a stochastic programming method can be applied. For a detailed review of stochastic programming solution techniques, we refer to the works of Birge and Louveaux (2011). However, methods assuming perfect information about the distribution of the uncertainty may suffer, as precise knowledge of the underlying probabilities is usually unavailable in practice. In addition, computational complexity might increase due to sample size. Hence, *robust optimization* becomes advantageous due to its distribution-free property and computational tractability.

Robust optimization is a leading methodology for handling optimization problems with uncertainty (Bertsimas et al., 2004). We focus on robust optimization where uncertainty is modeled as uncertain parameters belonging to a set defined as the *uncertainty set*. Our main goal is to solve the problem with respect to an uncertainty set, thereby identifying a robust solution.

Robust optimization attempts to ensure that the solution remains feasible and near optimal even if the parameters change (Bertsimas and Thiele, 2006). The first approach to model parameter uncertainty was proposed by Soyster (1973). He considered a linear optimization problem where columns of the coefficient matrix are uncertain and belong to a convex uncertainty set. He constructed a solution that is feasible for all data in the uncertainty set. Originally a worst-case scenario was proposed; but this is typically too conservative for practical implementations. To overcome this overconservatism, Ghaoui and Lebret (1997); Ben-Tal and Nemirovski (1998); Ghaoui et al. (1998); Ben-Tal and Nemirovski (1999), and Ben-Tal and Nemirovski (2000) studied ellipsoidal uncertainty sets by removing unlikely outcomes of uncertain parameters. The robust counterpart of some important generic convex optimization problems under ellipsoidal uncertainty sets are exactly or approximately tractable problems which can be solved by interior point methods. However, the robust counterpart of a linear programming problem becomes a second-order conic problem. Thus, the complexity of the problem increases with the ellipsoidal uncertainty set.

To avoid this increasing complexity, Bertsimas and Sim (2004), and Bertsimas et al. (2004) proposed a robust optimization approach with a polyhedral uncertainty set. This robust counterpart of linear programming problem remains a linear programming problem; thus, the method is more tractable, especially in large-scale settings. The level of conservatism can also be controlled via restricting the number of parameters that take their worst-case values (Bertsimas and Brown, 2009; Bertsimas and Sim, 2003, 2004; Conde, 2009; Natarajan et al., 2009; Yaman et al., 2007). Robust optimization with uncertainty sets is an evolving and relatively new method. We have provided only an overview of robust optimization literature; for a detailed review, we refer to the works of Bertsimas and Thiele (2006); Bertsimas et al. (2011) and Ben-Tal et al. (2009).

As mentioned in Section 4.1, we are inspired by a project scheduling problem observed at a large IT services delivery organization with cross-trained agents, heterogeneous projects, schedule disruptions, and service quality guarantees. Our goal is to find an effective assignment and schedule of tasks for each agent, capturing the system uncertainties. We design a novel robust scheduling model to solve this project scheduling problem.

4.3 The Models

We develop project scheduling models that describe the operations at a large IT services delivery organization (SDO). We first discuss the business setting and then describe the models. Our objective is to assign incoming projects' tasks to agents at the IT SDO so as to provide timely delivery to customers. The SDO employs several hundred agents, differentiated according to their skills, who process thousands of incoming customer projects each year. A project may consist of several tasks that must be completed in a prescribed order. An agent must have at least the necessary required skill level to complete the task: For example, suppose Java programming skill is required for a project's task. Then, this task must be assigned to an agent who has at least Java programming skills, but it may be assigned to an agent who has skills in addition to Java programming skills.

Based on the data sets we have analyzed, late deliveries on projects result primarily from interruptions to service that require a task to be set aside temporarily. Examples of such interruptions include cases where the agent may need more information from a customer before continuing with the task, authorization from a superior, and so forth. The resulting delay may be several days. An agent may be able to work on other tasks during this period, but only if the agent has been assigned other tasks. The schedule must therefore anticipate this possibility when assigning tasks.

Late deliveries on projects are also caused by variability in task processing times. Data analysis indicates that task processing times can differ by a factor of two or more for tasks of a given type. This appears to have a less significant impact than interruptions, since total processing times tend to be measured in hours rather than days. However, long processing times can upset the schedule if several occur in a row. There are project classes defined by the customer, required skill, type of project, country of origin and so on. Longer processing times are somewhat predictable based on the class, particularly the country of origin.

Interruptions usually occur as processing gets underway, and the agent finds that key information required to complete the task is missing. This, combined with the fact that processing times are short, suggest that there is no need to adopt pre-emptive scheduling model. (If the processing times were long, using a pre-emptive scheduling model might be necessary.) As the processing times are short in the IT service center, we treat the interrupted task as having a delayed start time. Under this modeling approach, the primary uncertainty is in the release time.

The schedule can be re-computed on a rolling basis as projects arrive. Our initial assumption is that tasks whose processing has already started are not rescheduled. However, this is not a limiting assumption since task processing times are typically less than one day.

We adopt a phased approach to addressing this problem. We first model the deterministic version of the project scheduling problem in Section 4.3.1. We then

Table 4.1: Nomenclature for the deterministic project scheduling problem.

Indices	
$j \in J$	task
i	agent
Sets	
J	set of tasks ($\{1, 2, ..., n\}$)
S_i	skill set of agent i
S'_j	required skill set to serve task j
Parameters	
r_j	release time of task j
d_j	deadline (or due date) of task j
p_j	processing time of task j
c_j	unit tardiness cost of task j
Variables	
y_j	agent assigned to task j
s_j	start time of task j

introduce a general robust model in Section 4.3.2. In Section 4.4, we introduce logic-based Benders decomposition. First, a three-stage decomposition is proposed to solve the robust scheduling problem capturing uncertain release times and processing times in Section 4.5. We describe this decomposition and its properties in Sections 4.5.1 - 4.5.3. Then, in Section 4.5.4, a two-stage decomposition is proposed by using the convexity analysis of Theorem 4.5.1 and proof of equivalent formulations of Theorem 4.5.3.

4.3.1 Deterministic Project Scheduling Model

We first state the deterministic version of the problem. The notation is summarized in Table 4.1.

The tardiness cost may reflect priorities (e.g., important clients). If only the completion time of a project matters, the cost is assigned to the final task of the project, and the other tasks have zero cost. The precedence graph (J, E) is a directed graph in which $(j, j') \in E$ when task j must finish before task j' starts.

Let $\alpha^+ = \max\{0, \alpha\}$. Assuming no preemption, the problem can be stated as in (4.1).

$$\begin{array}{lr} \min\ \sum_j c_j(s_j + p_j - d_j)^+ & \\ S'_j \subset S_{y_j},\ \ \text{all } j & (a) \\ s_j \geq r_j,\ \ \text{all } j & (b) \\ \text{noOverlap}\left((s_j|y_j = i), (p_j|y_j = i)\right),\ \ \text{all } i & (c) \\ s_j + p_j \leq s_{j'},\ \ \text{all } (j, j') \in E & (d) \end{array} \tag{4.1}$$

The objective function minimizes weighted tardiness of tasks. Constraint (a) ensures that agents have the skills for the tasks they are assigned. Constraints (b) states that tasks are processed after their release times. The noOverlap constraint in (c) has the form noOverlap(s, p), which requires that tasks $1, \ldots, n$ with processing times $p = (p_1, \ldots, p_n)$ be given start times $s = (s_1, \ldots, s_n)$ so that the tasks do not overlap. Constraint (c) therefore states that the tasks assigned to each agent i do not overlap. Constraint (d) enforces the precedence graph. An agent who will be unavailable for a certain period $[t, t']$ should be pre-assigned a dummy task with a fixed start time t and duration $t' - t$.

4.3.2 General Robust Model

The idea behind robust scheduling is to leave flexibility in the schedule so that it can be adjusted to accommodate unexpected delays without major disruption. The simplest sort of robust scheduling plans for the worst case. Suppose $\theta = (\theta_1, \ldots, \theta_m)$ is a tuple of uncertain parameters, such as tasks' processing times or delay costs. If we let $f(x, \theta)$ be the cost of a schedule x when θ represents a tuple of uncertain parameters, then the worst-case robust scheduling solves the problem $\min_x\{\max_\theta\{f(x, \theta)\}\}$.

The problem with worst-case scheduling is that it is too conservative. There is vanishingly small probability that every parameter will take its worst-case value, and it makes no sense to plan for this.

If we know the joint distribution of the random variables $\theta_1, \ldots, \theta_m$, then we can optimize over all possible realizations of θ that have probability at least ϵ:

$$\min_{x \in X} \left\{ \max_{\theta} \{ f(x, \theta) \mid Pr(\theta) \geq \epsilon \} \right\}$$

However, we don't know the joint distribution. It is hard enough to make realistic assumptions about the distribution of one parameter, and practically impossible to characterize the joint distribution of thousands of parameters. We will therefore abandon this approach.

A recently studied alternative (Bertsimas and Brown, 2009; Bertsimas and Sim, 2003, 2004; Conde, 2009; Natarajan et al., 2009; Yaman et al., 2007) is *restricted robust scheduling*, which limits how many parameters can take really bad values. We suppose that θ belongs to an *uncertainty set* Θ, which contains θ vectors that we believe are not too unlikely, without quantifying probabilities as illustrated (4.2).

$$\min_{x \in X} \left\{ \max_{\theta \in \Theta} \{ f(x, \theta) \} \right\} \tag{4.2}$$

r-restricted robust scheduling supposes that at most r parameters take their worst-case value. However, we will not solve our problem as an r-restricted robust scheduling problem since there is no evident upper bound on the number of delayed tasks in the SDO's service center. Instead, we consider uncertain parameters that belong to uncertainty sets. This still complicates solution of the problem, but we believe the decompositions described in the following sections can provide a practical solution.

In the IT SDO's scheduling problem, the uncertain parameters are the delays in the release times and the processing times. The delays in the release times are defined by Δr where Δr represents a tuple of the delays, $(\Delta r_1, \ldots, \Delta r_n)$ and belongs to uncertainty set R. The uncertain processing times are $p + \Delta p$ where Δp represents a tuple of the processing time's uncertain part, $(\Delta p_1, \ldots, \Delta p_n)$ and Δp belongs to uncertainty set P.

A difficulty with unpredictable processing times is that tasks assigned to a particular agent may overlap. We thus want to fix the sequence σ rather than the start times s of tasks when solving the worst-case subproblem, because start times may have to be adjusted to avoid task overlap. We therefore replace the start time variable s_j with a sequence variable σ_j, which represents the position of task j in sequence σ. Note that σ is a single sequence computed for all tasks and agents. Then, (4.2) becomes (4.3) where f defines the total weighted tardiness of all tasks in terms of the decision variables: σ, x, Δr, and Δp, representing the sequence, the assignment, the delays in the release times, and the uncertainty in the processing times for all tasks, respectively.

$$\min_{y,\sigma} \left\{ \max_{\substack{\Delta r \in R \\ \Delta p \in P}} \{ f(\sigma, y, r + \Delta r, p + \Delta p) \} \;\middle|\; S'_j \subset S_{y_j} \right\} \tag{4.3}$$

Now $f(\sigma, y, r + \Delta r, p + \Delta p)$ is calculated by first constructing a greedy schedule based on the given values of $(\sigma, y, r + \Delta r, p + \Delta p)$ and then observing the weighted tardiness of this schedule. The greedy schedule has each agent perform assigned tasks in the order given by σ.

Let $\mathrm{pr}(j, \sigma, y)$ be the task that immediately is sequenced before task j at the same agent y_j according to the sequence σ and assignment y. If we assume that σ observes the precedence relations in the graph (J, E), we can write

$$f(\sigma, y, r + \Delta r, p + \Delta p) = \sum_j c_j (s_j + p_j + \Delta p_j - d_j)^+ \tag{4.4}$$

where s_j is recursively defined for all $j = 1, \ldots, n$ by

$$s_j = \max \left\{ r_j + \Delta r_j,\ s_{\mathrm{pr}(j,\sigma,y)} + p_{\mathrm{pr}(j,\sigma,y)} + \Delta p_{\mathrm{pr}(j,\sigma,y)},\ \max_{(j',j) \in E} \{ s_{j'} + p_{j'} + \Delta p_{j'} \} \right\} \tag{4.5}$$

Each task j starts at the earliest possible time that is defined by the maximum of the three events formulated in (4.5): (i) the delayed release time of task j, (ii) the finishing time of the previously scheduled task just before task j at agent y_j, and (iii) the finishing time of all the predecessor tasks. Note that, $s_{\mathrm{pr}(j,\sigma,y)} = -\infty$ if task j is the first task assigned to some agent.

4.4 Logic Based Benders Decomposition

Logic-based Benders decomposition (Hooker and Ottosson, 2003) is a generalization of Benders decomposition accommodating a much wider range of problems. In contrast with the classical Benders method, the subproblem can in principle be any combinatorial problem, not necessarily a linear or nonlinear programming problem. For example, it can be a scheduling problem solved by constraint programming (CP), a method well suited to scheduling.

This flexibility has led to the application of logic-based Benders decomposition to planning and scheduling problems that naturally decompose into an assignment and a scheduling portion. Jobs are assigned to facilities by the Benders master problem using mixed integer programming (MILP), and the subproblem uses CP to schedule jobs on each facility. This approach can reduce solution times by several orders of magnitude relative to methods that use MILP or CP alone (Hooker, 2004, 2005b,a, 2006, 2007a; Jain and Grossmann, 2001; Thorsteinsson, 2001).

We apply logic-based Benders decomposition to the general robust problem represented in (4.2). First, write (4.2) as

$$\begin{aligned} &\min\ v \\ &v = \max_{\theta\in\Theta}\{f(x,\theta)\} \\ &x \in X \end{aligned} \tag{4.6}$$

The master problem for (4.6) is represented in (4.7), with x as the decision variable.

$$\begin{aligned} &\min\ z \\ &\text{Benders cuts} \\ &x \in X \end{aligned} \tag{4.7}$$

The objective in (4.7) is to minimize the objective function z when the decision variable x is in the feasible set of x vectors, X. The Benders cuts are formulated in terms of the decision variable x and put lower bounds on the objective function z as the logic-based Benders decomposition continues to iterate.

Given the optimal solution of the master problem $\bar{x}$, the subproblem is formulated in (4.8).

$$\begin{aligned} &\min\ v \\ &v = \max_{\theta\in\Theta}\{f(\bar{x},\theta\} \end{aligned} \tag{4.8}$$

or simply $\max_{\theta\in\Theta}\{f(\bar{x},\theta)\}$. The objective is to calculate the worst-case solution for the tuple of the uncertain parameters, θ, which is the only decision variable in the subproblem.

Benders cuts are easy to generate. Let (θ^*, v^*) solve the subproblem. Then for any x, the restricted worst-case cost is at least $f(x,\theta^*)$. So we have a Benders cut

$$z \geq f(x,\theta^*) \tag{4.9}$$

Normally, Benders cuts are obtained by solving an inference dual of the subproblem. This is unnecessary for the very simple Benders cut illustrated in (4.9). However, it may be possible to obtain stronger cuts from the dual.

Next, we investigate whether a similar technique can solve a robust project scheduling problem where the processing times and the delays in the release times of projects' tasks are uncertain. First, we introduce a robust scheduling model with three-stage decomposition in Section 4.5. Then, we simplify the three-stage decomposition into a two-stage decomposition as shown in Section 4.5.4.

4.5 Robust Scheduling Model with Three-Stage Decomposition

To simplify notation, let's assume all unit tardiness costs of tasks equal 1 (i.e., $c_j = 1$ in (4.1) for all j) for the remaining part of the chapter. Thus, the weighted tardiness of tasks equals the sum of each task's tardiness. To decompose the scheduling problem in (4.3), we write the master problem as

$$\begin{array}{lr} \min\ z & \\ S'_j \subset S_{y_j},\ \text{ all } j & (a) \\ \sigma_j < \sigma_{j'},\ \text{ all } (j,j') \in E & (b) \\ \text{Benders cuts} & (c) \\ y_j \in \{1,\ldots,m\},\ \text{ all } j & (d) \end{array} \tag{4.10}$$

The objective function minimizes the total tardiness when the decision variables y_j and σ represent the agent to whom task j is assigned, and the sequence of all tasks, respectively. Constraint (a) ensures that agents have the skills for the tasks they are assigned. Constraint (b) enforces the precedence graph. Constraints (c) are the Benders cuts that will be added as the Benders decomposition continues to iterate. The Benders cuts are written in terms of decision variables y_j and σ, and put lower bounds on the total tardiness, z, in the master problem.

Then, the subproblem is formulated as follows.

$$\begin{array}{lr} \max_{s,\Delta r,\Delta p}\ \sum_j (s_j + p_j + \Delta p_j - d_j)^+ & \\ s_j = \max\left\{r_j + \Delta r_j,\ s_{\text{pr}(j,\bar\sigma,\bar y)} + p_{\text{pr}(j,\bar\sigma,\bar y)} + \Delta p_{\text{pr}(j,\bar\sigma,\bar y)}\right\},\ \text{ all } j & (a) \\ \Delta p \in P,\ \ \Delta r \in R & (b) \end{array} \tag{4.11}$$

where $(\bar\sigma, \bar y)$ is the solution of the master problem. The starting time of each task, the uncertain part of the processing time, and the delay in the release time of each task are decision variables. Constraints (a) compute the starting time of task j by calculating the maximum of the delayed release time of task j and the finishing time of the task sequenced just before task j at agent y_j. Note that the precedence graph has already been considered while computing the sequence in the master problem, so in Constraint (b) we do not need to include the finishing time of task j's predecessors. If uncertainty sets P and R are polyhedra, this problem can be formulated as an MILP formulation since the sequence of tasks is fixed.

If $(s^*, \Delta r^*, \Delta p^*)$ solves the subproblem, the simplest Benders cut is the nogood cut

$$z \geq \begin{cases} \sum_j (s_j^* + p_j + \Delta p_j^* - d_j)^+ & \text{if } (\sigma, y) = (\bar{\sigma}, \bar{y}) \\ -\infty & \text{otherwise} \end{cases} \tag{4.12}$$

Note that the cut can be strengthened by finding a subset of the σ_js and y_js that, when fixed to $\bar{\sigma}_j$ and $\bar{y}_j$, result in the same bound.

One can also use the cut (4.9), which is now

$$z \geq f(\sigma, y, r + \Delta r^*, p + \Delta p^*) \tag{4.13}$$

where f represents the total tardiness function when $\sigma, y, \Delta r^*$, and Δp^* are fixed. It is not obvious how to impose this as a constraint in the master problem, because the value of f must be calculated for each σ, y exactly. In principle, one can incorporate the problem of finding the right start times s into a master problem (using only inequality constraints), because the master problem is a minimization problem. A different set of variables s^k must be used for each Benders cut k. Then the master problem becomes:

$$\begin{array}{l} \min\ z \\ S'_j \subset S_{y_j},\ \text{all } j \\ \sigma_j < \sigma_{j'},\ \text{all } (j,j') \in E \\ z \geq \sum_j z_j^k \\ z_j^k \geq s_j^k + p_j + \Delta p_j^k - d_j,\ \text{all } j,k \\ s_j^k \geq r_j + \Delta r_j^k,\ \text{all } j,k \\ s_j^k \geq s_{\text{pr}(j,\sigma,y)}^k + p_{\text{pr}(j,\sigma,y)} + \Delta p_{\text{pr}(j,\sigma,y)}^k,\ \text{all } j,k \\ s_j^k + p_j + \Delta p_j^k \leq s_{j'}^k,\ \text{all } (j,j') \in E, \text{all } k \\ s_j^k \geq 0,\ \text{all } j,k \\ y_j \in \{1,\ldots,m\} \end{array} \tag{4.14}$$

where the decision variables are z, z^k, y, σ, and s^k. $(\Delta r^k, \Delta p^k)$ is the solution of the kth subproblem. It is possible to formulate this as an MILP. However, the MILP model may become too large as the Benders cuts accumulate.

To overcome the increasing size of the problem, we propose a three-stage decomposition illustrated in Figure 4.1. The master problem assigns tasks to agents. Then, given the optimal assignment, Subproblem 1 finds an optimal sequence of tasks. This sequence is sent to the master problem via a Benders cut which is solved again. This iterative process continues until the master problem and Subproblem 1 have the same objective function value. Then, the optimal sequence and assignment are sent to Subproblem 2 that captures uncertainty by finding the worst-case given the uncertainty set. Hence, Subproblem 2 is solved, and the optimal starting times, the processing times, and the delays in the release times are sent as Benders cuts to the master problem which is solved again by iterating with Subproblem 1. This iterative process continues until the master problem and Subproblem 2 have the same objective function value.

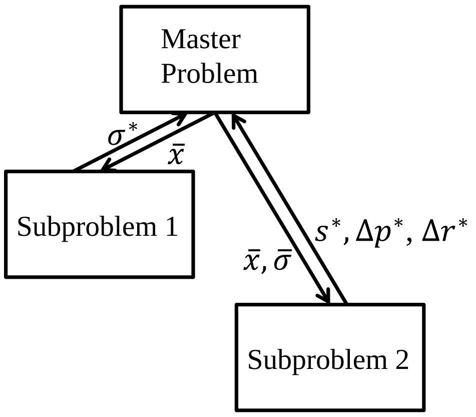


Figure 4.1: Three-stage decomposition.

The master problem is shown in (4.15) where x_{ij} is a binary variable indicating whether task j is assigned to agent i. If it is, x_{ij} equals 1; otherwise 0.

$$\begin{array}{ll} \min\ z & \\ S_j' \subset S_i, \ \text{all } j & (a) \\ \sum_i x_{ij} = 1, \ \text{all } j & (b) \\ \text{Relaxations} & (c) \\ \text{Benders cuts} & (d) \\ x_{ij} \in \{0,1\} & (e) \end{array} \tag{4.15}$$

The objective function minimizes the total tardiness. Constraint (a) ensures that agents have the skills for the tasks they are assigned. Constraint (b) enforces that every task should be assigned to an agent. Constraints (c) and (d) are relaxations and Benders cuts, respectively, putting lower bound on the total tardiness, z.

4.5.1 Solving the Sequencing Subproblem

A project's tasks are typically carried out by a single agent who has at least the necessary required skill level to complete the tasks in the IT service center. Thus, we can decompose the problem with respect to agents ignoring the precedence relations of tasks that are assigned to different agents. However, if a project is not completely assigned to an agent, then, due to precedence relations between tasks, we may need to decompose the problem with respect to sets of agents such that a set of agent is defined by the smallest set of agents assigned to the tasks having precedence relations. For example, suppose there are four tasks, and task 1 is the predecessor of task 2. Tasks 1 and 2 are assigned to agents 1 and 3, respectively. And the other tasks are assigned to agent 2. Hence, there are two sets of agents: $\{1,3\}$, and $\{2\}$, that we need for decomposing Subproblem 1 in this example.

For each iteration k and each agent i, we solve Subproblem 1 represented in (4.16) to find an optimal sequence of the tasks assigned to agent i without uncertainty.

$$\begin{array}{ll} \min\ z_i^k & \\ z_i^k \geq \sum_k (s_j^k + p_j - d_j)^+ & (a) \\ \texttt{disjunctive}\left((s_j \mid \bar{x}_{ij}=1),(p_j \mid \bar{x}_{ij}=1)\right) & (b) \\ s_j^k \geq s_{j'}^k + p_{j'} \quad \text{all } \ (j',j) \in E, \bar{x}_{ij} = \bar{x}_{ij'} = 1 & (c) \\ s_j^k \geq r_j, \ \text{ all } j & (d) \end{array} \tag{4.16}$$

In (4.16) for agent i and current iteration k, the objective is to minimize agent i's total tardiness. Constraint (a) computes the total tardiness for agent i. Given the optimal assignment of the master problem, $\bar{x}_{ij}$, constraint (b) computes a disjunctive sequence for the tasks assigned to agent i by the `disjunctive` global constraint, so only one task is served at a time. Constraint (c) enforces the precedence graph. Constraint (d) states that a task starts after its release time.

The optimal sequence computed by the Subproblem 1 is sent as a Benders cut to the master problem that is solved again. The iterative process between Subproblem 1 and the master problem continues until they have the same objective function value.

4.5.2 Solving the Uncertainty Subproblem

When Subproblem 1 and the master problem have the same objective function value, we solve Subproblem 2 represented in (4.17).

$$\begin{array}{ll} \max\ \sum_j (s_j + p_j + \Delta p_j - d_j)^+ & \\ s_j = \max\left\{r_j + \Delta r_j, s_{pr(j,\bar{\sigma},\bar{x})} + p_{pr(j,\bar{\sigma},\bar{x})} + \Delta p_{pr(j,\bar{\sigma},\bar{x})}\right\}, \ \text{ all } j & (a) \\ \Delta p \in P, \ \ \Delta r \in R & (b) \end{array} \tag{4.17}$$

Subproblem 2 captures the uncertainty sets of processing times and release times. The objective is to maximize the worst-case total tardiness within the uncertainty sets given sequence $\bar{\sigma}$ and assignment $\bar{x}$. Constraint (a) computes the starting time of each task j by considering whether they can start at their delayed release times or when the previously scheduled task just before task j at the same agent is finished. Constraint (b) constrains the uncertain parameters to take values from the respective uncertainty sets.

If we decide to use this three-stage decomposition, we can rewrite Subproblem 2 within a dynamic programming model, since dynamic programming can be effectively applied to handle uncertainty of the robust scheduling problem given the assignment and the sequence. At each stage in the dynamic programming model, we schedule a task in accordance with the given sequence, and the decision is to pick the task's uncertain parameters defined by the uncertainty set. To simplify our notation, let's assume that the delays in the release times are the only uncertainties at the service center. (Even if we might have assumed that only the delays in the release times are uncertain, it will be a valid assumption since the processing times of the tasks are very short in the IT service center.)

Next, we prove that the optimal values of the uncertain parameters lie in a subset of extreme points. This result simplifies the three-stage decomposition to a two-stage decomposition in Section 4.5.4.

4.5.3 Dynamic Programming Formulation

Let's define the decision variables and the states of the (backward) recursive relationship for the maximum-total tardiness problem. The decision variable at stage j is the delay of task j's release time. The value function is the total tardiness, where the states are the vector of slack values of the uncertainty set and the ending time of the previously scheduled task. Note that the delays are represented in the release times by Δr where Δr is a vector of the delays for each task: $\Delta r = [\Delta r_1, ..., \Delta r_n]$.

At each stage, the decision is to pick release times that maximize total tardiness in the schedule from that stage on. The uncertainty set of the delays in release times is defined by a polyhedral uncertainty set formulated as $A\,\Delta r \leq U$ and $\Delta r \geq \mathbf{0}$, where the dimension of the A matrix depends on the number of constraints defining the uncertainty set and the number of tasks. Given the sequence of tasks at each agent, the recursive relationship on the maximum-total tardiness is shown in (4.18) for stage j.

$$\begin{aligned} g_j(u_j, e_j) \quad &= \max_{\Delta r_j : A_{.,j} \Delta r_j \leq u_j} ((\max\{r_j + \Delta r_j, e_j\} + p_j - d_j)^+ \\ &\quad + g_{j+1}(u_j - A_{.,j}\Delta r_j, \max\{r_j + \Delta r_j, e_j\} + p_j)) \end{aligned} \tag{4.18}$$

where u_j is the vector representing the slack values of $A\,\Delta r \leq U$ at the j^{th} iteration, and e_j is the ending time of the previously scheduled task just before task j at the same agent. The boundary condition is

$$g_n(u_n, e_n) = \max_{\Delta r_n : A_{\cdot,n} \Delta r_n \leq u_n} \Big\{ \big(\max\{r_n + \Delta r_n, e_n\} + p_n - d_n \big)^+ \Big\} \tag{4.19}$$

Theorem 4.5.1. *The value function*

$$g_{j+1}\big(a - b\Delta r, \max\{\gamma + \Delta r, \delta\} + \epsilon\big) \tag{4.20}$$

is convex in Δr, where a, b, γ, δ, and ϵ are arbitrary constants.

Proof. The proof is by induction on $j = n-1, n-2, \ldots, 1$. When $j = n-1$, we have from (4.19) that (4.20) is

$$\max_{\Delta r_n} \Big\{ \big(\max\{r_n + \Delta r_n, \max\{\gamma + \Delta r, \delta\} + \epsilon\} + p_n - d_n \big)^+ \Big\}$$

It can be seen that this expression is convex in Δr by repeatedly applying the fact that $\max\{\alpha + \Delta r, \beta\}$ is convex in Δr for arbitrary constants α, β. And the maximum of two convex functions is a convex function. Now for the inductive hypothesis, we suppose that

$$g_{j+2}\big(a - b\Delta r, \max\{\gamma + \Delta r, \delta\} + \epsilon\big) \tag{4.21}$$

is convex in Δr, where a, b, γ, δ, and ϵ are arbitrary constants. Using (4.18), we can write (4.20) as follows:

$$\max_{\Delta r_{j+1}} \Big\{ \big(\max \big\{ r_{j+1} + \Delta r_{j+1}, \max\{\gamma + \Delta r, \delta\} + \epsilon \big\} + p_{j+1} - d_{j+1} \big)^{+} \\ + g_{j+2}\big(a - b\Delta r - A_{\cdot,j+1}\Delta r_{j+1}, \max \big\{ r_{j+1} + \Delta r_{j+1}, \max\{\gamma + \Delta r, \delta\} + \epsilon \big\} + p_{j+1}\big) \Big\}$$

The second line can be rearranged to yield

$$\max_{\Delta r_{j+1}} \Big\{ \big(\max \big\{ r_{j+1} + \Delta r_{j+1}, \max\{\gamma + \Delta r, \delta\} + \epsilon \big\} + p_{j+1} - d_{j+1} \big)^{+} \\ + g_{j+2}\big(a - b\Delta r - A_{\cdot,j+1}\Delta r_{j+1}, \max \big\{ \gamma + \Delta r, \max\{r_{j+1} + \Delta r_{j+1} - \epsilon, \delta\} + \epsilon + p_{j+1}\big) \Big\} \tag{4.22}$$

We wish to show that (4.22) is convex in Δr. But (4.22) has the form $\max_{\Delta r_{j+1}}\{B + C\}$. It can be seen as above that B is convex in Δr. Furthermore, C has the form (4.21) and is therefore convex by the inductive hypothesis. Thus $B + C$ is convex because it is the sum of convex functions. Finally, (4.22) is a maximum over convex functions and is therefore convex in Δr, as claimed. □

Now we can infer the convexity of the expression that is maximized in the recursion.

Corollary 4.5.2. *The expression maximized in recursion (4.19) is a convex function of Δr_j.*

Proof. Because (4.19) has the form $g_j(u_j, e_j) = \max_{\Delta r_j}\{B + C\}$, it suffices to show that B and C are convex in Δr_j. But B is clearly convex, and C is convex due to Theorem 4.5.1. □

This corollary implies extreme point solutions for a polyhedral uncertainty set, but it actually implies something stronger. We know that Δr_1 takes one of the two extreme values $\Delta r_1^{\min}$ or $\Delta r_1^{\max}$ (min or max of Δr_1 subject to $A\Delta r \leq U$). Also Δr_2

takes a min or max value subject to $A\Delta r \leq U$ and $\Delta r_1 = \Delta r_1^{\min}$, or a min or max value subject to $A\Delta r \leq U$ and $\Delta r_1 = \Delta r_1^{\max}$, and so forth. It follows that Δr takes an extreme point value, and in fact belongs to a special subset of extreme points. This subset is equal to the set of all extreme points when the uncertainty set is a simplex.

This results helps us to simplify the three-stage decomposition, and a two-stage decomposition is introduced in Section 4.5.4.

4.5.4 Robust Scheduling Model with Two-Stage Decomposition

By using Theorem 4.5.1, we show that the optimal solution of Subproblem 2, where worst-case delays are computed by maximizing the total tardiness given the uncertainty set and the sequence of tasks, lies at one of the extreme points of the uncertainty set. Thus, we simplify the three-stage decomposition to a two-stage decomposition. The notation is summarized in Table 4.2.

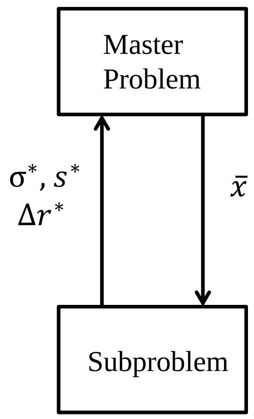


Figure 4.2: Two-stage decomposition.

The relation between the master problem and the subproblem is summarized in Figure 4.2. First, a master problem is solved assigning tasks to agents. Then, the

Table 4.2: Nomenclature for the two-stage decomposition for the robust scheduling problem.

Indices	
$j \in J$	task
i	agent
k	iteration
$\ell \in L$	extreme point of a polyhedron defining the uncertainty set, L, for the delays of the release times
Sets	
J	set of tasks ($\{1, 2, ..., n\}$)
S_i	set of agent i's skills
L	set of extreme points defining the uncertainty set for the delays of the release times
Parameters	
d_j	deadline (or due date) of task j
$\bar{x}_{ij}$	solution value of x_{ij} in the master problem
σ^*	robust sequence calculated at the subproblem
Δr_j^*	delay in the release time of task j in the subproblem
$s_{j,\ell}^*$	start time of task j in the subproblem when ℓ^{th} extreme point defines the delays
T_k^*	total tardiness at previous iteration k
Variables	
x_{ij}	$= 1$ if task j is assigned to agent i
T	total tardiness
$s_{j,\ell}$	start time of task j when ℓ^{th} extreme point of the uncertainty set L defines the delays
$\Delta r_{\ell j}$	delay in the release time of task j in the ℓ^{th} extreme point of the uncertainty set L
σ	sequence of tasks

optimal solution of the master problem becomes parameters for the subproblem where we schedule tasks with respect to the uncertainty set of the delays in the release times of the tasks.

The master problem is shown in (4.23). It is an assignment problem that computes an optimal assignment of tasks to agents at each iteration.

$$\begin{array}{ll} \min\ z & \\ S_j' \subset S_i, \ \ \text{all } j & (a) \\ \sum_i x_{ij} = 1, \ \ \text{all } j & (b) \\ \text{Relaxations} & (c) \\ \text{Benders cuts} & (d) \\ x_{ij} \in \{0,1\}, \ \ \text{all } i,j & (e) \end{array} \tag{4.23}$$

The objective function is to minimize tardiness in (4.23) where x_{ij} is the decision variable representing whether task j is assigned to agent i. Constraint (a) ensures each task is assigned to an agent that has at least required skills to serve that task. Each task is assigned to only one agent by Constraint (b). We have relaxations and Benders cuts as introduced in Sections 4.5.5 and 4.5.6 in Constraints (c) and (d), linking x_{ij} variables with the objective function z.

After solving the master problem, we find the optimal assignment at the current iteration. To find a robust schedule, the master problem communicates with the subproblem via sending the optimal assignment of the tasks to the agents, $\bar{x}_{ij}$. Thus, given the optimal assignment calculated at the master problem, we solve the subproblem in (4.24).

$$\begin{array}{ll} \min\ T & \\ T \geq \sum_j (s_{j\,\ell} + p_j - d_j)^+, \ \ \text{all } \ell & (a) \\ \text{noOverlap}\left((s_j | \bar{x}_{ij} = 1), (p_j | \bar{x}_{ij} = 1)\right), \ \ \text{all } i, \ell & (b) \\ s_{j\,\ell} \geq r_j + \Delta r_{\ell j}, \ \ \text{all } j, \ell & (c) \\ s_{j\,\ell} \geq s_{j'\,\ell} + p_{j'} \ \ \ \text{all } \ (j', j) \in E & (d) \end{array} \tag{4.24}$$

We find the schedule of the tasks assigned to the agents considering the delays of the release times via a minmax problem in (4.24). Let's define L the set of all extreme points of the polyhedron representing the delays of the release times (i.e., uncertainty set); r_j, p_j, and d_j are release time, processing time, and deadline of task j, respectively. The objective function in (4.24) is to minimize maximum tardiness, T, that is calculated for each extreme point ℓ in the uncertainty set L. Variable $s_{j\,\ell}$ is the starting time of task j when the delay is defined by the ℓth extreme point. Constraint (a) computes the total tardiness. Constraint (b) prevents overlapping of tasks at agent i via the global constraint, `noOverlap`, given the optimal assignment from the master problem, $\bar{x}_{ij}$. A task j can start only after its release time and the possible delay (if there is any) as illustrated in Constraint (c). Constraint (d) enforces the precedence graph.

Theorem 4.5.3. *The three-stage decomposition and the two-stage decomposition are equivalent optimization problems.*

Proof. We rewrite the two-stage problem's subproblem in (4.24). The two-stage problem minimizes (4.25) over all $\bar{x}$.

$$\min_{s_\ell} \max_{\ell} \left\{ \sum_j (s_{j\,\ell} + p_j - d_j)^+ \;\middle|\; \begin{array}{l} \text{noOverlap}\left((s_j|\bar{x}_{ij}=1),(p_j|\bar{x}_{ij}=1)\right), \;\; \text{all } i,\ell \\ s_{j\,\ell} \geq r_j + \Delta r_{\ell j}, \;\; \text{all } \ell \\ s_{j\,\ell} \geq s_{j'\,\ell} + p_{j'\,\ell}, \;\; \text{all } (j',j) \in E, \, \bar{x}_{ij} = \bar{x}_{ij'} = 1 \end{array} \right\} \tag{4.25}$$

where $s_\ell = [s_{1\,\ell}, \ldots, s_{n\,\ell}]$ is a vector of the starting times of tasks when the delay of release times is defined by the ℓth extreme point of the uncertainty set.

In the three-stage decomposition introduced in Section 4.5.4, the objective is to minimize the tardiness over possible sequences in Subproblem 1. In Subproblem 2, the worst-case delay is computed given the sequence. First, given sequence $\bar{\sigma}$, let's rewrite Subproblem 2 as follows.

$$\max_{\ell} \quad \min_{s_\ell} \left\{ \sum_j (s_{j\,\ell} + p_j - d_j)^+ \;\middle|\; \begin{array}{l} s_{j\,\ell} \text{ reflects } \bar{\sigma} \\ \text{noOverlap}\left((s_j|\bar{x}_{ij}=1),(p_j|\bar{x}_{ij}=1)\right), \ \text{ all } i,\ell \\ s_{j\,\ell} \geq r_j + \Delta r_{\ell j}, \ \text{ all } \ell \end{array} \right\} \tag{4.26}$$

where s_ℓ is defined again as the vector of the starting times of tasks when the delay of release times is defined by the ℓth extreme point of the uncertainty set.

Then, we combine Subproblem 1 and reformulated Subproblem 2 as illustrated in (4.26). The resulting formulation is shown in (4.27).

$$\min_{\sigma} \quad \max_{\ell} \quad \min_{s_\ell} \left\{ \sum_j (s_{j\,\ell} + p_j - d_j)^+ \;\middle|\; \begin{array}{l} s_{j\,\ell} \text{ reflects } \sigma \\ \text{noOverlap}\left((s_j|\bar{x}_{ij}=1),(p_j|\bar{x}_{ij}=1)\right), \ \text{ all } i,\ell \\ s_{j\,\ell} \geq r_j + \Delta r_{\ell j}, \ \text{ all } \ell \\ s_{j\,\ell} \geq s_{j'\,\ell} + p_{j'\,\ell}, \ \text{ all } (j',j) \in E, \, \bar{x}_{ij} = \bar{x}_{ij'} = 1 \end{array} \right\} \tag{4.27}$$

The three-stage problem minimizes (4.27) over all $\bar{x}$. The formulation in (4.27) is equivalent to formulation in (4.28) since $s_{j\,\ell}$ reflects σ.

$$\min_{s_\ell} \quad \max_{\ell} \quad \min_{s_\ell} \left\{ \sum_j (s_{j\,\ell} + p_j - d_j)^+ \;\middle|\; \begin{array}{l} \text{noOverlap}\left((s_j|\bar{x}_{ij}=1),(p_j|\bar{x}_{ij}=1)\right), \ \text{ all } i,\ell \\ s_{j\,\ell} \geq r_j + \Delta r_{\ell j}, \ \text{ all } \ell \\ s_{j\,\ell} \geq s_{j'\,\ell} + p_{j'\,\ell}, \ \text{ all } (j',j) \in E, \, \bar{x}_{ij} = \bar{x}_{ij'} = 1 \end{array} \right\} \tag{4.28}$$

But we can simplify (4.28) by deleting the redundant inner minimization over s_ℓ, which yields (4.25). Thus the two-stage and three-stage problems both minimize (4.25) over all $\bar{x}$ and are therefore equivalent. $\square$

4.5.5 Relaxations of the Subproblem for the Master Problem

We can strengthen the master problem with relaxations of the subproblem, since the subproblem relaxations allow the master problem to select reasonable assignments before many Benders cuts have been accumulated. The first relaxation is illustrated in (4.29).

$$\begin{array}{l} T \geq \sum_i T_i^D \\ T_i^D \geq \sum_{j \in J(0,d_k)} p_{ij} x_{ij} - d_k \qquad \forall i, k \\ T_i^D \geq 0 \qquad \forall i \end{array} \tag{4.29}$$

where $J(0, d_k)$ is the set of tasks whose time windows (i.e., release time and deadline) are between 0 and d_k (i.e., deadlines are less than d_k). For each agent and each distinct deadline of tasks, suppose all possible tasks that can start and need to be finished before the chosen distinct deadline are scheduled consecutively. If we assume that these tasks' deadlines equal to the chosen distinct deadline, then tardiness value for each agent, T_i^D is a valid relaxation computing a lower bound on the tardiness of agent i.

As a second relaxation, we use a bound on tardiness that is developed by Hooker (2007a). We assume $\min_j \{r_j\}$ is zero without loss of generality. (Note that if $\min_j \{r_j\} \neq 0$, this relaxation may become loose.) We index the tasks so $d_1 \leq d_2 \leq \cdots \leq d_n$.

The relaxation is then written as:

$$\begin{array}{l} T \geq \sum_{i=1}^m \sum_{k=1}^n \bar{T}_{ik} \\ \bar{T}_{ik} \geq 0 \\ \bar{T}_{ik} \geq (\displaystyle\sum_{j=1}^{k} p_{i\pi_{i(j)}} x_{i\pi_{i(j)}} - d_k) - (1 - x_{ik}) U_{ik} \end{array} \tag{4.30}$$

where $U_{ik} = \sum_{j=1}^k p_i \pi_{i(j)} - d_k$, π is a permutation of the indices for which $p_{\pi(1)} \leq \cdots \leq p_{\pi(n)}$.

4.5.6 Benders Cuts

The following nogood cut is added to the master problem at each iteration. We can also strengthen the cut by finding the smallest set of the tasks that results in the same optimal objective function value, T^*.

$$z \geq \left\{ \begin{array}{cc} T^* \text{ if } & x = \bar{x} \\ -\infty & \text{otherwise} \end{array} \right\} \tag{4.31}$$

4.6 Conclusion

We consider a practical project scheduling problem at a large IT services delivery organization (SDO) with cross-trained agents, heterogeneous projects, schedule disruptions, and service quality guarantees. Durations of projects' tasks, disruptions during task service, and project arrival times are uncertain. We estimate system parameters using real data from the IT SDO. Our goal is to identify an effective schedule of tasks and assignment of tasks to agents, capturing uncertainties in a robust scheduling model.

We capture the uncertainties by uncertainty sets and use logic-based Benders decomposition to solve the robust scheduling problem. Due to the processes of the SDO's IT service center, we assume the uncertainties are the delays in the release times of tasks and the processing times. First, we model a three-stage decomposition. In the master problem, tasks are assigned to agents. Given the optimal assignment of the master problem, a sequence is computed in Subproblem 1. Then, the worst-case solution given the uncertainty sets is calculated in Subproblem 2. We prove that given polyhedral uncertainty sets, the worst-case solution of the second subproblem lies at one of the extreme points. Thus, we show that we can simplify the three-stage decomposition to a two-stage decomposition. In the two-stage decomposition, we first compute the optimal assignment of tasks to agents at the master problem. Then, we solve a scheduling model with uncertainty in the subproblem. The equivalence of the three-stage and two-stage decompositions is also proven. In addition, relaxations and Benders cuts are introduced for the decompositions.

In summary, we introduce a novel robust scheduling method to solve a project scheduling problem with uncertainty. We apply robust optimization to capture uncertainty by uncertainty sets. Possible future research includes development of other Benders cuts and strengthening them in the decomposition since the bounds are unnecessarily weak and in many cases a proper subset of tasks are responsible for the objective function value.

5 CONCLUSION

Scheduling is an important decision-making process. If not managed properly, a penalty can be charged both in the form of loss of goodwill and proportional to the tardiness of the delivery. In this thesis, I aim to deepen the understanding of scheduling problems in practice, and hope to inspire more related future research.

In Chapter 2, I solve single-facility non-preemptive scheduling problems over a long time horizon, with tasks with deterministic durations but different release times and deadlines. I use a hybrid method: logic-based Benders decomposition to solve this pure scheduling problem. I decompose the long time horizon into time segments and consider two versions of the single-facility scheduling problem: *segmented* and *unsegmented*. In the *segmented* problem, each job must be completed within one time segment. In the *unsegmented* problem, jobs can overlap two or more segments. I compare classical methods and our logic-based Benders decomposition in terms of computation time and memory requirements. I find that for these problems, logic-based Benders scales up more effectively than state-of-the-art Constraint Programming (CP) and Mixed Integer Linear Programming solvers, especially for the segmented problem. I further find that logic-based Benders decomposition is not necessarily the fastest method, but clearly the most robust for the unsegmented problem. I suggest a strategy of applying CP first, and if it fails to solve the problem within a few seconds, switching to logic-based Benders decomposition. For segmented instances, logic-based Benders decomposition is always superior, and, thus should be used from the start.

In Chapter 3, I analyze a service center at a large, global, IT services delivery organization (SDO) with cross-trained agents, heterogeneous customer requests, and service level agreements. I model the system as a multi-server queueing system, applying approximation and bounding techniques to evaluate different control policies, to determine an appropriate static agent base and an effective request-assignment policy. I demonstrate a simple but effective request-assignment policy based on thresholds, and numerically compare this threshold-based policy and two naïve policies, capturing the IT SDO's service centers' processes. I reveal that applying the threshold-based policy might decrease the total cost without any major changes in the service center. I further find managerial insights about the potential benefits of a such policy, for example, how to set optimal threshold values, how to negotiate costs with customers, and what mix of agents to hire.

In Chapter 4, I study a similar problem as Chapter 3: a project scheduling problem at a large IT SDO with cross-trained agents, heterogeneous projects, and service quality guarantees. The durations of projects' tasks and arrival times are uncertain; in addition, projects are subject to random disruptions. The objective function is to minimize the weighted total tardiness. I identify an effective schedule of tasks and assignments to agents by utilizing a novel robust scheduling model based on logic-based Benders decomposition in which the uncertainties are captured by uncertainty sets. I prove convexity of the objective function under a structured uncertainty set (polyhedral). I show that the robust scheduling model is simplified because of this convexity.

Besides the future work mentioned in the conclusion of each chapter, there are other directions toward which this thesis can be extended. Characterizing the single-facility scheduling problems in Chapter 2 to reveal the phase-transition of instances (where average problem difficulty peaks and there is a mix of feasible and infeasible instances), especially when solving an instance becomes nontrivial, is an interesting direction for future research. Another potential future work is to explore different uncertainty sets capturing time varying projects' arrival rates in Chapter 4, as this is particularly useful to derive insights about the sensitivity of the robust scheduling model.

A MATRIX ANALYTIC METHOD

The 1D-infinite Markov chain is modeled as a quasi-birth-and-death (QBD) process. Level l denotes the l^{th} column. For instance, in System 1, the number of 1H requests in the system defines the level, whereas, in System 2, the number of 2H requests in the system defines the level. The generator matrix, Q, of the process is expressed as a block diagonal matrix:

$$Q = \begin{pmatrix} L^{(0)} & F^{(0)} & & & \\ B^{(1)} & L^{(1)} & F^{(1)} & & \\ & B^{(2)} & L^{(2)} & F^{(2)} & \\ & & \ddots & \ddots & \ddots \end{pmatrix}$$

L$^{(i)}$ encodes the (local) transitions within level i, for $i \geq 0$, F$^{(i)}$ encodes the (forward) transitions from level i to level $i+1$, for $i \geq 0$, and B$^{(i)}$ encodes the (backward) transitions from level i to level $i-1$, for $i \geq 1$. After incorporating busy period transitions, all submatrices will have finite dimensions. The matrices for System 1 are given explicitly in Section A.1.

The stationary probability of being in level i, $\overrightarrow{\pi_i}$, is given recursively by

$$\overrightarrow{\pi_i} = \overrightarrow{\pi_{i-1}} R^{(i)}.$$

$R^{(i)}$ records the expected number of visits to the states in level i between two visits to the level $i-1$. $\hat{l}$ is defined as the level QBD process starts repeating. Then, $R^{(l)}$ is given recursively by

$$F^{(l-1)} + R^{(l)} L^{(l)} + R^{(l)} R^{(l+1)} B^{(l+1)} = 0$$

when $l \leq \hat{l}$, and

$$R^{(l)} = R$$

when $l \geq \hat{l}$. R is the minimal solution to the following matrix quadratic equation:

$$F^{(\hat{l})} + RL^{(\hat{l})} + R^2 B^{(\hat{l})} = 0$$

$$\overrightarrow{\pi_0}(L^0 + R^{(1)}B^{(1)}) = \overrightarrow{0}$$

$$\overrightarrow{\pi_0} \sum_{l=0}^{\infty} \prod_{i=1}^{l} R^{(i)} \overrightarrow{1} = 1.$$

We can calculate the mean number of requests by using the stationary probabilities. For instance, in System 1, $E[N_{1L}]_{j_{1L},k_{1H}}$ and $E[N_{1H}]_{j_{1L},k_{1H}}$ denote the expected number of 1L and 1H requests, respectively, when the system is in state (j_{1L}, k_{1H}). Deriving the mean number of 1H requests is trivial as the exact number of 1H requests is tracked; see Figure 3.2. However, computing the number of 1L requests is more complicated and requires conditioning on the state of the Markov chain. For $i = \{0, 1, \ldots, n_{lL} - 1\}$, $E[N_{1L}]_{i_{1L},j_{1H}} = i$ for all j. For $i = t_{1L}^{+}$, $E[N_{1L}]_{t_{1L}^{+},j_{1H}}$ is the mean number of requests in an M/M/1 system given service rate $\mu_H + \mu_L$ plus the additional t_{1L} requests as the system is already busy. Using Little's Law, one can then calculate the mean response time of each class of request.

Let's look at L, F, and B matrices of System 1 (1L and 1H requests) introduced in Chapter 3 explicitly.

A.1 Matrices of System 1: 1H and 1L requests

Suppose we are solving the problem illustrated at Chapter 3. There are two agents at the service center: one high skilled and one low skilled. 1L and 1H requests arrive with rates λ_{1L} and λ_{1H}, respectively. A busy period begins when the number of 1L requests is 3 (i.e., $t_{1L} = 3$). Then, the submatrices are:

$$L^{(0)} = \begin{pmatrix} -\sigma_1 & \lambda_{1L} & 0 & 0 & 0 \\ \mu_L & -\sigma_2 & \lambda_{1L} & 0 & 0 \\ 0 & \mu_L + \mu_H & -\sigma_3 & \lambda_{1L} & 0 \\ 0 & 0 & t_1 & -\sigma_4 & t_{12} \\ 0 & 0 & t_2 & 0 & -\sigma_5 \end{pmatrix}$$

$$L^{(i)} = \begin{pmatrix} -\sigma_1 & \lambda_{1L} & 0 & 0 & 0 \\ \mu_L & -\sigma_2 & \lambda_{1L} & 0 & 0 \\ 0 & \mu_L & -\sigma_3 & \lambda_{1L} & 0 \\ 0 & 0 & t_1 & -\sigma_4 & t_{12} \\ 0 & 0 & t_2 & 0 & -\sigma_5 \end{pmatrix}$$

where σ_i is determined so that the sum of each row in the generator matrix Q becomes zero.

$$F^{(i)} = \lambda_{1H}\mathbf{I}$$

where $\mathbf{I}$ is an 5x5 identity matrix.

$$B^{(i)} = \begin{pmatrix} \mu_H & 0 & 0 & 0 & 0 \\ 0 & \mu_H & 0 & 0 & 0 \\ 0 & 0 & \mu_H & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

B MORE COMPUTATIONAL RESULTS FOR CHAPTER 3

Typical traffic at the motivating IT SDO's service centers is high for low complexity requests but moderate for high complexity requests as mentioned in Chapter 3. In addition, the total load of priority class 1 requests is lower than the total load of priority class 2 requests. Finally the number of high skilled agents is less than the number of low skilled agents. We generate more instances capturing these features and analyze their performance numerically in this section.

We consider two cases: (i) $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.4$, $\rho_{2H} = 1.1$, and (ii) $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.6$, $\rho_{2H} = 1.1$. For each case, there are two different agent bases: (a) $n_H = 2$, $n_L = 4$, and (b) $n_H = 2$, $n_L = 3$.

B.1 Stability of the System

Figures B.1 and B.2 depict the expected number of high skilled agents busy with 1L, 1H, 2L, and 2H requests as a function of t_{1L} and t_{2L}. The available number of high skilled agents is two which is represented by the dashed line. When the thresholds are low, the expected number of busy high skilled agents increases as low complexity requests are also served by the high skilled agents preempting high complexity requests. The expected number of busy high skilled agents in Figure B.1 is slightly less than the expected number of busy high skilled agents in Figure B.2 since the load of 2L requests is higher in Figure B.2. In addition, in Figures B.1 and B.2 when $t_{2L} = 3$, the expected number of busy high skilled agents is more than the available number of high skilled agents, thus these systems are not stable.

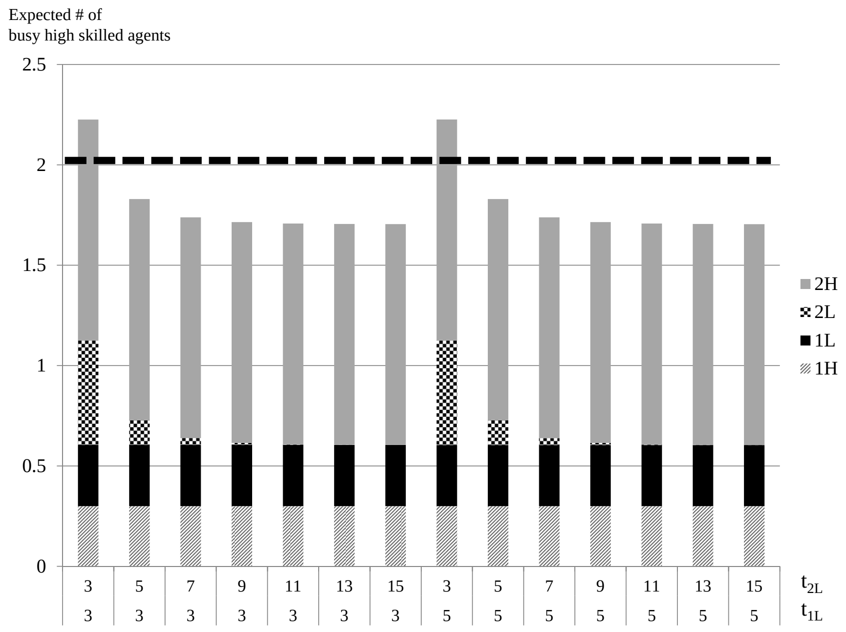


Figure B.1: Expected number of busy high skilled agents as a function of t_{1L} and t_{2L} when $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.4$, $\rho_{2H} = 1.1$, $n_L = 4$, and $n_H = 2$.

B.2 Effect of Threshold

B.2.1 Expected Response Times

The expected response time of 1H requests decreases and 1L requests increases as t_{1L} increases, until the expected response times converge to the case of separate queues at some point after which increasing t_{1L} further does not have any effect as illustrated in Figures B.3 and B.4, respectively, for Cases (i) and (ii). When we increase t_{1L}, 1L requests steal fewer cycles from the high skilled agent which decreases the expected response time of 1H requests. The expected response times of priority class 1 increases as the total number of agents decreases. Note that since priority class 1 requests have higher priority than priority class 2 requests, the expected response times of priority class 1 requests do not change if t_{2L} or load of priority class 2 requests changes.

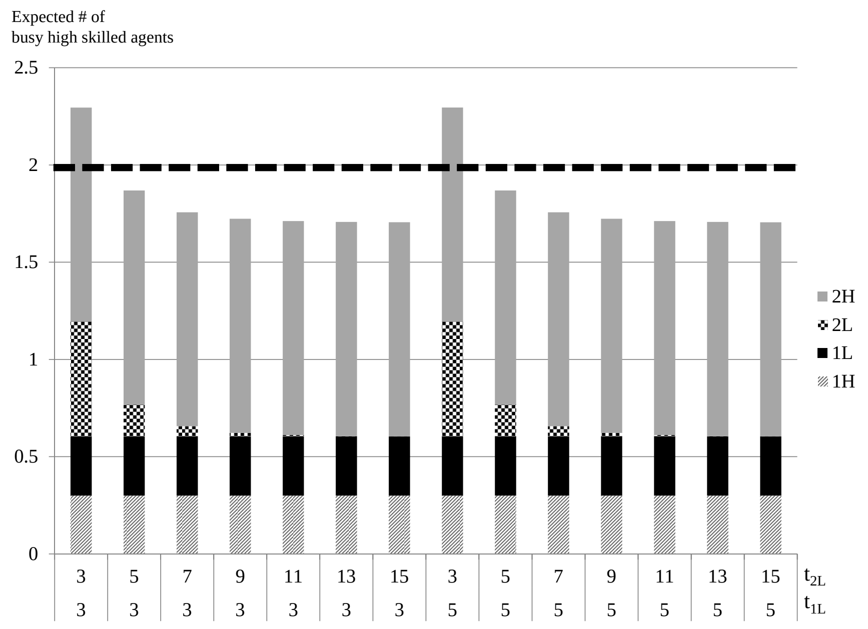


Figure B.2: Expected number of busy high skilled agents as a function of t_{1L} and t_{2L} when $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.6$, $\rho_{2H} = 1.1$, $n_L = 4$, and $n_H = 2$.

In addition, the expected response times of 1H and 1L requests increase when the number of low skilled agents decreases. Since the increase in 1H requests' expected response time is very small, this increase is not obvious in Figure B.3.

Similarly, expected response times of 2L and 2H requests are illustrated in Figures B.5 and B.6, respectively, as a function of t_{2L} when $t_{1L} = 5$. The impact of t_{2L} on 2H requests is higher than 2L requests because when t_{2L} is small, the high skilled agents' queue might become highly loaded for the 2H requests. In Case (ii) with a given agent base, the expected response times of priority class 2 requests are higher than the ones in Case (i) with the same agent base as the load of 2L requests is more in Case (ii). In addition, when the number of low skilled agents is 3, the expected response times are higher since 2L requests stay longer in the system and preempt 2H requests more often.

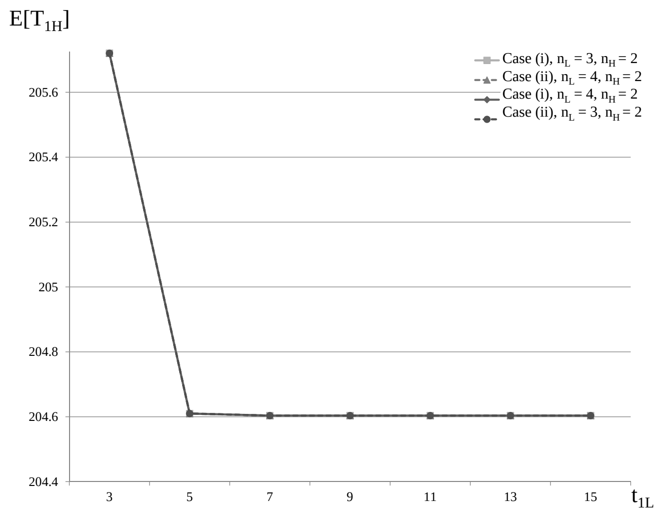


Figure B.3: Expected response time of 1H requests as a function of t_{1L}, $n_L \in \{3, 4\}$, $n_H = 2$. Case(i): $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.4$, $\rho_{2H} = 1.1$, and Case(ii): $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.6$, $\rho_{2H} = 1.1$.

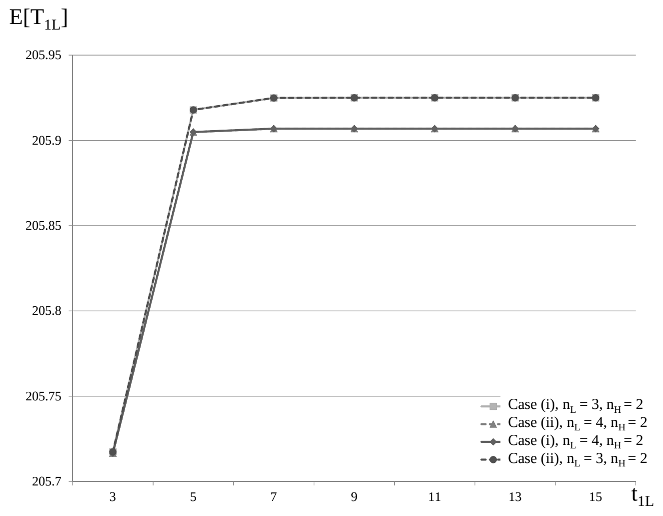


Figure B.4: Expected response time of 1L requests as a function of t_{1L}, $n_L \in \{3, 4\}$, $n_H = 2$. Case(i): $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.4$, $\rho_{2H} = 1.1$, and Case(ii): $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.6$, $\rho_{2H} = 1.1$.

B.2.2 Total Costs

In Chapter 3, we introduce two naïve policies potentially representing service centers'

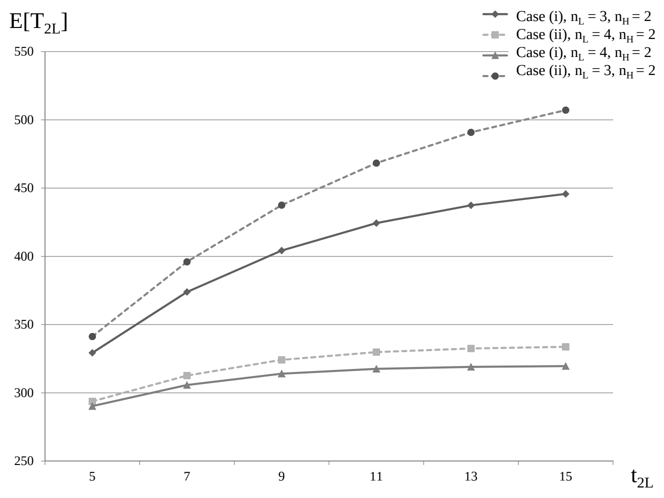


Figure B.5: Expected response time of 2L requests as a function of t_{2L} when $t_{1L} = 5$, $n_L \in \{3, 4\}$, $n_H = 2$. Case(i): $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.4$, $\rho_{2H} = 1.1$, and Case(ii): $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.6$, $\rho_{2H} = 1.1$.

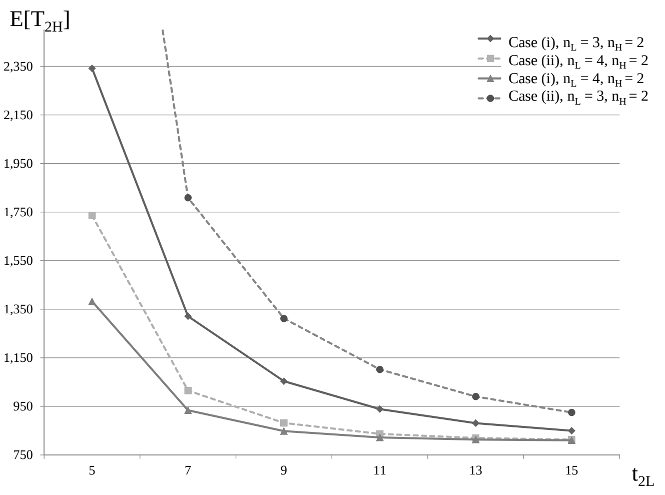


Figure B.6: Expected response time of 2H requests as a function of t_{2L} when $t_{1L} = 5$, $n_L \in \{3, 4\}$, $n_H = 2$. Case(i): $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.4$, $\rho_{2H} = 1.1$, and Case(ii): $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.6$, $\rho_{2H} = 1.1$.

operations with increasing cooperation: (i) Separate queue: high skilled agents serve only high complexity (1H and 2H) requests, and low skilled agents only serve low complexity (1L and 2L) requests, and (ii) Prioritized cycle stealing: high skilled agents serve high complexity requests and low skilled agents serve low complexity requests, priority class 1 customers have higher priority than priority class 2 customer requests, and if high skilled agents are idle, they can serve low complexity requests.

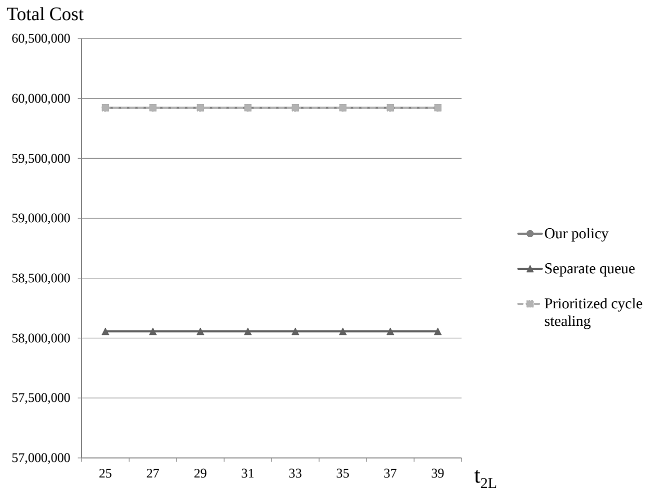


Figure B.7: Comparison of our policy and naïve policies when $t_{1L} = 35$, $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.4$, $\rho_{2H} = 1.1$, $C_{1L} = C_{1H} = 400$k, $C_{2L} = 250$k, $C_{2H} = 200$k, $C_L = 7,000$k, $C_H = 10,000$k, $n_L = 4$, and $n_H = 2$.

The total cost of 1L and 2L requests increases in the separate queue policy, but since the decrease in the cost of 2H requests is more than the increase in the low complexity requests' total cost, the separate queue policy performs the best in Figure B.7. Thus, hurting the low complexity requests by preventing them to preempt the high complexity requests improves the total cost. However, in Figure B.8, the separate queue policy performs the worst because as the load of 2L requests is higher, preventing the 2L requests to preempt the 2H requests hurts the system more and increases the cost of 2L requests more than it decreases the cost of 2H requests.

The total cost of a policy in Figure B.7 is lower than the total cost of the same policy illustrated in Figure B.8 since the total load of the system is higher in Case

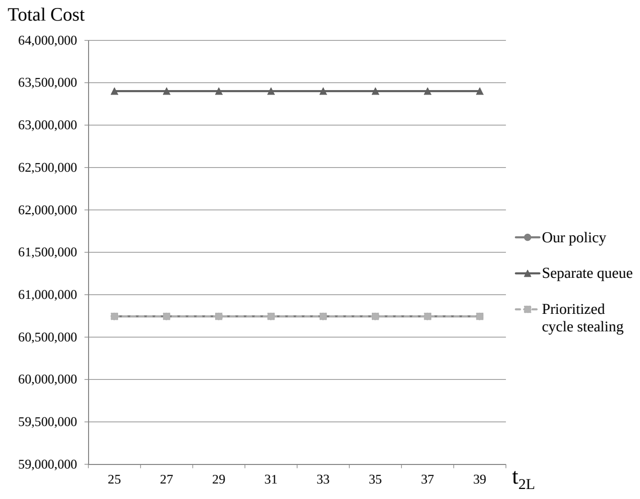


Figure B.8: Comparison of our policy and naïve policies when $t_{1L} = 35$, $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.6$, $\rho_{2H} = 1.1$, $C_{1L} = C_{1H} = 400$k, $C_{2L} = 250$k, $C_{2H} = 200$k, $C_L = 7,000$k, $C_H = 10,000$k, $n_L = 4$, and $n_H = 2$.

(ii) depicted in Figure B.8. In Figures B.9 and B.10, we analyze Case (i) and (ii) when the number of low skilled agents is 3. In both cases, the separate queue policy performs the worst. When we decrease the number of low skilled agents to 3 and apply separate queue policy, we hurt low complexity requests more than we improve the high complexity requests. Hence, our policy and prioritized cycle stealing policy perform better than the separate queue policy.

The total cost of our policy is lower than the prioritized cycle stealing policy with respect to t_{2L} as shown in Figures B.11 and B.12. A similar result is valid for Case (ii) when the number of low skilled agents is 4. If we increase t_{1L} and t_{2L} enough, the total costs of the prioritized cycle stealing policy and our policy should converge, because the expected response time converges at some point after which increasing t_{2L} further does not make any difference in the expected response times of priority class 2 requests. Thus, these examples allow us to understand when cooperation is important and how our policy might improve the total cost.

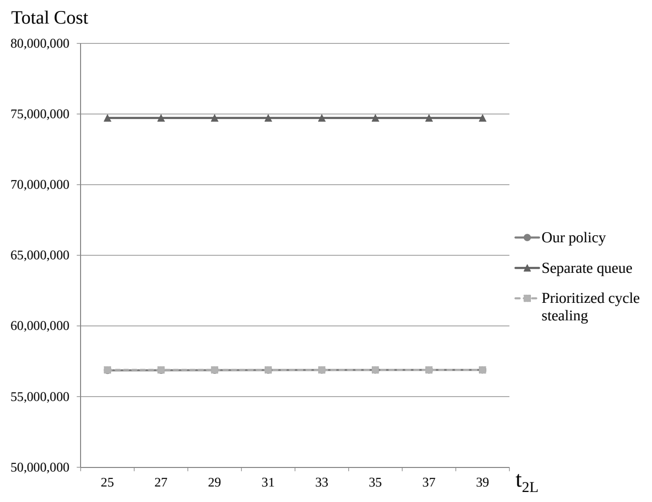


Figure B.9: Comparison of our policy and naïve policies when $t_{1L} = 35$, $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.4$, $\rho_{2H} = 1.1$, $C_{1L} = C_{1H} = 400$k, $C_{2L} = 250$k, $C_{2H} = 200$k, $C_L = 7,000$k, $C_H = 10,000$k, $n_L = 3$, and $n_H = 2$.

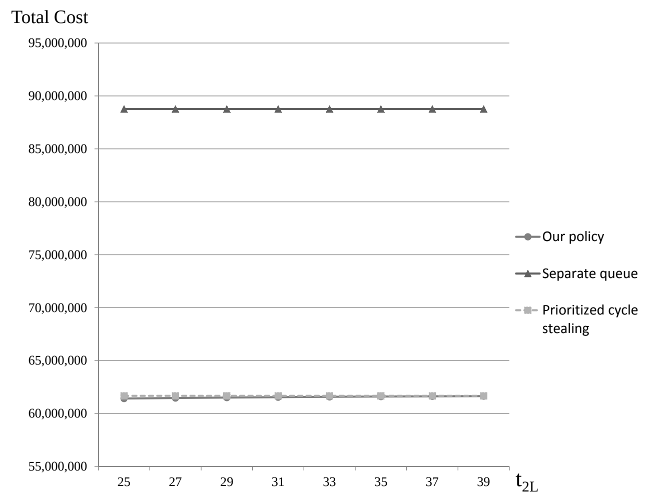


Figure B.10: Comparison of our policy and naïve policies when $t_{1L} = 35$, $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.6$, $\rho_{2H} = 1.1$, $C_{1L} = C_{1H} = 400$k, $C_{2L} = 250$k, $C_{2H} = 200$k, $C_L = 7,000$k, $C_H = 10,000$k, $n_L = 3$, and $n_H = 2$.

B.3 What Mix of Agents to Hire

One of our goals is to determine an appropriate static agent base. The objective is

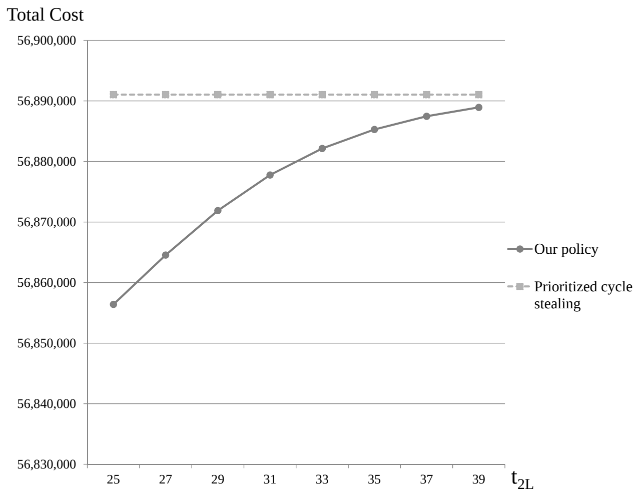


Figure B.11: Comparison of our policy and the prioritized cycle stealing policy when $t_{1L} = 35$, $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.4$, $\rho_{2H} = 1.1$, $C_{1L} = C_{1H} = 400$k, $C_{2L} = 250$k, $C_{2H} = 200$k, $C_L = 7,000$k, $C_H = 10,000$k, $n_L = 3$, and $n_H = 2$.

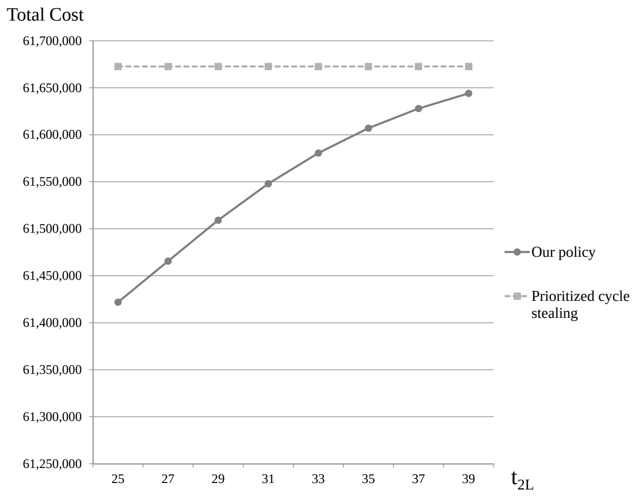


Figure B.12: Comparison of our policy and the prioritized cycle stealing policy when $t_{1L} = 35$, $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.6$, $\rho_{2H} = 1.1$, $C_{1L} = C_{1H} = 400$k, $C_{2L} = 250$k, $C_{2H} = 200$k, $C_L = 7,000$k, $C_H = 10,000$k, $n_L = 3$, and $n_H = 2$.

to minimize the total cost which is a combination of cost of agents and convex cost of expected response times. As mentioned in Chapter 3, there is a trade-off between the number of agents and the target times of requests. If the cost of an agent is high, even if hiring that agent improves expected response times of the requests, it may increase the total cost.

We generate representative unit costs for Cases (i) and (ii) and analyze whether hiring another low skilled agent is advisable as a function of t_{2L}, illustrated in Figures B.13 and B.14.

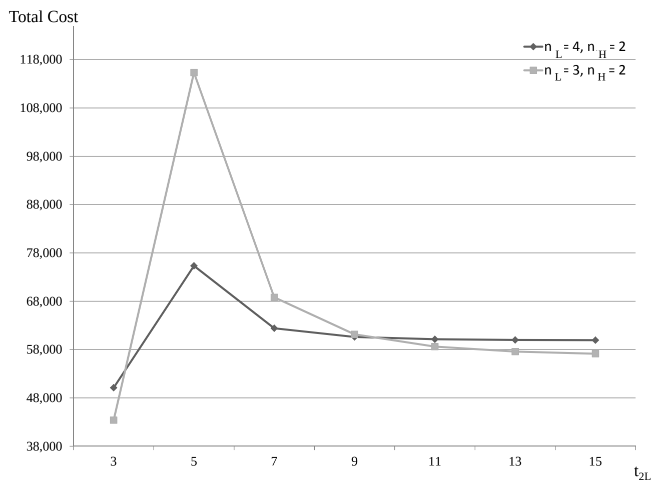


Figure B.13: Analysis of hiring a low skilled agent as a function of t_{2L} when $t_{1L} = 5$, $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.4$, $\rho_{2H} = 1.1$, $C_{1L} = C_{1H} = 400$k, $C_{2L} = 250$k, $C_{2H} = 200$k, $C_L = 7,000$k, and $C_H = 10,000$k.

Even if unit cost of 2L requests, C_{2L}, is higher than unit cost of 2H requests, C_{2H}, hiring a low skilled agent does not result in the lowest cost when $t_{2L} = 3$ as depicted in Figure B.13. Since 2L requests preempt 2H requests more often when $t_{2L} = 3$, hiring another low skilled agent is more expensive than the total cost change due to the increase in the expected response time of 2H requests and the decrease in the expected response time of 2L requests. If t_{2L} is between 5 and 9, hiring a low skilled agent results in the lowest cost. When $t_{2L} \geq 9$, hiring another low skilled agent is not necessary. In Figure B.14, we observe similar behavior of the total cost function.

However, since load of 2L requests is higher in Case (ii) in Figure B.14 than Case (i) in Figure B.13, hiring a low skilled agent has an important impact on the total cost and results in lowest cost except when $t_{2L} = 3$. Note that, given a fixed t_{2L} the total cost of Case (ii) is higher than the total cost of Case (i) since the load of 2L requests is higher.

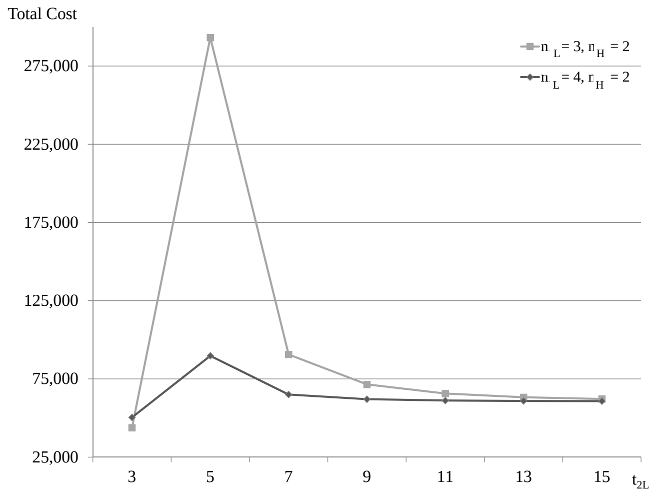


Figure B.14: Analysis of hiring a low skilled agent as a function of t_{2L} when $t_{1L} = 5$, $\rho_{1L} = 0.4$, $\rho_{1H} = 0.3$, $\rho_{2L} = 2.6$, $\rho_{2H} = 1.1$, $C_{1L} = C_{1H} = 400$k, $C_{2L} = 250$k, $C_{2H} = 200$k, $C_L = 7,000$k, and $C_H = 10,000$k.

Bibliography

I. Adan and J. Resing. Queueing theory, 2002. URL `http://www.win.tue.nl/~iadan/queueing.pdf`.

A. Aggoun and A. Vazacopoulos. Solving sports scheduling and timetabling problems with constraint programming. In S. Butenko, J. Gil-Lafuente, and P. M. Pardalos, editors, *Economics, Management and Optimization in Sports*, pages 243–264. Springer, New York, 2004.

S. R. Agnihothri, A. K. Mishra, and D. E. Simmons. Workforce cross-training decisions in field service systems with two job types. *The Journal of the Operational Research Society*, 54(4):410–418, 2003.

Z. Aksin, M. Armony, and V. Mehrota. The modern call center: A multi-disciplinary perspective on operations management research. *Production and Operations Management*, 16(6):665–668, 2007.

M. Armony. Dynamic routing in large-scale service systems with heterogenous servers. *Queueing Systems*, 51(3):287–329, 2005.

F. Babonneau, C. Beltran, A. Haurie, C. Tadonki, and J. P. Vial. Proximal-ACCPM: A versatile oracle based optimization method. In E. J. Kontoghiorghes and C. Gatu, editors, *Optimisation, Econometric and Financial Analysis*, volume 9 of *Advances in Computational Management Science*, pages 69–92. Springer, New York, 2007.

P. Baptiste, C. L. Pape, and W. Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems.* Kluwer, Dordrecht, 2001.

A. Y. Barlatta, A. M. Cohn, and O. Gusikhinc. A hybridization of mathematical programming and dominance-driven enumeration for solving shift-selection and task-sequencing problems. *Computers and Operations Research*, 37:1298–1307, 2010.

F. Baskett, M. C. K., R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22(2):248–260, 1975.

A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Mathematics of Operations Research*, 23:769–805, 1998.

A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25:1–13, 1999.

A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88:411–424, 2000.

A. Ben-Tal, L. E. Ghaoui, and A. Nemirovski. *Robust Optimization.* Princeton University Press, 2009.

J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

L. Benini, D. Bertozzi, A. Guerri, and M. Milano. Allocation and scheduling for MPSoCs via decomposition and no-good generation. In *Principles and Practice of Constraint Programming (CP 2005)*, volume 3709 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2005.

R. Bent and P. V. Hentenryck. Spatial, temporal, and hybrid decompositions for large-scale vehicle routing with time windows. In D. Cohen, editor, *Principles and Practice of Constraint Programming (CP 2010)*, volume 6308 of *Lecture Notes in Computer Science*, pages 99–113. Springer, 2010.

D. Bertsimas and D. B. Brown. Constructing uncertainty sets for robust linear optimization. *Operations Research*, 57(6):1483–1495, 2009.

D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical Programming Series B*, 98:49–71, 2003.

D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, Jan. 2004.

D. Bertsimas and A. Thiele. Robust and data-driven optimization: Modern decision-making under uncertainty. *Tutorials on Operations Research*, Chapter 4:122–195, 2006.

D. Bertsimas, D. Pachamanova, and M. Sim. Robust linear optimization under general norms. *Operations Research Letters*, 32:510–516, 2004.

D. Bertsimas, D. B. Brown, and C. Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464501, 2011.

J. R. Birge and F. Louveaux. *The robust project scheduling.* Springer, 2011.

L. Brown, N. Gans, A. Mandelbaum, A. Sakov, H. Shen, S. Zeltyn, and L. Zhao. Statistical analysis of a telephone call center: A queueing-science perspective. *Journal of the American Statistical Association*, 100:36–50, 2005.

H. Cambazard, P.-E. Hladik, A.-M. Déplanche, N. Jussien, and Y. Trinquet. Decomposition and learning for a hard real time task allocation problem. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2004.

M. T. Cezik and P. L'Ecuyer. Staffing multiskill call centers via linear programming and simulation. *Management Science*, 54(2):310–323, 2008.

Y. Chu and Q. Xia. Generating Benders cuts for a class of integer programming problems. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques* *in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 2004.

G. Codato and M. Fischetti. Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research*, 54:756–766, 2006.

E. Conde. A minmax regret approach to the critical path method with task interval times. *European Journal of Operational Research*, 197(1):235–242, August 2009.

A. I. Corréa, A. Langevin, and L. M. Rousseau. Dispatching and conflict-free routing of automated guided vehicles: A hybrid approach combining constraint programming and mixed integer programming. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 370–378. Springer, 2004.

R. S. Dembo. Scenario optimization. *Annals of Operations Research*, 30:63–80, 1991.

E. Demeulemeester and W. Herroelen. *The robust project scheduling.* now Publishers, 2009.

P. Enders, I. Adan, A. Scheller-Wolf, and G.-J. van Houtum. Inventory rationing for a system with heterogeneous customer classes. 2009.

M. M. Fazel-Zarandi and J. C. Beck. Solving a location-allocation problem with logic-based Benders' decomposition. In I. P. Gent, editor, *Principles and Practice of Constraint Programming (CP 2009)*, volume 5732 of *Lecture Notes in Computer Science*, pages 344–351. Springer, 2009.

S. French. *Sequencing and Scheduling.* John Wiley & Sons, 1982. ISBN 0470272295.

D. R. Fulkerson. Blocking and anti-blocking pairs of polyhedra. *Mathematical Programming*, 1:168–194, 1971.

N. Gans, G. Koole, and A. Mandelbaum. Telephone call centers: tutorial, review, and research prospects. *Manufacturing and Service Operations Management*, 5(2): 79–141, 2003.

O. Garnett, A. Mandelbaum, and M. I. Reiman. Designing a call center with impatient customers. *Manufacturing and Service Operations Management*, 4(3):208–227, 2002.

A. M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

L. E. Ghaoui and H. Lebret. Robust solutions to least-squares problems with uncertain data. *SIAM Journal on Matrix Analysis and Applications*, 18(4):1035–1064, Oct. 1997.

L. E. Ghaoui, F. Oustry, and H. Lebret. Robust solutions to uncertain semidefinite programs. *SIAM Journal on Optimization*, 9(1):33–52, 1998. ISSN 1052-6234.

L. Green. A queueing system with general-use and limited-use servers. *Operations Research*, 33(1):168–182, 1985.

M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, and M. S. Squillante. Cycle stealing under immediate dispatch task assignment. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 274–285, 2003a. ISBN 1-58113-661-7.

M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, and M. S. Squillante. Analysis of task assignment with cycle stealing under central queue. In *ICDCS '03:*

Proceedings of the 23rd International Conference on Distributed Computing Systems, 2003b.

M. Harchol-Balter, T. Osogami, A. Scheller-Wolf, and A. Wierman. Multi-server queueing systems with multiple priority classes. *Queueing Syst. Theory Appl.*, 51 (3-4):331–360, 2005. ISSN 0257-0130.

I. Harjunkoski and I. E. Grossmann. A decomposition approach for the scheduling of a steel plant production. *Computers and Chemical Engineering*, 25:1647–1660, 2001.

I. Harjunkoski and I. E. Grossmann. Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. *Computers and Chemical Engineering*, 26:1533–1552, 2002.

J. M. Harrison and M. J. López. Heavy traffic resource pooling in parallel-server systems. *Queueing Systems*, 33:339–368, 1999.

J. M. Harrison and A. Zeevi. Dynamic scheduling of a multiclass queue in the halfin-whitt heavy traffic regime. *Operations Research*, 52(2):243–257, 2004.

W. B. Henderson and W. L. Berry. Heuristic methods for telephone operator shift scheduling: an experimental analysis. *Management Science*, 22(12):1372–1380, 1976.

J. N. Hooker. Logic-based Benders decomposition. In *INFORMS National Meeting (INFORMS 1995)*, 1995.

J. N. Hooker. Inference duality as a basis for sensitivity analysis. In E. C. Freuder, editor, *Principles and Practice of Constraint Programming (CP 1996)*, volume 1118 of *Lecture Notes in Computer Science*, pages 224–236. Springer, 1996.

J. N. Hooker. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. Wiley, New York, 2000.

J. N. Hooker. A hybrid method for planning and scheduling. In M. Wallace, editor, *Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *Lecture Notes in Computer Science*, pages 305–316. Springer, 2004.

J. N. Hooker. Planning and scheduling to minimize tardiness. In *Principles and Practice of Constraint Programming (CP 2005)*, volume 3709 of *Lecture Notes in Computer Science*, pages 314–327. Springer, 2005a.

J. N. Hooker. A hybrid method for planning and scheduling. *Constraints*, 10:385–401, 2005b.

J. N. Hooker. An integrated method for planning and scheduling to minimize tardiness. *Constraints*, 11:139–157, 2006.

J. N. Hooker. Planning and scheduling by logic-based Benders decomposition. *Operations Research*, 55:588–602, 2007a.

J. N. Hooker. *Integrated Methods for Optimization*. Springer, 2007b.

J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

J. N. Hooker and H. Yan. Logic circuit verification by Benders decomposition. In V. Saraswat and P. V. Hentenryck, editors, *Principles and Practice of Constraint Programming: The Newport Papers*, pages 267–288, Cambridge, MA, 1995. MIT Press.

V. Jain and I. E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13:258–276, 2001.

R. G. Jeroslow. Representability in mixed integer programming, I: Characterization results. *Discrete Applied Mathematics*, 17:223–243, 1987.

E. P. C. Kao and K. S. Narayanan. Computing steady-state probabilities of a nonpreemptive priority multiserver queue. *ORSA Journal on Computing*, 2(3):211–218, 1990.

E. P. C. Kao and K. S. Narayanan. Modeling a multiprocessor system with preemptive priorities. *Management Science*, 37(2):185–197, 1991.

A. B. Keha, K. Khowala, and J. W. Fowler. Mixed integer programming formulations for single machine scheduling problems. *Computers and Industrial Engineering*, 56: 357–367, 2009.

G. Koole. Stochastic scheduling with event-based dynamic programming. *Mathematical Methods of OR*, 51(2):249–261, 2000.

C. Koulamas. The single-machine total tardiness scheduling problem: Review and extensions. *European Journal of Operational Research*, 202:1–7, 2010.

A. Mandelbaum. Call centers (centres): research bibliography with abstracts. faculty of industrial engineering and management. *Version*, 3, 2004.

A. Mandelbaum and W. A. Massey. Strong approximations for time dependent queues. *Mathematics of Operations Research*, 20(1):33–64, 1995.

C. T. Maravelias. A decomposition framework for the scheduling of single- and multistage processes. *Computers and Chemical Engineering*, 30:407–420, 2006.

C. T. Maravelias and I. E. Grossmann. Using MILP and CP for the scheduling of batch chemical processes. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization*

Problems (CPAIOR 2004), volume 3011 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2004a.

C. T. Maravelias and I. E. Grossmann. A hybrid MILP/CP decomposition approach for the continuous time scheduling of multipurpose batch plants. *Computers and Chemical Engineering*, 28:1921–1949, 2004b.

K. Natarajan, D. Pachamanova, and M. Sim. Constructing risk measures from uncertainty sets. *Operations Research*, 57(5):1129–1141, 2009.

M. F. Neuts. *Matrix-geometric solutions in stochastic models.* 1981.

T. Osogami and M. Harchol-Balter. A closed form solution for mapping general distributions to minimal ph distributions. In *In International Conference on Performance Tools TOOLS 2003*, pages 200–217. Springer, LNCS, 2003.

T. Osogami, M. Harchol-Balter, and A. Scheller-Wolf. Robustness and performance of threshold-based resource allocation policies. working paper, 2005a.

T. Osogami, M. Harchol-Balter, and A. Scheller-Wolf. Analysis of cycle stealing with switching times and thresholds. *Performance Evaluation*, 61(4):347–369, 2005b.

M. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

M. T. Pich, C. H. Loch, and A. D. Meyer. On uncertainty, ambiguity, and complexity in project management. *Management Science*, 48(8):1008–1023, 2002.

M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, 1995. ISBN 0-13-706757-7.

A. Pot, S. Bhulai, and G. Koole. A simple staffing method for multiskill call centers. *Manufacturing and Service Operations Management*, 10(3):421–428, 2008.

R. Rasmussen. Scheduling a triple round robin tournament for the best Danish soccer league. *European Journal of Operational Research*, 185:795–810, 2008.

R. Rasmussen and M. A. Trick. A Benders approach to the constrained minimum break problem. *European Journal of Operational Research*, 177:198–213, 2007.

M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queuing networks. *J. ACM*, 27(2):313–322, 1980.

R. Sadykov and L. A. Wolsey. Integer programming and constraint programming in solving a multimachine assignment scheduling problem with deadlines and release dates. *INFORMS Journal on Computing*, 18:209–217, 2006.

B. L. Schwartz. Queuing models with lane selection: A new class of problems. *Operations Research*, 22(2):331–339, 1974.

M. Segal. The operator-scheduling problem: A network-flow approach. *Operations Research*, 22(4):808–823, 1974.

R. A. Shumsky. Approximation and analysis of a call center with flexible and specialized servers. *OR Spectrum*, 26(3):307–330, 2004.

A. L. Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(5):1154–1157, 1973.

D. A. Stanford and W. K. Grassmann. The bilingual server system: A queueing model featuring fully and partially qualified servers. *INFOR*, 31(4):261–277, 1993.

D. A. Stanford and W. K. Grassmann. *Bilingual server call centers.* Analysis of Communication Networks: Call Centers, Traffic and Performance, American Mathematical Society, 2000.

S. A. Tarim and I. Miguel. A hybrid Benders' decomposition method for solving stochastic constraint programs with linear recourse. In B. Hnich, M. Carlsson, F. Fages, and F. Rossi, editors, *Recent Advances in Constraints (CSCLP 2005)*, volume 3978 of *Lecture Notes in Computer Science*, pages 133–148. Springer, 2006.

D. Terekhov, J. C. Beck, and K. N. Brown. Solving a stochastic queueing design and control problem with constraint programming. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI 2005)*, pages 261–266, 2005.

E. Thorsteinsson. Branch and check: A hybrid framework integrating mixed integer programming and constraint logic programming. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP 2001)*, volume 2239 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2001.

C. Timpe. Solving planning and scheduling problems with combined integer and constraint programming. *OR Spectrum*, 24:431–448, 2002.

R. B. Wallace and W. Whitt. A staffing algorithm for call centers with skill-based routing. *Manufacturing and Operations Management*, 7(4):276–294, 2005.

W. Whitt. Improving service by informing customers about anticipated delays. *Management Science*, 45:192–207, 1999.

W. Whitt. Sensitivity of performance in the erlang-a queueing model to changes in the model parameters. *Operations Research*, 54:247–260, 2002.

H. Yaman, O. E. Karaşan, and M. c. Pınar. Restricted robust uniform matroid maximization under interval uncertainty. *Mathematical Programming*, 110(2):431–441, 2007.

J. W. Yen and J. R. Birge. A stochastic programming approach to the airline crew scheduling problem. *Transportation Science*, 40(1):3–14, Feb. 2006.