Scheduling Multiclass Queueing Networks and Job Shops using Fluid and Semidefinite Relaxations

by

Jayachandran Sethuraman

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1999

Author .....
Department of Electrical Engineering and Computer Science
August 9, 1999

Certified by .....
Dimitris J. Bertsimas
Boeing Leaders For Manufacturing Professor of Operations Research
Thesis Supervisor

Accepted by .....
James B. Orlin
Co-director, Operations Research Center

Scheduling Multiclass Queueing Networks and Job Shops using Fluid and Semidefinite Relaxations

by

Jayachandran Sethuraman

Submitted to the Department of Electrical Engineering and Computer Science
on August 9, 1999, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

Queueing networks serve as useful models for a variety of problems arising in modern communications, computer, and manufacturing systems. Since the optimal control problem for queueing networks is well-known to be intractable, an important theme of research during the last two decades has been the development of tractable approximations, and the use of these approximations to determine optimal controls. *Fluid relaxations* are an important class of such approximations that have received much attention in recent years. The central aim of this dissertation is to demonstrate the usefulness of fluid relaxations in solving a variety of scheduling problems.

In the first part of this dissertation, we explore the role of fluid relaxations in solving traditional job shop problems. For the job shop problem with the objective of minimizing makespan, we construct a schedule that is guaranteed to be within a constant of the optimal. In particular, our schedule is asymptotically optimal. We prove an analogous asymptotic-optimality result for the objective of minimizing holding costs. For both objectives, we report impressive computational results on benchmark instances chosen from the OR library. Our algorithms use fluid relaxations in two different ways: first, the optimal fluid cost is used as a lower bound for the original problem; and second, a discrete schedule is constructed by appropriately rounding an optimal fluid solution.

In the second part of this dissertation we study the optimal control problem for multiclass queueing networks in steady-state. A key difficulty is that the fluid relaxation, being transient in nature, does not readily yield a lower bound for the steady-state problem. For this reason, we use a class of lower bounds, based on semidefinite relaxations, first proposed by Bertsimas and Niño-Mora. Interestingly, one of the shortcomings of these relaxations is that there is no natural way to derive policies based on them. Thus, while our lower bounds are based on semidefinite relaxations, our policies are based on a heuristic interpretation of optimal fluid solutions. We consider objective functions involving both first and second moments of queue-lengths (equivalently, delays). Our computational results show the effectiveness of this approach in obtaining good policies for multiclass queueing networks.

Finally, we turn our attention to the problem of computing the optimal fluid solution efficiently. We develop a general method to solve the optimal control problem for fluid tandem networks, and identify polynomially solvable special cases.

Thesis Supervisor: Dimitris J. Bertsimas
Title: Boeing Leaders For Manufacturing Professor of Operations Research

Acknowledgments

I thank my advisor, Dimitris Bertsimas, for supervising my dissertation with his inimitable care, enthusiasm, and patience; for being a rich source of problems and ideas; and for being generous with his time. I am constantly amazed at the breadth of his knowledge, and the unique perspective he brings even to the most mundane of topics. I thank John Tsitsiklis and Andreas Schulz for serving on my thesis committee. Their numerous suggestions and advice led to substantial improvements in both form and content.

During my years at MIT, I had the good fortune to be associated with many wonderful people. In particular, I thank: Florin Avram, with whom most of the results in Chapter 5 were obtained; David Gamarnik, for working with me on an earlier version of Chapter 3, and for several discussions on fluid models; Anant Balakrishnan and Nitin Patel, for their attempts to teach me *O.R. practice*, and for supporting me during my initial years; Tom Magnanti, Rob Freund, and Ricky Vohra, for many helpful discussions and advice; Paulette Mosley, Laura Rose, and Danielle Bonaventura, for shielding me from the MIT bureaucracy; Anand Arunachalam and Wilfred Gomes, for their help and friendship; and all my friends at the OR Center for making my stay at MIT thoroughly enjoyable. In addition, my thanks to Costis Maglaras, for informing me of his results, and for discussing and clarifying the details patiently during the last year; and Karl Sigman, for some insightful observations during my visit to Columbia.

My summer internships at IBM T. J. Watson Research Center were wonderful opportunities to keep in touch with my interests in Computer Science. I am grateful to Mark Squillante for hosting my IBM visits, and for introducing me to a variety of interesting research questions. His enthusiasm, kindness, and sense of humor made my visits to Watson truly memorable. I also thank Mark (and IBM Research) for supporting me with a fellowship during my final year.

Chung-Piaw Teo has the rare (and enviable) gift of making you work at a higher level than you think you are capable of, and I consider myself very fortunate to have known and to have worked with him. In the spring of 1996, I got a glimpse of the Book when Chung-Piaw showed me *a truly marvelous proof* of a stable matching result, which, fortunately, was *short enough to fit into the margin.* This was the beginning of many exciting discussions, which are among the most treasured moments of my MIT life. I am grateful to him for sharing his thoughts on many research problems, for teaching me most of what I know about discrete optimization, and for his thoughtful responses to many of my questions.

Finally, I thank my wife, Viji, for her love, inspiration, and friendship; and our parents, whose sacrifices have enabled us to live our dreams. This work is dedicated to them with affection and respect.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The theory of queueing networks is presently undergoing a period of intensive research, motivated by the need to understand and control the behavior of modern communications, computer and manufacturing systems. Recognizing the importance and the inherent intractability of optimal control of queueing networks, the research community has focused its attention, for the most part, on developing tractable approximations. An important theme that has emerged during the last two decades is the study of approximations to queueing network models via various natural space and time scalings, and the use of these approximations to determine optimal controls. *Fluid relaxations* are an important class of such approximations that have received much attention in recent years. The central aim of this dissertation is to demonstrate the usefulness of fluid relaxations in solving a variety of scheduling problems.

1.1 Preliminaries

Multiclass queueing networks. In a multiclass queueing network, we are given a set of machines to process job classes that may differ in their arrival processes, service requirements, routes through the network, cost per unit of time spent in the system, etc. Processing times are assumed to be stochastic with known distributions. The fundamental optimization problem that arises in these networks is to determine a scheduling policy so as to optimize a specified performance objective. Optimal control of such networks typically involve *sequencing* decisions: a sequencing policy specifies the job-type to be served at each machine at each point in time. Extensions to this basic model involve *routing* and *input*

control: a *routing* policy specifies the route of each job, whereas an *input control* policy specifies when to admit jobs into the network. Some common objectives include minimizing steady-state average holding costs (equivalently, minimizing a linear combination of expected sojourn times of each customer class), maximizing throughput, etc.

The optimal control problem for multiclass queueing networks is extremely difficult; problems of realistic size, with very few exceptions, cannot be solved exactly in reasonable time. In fact, Papadimitriou and Tsitsiklis [59] have shown that certain natural versions of this problem are provably intractable: even if $\mathcal{P} = \mathcal{NP}$, this class of problems does not have efficient algorithms.

Job Shops. Job shop scheduling problems arise in a variety of settings, especially in a factory or assembly line. A job shop has J heterogeneous machines, and N jobs. Each job consists of an ordered set of *tasks*, each of which must be processed on a specified machine. (Different tasks of the same job may be processed on the same machine.) The processing time of each task (at its associated machine) is deterministic and known. The goal is to schedule all the tasks such that (i) no more than one task is assigned to a machine at any point in time, (ii) no two tasks belonging to the same job are scheduled simultaneously, and (iii) a pre-specified objective function is optimized. Some typical objectives are as follows:

- Minimize the *makespan* of the schedule, which is defined as the completion time of the last job.
- Minimize the *weighted completion time*, $\sum w_i C_i$, where w_i is a (non-negative) weight associated with job i, and C_i is its completion time.

There has been a large literature on job shop scheduling problems over the last forty years (see Lawler et. al. [47]). Not surprisingly, all but the most restrictive versions of the job shop problem are $\mathcal{NP}$-hard. In spite of all the attention, surprisingly little was known about approximation algorithms for job shop scheduling problems until recently.

Discussion. Optimal control of multiclass queueing networks shares a number of features with the job shop problem. The intractability of the job shop problem can be traced to two of its features: it is both *discrete* and *dynamic*. While sharing these features with the job shop problem, the optimal control problem for multiclass queueing networks has

an additional complicating feature — *stochasticity.* In the last decade, many researchers have adopted the philosophical view that the most important components of the optimal control problem are the discrete and dynamic nature of the problem. This view relies on the assumption that while stochasticity plays an important role, its effect is primarily of "second-order," and that the structure of an optimal policy is essentially revealed by a deterministic, dynamic model whose arrival and service rates are the mean rates for the stochastic network. This deterministic approximation to multiclass queueing networks leads naturally to a *job shop*-like problem with deterministic arrivals, which is a hard problem in its own right. A (relatively) tractable approximation can be obtained by further relaxing the *discrete* nature of the jobs: i.e., by treating jobs as a *continuous fluid*, and by allowing a machine to work simultaneously on multiple job-types. Such a relaxation is termed a *fluid relaxation*, and has been the focus of much attention during the last decade. Inspired by this approach for the optimal control of multiclass queueing networks, we explore, in the first part of this dissertation, the role played by fluid relaxations in solving traditional job shop problems. We emphasize that fluid relaxations are used in two different ways: first, the optimal fluid cost provides a lower bound for the discrete problem; and second, appropriately rounding the optimal fluid solution results in a near-optimal schedule for the original job shop problem.

In the second part of the dissertation we study the optimal control problem for multiclass queueing networks in steady-state. A key difficulty is that the fluid relaxation, being transient in nature, does not readily yield a lower bound for the steady-state problem. Thus, we are led naturally to consider alternative methods for deriving good lower bounds. To this end, we study the effectiveness of semidefinite relaxations, first proposed by Bertsimas and Niño-Mora [12]. Interestingly, one of the shortcomings of the semidefinite relaxations is that there is no natural way to derive policies based on them. Thus, we use semidefinite relaxations to compute lower bounds, and derive policies based on a heuristic interpretation of optimal fluid solutions.

In the final part of the dissertation we turn our attention to the problem of computing the optimal fluid solution. We develop a general method to solve the optimal control problem for fluid tandem networks, and identify special cases that can be solved in time polynomial in the number of machines.

1.2 Contributions

The main contributions of this dissertation are as follows:

1. **Minimizing makespan in job shops:** We provide an efficient algorithm to round an optimal fluid solution such that the resulting schedule is asymptotically optimal. In fact, the schedule obtained is within an additive constant of a trivial lower bound. Our results also imply that the fluid relaxation is asymptotically exact. We extend our results to a model in which deterministic arrivals occur over a finite horizon. Our algorithm produces extremely good schedules in practice, as demonstrated by computational results on benchmarks chosen from the OR library.

2. **Minimizing holding costs in job shops:** We provide an efficient algorithm to round an optimal fluid solution such that the resulting schedule is asymptotically optimal. In fact we show that the error incurred by our solution, relative to the optimal fluid cost, is $O(1/N)$, where N, the multiplicity of the job types, goes to ∞. Our results also imply that the fluid relaxation is asymptotically exact. We extend our results to a (suitably restricted) model in which deterministic arrivals occur over a finite horizon. Essentially, we show that our results hold as long as the horizon is in the "fluid regime." Computational results on benchmarks from the OR library show that our algorithm produces good schedules in practice. These results can be viewed as a first step toward solving similar problems in a stochastic, dynamic setting (multiclass queueing network model).

3. **Steady-state control of multiclass queueing networks:** We study the effectiveness of the *achievable region* approach in addressing a variety of problems arising in performance evaluation and optimization of multiclass queueing networks. We do this by carrying out an extensive computational investigation of a class of lower bounds, developed by Bertsimas and Niño-Mora [12], for a variety of models including open networks, closed networks, routing etc. For all of these problems we derive policies based on a heuristic interpretation of optimal solutions to the associated fluid relaxation.

4. **Optimal control of fluid tandem networks:** We provide a polynomial time algorithm for the problem of optimal control of fluid tandem networks in which the

holding costs are nondecreasing along the route. Our approach is *algebraic* and does not use time-discretization explicitly: We show that an optimal control for a fluid tandem network with non-decreasing holding costs can be computed by solving a *single* convex programming problem. Further, we show that this convex program can be solved efficiently as a shortest-path problem on a graph.

1.3 Dissertation Outline

The rest of this dissertation is organized as follows. In Chapters 2 and 3 we describe algorithms to compute asymptotically optimal solutions to deterministic job shop problems; In Chapter 2, we consider the makespan objective, while in Chapter 3 we consider the holding costs objective. Chapter 4 discusses the use of fluid relaxations in steady-state control of queueing networks; here, we investigate the power of semidefinite relaxations to compute lower bounds, and compare these lower bounds to the costs of fluid-based policies whenever applicable. In Chapter 5 we discuss the complexity of computing an optimal solution to the fluid tandem network. We end with a summary of our results, followed by a brief discussion of open problems.

Chapter 2

Asymptotically Optimal Solutions for Minimizing Makespan in Job Shops

2.1 Introduction

The job shop scheduling problem is a fundamental problem in Operations Research and Computer Science, and has been studied extensively from a variety of perspectives. The job shop scheduling problem is a well-known difficult problem, and is $\mathcal{NP}$-hard. For this reason substantial amount of research has been devoted to finding approximately optimal solutions for job shop problems. There are several ways to measure the degree of sub-optimality of a given solution. A fairly standard measure is the *ratio* of the cost of the given solution to that of an (easily computable) lower bound on the cost. In this chapter, we take a slightly different point of view: the degree of sub-optimality of a given solution S is defined as the extra cost incurred by S compared to an easily computable lower bound. Using a *fluid relaxation*, we describe an algorithm to compute near-optimal schedules efficiently, and show that the extra cost incurred can be bounded above by a constant, independent of the number of jobs.

The job shop scheduling problem is the the problem of scheduling a set of I job types on J machines. Job type i consists of J_i stages (also referred to as "tasks"), each of which must be completed on a particular machine. The pair (i, k) represents the k^{th} stage of

the i^{th} job type, and has processing time $p_{i,k}$. Suppose that we have n_i jobs of type i. Our objective is to find a schedule that minimizes the makespan, which is defined as the maximum completion time of the jobs; in the standard scheduling notation, this problem is denoted as $J||C_{\max}$.

We impose the following restrictions on the schedule.

1. The schedule must be non-preemptive. That is, once a machine begins processing a stage of a job, it must complete that stage before doing anything else.

2. Each machine may work on at most one task at any given time.

3. For $k > 1$, stage k of a job can begin only after the completion of its $(k-1)^{\text{st}}$ stage.

The classical job shop scheduling problem involves exactly one job from each type, i.e., the initial vector of job types is $(1, 1, \ldots, 1)$. The job shop scheduling problem is notoriously difficult to solve exactly, even if the sizes of the instances are relatively small. As an example, a specific instance involving 10 machines and 10 jobs posed in a book by Muth and Thompson [57] in 1963 remained unsolved for over 20 years until solved by Carlier and Pinson [18] in 1985.

Our overall approach for these problems draws on on ideas from two distinct communities, and is inspired by the recent paper of Bertsimas and Gamarnik [8]. First, we consider a relaxation for the job shop scheduling problem called the fluid relaxation, in which we replace discrete jobs with the flow of a continuous fluid. The motivation for this approach comes from optimal control of multiclass queueing networks, which are stochastic and dynamic generalizations of job shops. For the makespan objective, the optimal solution of the fluid control problem can be computed in closed form and provides a lower bound $C_{\max}$ to the job shop scheduling problem; see Weiss [73], Bertsimas and Gamarnik [8].

Our second idea is motivated by the literature on *fair queueing*, which addresses the question of emulating a *given* head-of-the-line processor sharing discipline without preempting jobs. Processor sharing disciplines were originally proposed as an idealization of time-sharing in computer systems. In a time-sharing discipline, the processor cycles through the jobs, giving each job a small quantum of service; processor-sharing is the discipline obtained as the quantum length approaches zero. Processor sharing disciplines are attractive from the point of view of congestion control in large scale networks because of their inherent fairness. For this reason, the question of emulating a given processor sharing discipline using a

non-preemptive discipline (while retaining its attractive features) has received a lot of attention in the flow control literature; several simple and elegant schemes, classified under the generic name of *fair queueing,* have been proposed (see Demers, Keshav and Shenker [26], Greenberg and Madras [34], Parekh and Gallager [60, 61]). Of particular relevance to our work is a fair queueing discipline called fair queueing based on start times (FQS). Under this discipline, whenever a scheduling decision is to be made, the job selected is the one that *starts earliest* in the underlying processor sharing discipline. A comprehensive review of related work appears in the survey of Zhang [76].

Our algorithm can be viewed as a natural outcome of combining these two ideas. We use the fluid relaxation to compute the underlying processor sharing discipline (i.e., the rate at which the machines work on various job classes), and then draw on ideas from the fair queueing literature. An important difficulty that must be addressed is that our fluid relaxation approximates jobs by a "continuous fluid," whereas in reality, jobs are "discrete entities." This necessitates a more careful definition of "start times," while attempting to use a discipline like FQS.

In recent years, considerable progress has been made in the deterministic scheduling community in providing approximation algorithms for scheduling problems based on linear programming relaxations. In this framework, a natural linear programming relaxation of the scheduling problem is solved first, which results in LP start/completion times for each job. A typical heuristic is to then schedule the jobs in the order of their LP start times or LP completion times. For a review of this approach, we refer the readers to the papers by Hall [36], Karger, Stein, and Wein [43], Hall, Schulz, Shmoys, and Wein [37], and the references therein. As we shall see, our scheduling algorithm can be viewed as a generalization of this idea to a dynamic setting in which the "LP start times" are computed on-the-fly. In other words, the "LP start times" at time t are computed based on both the continuous relaxation, and all of the jobs that have been scheduled prior to t.

Results. We describe an efficient algorithm to round an optimal fluid solution such that the resulting schedule is asymptotically optimal. Essentially, we show that rounding an optimal fluid solution appropriately results in a schedule that incurs $O(1)$ extra cost compared to a trivial lower bound for the job shop problem; this trivial lower bound coincides with the optimal cost of the fluid relaxation. To put our result in perspective, consider

the classical job shop scheduling problem, which has exactly one job of each type. In this case, the combinatorial structure of the job scheduling problem makes the problem very complicated to solve. Interestingly, the results of this chapter imply that as the number of jobs increases, the combinatorial structure of the problem is increasingly less important, and as a result, a fluid approximation of the problem becomes increasingly exact. The results of this chapter also imply that a continuous approximation to the job shop problem is asymptotically exact.

Related work. As we discussed earlier there is an extensive literature on both job shop problems, and fluid relaxations. We now provide a very brief overview of some of these results, which will also serve to place our results in perspective.

In spite of enormous attention, very little was known about approximation algorithms for job shops until recently. Gonzalez and Sahni [33] proved the following obvious result: any algorithm in which at least one machine is operating at any point in time is within a factor of J of the optimal. Interesting approximation algorithms for shop scheduling problems appeared in the Soviet literature in the mid seventies: these were based on geometric arguments, and were discovered independently by Belov and Stolin [6], Sevast'yanov [66], and Fiala [29]. The results based on this approach are in the spirit of our results: typically, these approaches produced schedules for which the length could be bounded by $C_{\max}+O(1)$. For example, for the job shop problem, Sevast'yanov [67, 68] proposed a polynomial-time algorithm that delivers a schedule with additive error at most $O(JJ_{\max}^3 P_{\max})$, where J is the number of machines, $J_{\max}$ is the maximum number of stages of any job-type, and $P_{\max}$ is the maximum processing time over all tasks. Our algorithm, while delivering a schedule with an $O(1)$ additive error, has two distinct advantages: (i) the error bound is substantially better, and (ii) our algorithm is substantially simpler to implement. Leighton, Maggs, and Rao [48], motivated by a packet routing application, considered a restricted version of the job shop problem in which all of the processing times are identically equal to 1. (The job shop problem remains strongly $\mathcal{NP}$-hard even under this restriction.) Leighton, Maggs and Rao [48] showed the existence of a schedule with length $O(C_{\max} + P_{\max})$. Unfortunately, their result was not algorithmic, as it relied on a non-constructive probabilistic argument based on the Lovász Local Lemma; they also discovered a randomized algorithm that delivers a schedule of length at most $O(C_{\max} + P_{\max} \log N)$ with high probability, where N

is the number of jobs to be scheduled. Motivated by these results, Shmoys, Stein and Wein [70] described a polynomial-time randomized algorithm for job shop scheduling that, with high probability, yields a schedule of length $O(\frac{\log^2(JP_{\max})}{\log\log(JP_{\max})}C_{\max})$. More importantly, they describe a $(2+\epsilon)$-approximation algorithm when J and $P_{\max}$ are constants (as is the case in our setting). An interesting recent development is the discovery of polynomial time approximation schemes for the job shop scheduling problem; this result was discovered by Jansen, Solis-Oba, and Sviridenko [41, 40]. While all of these algorithms serve an important role — that of classifying these problems in the complexity hierarchy according to the ease of their approximability — none of these algorithms is practical. In sharp contrast, the algorithm proposed in this chapter is accompanied by a guarantee of a small additive error, is easy to implement, and practical, as demonstrated by extensive computational results.

Fluid relaxations have also been the subject of intensive research during the last decade. An important breakthrough was achieved by Dai [22], who showed that global stability analysis of multiclass queueing networks is possible by focusing on their deterministic counterparts; see also the papers by Chen [19], Chen and Mandelbaum [20], Dai and Meyn [23], Dai and Weiss [24] and Rybko and Stolyar [65]. Spurred by the success of these ideas in analyzing stability, there has been a growing literature in finding near-optimal scheduling policies using fluid relaxations; since most of this literature is more relevant to the work described in chapter 3, we defer its discussion until then. As mentioned earlier, the fluid relaxation associated with the makespan objective can be solved in closed form; this is discussed by Weiss [73] (see also Bertsimas and Gamarnik [8]).

Of particular relevance are two recent papers that use fluid relaxations to address the job shop problem with the makespan objective. Bertsimas and Gamarnik [8] address the same problem, and describe an algorithm to compute a schedule of length $C_{\max}+O(\sqrt{C_{\max}})$. Dai and Weiss [25] consider the same problem, but assume (generally distributed) stochastic processing times. They describe an algorithm to compute a near-optimal schedule with high probability.

Structure of the chapter. In §2.2, we consider the makespan objective, and describe an algorithm that provides an asymptotically optimal schedule. These results are extended in §2.3 to a model in which deterministic arrivals occur over a finite horizon. In §2.4, we present computational results on a variety of job shop instances from the OR library. §2.5

contains some concluding remarks.

2.2 Makespan

This section considers the job shop problem with the objective of minimizing makespan, and is structured as follows: In §2.2.1, we define the job shop scheduling problem formally, and discuss the notation. In §2.2.2, we describe the fluid relaxation for the job shop scheduling problem. and discuss its solution; this section is reproduced from Bertsimas and Gamarnik [8] and is included here primarily for the sake of completeness. In §2.2.3, we provide an algorithm, called the *fluid synchronization algorithm* (FSA), and prove that FSA yields a schedule that is asymptotically optimal.

2.2.1 Problem Formulation and Notation

In the job shop scheduling problem there are J machines $\sigma_1, \sigma_2, \ldots, \sigma_J$ which process I different types of jobs. Each job type is specified by the sequence of machines to be processed on, and the processing time on each machine. In particular, jobs of type i, $i = 1, 2, \ldots, I$ are processed on machines $\sigma_1^i, \sigma_2^i, \ldots, \sigma_{J_i}^i$ in that order, where $1 \leq J_i \leq J_{\max}$. The time to process a type i job on machine σ_k^i is denoted by $p_{i,k}$. Throughout, we assume that $p_{i,k}$ are integers.

The jobs of type i that have been processed on machines $\sigma_1^i, \ldots, \sigma_{k-1}^i$ but not on machine σ_k^i, are queued at machine σ_k^i and are called "type i jobs in stage k" or "class (i,k)" jobs. We will also think of each machine σ_j as a collection of all type and stage pairs that it processes. Namely, for each $j = 1, 2, \ldots, J$

$$\sigma_j = \{(i,k) : \sigma_j = \sigma_k^i,\ 1 \leq i \leq I,\ 1 \leq k \leq J\}.$$

There are n_i jobs for each type i initially present at their corresponding first stage. Our objective is to minimize the makespan, i.e., to process all the $n_1 + n_2 + \cdots + n_I$ jobs on machines $\sigma_1, \ldots, \sigma_J$, so that the time taken to process all the jobs is minimized.

Machine σ_j requires a certain processing time to process jobs that eventually come to it, which is

$$C_j = \sum_{(i,k) \in \sigma_j} p_{i,k} n_i.$$

The quantity C_j is called the congestion of machine σ_j. We denote the maximum congestion by

$$C_{\max} \equiv \max_{j=1,\ldots,J} C_j.$$

The following proposition is immediate.

Proposition 2.1 *The minimum makespan C^* of the job shop scheduling problem satisfies:*

$$C^* \geq C_{\max}.$$

We define a few other useful quantities. For machine σ_j, let

$$U_j = \sum_{(i,k)\in\sigma_j} p_{i,k},$$

and

$$P_j = \max_{(i,k)\in\sigma_j} p_{i,k}. \tag{2.1}$$

Namely, U_j is the workload of machine σ_j when only one job per type is present, and P_j is the maximum processing time at σ_j. Finally, let

$$U_{\max} = \max_{1\leq j\leq J} U_j, \tag{2.2}$$

and

$$P_{\max} = \max_{1\leq j\leq J} P_j. \tag{2.3}$$

In the next section we consider a fluid (fractional) version of this problem, in which the number of jobs n_i of type i can take arbitrary positive real values, and machines are allowed to work simultaneously on several types of jobs (the formal description of the fluid job shop scheduling problem is provided in §2.2.2). For the fluid relaxation, we show that a simple algorithm leads to a makespan equal to $C_{\max}$, and is, therefore, optimal.

2.2.2 The Fluid Job Shop Scheduling Problem

In this section we describe a fluid version of the job shop scheduling problem. The input data for the fluid job shop scheduling problem is the same as for the original problem. There are J machines $\sigma_1, \sigma_2, \ldots, \sigma_J$, I job types, each specified by the sequence of machines σ_k^i,

$k = 1, 2, \ldots, J_i$, $J_i \leq J$ and the sequence of processing times $p_{i,k}$ for type i jobs in stage k. We introduce the notation $\mu_{i,k} = 1/p_{i,k}$ that represents the rate of machine σ_k^i on a type i job. The number of type i jobs initially present, denoted by x_i, takes nonnegative real values.

In order to specify the fluid relaxation we introduce some notation. We let $x_{i,k}(t)$ be the total (fractional in general) number of type i jobs in stage k at time t. We call this quantity the fluid level of type i in stage k at time t. We denote by $T_{i,k}(t)$ the total time the machine σ_k^i works on type i jobs in stage k during the time interval $[0, t)$. Finally $1\{A\}$ denotes the indicator function for the set A.

The fluid relaxation associated with the problem of minimizing makespan can be formulated as follows:

$$\text{minimize} \quad \int_0^\infty 1\left\{\sum_{1\leq i\leq I, 1\leq k\leq J} x_{i,k}(t) > 0\right\} dt \tag{2.4}$$

$$\text{subject to} \quad x_{i,1}(t) = x_i - \mu_{i,1} T_{i,1}(t), \qquad i = 1, 2, \ldots, I, \ t \geq 0, \tag{2.5}$$

$$x_{i,k}(t) = \mu_{i,k-1} T_{i,k-1}(t) - \mu_{i,k} T_{i,k}(t), \quad k = 2, \ldots, J_i, \ i = 1, 2, \ldots, I, \ t \geq 0, \tag{2.6}$$

$$0 \leq \sum_{(i,k)\in\sigma_j} (T_{i,k}(t_2) - T_{i,k}(t_1)) \leq t_2 - t_1, \ \forall\, t_2 > t_1, \ t_1, t_2 \geq 0, \ j = 1, 2, \ldots, J, \tag{2.7}$$

$$x_{i,k}(t) \geq 0, \quad T_{i,k}(t) \geq 0. \tag{2.8}$$

The objective function (2.4) represents the total time that at least one of the fluid levels is positive. It corresponds to the minimum makespan schedule in the discrete problem. Equations (2.5) and (2.6) represent the dynamics of the system. The fluid level of type i in stage k at time t is the initial number of type i jobs in stage k (x_i for $k = 1$, zero for $k > 1$) plus the number of type i jobs processed in stage $k-1$ during $[0, t)$ (given by $\mu_{i,k-1} T_{i,k-1}(t)$), minus the number of type i jobs processed in stage k during $[0, t)$ (given by $\mu_{i,k} T_{i,k}(t)$). Constraint (2.7) is just the aggregate feasibility constraint for machine σ_j.

Similar to the definition for the discrete problem, we define congestion in machine σ_j as

$$C_j = \sum_{(i,k)\in\sigma_j} p_{i,k} x_i, \tag{2.9}$$

and the maximal congestion as

$$C_{\max} = \max_{1\leq j\leq J} C_j. \tag{2.10}$$

We next show that the fluid relaxation can be solved in closed form.

Proposition 2.2 *The fluid relaxation (2.4) has an optimal value equal to the maximum congestion* $C_{\max}$.

Proof: We first show that the maximum congestion $C_{\max}$ is a lower bound on the optimal value of the fluid relaxation. For any positive time t and for each $i \leq I$, $k \leq J_i$, we have from (2.5), (2.6):

$$\sum_{l=1}^{k} x_{i,l}(t) = x_i - \mu_{i,k} T_{i,k}(t).$$

For each machine σ_j we obtain:

$$\sum_{(i,k)\in\sigma_j} p_{i,k} \sum_{l=1}^{k} x_{i,l}(t) = \sum_{(i,k)\in\sigma_j} p_{i,k} x_i - \sum_{(i,k)\in\sigma_j} T_{i,k}(t) \geq C_j - t,$$

where the last inequality follows from the definition of C_j and Constraint (2.7) applied to $t_1 = 0$, $t_2 = t$. It follows then, that the fluid levels are positive for all times t smaller than C_j. Therefore, the objective value of the fluid relaxation is at least $\max_j C_j = C_{\max}$.

We now construct a feasible solution that achieves this value. For each $i \leq I$, $k \leq J_i$ and each $t \leq C_{\max}$ we let

$$\begin{aligned}
T_{i,k}(t) &= \frac{p_{i,k} x_i}{C_{\max}} t, \\
x_{i,1}(t) &= x_i - \mu_{i,1} T_{i,1}(t) = x_i - \frac{x_i}{C_{\max}} t, \quad i = 1, \ldots, I, \\
x_{i,k}(t) &= 0, \qquad\qquad\qquad\qquad\qquad\qquad k = 2, 3, \ldots, J_i, \; i = 1, \ldots, I.
\end{aligned}$$

For all $t > C_{\max}$ we set $T_{i,k}(t) = p_{i,k} x_i$, $x_{i,k}(t) = 0$. Clearly, this solution has an objective value equal to $C_{\max}$. We now show that this solution is feasible. It is nonnegative by construction. Also by construction, Eq. (2.5) is satisfied for all $t \leq C_{\max}$. In particular, $x_{i,1}(C_{\max}) = 0$, $i = 1, 2, \ldots, I$. Moreover, for all i, $k = 2, 3, \ldots, J_i$ and $t \leq C_{\max}$ we have:

$$\mu_{i,k-1} T_{i,k-1}(t) - \mu_{i,k} T_{i,k}(t) = p_{i,k-1}^{-1} T_{i,k-1}(t) - p_{i,k}^{-1} T_{i,k}(t) = \frac{x_i}{C_{\max}} t - \frac{x_i}{C_{\max}} t = 0 = x_{i,k}(t),$$

and Eq. (2.6) is satisfied. Finally, for any $t_1 < t_2 \leq C_{\max}$ and for any machine σ_j, we have:

$$\sum_{(i,k)\in\sigma_j} (T_{i,k}(t_2) - T_{i,k}(t_1)) = \sum_{(i,k)\in\sigma_j} \left(\frac{p_{i,k} x_i}{C_{\max}} t_2 - \frac{p_{i,k} x_i}{C_{\max}} t_1 \right) = \frac{C_j}{C_{\max}} (t_2 - t_1) \leq t_2 - t_1,$$

and Constraint (2.7) is satisfied. Note, that for the constructed solution $x_{i,k}(C_{\max}) = 0$ for all $i \leq I, k \leq J_i$. Therefore the feasibility for times $t > C_{\max}$ follows trivially. ■

Let $u_{i,k}$ be the fraction of effort allocated by machine σ_k^i to processing (i,k) jobs in the constructed optimal solution. Clearly,

$$u_{i,k} = \frac{d\, T_{i,k}(t)}{dt} = \frac{p_{i,k} x_i}{C_{\max}}.$$

The constructed solution has a structure resembling a processor sharing policy. It calculates the maximal congestion $C_{\max}$ and allocates a proportional effort to different job types within each machine to achieve the target value $C_{\max}$. Such an optimal policy is possible, since we relaxed the integrality constraint on the number of jobs and allowed machines to work simultaneously on several job types. In the following section we use this fluid solution to construct an asymptotically optimal solution for the original discrete job shop scheduling problem.

2.2.3 The Fluid Synchronization Algorithm

We now provide an efficient algorithm to discretize a fluid solution. We start with some definitions, followed by a formal description of our discretization algorithm. We illustrate our algorithm on a small example, and also discuss the motivation behind our approach. We conclude this section with a formal statement of our main result, along with a complete proof. In the rest of this section the term "discrete network" will refer to the (discrete) job shop scheduling problem, and the term "fluid relaxation" will refer to the fluid job shop scheduling problem. The term "discrete schedule" will refer to the schedule of the jobs in the discrete network. Finally, whenever we use σ_j to refer to a machine, we assume that the reference is to the machine in the discrete network, unless stated otherwise.

Definitions. We first present some useful definitions, and describe our algorithm; the motivation behind these definitions will be described subsequently.

> **Discrete Start time ($DS_{i,k}(n)$):** This is the start time of the n^{th} (i,k) job in the discrete network, i.e., the time at which the n^{th} (i,k) job is scheduled for processing in the (discrete) job shop.

Discrete Completion time ($DC_{i,k}(n)$): This is the completion time of the n^{th} (i,k) job in the discrete network. In particular,

$$DC_{i,k}(n) = DS_{i,k}(n) + p_{i,k}. \tag{2.11}$$

Fluid Start time ($FS_{i,k}(n)$): This is the start time of the n^{th} (i,k) job in the fluid relaxation, and is given by

$$FS_{i,k}(1) = 0, \tag{2.12}$$

$$FS_{i,k}(n) = FS_{i,k}(n-1) + \frac{C_{\max}}{n_i}, \quad n > 1. \tag{2.13}$$

Fluid Completion time ($FC_{i,k}(n)$): This is the completion time of the n^{th} (i,k) job in the fluid relaxation, and is given by

$$FC_{i,k}(n) = FS_{i,k}(n) + \frac{C_{\max}}{n_i}. \tag{2.14}$$

Nominal Start time ($NS_{i,k}(n)$): The nominal start time of the n^{th} (i,k) job is defined as follows.

$$NS_{i,1}(n) = FS_{i,1}(n), \tag{2.15}$$

$$NS_{i,k}(1) = DS_{i,k-1}(1) + p_{i,k-1}, \quad k > 1, \tag{2.16}$$

$$NS_{i,k}(n) = \max\left\{NS_{i,k}(n-1) + \frac{C_{\max}}{n_i},\ DS_{i,k-1}(n) + p_{i,k-1}\right\}, \quad n,k > 1. \tag{2.17}$$

Nominal Completion time ($NC_{i,k}(n)$): The nominal completion time of the n^{th} (i,k) job is defined as follows.

$$NC_{i,k}(n) = NS_{i,k}(n) + \frac{C_{\max}}{n_i}. \tag{2.18}$$

Remark. As a convention, we define $DS_{i,0}(n) = DC_{i,0}(n) = 0$, for all i,n. Similarly, we define $p_{i,0} = 0$ for all i,n.

Each job in the discrete network is assigned a *status* at each of its stages, which is one of *not available*, *available*, *in progress*, or *departed*. The status of the n^{th} (i,k) job at time t is:

- *not available*, if $0 \leq t < DC_{i,k-1}(n)$.
- *available*, if $DC_{i,k-1}(n) \leq t < DS_{i,k}(n)$.
- *in progress*, if $DS_{i,k}(n) \leq t < DC_{i,k}(n)$.
- *departed*, if $t \geq DC_{i,k}(n)$.

We define the *queue-length* of class (i,k) jobs at time t to be the total number of class (i,k) jobs that are *available* or *in progress* at time t.

The following lemma is an easy consequence of our definitions.

Lemma 2.1

(a) The fluid start time and fluid completion time of the n^{th} class (i,k) job satisfy

$$FS_{i,k}(n) = (n-1)\frac{C_{\max}}{n_i}, \quad n = 1, 2, \ldots, n_i. \tag{2.19}$$

$$FC_{i,k}(n) = n\frac{C_{\max}}{n_i}, \quad n = 1, 2, \ldots, n_i. \tag{2.20}$$

(b) The nominal start time, $NS_{i,k}(n)$, and the nominal completion time, $NC_{i,k}(n)$, of the n^{th} (i,k) job can be computed at time $DS_{i,k-1}(n)$. In particular, $NS_{i,k}(n)$ and $NC_{i,k}(n)$ can be computed no later than the time at which the n^{th} (i,k) job becomes available.

(c) The nominal completion time of the n^{th} class (i,k) job satisfies

$$NC_{i,1}(n) = n\frac{C_{\max}}{n_i}, \qquad n = 1, 2, \ldots, n_i, \tag{2.21}$$

$$NC_{i,k}(1) = DC_{i,k-1}(1) + \frac{C_{\max}}{n_i}, \qquad k > 1, \tag{2.22}$$

$$NC_{i,k}(n) = \max\left\{NC_{i,k}(n-1),\ DC_{i,k-1}(n)\right\} + \frac{C_{\max}}{n_i}, \qquad n, k > 1. \tag{2.23}$$

Proof. Part (a) is an immediate consequence of the definitions of $FS_{i,k}(n)$ and $FC_{i,k}(n)$. For part (b), we first suppose that $k > 1$, and expand the recurrence relation given by

Eqs. (2.16) and (2.17) to obtain

$$\begin{aligned} NS_{i,k}(n) &= \max\left\{NS_{i,k}(n-1) + \frac{C_{\max}}{n_i},\ DS_{i,k-1}(n) + p_{i,k-1}\right\}, \\ &= \max_{1\le r\le n}\left\{DS_{i,k-1}(r) + p_{i,k-1} + (n-r)\frac{C_{\max}}{n_i}\right\}. \end{aligned} \tag{2.24}$$

All of the terms involved in Eq. (2.24) become known when the n^{th} $(i,k-1)$ job is scheduled for service in the discrete network, i.e., at time $DS_{i,k-1}(n)$; this proves part (b) for $k > 1$. For $k = 1$, the nominal start times are determined (at time zero) by Eq. (2.15), which completes the proof. Part (c) follows from the definition of $NS_{i,k}(n)$ and $NC_{i,k}(n)$. ■

Description of FSA. Scheduling decisions in the discrete network are made at well-defined *scheduling epochs.* Scheduling epochs for machine σ_j are instants of time at which either some job completes service at σ_j or some job arrives to an idle machine σ_j. Suppose machine σ_j has a scheduling epoch at time t. Among all the *available* jobs at machine σ_j, our algorithm schedules the one with the *smallest nominal start time.* This scheduling decision, in turn, determines the nominal start time of this job at its next stage.

Lemma 2.1 ensures that the nominal start time of a job is determined no later than the time at which it becomes available. Thus, our algorithm is well-defined — every job that is *available* for scheduling will have its *nominal start time* already determined. A complete description of the algorithm appears in Figure 2-1.

Example: Consider the network of Figure 2-2: there are two machines and two types of jobs. Type 1 jobs require 2 units of processing at machine 1, followed by 4 units of processing at machine 2. Type 2 jobs require 1 unit of processing at machine 2, followed by 4 units of processing at machine 1. Initially, we are given 3 jobs of type 1 and 6 jobs of type 2; our objective is to find a schedule that minimizes the makespan. As before, we use (i,k) to denote type i jobs at stage k. The optimal makespan of the associated fluid job shop scheduling problem, with the corresponding optimal fluid controls are given by:

$$C_{\max} = 30,$$

$$\text{(at machine 1)} \quad u_{1,1} = 0.2, \quad u_{2,2} = 0.8,$$

Initialization:

Set $NS_{i,1}(n) = (n-1)\frac{C_{\max}}{n_i}$, for $n = 1, 2, \ldots, n_i, \quad i = 1, 2, \ldots, I$.

Declare all jobs in stage 1 as *available.*

For $j = 1, 2, \ldots, J$: set machine j to have a scheduling epoch.

While (jobs remain to be processed) {

Process job completions

For $j = 1, 2, \ldots, J$:

if machine j has a scheduling epoch at *current-time*

if the n^{th} job $(i,k) \in \sigma_j$ just completed service

if $k < J_i$, declare job n of class $(i, k+1)$ as *available.*

if $(i, k+1) \in \sigma'_j$ and j' is idle, set machine j' to have a scheduling epoch at *current-time.*

Schedule jobs at machines that have scheduling epochs

For $j = 1, 2, \ldots, J$:

if machine j does not have any jobs available

next-epoch(j) $= \infty$

else

if machine j has a scheduling epoch at *current-time*

Schedule an available job with the smallest *nominal start time.*

If the n^{th} (i,k) job is scheduled

Set $DC_{i,k}(n) =$ *current-time* $+p_{i,k}$.

if $k < J_i$,

If $n = 1$ set $NS_{i,k+1}(n) = DC_{i,k}(1)$

else set $NS_{i,k+1}(n) = \max\left\{DC_{i,k}(n), NS_{i,k+1}(n-1) + C_{\max}/n_i\right\}$.

next-epoch(j) $=$ *current-time* $+ p_{i,k}$.

else

next-epoch(j) $= \infty$.

Prepare for the next epoch

next-time $= \min_{j \in \{1,2,\ldots,J\}}$ *next-epoch(j).*

current-time $:=$ *next-time*

For $j = 1, 2, \ldots, J$:

if *next-epoch(j)* $=$ *current-time,*

set machine j to have a scheduling epoch at *current-time.*

}

Figure 2-1: Discretization Algorithm: FSA.

At	(1,1)			(2,2)					
M_1	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	0	10	20	-	-	-	-	-	-
status	a	a	a	na	na	na	na	na	na

At	(1,2)			(2,1)					
M_2	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	-	-	-	0	5	10	15	20	25
status	na	na	na	a	a	a	a	a	a

Table 2.1: State of the system at $t = 0$.

(at machine 2) $u_{2,1} = 0.2, \quad u_{1,2} = 0.4.$

We now step through algorithm FSA and illustrate how a discrete schedule is determined.

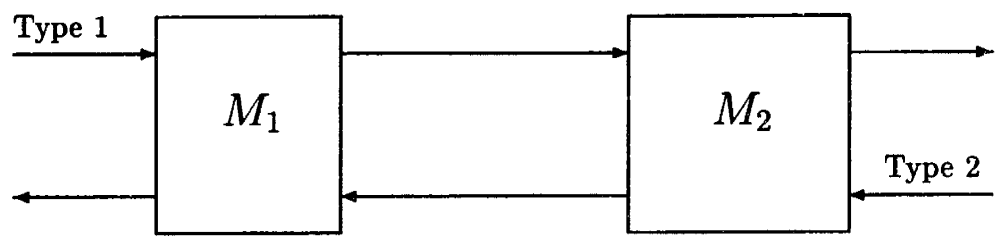

Figure 2-2: A two station network.

$n_1 = 3; p_{1,1} = 2, p_{1,2} = 4$
$n_2 = 6; p_{2,1} = 1, p_{2,2} = 4$
Optimal Fluid Solution: $C_{\max} = 30; u_{1,1} = 0.2, u_{1,2} = 0.4, u_{2,1} = 0.2, u_{2,2} = 0.8$

t = 0: We first perform the initialization step: We compute $NS_{1,1}(n)$ for $n = 1, 2, 3$; and all three jobs of type $(1, 1)$ are declared available at M_1. Similarly, at M_2, we compute $NS_{2,1}(n)$ for $n = 1, \ldots, 6$, and all six jobs of type $(2, 1)$ are declared available. The "state" of the system seen by the machines M_1 and M_2 is shown in Table 2.1.

Remark: A few words of explanation about the tables are in order: the tables shown at time t present the state of the system as seen by the machines *prior* to the scheduling decisions made at time t. As a consequence of the scheduling decisions made at time t, some additional nominal start times may get defined — we shall explicitly state these in our discussion. The "status" row in each table indicates the status of each job, and is one of "unavailable" (na), "available" (a), "in progress" (p) or "departed" (d). In illustrating

At M_1	(1,1) job 1	(1,1) job 2	(1,1) job 3	(2,2) job 1	(2,2) job 2	(2,2) job 3	(2,2) job 4	(2,2) job 5	(2,2) job 6
$NS_{i,k}(n)$	0	10	20	1	-	-	-	-	-
status	p	a	a	a	na	na	na	na	na

At M_2	(1,2) job 1	(1,2) job 2	(1,2) job 3	(2,1) job 1	(2,1) job 2	(2,1) job 3	(2,1) job 4	(2,1) job 5	(2,1) job 6
$NS_{i,k}(n)$	2	-	-	0	5	10	15	20	25
status	na	na	na	d	a	a	a	a	a

Table 2.2: State of the system at $t = 1$.

the algorithm on this example, we shall exhibit similar tables for each of the machines at all scheduling epochs.

Example (contd.): We now return to our example, and consider how the machines M_1 and M_2 make their scheduling decisions at time $t = 0$. At M_1, job 1 of type (1,1) has the smallest nominal start time among all available jobs, and so is scheduled. Similarly, at M_2, job 1 of type $(2,1)$ has the smallest nominal start time among all available jobs, and so is scheduled. These decisions determine the values of $NS_{1,2}(1)$ and $NS_{2,2}(1)$; using Eq. (2.16), we see that $NS_{1,2}(1) = 2$ and $NS_{2,2}(1) = 1$. The next scheduling epoch is for M_2 at $t = 1$.

t = 1: The state of the system is summarized in Table 2.2. Only M_2 has a scheduling epoch. Among all the available jobs at M_2, the second $(2,1)$ job has the smallest nominal start time, and so is scheduled for service. This determines the value $NS_{2,2}(2)$, which is computed (using Eq. (2.17)) as follows.

$$\begin{aligned} NS_{2,2}(2) &= \max\left\{NS_{2,2}(1) + \frac{C_{\max}}{n_i}, DS_{2,1}(2) + p_{2,1}\right\}, \\ &= \max\{1+5, 1+1\} = 6. \end{aligned}$$

The next scheduling epoch is at $t = 2$, for both M_1 and M_2.

t = 2: As before, the state of the system at $t = 2$ is summarized in Table 2.3. The available job with the smallest nominal start time at M_1 is the first $(2,2)$ job; so this job is scheduled for service at M_1. Similarly, the available job with the smallest nominal start time at M_2 is the first $(1,2)$ job, which is scheduled for service. Since both of the jobs scheduled leave

At		(1,1)				(2,2)			
M_1	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	0	10	20	1	6	-	-	-	-
status	d	a	a	a	a	na	na	na	na

At		(1,2)				(2,1)			
M_2	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	2	-	-	0	5	10	15	20	25
status	a	na	na	d	d	a	a	a	a

Table 2.3: State of the system at $t = 2$.

At		(1,1)				(2,2)			
M_1	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	0	10	20	1	6	-	-	-	-
status	d	a	a	d	a	na	na	na	na

At		(1,2)				(2,1)			
M_2	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	2	-	-	0	5	10	15	20	25
status	d	na	na	d	d	a	a	a	a

Table 2.4: State of the system at $t = 6$.

the network, no additional nominal start times need to be computed. The next scheduling epoch is at $t = 6$, for both M_1 and M_2.

t = 6: The state of the system is summarized in Table 2.4. The available job with the smallest nominal start time at M_1 is the second $(2,2)$ job; so this job is scheduled for service at M_1. Since this job leaves the network after its service, it does not determine any additional nominal start times. Similarly, the job with the smallest nominal start time at M_2 is the third $(2,1)$ job, which is scheduled for service; this determines $NS_{2,2}(3) = 11$. The next scheduling epoch is at $t = 7$, for M_2.

t = 7: The state of the system is summarized in Table 2.5. Machine M_2 schedules job 4 of class $(2,1)$, which forces $NS_{2,2}(4) = 16$. The next scheduling epoch is at $t = 8$, for M_2.

t = 8: The state of the system is summarized in Table 2.6. Machine M_2 schedules job 5 of class $(2,1)$, which forces $NS_{2,2}(5) = 21$. The next scheduling epoch is at $t = 9$, for M_2.

At		(1,1)				(2,2)			
M_1	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	0	10	20	1	6	11	-	-	-
status	d	a	a	d	p	a	na	na	na

At		(1,2)				(2,1)			
M_2	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	2	-	-	0	5	10	15	20	25
status	d	na	na	d	d	d	a	a	a

Table 2.5: State of the system at $t = 7$.

At		(1,1)				(2,2)			
M_1	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	0	10	20	1	6	11	16	-	-
status	d	a	a	d	p	a	a	na	na

At		(1,2)				(2,1)			
M_2	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	2	-	-	0	5	10	15	20	25
status	d	na	na	d	d	d	d	a	a

Table 2.6: State of the system at $t = 8$.

t = 9: The state of the system is summarized in Table 2.7. Machine M_2 schedules job 6 of class $(2,1)$, which forces $NS_{2,2}(6) = 26$. The next scheduling epoch is at $t = 10$, for both M_1 and M_2.

t = 10: The state of the system is summarized in Table 2.8. Machine M_2 does not have any jobs to process and hence idles. The job with the smallest nominal start time at machine M_1 is job 2 of class $(1,1)$. The next scheduling epoch is at $t = 12$ for M_1.

By now, the mechanics of the algorithm are clear, and so we end the discussion of this example at this point. We note that the rest of the schedule can be computed easily: observe that the nominal start times of all the jobs that require processing at M_1 have been determined already; this dictates the order in which jobs get processed. At M_2, only jobs 2 and 3 of class $(1,2)$ require processing, and they will be scheduled in that order. The schedule determined by algorithm FSA appears in Table 2.9. We note that in this example,

At		(1,1)				(2,2)			
M_1	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	0	10	20	1	6	11	16	21	-
status	d	a	a	d	p	a	a	a	na

At		(1,2)				(2,1)			
M_2	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	2	-	-	0	5	10	15	20	25
status	d	na	na	d	d	d	d	d	a

Table 2.7: State of the system at $t = 9$.

At		(1,1)				(2,2)			
M_1	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	0	10	20	1	6	11	16	21	26
status	d	a	a	d	d	a	a	a	a

At		(1.2)				(2,1)			
M_2	job 1	job 2	job 3	job 1	job 2	job 3	job 4	job 5	job 6
$NS_{i,k}(n)$	2	-	-	0	5	10	15	20	25
status	d	na	na	d	d	d	d	d	d

Table 2.8: State of the system at $t = 10$.

Time	Queue length			
t	(1,1)	(1,2)	(2,1)	(2,2)
0	3*	0	6*	0
1	3	0	5*	1
2	2	1*	4	2*
6	2	0	4*	1*
7	2	0	3*	2
8	2	0	2*	3
9	2	0	1*	4
10	2*	0	0	4
12	1	1	0	4*
16	1	1*	0	3*
20	1*	0	0	2
22	0	1*	0	2*
26	0	0	0	1*
30	0	0	0	0

Table 2.9: Discrete schedule computed by algorithm FSA.
(* indicates a job was scheduled at that time)

the schedule determined by algorithm FSA has a makespan of 30, which equals the lower bound of $C_{\max} = 30$; thus the schedule shown in Table 2.9 is clearly optimal. ■

Motivation: The key motivation behind our approach is to schedule jobs in a way that keeps us "close" to the optimal fluid solution. Since the optimal fluid cost is a lower bound, we expect this to be a good strategy. The notion of "closeness" is formalized in our definition of *nominal start times* of jobs. The *nominal start time*, $NS_{i,k}(n)$, of the n^{th} (i,k) job reflects the ideal time by which it should have been scheduled.

Since our main objective is to get "close" to the optimal fluid solution, a natural idea is to set the *nominal start time* of the n^{th} (i,k) job to be its start time in the fluid relaxation $(FS_{i,k}(n))$. While this is reasonable in a single machine setting, this does not give much information in a network setting. To illustrate this, consider the n^{th} job of class (i,k). Its fluid start time, $FS_{i,k}(n)$, is identically equal to its fluid start times at stages $1, 2, \ldots k-1$, and is an artifact of the continuous nature of the fluid relaxation. In contrast, in the actual problem, even if the machine at stage (i,k) could process arrivals continuously, job n cannot start at stage k unless it has completed processing at stage at $k-1$ in the discrete network! Our definition of *nominal start time* can be viewed as a correction to the *fluid start time* to account for this effect. Another way to understand the relationship is to observe the

similarity in the definitions of $FS_{i,k}(n)$ and $NS_{i,k}(n)$, which are reproduced below.

$$\begin{aligned}
FS_{i,k}(1) &= 0, \\
FS_{i,k}(n) &= FS_{i,k}(n-1) + \frac{C_{\max}}{n_i}, \quad n > 1, \\
NS_{i,1}(n) &= FS_{i,1}(n), \\
NS_{i,k}(1) &= DC_{i,k-1}(1), \quad k > 1, \\
NS_{i,k}(n) &= \max\left\{NS_{i,k}(n-1) + \frac{C_{\max}}{n_i},\ DC_{i,k-1}(n)\right\}, \quad n > 1, k > 1.
\end{aligned}$$

In the definition of nominal start times, if we ignore the terms involving the discrete network, we obtain exactly the fluid start times! Our approach is inspired by (and related to) research that deals with generalized processor sharing approaches to flow control (see Parekh and Gallager [60, 61]), fair queueing (see Demers, Keshav and Shenker [26], Greenberg and Madras [34]). In fact, our definition of nominal start times can be viewed as a natural adaptation of the notion of *virtual start times*, used by Greenberg and Madras [34], to this setting.

Analysis and Results: We now provide a complete analysis of the algorithm shown in Figure 2-1, and prove our main result, which is stated as Theorem 2.4. In what follows we will make repeated use of the start times $FS_{i,k}(n)$, $NS_{i,k}(n)$, and $DS_{i,k}(n)$, and the completion times $FC_{i,k}(n)$, $NC_{i,k}(n)$, and $DC_{i,k}(n)$; these quantities are defined by Eqs. (2.12)-(2.18), and further simplified in Eqs. (2.19)-(2.23). The proof of Theorem 2.4 involves understanding the relationships between the various start times and completion times for a fixed job. In particular, suppose we consider job n of class (i,k); our objective is to establish a strong relationship between $FC_{i,k}(n)$ and $DC_{i,k}(n)$ — these are, respectively, the completion time of this job in the fluid relaxation and in the discrete network. We establish such a relationship using the nominal completion time, $NC_{i,k}(n)$, as an intermediary.

Our first result relates the discrete start time, $DS_{i,k}(n)$, to the nominal start time, $NS_{i,k}(n)$. Specifically, we show that

$$DS_{i,k}(n) \leq NS_{i,k}(n) + (P_{\sigma_k^i} + U_{\sigma_k^i}), \tag{2.25}$$

where σ_k^i is the machine that processes class (i,k) jobs, $P_{\sigma_k^i}$ is the maximum processing time among all the job classes processed by machine σ_k^i, and $U_{\sigma_k^i}$ is the workload of machine σ_k^i

when there is exactly one job of each type present. A result in this spirit, but for completion times, appears in the literature; it was first proved by Greenberg and Madras [34] for uniform processor sharing systems, and was generalized by Parekh and Gallager [60] to generalized processor sharing systems.

First, we develop the machinery necessary to prove Eq. (2.25). To this end, we focus on the particular machine $\sigma_k^i = \sigma_j$ at which class (i,k) is processed: as we shall see, only the classes that are processed at machine σ_j play a role in establishing Eq. (2.25). To avoid cumbersome notation, we drop the usual (i,k) notation for classes; instead we let R be the set of classes processed by machine σ_j, and use r to denote a generic element of R; these conventions will be in effect until we establish Eq. (2.25).

We start with a few definitions. Let $r \in R$; we define $T_r^c(t)$ and $T_r^d(t)$ as follows:

$$T_r^c(t) = \begin{cases} (n-1)\ p_r + u_r\ (t - NS_r(n)), & \text{for } NS_r(n) \le t < NC_r(n); \\ n\ p_r, & \text{for } NC_r(n) \le t < NS_r(n+1). \end{cases} \tag{2.26}$$

$$T_r^d(t) = \begin{cases} (n-1)\ p_r + (t - DS_r(n)), & \text{for } DS_r(n) \le t < DC_r(n); \\ n\ p_r, & \text{for } DC_r(n) \le t < DS_r(n+1). \end{cases} \tag{2.27}$$

Thus, $T_r^d(t)$ can be interpreted as the total amount of time devoted to processing class r jobs in $[0,t)$ in the discrete network; And $T_r^c(t)$ admits the same interpretation for the "continuous" schedule defined by the nominal start times, in which a job of class r is processed continuously at rate u_r. We also note that $T_r^c(t)$ is continuous: to prove this, we need only check continuity at $t = NC_r(n)$. Suppose $r = (i,k)$; Recall that

$$u_r = \frac{n_i p_r}{C_{\max}},$$

and

$$NC_r(n) - NS_r(n) = \frac{C_{\max}}{n_i}.$$

Using these observations in Eq. (2.26) at $t = NC_r(n)$, we have

$$\begin{aligned} T_r^c(t) &= (n-1)\ p_r + u_r(NC_r(n) - NS_r(n)) \\ &= (n-1)\ p_r + \frac{n_i p_r}{C_{\max}} \frac{C_{\max}}{n_i} \end{aligned}$$

$$= n\,p_r,$$

which is consistent with Eq. (2.26) when $t = NC_r(n)$.

We define a potential function, $\phi_r(t)$, for class r jobs at time t as follows:

$$\phi_r(t) = \max\{T_r^c(t) - T_r^d(t), -p_r\}.$$

Our main motivation in defining the potential function, $\phi_r(t)$, is to capture the extent to which the "discrete" schedule is *behind* the "continuous" schedule on class r jobs up to time t. For this reason, we penalize the discrete schedule for falling behind the continuous schedule, but give credit for *at most one job* when the discrete schedule is ahead of the continuous schedule. We now proceed to derive some useful properties of $\phi_r(t)$, stated as a series of lemmas.

Lemma 2.2 *Let t be a scheduling epoch at machine σ_j. Then the following two statements are equivalent.*

(a) For some $n \geq 1$, job n of class r is such that $NS_r(n) < t \leq DS_r(n)$.

(b) $\phi_r(t) > 0$.

Proof:

(a) $\Rightarrow$ (b):

From Eq. (2.26), we have

$$NS_r(n) < t \quad \Rightarrow \quad T_r^c(t) > (n-1)p_r. \tag{2.28}$$

Similarly, from Eq. (2.27), we have

$$DS_r(n) \geq t \quad \Rightarrow \quad T_r^d(t) \leq (n-1)p_r. \tag{2.29}$$

Simplifying Eqs. (2.28) and (2.29), we obtain

$$T_r^c(t) - T_r^d(t) > 0.$$

Noting that $\phi_r(t) = \max\{T_r^c(t) - T_r^d(t), -p_r\}$, we conclude that $\phi_r(t) > 0$.

(b) $\Rightarrow$ (a):

By definition,

$$\phi_r(t) > 0 \;\Rightarrow\; T_r^c(t) - T_r^d(t) > 0.$$

Since t is a scheduling epoch in the discrete network, $T_r^d(t)$ should be an integral multiple of p_r, i.e., $T_r^d(t) = lp_r$ for some $l \geq 0$. Since $T_r^c(t) > T_r^d(t)$, the $(l+1)^{\text{st}}$ job of class r satisfies $NS_r(l+1) < t \leq DS_r(l+1)$. ■

Lemma 2.3 *Let t be a scheduling epoch at machine σ_j, and suppose $\phi_r(t) > 0$ for some r. Suppose the job scheduled to start at machine σ_j at time t belongs to class r'. Then $\phi_{r'}(t) > 0$.*

Proof: Since $\phi_r(t) > 0$, by Lemma 2.2, there exists n such that $NS_r(n) < t \leq DS_r(n)$. In particular, σ_j cannot starve as there is at least one job waiting to be served. If $r' = r$, we are done, as $\phi_r(t) > 0$ by assumption. If not, let n' be the job of class r' that was scheduled at time t. Since algorithm FSA selected n' over n, $NS_{r'}(n') \leq NS_r(n)$; this is because, at any scheduling epoch, algorithm FSA schedules a job with the smallest nominal start time. In particular, $NS_{r'}(n') < t = DS_{r'}(n')$. Using Lemma 2.2, we conclude that $\phi_{r'}(t) > 0$. ■

A *busy period* for machine σ_j begins when a job arrival (from its previous stage) to σ_j finds it idle; similarly, a *busy period* for machine σ_j ends when a job departure from σ_j leaves it idle.

Lemma 2.4 *Let t be a scheduling epoch at machine σ_j that begins a busy period. Then,*

$$\phi_r(t) \;\leq\; 0. \tag{2.30}$$

Proof: Let $W_r(t)$ is the sum of the processing times of all the class r jobs that arrive prior to time t. Since t is a scheduling epoch at σ_j that begins a busy period, $T_r^d(t) \;=\; W_r(t)$. Also, $T_r^c(t) \;\leq\; W_r(t)$, because $W_r(t)$ represents the total amount of class r work that has arrived up to time t. Thus, $T_r^c(t) - T_r^d(t) \;\leq\; 0$, for every r. By definition,

$$\phi_r(t) \;=\; \max\{T_r^c(t) - T_r^d(t), -p_r\} \;\leq\; 0.$$

■

Lemma 2.5 *Let t be a scheduling epoch at machine σ_j. Then,*

$$\sum_{r=1}^{R} \phi_r(t) \leq 0. \tag{2.31}$$

Proof: Let $t_1, t_2, \ldots, t_b$ be the list of scheduling epochs during an arbitrary busy period in σ_j. (Note that t_b is the epoch that concludes the busy period.) We will prove this lemma using induction on scheduling epochs. By Lemma 2.4, the result is true at t_1, the beginning of the busy period. For $l \leq b$, suppose that Eq. (2.31) holds for $t = t_1, t_2, \ldots t_{l-1}$. We now prove that Eq. (2.31) holds for $t = t_l$. Since t_{l-1} does not conclude a busy period, some job is scheduled to start in machine σ_j at time t_{l-1}; let r' be the class of this job. By definition, $t_l = t_{l-1} + p_{r'}$. We next relate the potential functions at time t_{l-1} and t_l for each job class. In doing this, we shall use the following inequalities that $\phi_r(t)$ satisfies:

$$\phi_r(t) \geq T_r^c(t) - T_r^d(t), \tag{2.32}$$

$$\phi_r(t) + p_r \geq 0. \tag{2.33}$$

Since class r jobs are allocated a fraction u_r of effort in the continuous schedule, we have:

$$T_r^c(t_l) \leq T_r^c(t_{l-1}) + u_r p_{r'}. \tag{2.34}$$

Also, at the machine σ_j, since a job of class r' is scheduled in the interval $[t_{l-1}, t_l)$, we have

$$T_{r'}^d(t_l) = T_{r'}^d(t_{l-1}) + p_{r'}. \tag{2.35}$$

and

$$T_r^d(t_l) = T_r^d(t_{l-1}), \quad \text{for } r \neq r'. \tag{2.36}$$

Thus,

$$\begin{aligned}
\phi_{r'}(t_l) &= \max\{T_{r'}^c(t_l) - T_{r'}^d(t_l), -p_{r'}\} \\
&\leq \max\{T_{r'}^c(t_{l-1}) - T_{r'}^d(t_{l-1}) + u_{r'}p_{r'} - p_{r'}, -p_{r'}\} \\
&\qquad\qquad \text{(by Eq. (2.34) for } r = r', \text{ and Eq. (2.35))} \\
&\leq \max\{\phi_{r'}(t_{l-1}) - p_{r'}(1 - u_{r'}), -p_{r'}\}. \qquad \text{(by Eq. (2.32)).}
\end{aligned} \tag{2.37}$$

For $r \neq r'$, we have

$$\begin{aligned}
\phi_r(t_l) &= \max\{T_r^c(t_l) - T_r^d(t_l), -p_r\} \\
&\leq \max\{T_r^c(t_{l-1}) - T_r^d(t_{l-1}) + u_r p_{r'}, -p_r\} \\
&\qquad\qquad \text{(by Eq. (2.34) for } r \neq r', \text{ and Eq. (2.36))} \\
&\leq \max\{\phi_r(t_{l-1}) + u_r p_{r'}, -p_r\} \qquad \text{(by Eq. (2.32)).} \\
&= \phi_r(t_{l-1}) + u_r p_{r'}. \qquad \text{(by Eq. (2.33)).} \qquad (2.38)
\end{aligned}$$

We now consider two possibilities depending on whether $\phi_{r'}(t_{l-1}) > 0$ or not.

Case 1: $\phi_{r'}(t_{l-1}) > 0$**:**

Since $\phi_{r'}(t_{l-1}) > 0$, and $u_{r'} \geq 0$,

$$\phi_{r'}(t_{l-1}) - p_{r'}(1 - u_{r'}) \; > \; -p_{r'}. \qquad (2.39)$$

From Eqs. (2.39) and (2.37), we obtain

$$\phi_{r'}(t_l) \; = \; \phi_{r'}(t_{l-1}) - p_{r'}(1 - u_{r'}). \qquad (2.40)$$

Thus,

$$\begin{aligned}
\sum_{r=1}^{R} \phi_r(t_l) &= \phi_{r'}(t_l) + \sum_{r:r\neq r'} \phi_r(t_l) \\
&\leq \phi_{r'}(t_{l-1}) - p_{r'}(1 - u_{r'}) + \sum_{r:r\neq r'} (\phi_r(t_{l-1}) + u_r p_{r'}) \\
&\qquad \text{(by Eqs. (2.38) and (2.40))} \\
&= \sum_{r=1}^{R} \phi_r(t_{l-1}) - p_{r'}\left(1 - \sum_{r=1}^{R} u_r\right) \\
&\leq \sum_{r=1}^{R} \phi_r(t_{l-1}) \qquad \left(\sum_{r=1}^{R} u_r \leq 1\right) \\
&\leq 0. \qquad \text{(by the induction hypothesis).}
\end{aligned}$$

Case 2: $\phi_{r'}(t_{l-1}) \leq 0$**:**

Since a job of class r' is scheduled at t_{l-1}, and since $\phi_{r'}(t_{l-1}) \leq 0$, we use Lemma 2.3 to

conclude that there cannot be any job class with positive potential, i.e.,

$$\phi_r(t_{l-1}) \leq 0, \quad \text{for all } r. \tag{2.41}$$

From Eqs. (2.41) and (2.38), we have

$$\phi_r(t_l) \leq u_r p_{r'}, \quad r \neq r'. \tag{2.42}$$

Similarly, from Eqs. (2.41) and (2.37), we obtain

$$\begin{aligned} \phi_{r'}(t_l) &\leq \max\{-p_{r'}(1-u_{r'}), -p_{r'}\} \\ &= -p_{r'}(1-u_{r'}). \end{aligned} \tag{2.43}$$

Adding Eqs. (2.42) and (2.43), we have

$$\begin{aligned} \sum_{r=1}^{R} \phi_r(t_l) &\leq p_{r'}\Big(\sum_{r=1}^{R} u_r - 1\Big) \\ &\leq 0. \qquad \Big(\sum_{r=1}^{R} u_r \leq 1\Big). \end{aligned}$$

In either case, we have shown that

$$\sum_{r=1}^{R} \phi_r(t_l) \leq 0,$$

completing the induction. ■

We are now ready to establish Eq. (2.25).

Theorem 2.1 *Let $NS_{i,k}(n)$ be the nominal start time of the n^{th} (i,k) job; let $DS_{i,k}(n)$ be its start time in the discrete network. Then,*

$$DS_{i,k}(n) \leq NS_{i,k}(n) + (P_{\sigma_k^i} + U_{\sigma_k^i}). \tag{2.44}$$

Proof: We let $r = (i,k)$, and let R be the set of all job classes processed by machine σ_k^i. For convenience, we also let $\sigma_j = \sigma_k^i$. If $DS_r(n) \leq NS_r(n)$, the lemma is trivially true. Suppose $DS_r(n) > NS_r(n)$. Let t be the first time instant in $[NS_r(n), \infty)$ at which the

discrete network has a scheduling epoch. Note that $t \leq DS_r(n)$, since, by definition, t is the first time instant at which the job under consideration could be scheduled in the discrete network. Our plan for the proof is to consider the sum of the processing times, S, of all jobs that are processed at machine σ_j in the discrete network in the interval $[t, DS_r(n))$. Clearly, $DS_r(n) = t + S$, since algorithm FSA does not idle when jobs are available to be processed. We will show that all such jobs have nominal start times at most t, and then proceed to find an upper bound on the number of such jobs, thus providing an upper bound on S.

Consider a job, say job n' of class r', that was scheduled in the discrete network in the interval $[t, DS_r(n))$. Since job n of class r was a candidate during this period, and since it was not selected, we have

$$NS_{r'}(n') \leq NS_r(n). \tag{2.45}$$

This is because algorithm FSA always selects a job with the smallest nominal start time. From Eq. (2.45) and the fact that $NS_r(n) \leq t$, we obtain

$$NS_{r'}(n') \leq t. \tag{2.46}$$

Thus, we have established that the jobs processed during the interval $[t, DS_r(n))$ have nominal start times at most t. By Lemma 2.5,

$$\phi_{r'}(t) > 0, \tag{2.47}$$

if a job belonging to class r' is scheduled in the discrete network in the interval $[t, DS_r(n))$.

Let

$$B_{r'}(t) = \{n' \mid NS_{r'}(n') < t \leq DS_{r'}(n')\}.$$

In other words, $B_{r'}(t)$ consists of those class r' jobs that have started in the continuous schedule before time t, but start in the discrete network at or after t. From Eq. (2.46), the set of jobs that are processed in the discrete network during $[t, DS_r(n))$ is a subset of $\cup_{r'} B_{r'}(t)$. Thus, we are naturally led to considering the cardinality of $B_{r'}(t)$.

If $B_{r'}(t) \neq \emptyset$, let

$$B_{r'}(t) = \{a+1, a+2, \ldots, a+l\}, \qquad a \geq 1,$$

i.e., $|B_{r'}(t)| = l$. (The particular form of $B_{r'}(t)$ follows from the fact that jobs within a class are served in FCFS manner.) Since the $(a+l)^{\text{th}}$ job has started service in the continuous schedule, the $(a+l-1)^{\text{st}}$ job has completed service in the continuous schedule. Thus,

$$T^c_{r'}(t) \;>\; (a+l-1)\; p_{r'},$$

and

$$T^d_{r'}(t) \;=\; a\; p_{r'}.$$

Thus,

$$\phi_{r'}(t) \;=\; \max\{T^c_{r'}(t) - T^d_{r'}(t), -p_{r'}\} \;=\; T^c_{r'}(t) - T^d_{r'}(t) \;>\; (l-1)p_{r'}.$$

Thus,

$$(l-1) \;<\; \frac{\phi_{r'}(t)}{p_{r'}},$$

which implies

$$|B_{r'}(t)| \;=\; l \;\leq\; \frac{\phi_{r'}(t)}{p_{r'}}. \tag{2.48}$$

From Eq. (2.48), the total time required to process all the jobs in $B_{r'}(t)$ in the discrete network is either zero (if $B_{r'}(t) = \emptyset$) or at most $|B_{r'}(t)|p_{r'} \;\leq\; \phi_{r'}(t)$. (Note that $\phi_{r'}(t) > 0$ from Eq. (2.47).) Thus, the total time required to process all the jobs in $B_{r'}(t)$ is at most $\max\{\phi_{r'}(t), 0\}$.

Thus, the total time S required to process all the jobs scheduled in the discrete network during $[t, DS_r(n))$ satisfies:

$$\begin{aligned} DS_r(n) - t \;\leq\; S \;&\leq\; \sum_{r'=1}^{R} \max\{\phi_{r'}(t), 0\} \\ &=\; \sum_{r':\phi_{r'}(t)>0} \phi_{r'}(t) \\ &\leq\; \sum_{r'=1}^{R} (\phi_{r'}(t) + p_{r'}) \qquad \text{(by Eq. (2.33))} \\ &\leq\; \sum_{r'=1}^{R} p_{r'}. \qquad \text{(by Lemma 2.5).} \end{aligned} \tag{2.49}$$

Moreover, t is defined as the first scheduling epoch for the discrete network after time

$NS_r(n)$. In the interval $[NS_r(n), t]$, if some job, say of class $\hat{r}$, is being processed, then

$$t - NS_r(n) \leq p_{\hat{r}}. \tag{2.50}$$

From Eqs. (2.49) and (2.50), we obtain

$$\begin{aligned} DS_r(n) - NS_r(n) &\leq \sum_{r'=1}^{R} p_{r'} + \max_{r'=1,\ldots R} p_{r'} \\ &= U_j + P_j. \end{aligned}$$

■

We now use Theorem 2.1 to establish a relationship between $NC_{i,k}(n)$ and $FC_{i,k}(n)$.

Theorem 2.2 *Let $NC_{i,k}(n)$ be the nominal completion time of the n^{th} (i,k) job, and let $FC_{i,k}(n)$ be its completion time in the fluid relaxation. Then,*

$$NC_{i,k}(n) \leq FC_{i,k}(n) + \sum_{l=1}^{k-1} (2\ P_{\sigma_l^i} + U_{\sigma_l^i}). \tag{2.51}$$

Proof: We fix a job type i, and prove this result by induction on the stage number. The base case for the induction is easy: for $k = 1$, $NC_{i,k}(n)$ and $FC_{i,k}(n)$ are identical (Eqs. (2.14), (2.15), and (2.18)). Suppose the lemma is true for all $(i, k-1)$ jobs, $k \geq 2$. Consider the first (i,k) job.

$$\begin{aligned} NC_{i,k}(1) &= DC_{i,k-1}(1) + \frac{C_{\max}}{n_i} \qquad \text{(by Eq. (2.22))} \\ &= DS_{i,k-1}(1) + p_{i,k-1} + \frac{C_{\max}}{n_i} \qquad \text{(by Eq. (2.11))} \\ &\leq NS_{i,k-1}(1) + (P_{\sigma_{k-1}^i} + U_{\sigma_{k-1}^i}) + p_{i,k-1} + \frac{C_{\max}}{n_i} \qquad \text{(by Theorem 2.1)} \\ &= NC_{i,k-1}(1) + (P_{\sigma_{k-1}^i} + U_{\sigma_{k-1}^i}) + p_{i,k-1} \qquad \text{(by Eq. (2.18))} \\ &\leq FC_{i,k-1}(1) + \sum_{l=1}^{k-2} (2\ P_{\sigma_l^i} + U_{\sigma_l^i}) + (P_{\sigma_{k-1}^i} + U_{\sigma_{k-1}^i}) + p_{i,k-1} \\ &\qquad \text{(by the induction hypothesis)} \\ &\leq FC_{i,k}(1) + \sum_{l=1}^{k-1} (2\ P_{\sigma_l^i} + U_{\sigma_l^i}). \qquad \text{(by Eq. (2.20))} \end{aligned}$$

Thus, the Lemma is true for the first (i,k) job. Suppose it is true for the first $(n-1)$

(i,k) jobs. Consider the n^{th} (i,k) job $(n > 1,\ k > 1)$. From Eq. (2.23),

$$NC_{i,k}(n) \;=\; \max\left\{NC_{i,k}(n-1),\ DC_{i,k-1}(n)\right\} \;+\; \frac{C_{\max}}{n_i}.$$

We consider the two cases.

Case 1: $NC_{i,k}(n) \;=\; NC_{i,k}(n-1) + C_{\max}/n_i$.

In this case, we have:

$$\begin{aligned}
NC_{i,k}(n) &= NC_{i,k}(n-1) + \frac{C_{\max}}{n_i} \\
&\leq FC_{i,k}(n-1) + \sum_{l=1}^{k-1}(2\ P_{\sigma_l^i} + U_{\sigma_l^i}) + \frac{C_{\max}}{n_i} \\
&\qquad\qquad \text{(by the induction hypothesis)} \\
&\leq FC_{i,k}(n) + \sum_{l=1}^{k-1}(2\ P_{\sigma_l^i} + U_{\sigma_l^i}). \qquad \text{(by Eq. (2.20))}
\end{aligned}$$

Case 2: $NC_{i,k}(n) \;=\; DC_{i,k-1}(n) + C_{\max}/n_i$.

In this case, we have:

$$\begin{aligned}
NC_{i,k}(n) &= DC_{i,k-1}(n) + \frac{C_{\max}}{n_i} \qquad \text{(by Eq. (2.23))} \\
&= DS_{i,k-1}(n) + p_{i,k-1} + \frac{C_{\max}}{n_i} \qquad \text{(by Eq. (2.11))} \\
&\leq NS_{i,k-1}(n) + (P_{\sigma_{k-1}^i} + U_{\sigma_{k-1}^i}) + p_{i,k-1} + \frac{C_{\max}}{n_i} \qquad \text{(by Theorem 2.1)} \\
&= NC_{i,k-1}(n) + (P_{\sigma_{k-1}^i} + U_{\sigma_{k-1}^i}) + p_{i,k-1} \qquad \text{(by Eq. (2.18))} \\
&\leq FC_{i,k-1}(n) + \sum_{l=1}^{k-2}(2\ P_{\sigma_l^i} + U_{\sigma_l^i}) + (P_{\sigma_{k-1}^i} + U_{\sigma_{k-1}^i}) + p_{i,k-1} \\
&\qquad\qquad \text{(by the induction hypothesis)} \\
&\leq FC_{i,k}(n) + \sum_{l=1}^{k-1}(2\ P_{\sigma_l^i} + U_{\sigma_l^i}). \qquad \text{(by Eq. (2.20))}
\end{aligned}$$

■

The following theorem relates $DC_{i,k}(n)$ to $FC_{i,k}(n)$, and is an immediate consequence of Theorems 2.1 and 2.2.

Theorem 2.3 *Let $FC_{i,k}(n)$ ($DC_{i,k}(n)$) be the completion time of the n^{th} (i,k) job in the fluid relaxation (discrete network). Then,*

$$DC_{i,k}(n) \;\leq\; FC_{i,k}(n) \;+\; \sum_{l=1}^{k}(2\; P_{\sigma_l^i} + U_{\sigma_l^i}). \tag{2.52}$$

Proof: From Theorem 2.1, we have

$$DS_{i,k}(n) \leq NS_{i,k}(n) + (P_{\sigma_k^i} + U_{\sigma_k^i}).$$

Since $p_{i,k} \leq C_{\max}/n_i$, we have

$$\begin{aligned} DC_{i,k}(n) &= DS_{i,k}(n) + p_{i,k} \\ &\leq NS_{i,k}(n) + (P_{\sigma_k^i} + U_{\sigma_k^i}) + p_{i,k} \\ &= NC_{i,k}(n) - \frac{C_{\max}}{n_i} + (P_{\sigma_k^i} + U_{\sigma_k^i}) + p_{i,k} \\ &\leq NC_{i,k}(n) + (P_{\sigma_k^i} + U_{\sigma_k^i}). \qquad (\text{since } p_{i,k} \leq C_{\max}/n_i). \end{aligned} \tag{2.53}$$

From Eqs. (2.53) and (2.51), we obtain Eq. (2.52). ■

We are now ready to state our main result.

Theorem 2.4 *Consider a job shop scheduling problem with I job types and J machines $\sigma_1, \sigma_2, \ldots, \sigma_J$. Given initially n_i jobs of type $i = 1, 2, \ldots, I$, the algorithm FSA produces a schedule with makespan time C_{D} such that*

$$C_{\max} \;\leq\; C^* \;\leq\; C_{\mathrm{D}} \;\leq\; C_{\max} + \max_{i=1,2,\cdots,I} \sum_{l=1}^{J_i}(2\; P_{\sigma_l^i} + U_{\sigma_l^i}). \tag{2.54}$$

In particular,

$$\frac{C_{\mathrm{D}}}{C^*} \leq \frac{C_{\mathrm{D}}}{C_{\max}} \to 1, \tag{2.55}$$

as

$$\sum_{i=1}^{I} n_i \to \infty,$$

where C^* is the optimal makespan.

Proof: From Eqs. (2.12), (2.13) and (2.14), we see that

$$FC_{i,J_i}(n_i) \;=\; C_{\max}$$

for all $i \in I$. Using Theorem 2.3,

$$\begin{aligned} DC_{i,J_i}(n_i) \;&\leq\; FC_{i,J_i}(n_i) + \sum_{l=1}^{J_i} (2\,P_{\sigma_l^i} + U_{\sigma_l^i}) \\ &=\; C_{\max} + \sum_{l=1}^{J_i} (2\,P_{\sigma_l^i} + U_{\sigma_l^i}). \end{aligned}$$

The result follows by observing that

$$C_D \;=\; \max_{i \in I} \{DC_{i,J_i}(n_i)\}.$$

■

From Theorem 2.4, we see that the additive error of the schedule computed by algorithm FSA is bounded from above by $J_{\max}(2\,P_{\max} + U_{\max})$, which is substantially smaller than the guarantees provided by Sevast'yanov's algorithm [67, 68]. We note that an additive error of $(J_{\max} - 1)P_{\max}$ is necessary for any algorithm that uses the optimal fluid cost for comparison purposes: for example, consider a simple flow shop with J stages, and let the processing time at each stage be P. If there are N jobs to start with, the optimal fluid cost is NP, whereas the optimal makespan is $(N + J - 1)P$. An interesting open problem is to find the "optimal" additive error for algorithms based on fluid relaxations, and to design algorithms that achieve this additive error.

2.3 Makespan with Deterministic Arrivals

This section generalizes the results of §2.2 to a model in which external arrivals are permitted over a (suitably restricted) finite horizon $[0, T^*]$. The objective is to minimize the time required to process all the initial jobs plus the jobs that arrive during $[0, T^*]$. This section is structured as follows: In §2.3.1, we formally define the model considered; The associated fluid relaxation and its solution is discussed in §2.3.2. In §2.3.3, we prove that the *fluid synchronization algorithm* (FSA) yields a schedule that is asymptotically optimal.

2.3.1 Problem Formulation

The model considered here is identical to that of §2.2.2, except that external arrivals are permitted. We assume that type i jobs arrive to the network in a deterministic fashion at rate λ_i. The traffic intensity at machine σ_j, denoted by ρ_j, is defined as

$$\rho_j = \sum_{(i,k)\in\sigma_j} p_{i,k}\lambda_i. \tag{2.56}$$

Our objective is to minimize the time required to process all the $n_1 + n_2 + \cdots + n_I$ jobs, plus the jobs that arrive in the interval $[0, T^*]$, where

$$T^* = \max_j \frac{\sum_{(i,k)\in\sigma_j} p_{i,k} n_i}{1 - \rho_j}. \tag{2.57}$$

Remarks:

- Observe that since arrivals to the network after T^* are irrelevant for our objective, we may assume that arrivals occur over a finite horizon $[0, T^*]$.
- In considering the asymptotics, we let $n \to \infty$; this will result in $T^* \to \infty$ as well, as specified in Eq. (2.57). We emphasize that T^* is implicitly determined by the choice of λ_i and n_i, and is *not* part of the input.

As before, we define the congestion of machine σ_j as

$$C_j = \sum_{(i,k)\in\sigma_j} p_{i,k}(n_i + \lambda_i T^*). \tag{2.58}$$

We denote the maximum congestion by

$$C_{\max} \equiv \max_{j=1,\ldots,J} C_j.$$

The following proposition is immediate.

Proposition 2.3 *The minimum makespan C^* of the job shop scheduling problem satisfies:*

$$C^* \geq C_{\max}.$$

We next show that $C_{\max} = T^*$.

Proposition 2.4 *The maximum congestion $C_{\max}$ of the job shop satisfies:*

$$C_{\max} = T^*.$$

Proof: Let j be such that

$$T^* = \frac{\sum_{(i,k)\in\sigma_j} p_{i,k} n_i}{1-\rho_j}.$$

Then,

$$\begin{aligned} C_j &= \sum_{(i,k)\in\sigma_j} p_{i,k}(n_i + \lambda_i T^*) \\ &= \sum_{(i,k)\in\sigma_j} p_{i,k} n_i + \rho_j T^* \qquad \text{(by Eq. (2.56))} \\ &= \sum_{(i,k)\in\sigma_j} p_{i,k} n_i + \rho_j \frac{\sum_{(i,k)\in\sigma_j} p_{i,k} n_i}{1-\rho_j} \qquad \text{(by choice of } j\text{)} \\ &= T^*, \end{aligned}$$

which shows that $C_{\max} \geq T^*$. Now, we show that $C_{j'} \leq T^*$ for an arbitrary machine j'. We have

$$\begin{aligned} C_{j'} &= \sum_{(i,k)\in\sigma_{j'}} p_{i,k}(n_i + \lambda_i T^*) \\ &= \sum_{(i,k)\in\sigma_{j'}} p_{i,k} n_i + \rho_{j'} T^* \qquad \text{(by Eq. (2.56))} \\ &\leq (1-\rho_{j'})T^* + \rho_{j'} T^* \qquad \text{(by definition of } T^*\text{)} \\ &= T^*, \end{aligned}$$

which proves that $C_{\max} \leq T^*$. ■

We next consider the associated fluid relaxation and show that a simple algorithm leads to a makespan equal to $C_{\max}$, and is, therefore, optimal.

2.3.2 The Fluid Job Shop Scheduling Problem

The fluid relaxation associated with the model considered in §2.3.1 is as follows:

$$\text{minimize} \quad \int_0^\infty 1\left\{\sum_{1\leq i\leq I, 1\leq k\leq J} x_{i,k}(t) > 0\right\} dt \tag{2.59}$$

$$\text{subject to} \quad x_{i,1}(t) = x_i + \lambda_i \min(t, T^*) - \mu_{i,1} T_{i,1}(t), \quad i = 1, 2, \ldots, I, \ t \geq 0, \tag{2.60}$$

$$x_{i,k}(t) = \mu_{i,k-1} T_{i,k-1}(t) - \mu_{i,k} T_{i,k}(t), \quad k = 2, \ldots, J_i, \ i = 1, 2, \ldots, I, \ t \geq 0, \tag{2.61}$$

$$0 \leq \sum_{(i,k) \in \sigma_j} (T_{i,k}(t_2) - T_{i,k}(t_1)) \leq t_2 - t_1, \ \forall \ t_2 > t_1, \ t_1, t_2 \geq 0, \ j = 1, 2, \ldots, J, \tag{2.62}$$

$$x_{i,k}(t) \geq 0, \quad T_{i,k}(t) \geq 0. \tag{2.63}$$

The objective function (2.59) represents the total time that at least one of the fluid levels is positive, and corresponds to the minimum makespan schedule in the discrete problem. The only difference from the model of §2.2.2 is in Eq. (2.60), where the additional term $\lambda_i \min(t, T^*)$ represents the (fractional) number of external arrivals of type i jobs up to time t.

Similar to the definition for the discrete problem, we define congestion in machine σ_j as

$$C_j = \sum_{(i,k) \in \sigma_j} p_{i,k}(x_i + \lambda_i T^*) \tag{2.64}$$

and the maximal congestion as

$$C_{\max} = \max_{1 \leq j \leq J} C_j. \tag{2.65}$$

We next show that the fluid relaxation can be solved explicitly.

Proposition 2.5 *The fluid relaxation (2.59) has an optimal value equal to the maximum congestion* $C_{\max}$.

Proof: We first show that the maximum congestion $C_{\max}$ is a lower bound on the optimal value of the control problem. For any positive time t and for each $i \leq I$, $k \leq J_i$, we have from Eqs. (2.60), (2.61):

$$\sum_{l=1}^{k} x_{i,l}(t) = x_i + \lambda_i \min(t, T^*) - \mu_{i,k} T_{i,k}(t).$$

Let $t \leq T^*$, and let j be an index that achieves the minimum in Eq. (2.57). For machine σ_j we obtain:

$$\begin{aligned} \sum_{(i,k) \in \sigma_j} p_{i,k} \sum_{l=1}^{k} x_{i,l}(t) &= \sum_{(i,k) \in \sigma_j} p_{i,k}(x_i + \lambda_i \min(t, T^*)) - \sum_{(i,k) \in \sigma_j} T_{i,k}(t) \\ &\geq \sum_{(i,k) \in \sigma_j} p_{i,k}(x_i + \lambda_i t) - t \qquad \text{(Constraint (2.62) applied to} \end{aligned}$$

$$t_1 = 0, t_2 = t)$$

$$= \sum_{(i,k)\in\sigma_j} p_{i,k}x_i - (1-\rho_j)t \qquad \text{(by Eq. (2.56))}$$

$$= (1-\rho_j)(T^* - t) \qquad \text{(by the choice of } j\text{, and Eq. (2.57))}$$

It follows then, that the fluid levels at σ_j are positive for all times t smaller than T^*. Therefore, the objective value of the fluid relaxation is at least T^*, which, by Proposition 2.4, equals $C_{\max}$.

We now construct a feasible solution that achieves this value. For each $i \leq I$, $k \leq J_i$ and each $t \leq C_{\max}$ we let

$$\begin{aligned} T_{i,k}(t) &= \frac{p_{i,k}(x_i + \lambda_i C_{\max})}{C_{\max}} t, \\ x_{i,1}(t) &= x_i + \lambda_i t - \mu_{i,1} T_{i,1}(t) = x_i + \lambda_i t - \frac{(x_i + \lambda_i C_{\max})}{C_{\max}} t, \quad i = 1, \ldots, I, \\ x_{i,k}(t) &= 0, \qquad\qquad k = 2, 3, \ldots, J_i, \ i = 1, \ldots, I. \end{aligned}$$

For all $t > C_{\max}$ we set $T_{i,k}(t) = p_{i,k}(x_i + \lambda_i C_{\max})$, $x_{i,k}(t) = 0$. Clearly, this solution has an objective value equal to $C_{\max}$. We now show that this solution is feasible. It is nonnegative by construction. Also by construction, Eq. (2.60) is satisfied for all $t \leq C_{\max}$. In particular, $x_{i,1}(C_{\max}) = 0$, $i = 1, 2, \ldots, I$. Moreover, for all i, $k = 2, 3, \ldots, J_i$ and $t \leq C_{\max}$ we have:

$$\mu_{i,k-1} T_{i,k-1}(t) - \mu_{i,k} T_{i,k}(t) = \frac{(x_i + \lambda_i C_{\max})}{C_{\max}} t - \frac{(x_i + \lambda_i C_{\max})}{C_{\max}} t = 0 = x_{i,k}(t),$$

and Eq. (2.61) is satisfied. Finally, for any $t_1 < t_2 \leq C_{\max}$ and for any machine σ_j, we have:

$$\begin{aligned} \sum_{(i,k)\in\sigma_j} (T_{i,k}(t_2) - T_{i,k}(t_1)) &= \sum_{(i,k)\in\sigma_j} \left(\frac{p_{i,k}(x_i + \lambda_i C_{\max})}{C_{\max}} t_2 - \frac{p_{i,k}(x_i + \lambda_i C_{\max}}{C_{\max}} t_1 \right) \\ &= \frac{C_j}{C_{\max}} (t_2 - t_1) \\ &\leq t_2 - t_1, \end{aligned}$$

and Constraint (2.62) is satisfied. Note, that for the constructed solution $x_{i,k}(C_{\max}) = 0$ for all $i \leq I, k \leq J_i$. Therefore the feasibility for times $t > C_{\max}$ follows trivially. ■

In the following section we prove that FSA yields an asymptotically optimal solution

for the original discrete job shop scheduling problem.

2.3.3 The Fluid Synchronization Algorithm

Recall that algorithm FSA of §2.2.3 relied on the notion of *nominal start times.* Our plan for this section is quite simple: we describe an analogous definition of *nominal start times* for the model with arrivals, and argue that all of the results of §2.2.3 carry over under this new definition also.

Definitions: The definitions of $DS_{i,k}(n)$ and $DC_{i,k}(n)$ are the same as before. We now present the definitions of $FS_{i,k}(n)$ and $NS_{i,k}(n)$. In the following, we let n index the jobs; the first n_i jobs are the ones that are initially present in the network. Jobs $n_i+1,\ldots,n_i+e_i$ are the type i jobs that arrived from outside in the interval $[0,T^*]$. Let $a_i(l)$ be the arrival time of the l^{th} external arrival of type i, for $l=1,2,\ldots,e_i$.

Fluid Start time ($FS_{i,k}(n)$): This is the start time of the n^{th} (i,k) job in the fluid relaxation, and is given by

$$FS_{i,k}(1) = 0, \tag{2.66}$$

$$FS_{i,k}(n) = FS_{i,k}(n-1) + \frac{C_{\max}}{(n_i+\lambda_i C_{\max})}, \quad n>1. \tag{2.67}$$

Nominal Start time ($NS_{i,k}(n)$): The nominal start time of the n^{th} (i,k) job is defined as follows.

$$NS_{i,1}(n) = FS_{i,1}(n), \quad n=1,2,\ldots,n_i, \tag{2.68}$$

$$NS_{i,1}(n_i+l) = \max\{FS_{i,1}(n_i+l), a_i(l)\} \quad l=1,2,\ldots,e_i, \tag{2.69}$$

$$NS_{i,k}(1) = DS_{i,k-1}(1)+p_{i,k-1}, \quad k>1, \tag{2.70}$$

$$NS_{i,k}(n) = \max\left\{NS_{i,k}(n-1) + \frac{C_{\max}}{(n_i+\lambda_i C_{\max})}, DS_{i,k-1}(n) + p_{i,k-1}\right\}, \quad n,k>1. \tag{2.71}$$

As before, at every scheduling epoch for the discrete network, algorithm FSA schedules a job with the smallest nominal start time. To prove that FSA yields an asymptotically optimal schedule, we need to prove analogs of Theorems 2.1 and 2.2. Clearly, Theorem 2.1 remains true in this setting as well. In the proof of Theorem 2.2, we used $NS_{i,1}(n) =$

$FS_{i,1}(n)$ for all i, n to establish the basis for the induction; this must be established for the model under consideration for $n_i < n \leq n_i + e_i$. It is easy to see that the proof of Theorem 2.2 would follow if we can prove that $NS_{i,1}(n) = FS_{i,1}(n)$ for all i, n.

Lemma 2.6 *Let $FS_{i,k}(n)$ and $NS_{i,k}(n)$ be defined as in Eqs. (2.66) and (2.71). Then,*

$$NS_{i,1}(n) = FS_{i,1}(n), \qquad \forall\ i, n. \tag{2.72}$$

Proof: Fix a job type i. The Lemma is trivially true (by Eq. (2.68)) for $n \leq n_i$. To establish Eq. (2.72) for $n > n_i$, we only need to show that for any $1 \leq l \leq e_i$,

$$FS_{i,1}(n_i + l) \geq a_i(l) \tag{2.73}$$

Since arrivals are deterministic,

$$a_i(l) = \frac{l-1}{\lambda_i}. \tag{2.74}$$

From Eq. (2.66),

$$FS_{i,1}(n_i + l) = (n_i + l - 1)\ \frac{C_{\max}}{n_i + \lambda_i C_{\max}}. \tag{2.75}$$

Thus,

$$\begin{aligned} \frac{FS_{i,1}(n_i + l)}{a_i(l)} &= (n_i + l - 1)\ \frac{C_{\max}}{n_i + \lambda_i C_{\max}}\ \frac{\lambda_i}{l-1} \\ &= \frac{n_i C_{\max}\lambda_i + (l-1)C_{\max}\lambda_i}{n_i(l-1) + (l-1)C_{\max}\lambda_i} \\ &\geq 1 \qquad (\text{because } e_i \leq \lambda_i C_{\max} + 1). \end{aligned}$$

■

Thus, Theorems 2.1 and 2.2 remain true for this model, which in turn imply Theorem 2.3. These observations prove the following analog of Theorem 2.4.

Theorem 2.5 *Consider a job shop scheduling problem with I job types and J machines $\sigma_1, \sigma_2, \ldots, \sigma_J$. Suppose we are given initially n_i jobs of type $i = 1, 2, \ldots, I$; suppose also that external arrivals of type i jobs occur deterministically at rate λ_i over the horizon $[0, T^*]$, where T^* is given by Eq. (2.57). Then, the algorithm FSA produces a schedule*

with makespan time C_{D} such that

$$C_{\max} \leq C^* \leq C_{\mathrm{D}} \leq C_{\max} + \max_{i=1,2,\cdots,I} \sum_{l=1}^{J_i} (2\ P_{\sigma_l^i} + U_{\sigma_l^i}). \tag{2.76}$$

In particular,

$$\frac{C_{\mathrm{D}}}{C^*} \leq \frac{C_{\mathrm{D}}}{C_{\max}} \to 1, \tag{2.77}$$

as

$$\sum_{i=1}^{I} n_i \to \infty,$$

where C^* is the optimal makespan.

From Theorem 2.5, we see that the additive error of the schedule computed by algorithm FSA is bounded from above by $J_{\max}(2\ P_{\max} + U_{\max})$. As before, considering a flow shop establishes that an an additive error of $(J_{\max} - 1)P_{\max}$ is necessary for any algorithm that uses the optimal fluid cost for comparison purposes. Improved results for the model of §2.2 will directly result in improvements for the model with arrivals, which makes the problem of designing algorithms that achieve the optimal additive error both interesting and important.

2.4 Computational Results

In this section, we report computational results on the proposed algorithms. In §2.4.1 we consider the case of deterministic processing times; §2.4.2 considers the case of stochastic processing times.

2.4.1 Deterministic processing times

We now present computational results based on algorithm FSA. For our first experiment, we consider the famous 10 by 10 instance in Muth and Thompson [57] with $n_i = N$ jobs present and vary N. Table 2.10 shows the performance of algorithm FSA for N ranging from 1 to 1000. The most striking observation from Table 2.10 is that the gap between C_D and $C_{\max}$ is insensitive to N (as guaranteed by our Theorem 2.4).

We have performed an extensive computational study of several benchmark job shop instances available as part of the OR library (http://mscmga.ms.ic.ac.uk/info.html). The results reported in Table 2.11 are for 80 benchmarks. The number of machines ranged

N	$C_{\max}$	C_D	$C_D - C_{\max}$	$(C_D - C_{\max})/C_{\max}$
1	631	1189	558	0.8843
2	1262	1820	558	0.4421
3	1893	2546	653	0.3450
4	2524	3073	549	0.2175
5	3155	3740	585	0.1854
6	3786	4313	527	0.1392
7	4417	4975	558	0.1263
8	5048	5608	560	0.1109
9	5679	6170	491	0.0865
10	6310	6842	532	0.0843
20	12620	13159	539	0.0427
30	18930	19509	579	0.0306
40	25240	25779	539	0.0214
50	31550	32041	491	0.0156
100	63100	63639	539	0.0085
1000	631000	631584	584	0.0009

Table 2.10: Computational results for the 10 x 10 example.

from 6 to 20, and the number of job types ranged from 6 to 50. For each benchmark, we report results for $N = 1$, $N = 2$, $N = 5$, $N = 10$ and $N = 100$, and $N = 500$. The lower bound based on the fluid relaxation, $C_{\max}$, is shown in the second column, and is valid for $N = 1$; the lower bound for $N = n$ is $n\, C_{\max}$. The subsequent columns report the value of $C_D - n\, C_{\max}$. Again, we see that the gap between C_D and $C_{\max}$ is insensitive to N.

2.4.2 Stochastic processing times

We now turn our attention to the case in which the processing times are stochastic. We consider the same set of 80 instances from the OR library. We assume that the processing times are independent, and geometrically distributed with mean $p_{i,k}$, where $p_{i,k}$ is the processing time in the deterministic case. In Table 2.12, we report our results. Since the results are insensitive to N, we only report results for $N = 1$, $N = 2$, $N = 5$ and $N = 10$. For each value of N, we performed 10000 experiments; the average, minimum, and maximum values of C_D are reported in Table 2.12. For comparison purposes, we provide the average value of the "congestion lower bound." (column "lb."), the expected value of C_D (column "avg.") and the maximum value of C_D (column "max.").

From Table 2.12, we see that the difference between $C_D(avg.)$ and $C_{\max}$ is insensitive to N.

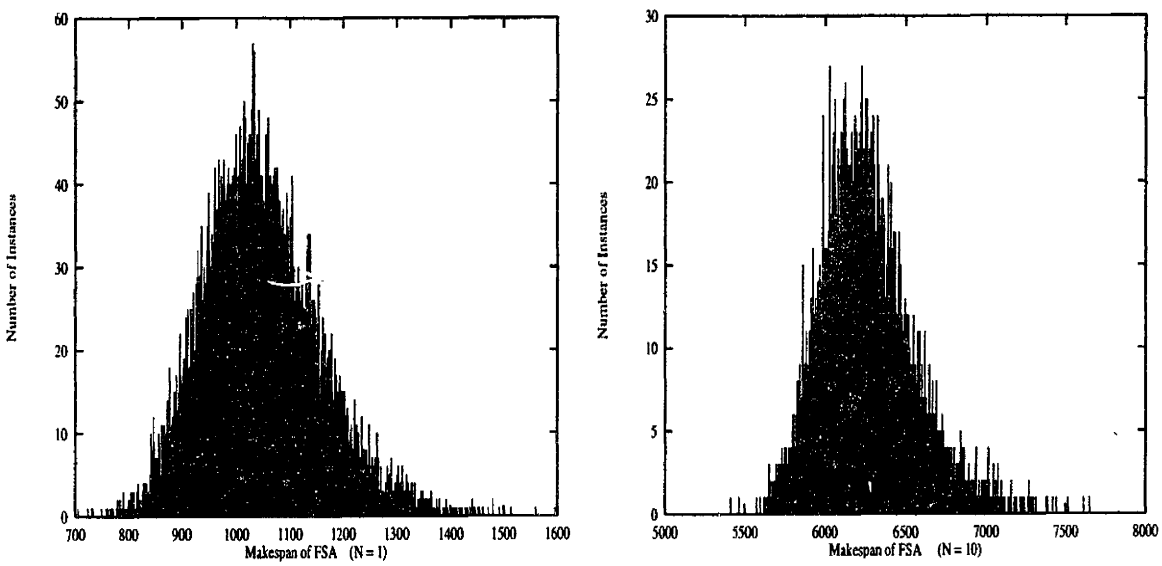

Figure 2-3: Distribution of Makespan for abz7

Benchmark	C_{max} (N = 1)	$C_D - N\,C_{max}$					
		N = 1	N = 2	N = 5	N = 10	N = 100	N = 500
abz5	868	599	452	457	443	567	475
abz6	688	368	350	412	234	270	256
abz7	556	233	255	227	194	43	79
abz8	566	294	269	140	131	84	93
abz9	563	366	234	135	122	110	102
ft20	1119	526	477	426	341	54	54
la01	666	106	44	85	86	86	86
la02	635	219	98	100	100	100	100
la03	588	174	115	46	0	0	0
la04	537	158	153	114	100	132	132
la05	593	17	0	0	0	0	0
la06	926	0	0	0	0	0	0
la07	869	219	105	67	67	67	67
la08	863	117	92	0	0	0	0
la09	951	67	23	0	0	0	0
la10	958	48	7	7	7	7	7
la11	1222	50	0	0	0	0	0
la12	1039	0	0	0	0	0	0
la13	1150	49	0	0	0	0	0
la14	1292	0	0	0	0	0	0
la15	1207	380	232	46	44	44	44
la16	660	520	477	328	335	277	277
la17	683	260	242	181	259	259	259
la18	623	421	373	346	262	219	264
la19	685	298	148	32	34	34	34
la20	744	528	345	175	160	160	160
la21	935	471	432	172	171	160	160
la22	830	482	374	439	439	439	439
la23	1032	250	49	142	0	0	0
la24	857	331	186	129	129	59	59
la25	864	344	408	99	23	29	29
la26	1218	154	236	11	0	4	4
la27	1188	456	421	401	280	121	121
la28	1216	258	149	60	157	157	157
la29	1105	473	413	226	236	215	215
la30	1355	293	14	0	0	0	0
la31	1784	150	0	0	0	0	0
la32	1850	260	79	0	0	0	0
la33	1719	154	73	66	66	66	66
la34	1721	240	41	0	0	0	0
la35	1888	254	8	8	8	8	8
la36	1028	488	393	468	388	409	406

continued on next page

continued from previous page							
	$C_{\max}$	$C_D - N\,C_{\max}$					
Benchmark	(N = 1)	N = 1	N = 2	N = 5	N = 10	N = 100	N = 500
la37	980	881	711	530	330	406	406
la38	876	590	610	345	373	99	99
la39	1012	520	497	557	604	679	679
la40	1027	504	346	278	233	193	161
orb01	643	736	720	720	740	740	740
orb02	671	336	322	246	271	134	243
orb03	624	781	792	829	798	819	819
orb04	759	566	447	337	337	371	371
orb05	630	525	557	397	390	390	390
orb06	659	671	731	696	746	746	746
orb07	286	189	162	162	140	132	132
orb08	585	640	823	763	755	774	725
orb09	661	528	492	554	448	448	448
orb10	652	651	553	415	415	408	408
swv01	1219	935	996	996	996	996	996
swv02	1259	898	828	828	828	828	828
swv03	1178	841	772	834	834	834	834
swv04	1161	964	895	842	737	608	645
swv05	1235	783	782	767	767	767	767
swv06	1229	1290	1161	1130	1103	1067	1067
swv07	1128	1111	1096	1156	999	977	977
swv08	1330	1174	1062	983	1009	1091	1091
swv09	1266	1232	1235	1189	1189	1189	1127
swv10	1159	1147	1659	1353	1370	1286	1286
swv11	2808	1619	1619	1619	1619	1619	1619
swv12	2829	1854	1830	1742	1602	888	902
swv13	2977	1852	1774	1752	1736	1752	1752
swv14	2842	1779	1684	1684	1684	1684	1684
swv15	2762	1858	1931	1877	1787	898	1013
swv16	2924	27	54	54	54	54	54
swv17	2794	168	171	146	114	114	114
swv18	2852	122	99	108	108	108	108
swv19	2843	252	67	0	0	0	0
swv20	2823	45	0	0	0	0	0
yn1	643	473	409	287	226	222	232
yn2	686	509	418	357	333	288	318
yn3	659	476	392	292	187	213	215
yn4	676	521	563	517	430	340	326

Table 2.11: Computational results for Job Shop instances in OR-Library.

Benchmark	C_D											
	N = 1			N = 2			N = 5			N = 10		
	lb.	avg.	max.	lb.	avg.	max.	lb.	avg.	max.	lb.	avg.	max.
abz5	1215	1830	2875	2162	2860	3994	4867	5665	7241	9268	10124	12247
abz6	965	1443	2187	1721	2261	3332	3902	4512	5828	7459	8121	10131
abz7	719	1043	1492	1325	1670	2142	3057	3415	4094	5883	6239	7264
abz8	740	1091	1541	1349	1731	2164	3091	3492	4128	5974	6350	7227
abz9	730	1077	1517	1327	1697	2194	3044	3429	4219	5849	6210	7483
ft06	57	82	179	103	131	236	237	271	389	456	493	702
ft10	854	1541	2466	1511	2333	3413	3429	4447	5897	6555	7747	9520
ft20	1356	1818	2999	2520	3051	4411	5997	6618	8511	11648	12309	14895
la01	828	1003	2053	1526	1720	2957	3547	3760	5600	6846	7071	9371
la02	790	987	1801	1450	1677	2833	3397	3653	5384	6575	6853	9385
la03	722	901	1622	1314	1525	2567	3067	3304	4600	5977	6187	8279
la04	746	932	1784	1353	1581	2790	3077	3361	4978	5849	6197	8073
la05	704	795	1564	1320	1415	2218	3117	3213	4874	6055	6154	7942
la06	1096	1205	2068	2042	2175	3476	4793	4932	7119	9376	9504	12384
la07	1031	1230	2014	1926	2144	3245	4521	4762	6439	8835	9104	11668
la08	1062	1207	2073	1975	2146	3480	4595	4764	6521	8882	9022	11571
la09	1167	1277	2236	2187	2317	3329	5121	5259	7228	9973	10105	12541
la10	1134	1243	2174	2124	2249	3497	4978	5102	6711	9733	9861	12840
la11	1429	1538	2622	2679	2821	4239	6308	6456	8733	12369	12514	15429
la12	1281	1359	2246	2371	2487	4094	5619	5745	7745	10904	11026	13234
la13	1380	1476	2255	2597	2700	3844	6141	6245	7980	11949	12058	14831
la14	1474	1545	2583	2730	2818	4177	6622	6711	9077	12987	13076	16479
la15	1444	1660	2616	2701	2952	4101	6347	6635	8673	12418	12710	15321
la16	887	1427	2605	1593	2210	3317	3587	4315	5609	6859	7680	9873
la17	832	1205	1998	1532	1947	3119	3540	4007	5930	6907	7375	9947
la18	876	1338	2445	1556	2089	2930	3528	4148	5314	6731	7384	9075
la19	904	1313	2301	1616	2035	2902	3682	4094	5562	7065	7457	9590
la20	916	1417	2282	1645	2199	3131	3820	4380	5916	7490	7988	10293
la26	1514	1888	2826	2808	3190	4516	6529	6892	8978	12659	12951	16119
la27	1547	1964	2961	2847	3285	4618	6547	7001	8800	12618	13046	15656
la28	1539	1946	2868	2791	3233	4329	6503	6925	8664	12615	12956	15439
la29	1433	1917	3168	2591	3142	4345	5988	6617	8187	11507	12164	14360
la30	1595	2061	3112	2927	3416	4476	6906	7272	9274	13623	13853	17071
la31	2112	2438	3594	3904	4240	5364	9317	9602	11573	18187	18416	22321
la32	2241	2590	3882	4155	4535	5777	9741	10074	11969	19041	19293	23593
la33	2058	2386	3646	3846	4191	5738	9027	9373	11723	17486	17809	21123
la34	2112	2485	3371	3908	4297	6023	9157	9538	11565	17765	18105	20794
la35	2189	2599	3725	4119	4512	6251	9788	10102	12940	19338	19587	23496
la36	1284	2022	3277	2306	3140	4413	5305	6261	8273	10387	11350	14936
la37	1358	2129	3331	2426	3247	4220	5518	6343	7737	10536	11359	13481

continued on next page

Benchmark	C_D											
	N = 1			N = 2			N = 5			N = 10		
	lb.	avg.	max.	lb.	avg.	max.	lb.	avg.	max.	lb.	avg.	max.
la38	1221	1978	2995	2181	3004	4042	4954	5794	7553	9440	10218	12679
la39	1266	2018	2912	2321	3120	4396	5347	6214	8002	10434	11307	14060
la40	1278	1978	2803	2329	3057	4414	5397	6161	7905	10502	11247	13371
orb01	901	1650	2642	1592	2516	3726	3591	4802	6200	6876	8330	10281
orb02	901	1392	2211	1620	2165	3406	3697	4283	5721	7101	7693	9804
orb03	872	1716	2904	1546	2590	3754	3478	4842	6350	6603	8286	10460
orb04	936	1602	2542	1700	2436	3553	3915	4739	6556	7651	8475	10782
orb05	830	1382	2181	1498	2133	3020	3433	4170	5991	6568	7374	9588
orb06	914	1640	2593	1627	2514	3861	3648	4806	7104	6947	8337	10529
orb08	784	1544	2534	1402	2323	3450	3179	4367	5869	6117	7496	9938
orb09	881	1489	2621	1567	2279	3386	3572	4409	5851	6860	7795	9778
orb10	918	1535	2305	1636	2362	3803	3679	4555	6233	6961	7983	10282
swv01	1469	2428	3414	2710	3818	4901	6350	7719	9783	12448	14003	16896
swv02	1485	2445	3762	2733	3853	5226	6471	7845	9897	12675	14211	17122
swv03	1483	2391	3834	2729	3795	5012	6351	7689	9472	12407	13944	16265
swv04	1499	2479	3577	2751	3900	5386	6335	7791	10006	12262	13982	16404
swv05	1499	2488	3754	2744	3882	5615	6463	7855	10033	12533	14091	17243
swv06	1589	2925	3949	2880	4457	5898	6646	8585	10564	12852	15085	17874
swv07	1505	2791	3943	2703	4233	5468	6185	8065	9901	11900	14079	16023
swv08	1624	3021	4229	2966	4596	5766	6866	8843	10438	13520	15599	18570
swv09	1564	2904	3990	2849	4433	5777	6605	8536	10773	12818	15004	18664
swv10	1602	2927	4188	2910	4474	5789	6625	8628	10493	12716	15060	17708
swv11	3277	5105	6780	6177	8211	9854	14713	17040	19262	28919	31509	35028
swv12	3294	5147	6685	6243	8220	10203	14915	17155	20037	29183	31712	34906
swv13	3414	5250	6646	6518	8499	10868	15531	17766	20427	30555	33078	36985
swv14	3206	4987	6406	6047	8029	10065	14503	16727	19367	28625	31040	35305
swv15	3229	5018	6470	6097	8042	9828	14536	16739	19214	28534	30961	34068
swv16	3273	3550	4972	6169	6473	8155	14831	15057	18463	29337	29513	34540
swv17	3212	3493	4712	6078	6337	7623	14540	14738	17922	28478	28646	32907
swv18	3218	3513	4942	6096	6383	8295	14702	14928	17747	28960	29151	33916
swv19	3322	3656	4846	6249	6565	8042	14896	15148	17489	29215	29420	33864
swv20	3200	3478	4581	6077	6310	7903	14491	14638	17261	28578	28663	32525
yn1	891	1435	2011	1596	2166	2692	3640	4219	5063	6951	7527	8538
yn2	898	1465	2040	1621	2237	2816	3715	4398	5390	7150	7861	9038
yn3	882	1433	1962	1596	2186	2775	3671	4277	4922	7033	7620	9007
yn4	917	1537	2081	1656	2365	2943	3768	4599	5388	7241	8115	9537

Table 2.12: Makespan: stochastic processing times

A more interesting experiment is to study the distribution of C_D, and compare it to $C_{\max}$. The distribution of the makespan returned by algorithm FSA for the benchmark

abz7 is displayed in Figures 2-3. This is representative of the distributions observed for the other benchmarks.

Benchmark	$C_{\max}$ (N = 1)	ϵ	$Pr[(C_D - N\ C_{\max}) > f\ \epsilon]$				
			N = 1				
			f = 0.1	f = 0.15	f = 0.2	f = 0.25	f = 0.5
abz7	556	9540	0.22	0.01	0.001	0	0
ft10	631	8290	0.11	0	0	0	0
la10	958	5760	0.13	0.0001	0	0	0
swv10	1159	20385	0.17	0.0002	0	0	0

Table 2.13: Distributions for selected benchmarks: N = 1

Benchmark	$C_{\max}$ (N = 1)	ϵ	$Pr[(C_D - N\ C_{\max}) > f\ \epsilon]$				
			N = 10				
			f = 0.1	f = 0.15	f = 0.2	f = 0.25	f = 0.5
abz7	556	9540	0.891	0.691	0.031	0.001	0
ft10	631	8290	0.909	0.538	0.044	0	0
la10	958	5760	0.781	0.587	0.02	0	0
swv10	1159	20385	0.9972	0.7579	0.1648	0.0013	0

Table 2.14: Distributions for selected benchmarks: N = 10

Next we present results on the tail probabilities for four of the benchmarks: abz7, ft10, la10, and swv10; For each of the four benchmarks we compute the value of the "error term" $\epsilon = J_{\max}(2\ E[P_{\max}] + E[U_{\max}])$ of Theorem 2.4; In Tables 2.13 and 2.14, we provide the (empirical) probability that the additive error of the schedule returned by algorithm FSA exceeds $f\ \epsilon$, for various values of f.

2.5 Conclusions

In this chapter we considered the job shop problem with the makespan objective. We presented an algorithm to compute an asymptotically optimal solution, based on rounding an

optimal fluid solution. Essentially, we show that rounding an optimal fluid solution appropriately results in a schedule that incurs $O(1)$ extra cost compared to a trivial lower bound for the job shop problem; this trivial lower bound coincides with the optimal cost of the fluid relaxation. Our results imply that as the number of jobs increases, the combinatorial structure of the problem is increasingly less important, and as a result, a fluid approximation of the problem becomes increasingly exact. The results of this chapter also imply that a continuous approximation to the problem is asymptotically exact. We then discussed an extension of our algorithm to the case in which deterministic arrivals over a (suitably restricted) finite horizon. Finally, we demonstrated the efficacy of our algorithm on a variety of benchmarks chosen from the OR library.

Chief among the open problems that remain is to extend these results to the case when processing times are stochastic according to a known distribution. We presented computational results for the case when processing times were geometrically distributed, and demonstrated that the relative error is small. It would be interesting to prove a theoretical guarantee for our algorithm in this setting. A closer examination of the techniques used by Dai and Weiss [25] might be useful in this regard. Another open problem is to improve the bound on the additive error; while it appears that these techniques cannot lead to an additive error smaller than $J_{\max}(P_{\max} + U_{\max})$, we do not have concrete evidence of this fact. Of course, there could be alternative methods that lead to smaller additive error, and those are of interest as well. As we pointed out before, an additive error better than $(J_{\max} - 1)\ P_{\max}$ cannot be achieved, if we use the optimal fluid cost for comparison purposes; an interesting open problem is to determine the optimal additive error for algorithms based on fluid relaxations.

Chapter 3

Asymptotically Optimal Solutions for Minimizing Holding Costs in Job Shops

3.1 Introduction

In this chapter we consider the job shop scheduling problem with the objective of minimizing holding costs. Recent results show that for this objective, the job shop problem does not have a polynomial time approximation scheme; thus, in terms of approximability, this is a harder objective than the makespan. Our main result is an algorithm, based on fluid relaxations, that yields asymptotically optimal schedules.

As before, we have a set of I job types on J machines. Job type i consists of J_i stages (also referred to as "tasks"), each of which must be completed on a particular machine. The pair (i,k) represents the k^{th} stage of the i^{th} job, and has processing time $p_{i,k}$. Suppose that we have n_i jobs of type i. We are given non-negative weights $w_{i,k}$ for type i jobs at stage k; our objective is to find a schedule that minimizes

$$\int_{t=0}^{\infty} \sum_{i=1}^{I} \sum_{k=1}^{J_i} w_{i,k} n_{i,k}(t) dt,$$

where $n_{i,k}(t)$ is the number of type i jobs in stage k at time t. We note that our objective generalizes several well-studied special cases, including minimizing weighted completion

time of jobs ($w_{i,k} = w_i$ and is independent of k).

We impose the following restrictions on the schedule.

1. The schedule must be non-preemptive. That is, once a machine begins processing a stage of a job, it must complete that stage before doing anything else.

2. Each machine may work on at most one task at any given time.

3. For $k > 1$, stage k of a job can begin only after the completion of its $(k-1)^{\text{st}}$ stage.

Continuing our work from chapter 2, we adopt an approach based on a fluid relaxation, in which we replace discrete jobs with the flow of a continuous fluid. We then adapt the fluid synchronization algorithm of chapter 2 to round the given optimal fluid solution, and obtain a discrete schedule. The motivation for this approach comes from optimal control of multiclass queueing networks, which are stochastic and dynamic generalizations of job shops. In recent years there has been considerable progress in solving the fluid relaxation in multiclass queueing networks. Focusing on objective functions that minimize holding costs, Avram, Bertsimas, and Ricard [5] show that by using the Pontryagin maximum principle, we can find an optimal solution to the fluid relaxation explicitly. However, the description of the optimal fluid solution, while insightful for the original problem, involves the enumeration of an exponential number of cases. Luo and Bertsimas [52], building upon the work of Pullan [63], use the theory of continuous linear programming to propose a convergent numerical algorithm for the problem that is able to solve efficiently problems involving hundreds of machines and job types; we use this algorithm in solving the fluid relaxations throughout this work.

Results. We describe an efficient algorithm to round an optimal fluid solution such that the resulting schedule is asymptotically optimal; the specific asymptotics we consider is a series of job shop problems in which the number of type i jobs is $\alpha_i N$, and $N \to \infty$. Essentially, we show that rounding an optimal fluid solution appropriately results in a schedule that incurs $O(N)$ extra cost compared to the optimal cost of the fluid job shop; this, coupled with the fact that the optimal fluid cost is $O(N^2)$, results in an asymptotically optimal schedule. Consider the classical job shop scheduling problem, which has exactly one job of each type. In this case, the combinatorial structure of the job scheduling problem makes the problem very complicated to solve. Interestingly, the results of this chapter imply

that as the number of jobs increases, the combinatorial structure of the problem becomes increasingly less important, and as a result, a fluid approximation of the problem becomes increasingly exact.

Related work. The job shop problem with weighted completion time objective has received little attention in the research literature. In contrast, fluid relaxations with holding costs objective have been studied extensively, because this is a natural formulation considered by queueing theorists. We provide a very brief overview of some of these results, which will also serve to place our results in perspective.

Fluid relaxations have been the subject of intensive research during the last decade. An important breakthrough was achieved by Dai [22], who established that global stability analysis of multiclass queueing networks is possible by focusing on their deterministic counterparts. Spurred by the success of these ideas in analyzing stability, there has been a growing literature in finding near-optimal scheduling policies using fluid relaxations; Papers that address this issue include the ones by Avram, Bertsimas and Ricard [5], Atkins and Chen [4], Chen and Yao [21], Eng, Humphrey and Meyn [27], Meyn [55], and Meyn [56]. All of these papers formulate the fluid relaxation, find a fluid solution (optimal or otherwise), and then heuristically interpret the fluid solution to derive a discrete policy; except for Meyn [56], none of these papers presents any performance analysis of the derived discrete policy (except in fairly restricted settings). Meyn [56] discusses a policy-iteration algorithm and demonstrates its (quicker) convergence to optimality when initiated with an optimal fluid solution. Although this establishes an intimate connection between the large-state behavior of a multiclass queueing network and its fluid model, this property does not seem to be directly usable in designing a good policy. In recent work, Maglaras [53], building on the BIGSTEP approach of Harrison [38], proposed a class of policies based on solving fluid relaxations repeatedly. For any policy in this class, the nominal length of a review period is computed; based on the queue-lengths at the beginning of each review period, the length of the nominal review period, and the planned "safety-stocks" for each class, a fluid-type problem is solved. A solution to this fluid-type problem is then used to derive a processing plan for that review period. The next review of the system is conducted as soon as this processing plan is complete, which could be different from the next nominal review time instant because of the stochastic nature of the processing times. This is an example

of a "discrete-review" policy. In his paper, Maglaras [53] proves the stability of a fairly broad range of discrete-review policies, and establishes their *fluid-scale asymptotic optimality*. This criterion is essentially what we use in §3.3, where we consider job shop problems with arrivals. An important distinction is that Maglaras considers a steady-state problem, but proves performance guarantees of a transient nature. Our work, in contrast, considers a transient problem to begin with. Thus, when restricted to transient problems, it may be possible to use results of Maglaras [53] to obtain asymptotically optimal schedules. (The scheme presented in Maglaras [53] appears to use the fact that effective arrival rates of each class is strictly positive, and so has to be modified suitably to address the job shop problem without arrivals). However, our approach has two distinct advantages: first, we provide an explicit rate of convergence to optimality; and second, we solve the fluid relaxation *once*, and do not resolve it at intermediate points in time.

An interesting recent development is the work of Queyranne and Sviridenko [64] in which they consider approximation algorithms for shop scheduling problems with minsum objective. Their main result is the following: if there exists a polynomial-time algorithm for a class of multiprocessor job shop problems that guarantees a makespan no larger than ϵ times the trivial lower bound (the so-called *congestion-dilation* bound), then they describe a polynomial-time algorithm for minimizing the weighted completion time that is within a factor of 8ϵ of the optimum. (In fact, their algorithm works for a generalization in which release dates are also given.) Their algorithm involves use of the approximation scheme for the makespan objective. Note that the polynomial-time approximation schemes for the makespan objective do not satisfy always satisfy the hypothesis of their statement: the work of Queyranne and Sviridenko [64] requires a makespan guarantee that is within a factor of ϵ of a *lower bound*, not the optimal makespan itself. In fact, recent results of Hoogeveen, Schuurman and Woeginger [39] show that the job shop problem with the objective of minimizing weighted completion time does not have a polynomial-time approximation scheme.

Structure of the chapter. In §3.2, we consider the holding costs objective, and describe an algorithm that provides an asymptotically optimal schedule. These results are extended in §3.3 to a model in which deterministic arrivals occur over a (suitably restricted) finite horizon. In §3.4, we present computational results on a variety of job shop instances from the OR library. §3.5 contains some concluding remarks.

3.2 Holding costs

This section considers the job shop problem with the objective of minimizing holding costs, and is structured as follows: In §3.2.1, we define the job shop scheduling problem formally, and discuss the notation. In §3.2.2, we describe the fluid relaxation for the job shop scheduling problem. and briefly discuss methods for its solution. In §3.2.3, we describe an analogous *fluid synchronization algorithm* (FSA-HC) that yields an asymptotically optimal schedule for the original discrete job shop scheduling problem.

3.2.1 Problem Formulation and Notation

In this section we turn to a different objective—that of minimizing holding costs. The set up is exactly the same as before except that each class (i,k) job incurs a cost $w_{i,k}$ for each unit of time spent in the system. As before, we are given n_i jobs of type i, and our objective is to process all these jobs so as to minimize the total holding costs incurred; a formal description of our objective follows.

Let $n_{i,k}(t)$ be the number of (i,k) jobs at machine σ_k^i at time t. Let $w_{i,k}$ be non-negative integer holding cost rates associated with (i,k) jobs. Our objective is to find a scheduling policy that minimizes

$$\int_{t=0}^{\infty} \sum_{i=1}^{I} \sum_{k=1}^{J_i} w_{i,k} n_{i,k}(t).$$

As before, we consider a continuous relaxation of this problem, in which the number of (i,k) jobs is allowed to assume arbitrary non-negative real values, and machines are allowed to work simultaneously on several types of jobs. This relaxation is called the fluid job-shop scheduling problem.

3.2.2 The Fluid Job shop scheduling problem

In this section we describe a continuous relaxation of the job-shop scheduling problem. In a fluid job-shop, there are J machines $\sigma_1, \sigma_2, \ldots, \sigma_J$ and I job types. Each job type is specified by the sequence of machines $\sigma_k^i, k = 1, 2, \ldots J_i$ it has to be processed on; the processing time of a type i job on machine σ_k^i is a positive real number $p_{i,k}$. For convenience, we let $\mu_{i,k} = 1/p_{i,k}$; we can think of $\mu_{i,k}$ as the rate at which machine σ_k^i processes (i,k) jobs. As before we refer to type i jobs which have been processed on machines $\sigma_1^i, \sigma_2^i, \ldots, \sigma_{k-1}^i$

but not on machine σ_k^i as jobs of class (i,k) or (i,k) jobs. We let $x_{i,k}(t)$ to be the number of jobs of class (i,k) at time t. The number of type i jobs initially present, $x_{i,1}(0)$, is also denoted by x_i and can take arbitrary non-negative values; we assume that $x_{i,k}(0) = 0$ for $k > 1$. In contrast to the discrete problem, the number of (i,k) jobs at time t can assume arbitrary non-negative real values; for that reason, we think of this as the fluid level of class (i,k) at time t. Let $T_{i,k}(t)$ be the total amount of time machine σ_k^i works on class (i,k) jobs in the interval $[0,t)$. We first present all of the constraints.

$$x_{i,1}(t) = x_i - \mu_{i,1}T_{i,1}(t), \qquad i = 1,2,\ldots,I,\ t \geq 0, \tag{3.1}$$

$$x_{i,k}(t) = \mu_{i,k-1}T_{i,k-1}(t) - \mu_{i,k}T_{i,k}(t), \qquad k = 2,\ldots,J_i,\ i = 1,2,\ldots,I,\ t \geq 0, \tag{3.2}$$

$$0 \leq \sum_{(i,k)\in\sigma_j} (T_{i,k}(t_2) - T_{i,k}(t_1)) \leq t_2 - t_1, \forall\ t_2 > t_1,\ t_1,t_2 \geq 0,\ j = 1,2,\ldots,J, \tag{3.3}$$

$$x_{i,k}(t) \geq 0,\ \ T_{i,k}(t) \geq 0. \tag{3.4}$$

Constraints (3.1) and (3.2) capture the dynamics of the system. These equations merely state that the fluid level of class (i,k) at time t is the initial fluid level plus the amount of fluid that has arrived from class $(i,k-1)$ by time t minus the amount of class (i,k) fluid that has been processed by machine σ_k^i by time t. Constraints (3.4) reflect the fact that the fluid level of class (i,k) and the amount of time allocated by machine σ_k^i to class (i,k) are non-negative. Constraint (3.3) is the capacity constraint for each machine—the total amount of time devoted to processing by machine j in an interval $[t_1,t_2)$ cannot exceed the length of the interval $t_2 - t_1$. Our objective function for the fluid job shop is

$$\sum_{i=1}^{I}\sum_{k=1}^{J_i}\int_0^\infty w_{i,k}x_{i,k}(t)dt.$$

The problem of finding an efficient algorithm to solve the fluid control problem is still open. However, based on several structural properties for this class of problems (see Anderson and Nash [3]), discretization-based methods have been proposed by Pullan [63], and Luo & Bertsimas [52] to solve these problems in practice. An implementation of the algorithm due to Luo and Bertsimas performs very well in practice, which we use in our computational study.

A key property of the fluid job shop problem that we shall make use of extensively is stated as Proposition 3.1; its proof can be found in Anderson & Nash [3].

Proposition 3.1 *There exists an optimal solution for the fluid job shop scheduling problem such that $x(t)$ is piecewise linear with a finite number of pieces.*

Proposition 3.1 guarantees the existence of an optimal fluid solution with piecewise constant control. This property enables us to use the machinery developed for the makespan objective repeatedly to obtain asymptotically optimal schedules.

3.2.3 The fluid synchronization algorithm

In this section we describe an algorithm to discretize an optimal fluid solution for the holding costs objective. Our discretization algorithm is based on repeated applications of a variant of the algorithm FSA developed for the makespan objective. Two key points to note about algorithm FSA are:

- FSA can be applied to any fluid solution as long as the fluid relaxation serves at constant rate.
- Since the analysis of §2.2.3 yielded job-by-job guarantees, we expect an algorithm based on FSA to yield good schedules for *any* objective based on completion times of jobs.

However, there is one important difficulty in using FSA directly. For the holding costs objective, processing a job "too soon" may be just as bad as processing a job "too late." For example, consider the n^{th} (i,k) job, and suppose $w_{i,k} \ll w_{i,k+1}$. If $DS_{i,k}(n) \ll NS_{i,k}(n)$, then this job is processed sooner than necessary at stage k, thereby reaching stage $(k+1)$ substantially earlier, and, therefore, accumulating holding costs at a much higher rate. This is in sharp contrast to the makespan objective, where there was no incentive for a machine to idle. We overcome this difficulty by modifying our definition of when a job becomes *available*. This variant of FSA is called the *Revised Fluid Synchronization Algorithm* ($RFSA$), and is as follows.

Description of RFSA. The only difference between FSA and $RFSA$ is the definition of when a job becomes available. For each job in the discrete network, we assign a status; the status of the n^{th} (i,k) job at time t is:

- *not available*, if $0 \leq t < \max\{DC_{i,k-1}(n), NS_{i,k}(n)\}$;

- *available*, if $\max\{DC_{i,k-1}(n), NS_{i,k}(n)\} \leq t < DS_{i,k}(n)$;
- *in progress*, if $DS_{i,k}(n) \leq t < DC_{i,k}(n)$;
- *departed*, if $t \geq DC_{i,k}(n)$.

Scheduling decisions in the discrete network are made at well-defined *scheduling epochs.* Scheduling epochs for machine σ_j are instants of time at which either some job completes service at σ_j and there is at least one *available* job at σ_j, or some job *becomes* available at an idle machine σ_j. Suppose machine σ_j has a scheduling epoch at time t. Among all the *available* jobs at machine σ_j, our algorithm schedules the one with the *smallest nominal start time.* This scheduling decision, in turn, determines the nominal start time of this job at its next stage. The key difference between FSA and $RFSA$ is thus in the definition of *available* jobs. (In FSA job n of class (i,k) is declared as available at time $DC_{i,k-1}(n)$.)

Elementary results for RFSA. The following theorem relates the fluid and discrete completion times of a job when the discrete schedule is computed using the $RFSA$.

Theorem 3.1 *Let $DC_{i,k}(n)$ be the completion time of the n^{th} (i,k) job in the discrete schedule computed by RFSA, and let $FC_{i,k}(n)$ be its completion time in the fluid relaxation. Then,*

$$DC_{i,k}(n) \leq FC_{i,k}(n) + \sum_{l=1}^{k}(2\ P_{\sigma_l^i} + U_{\sigma_l^i}). \tag{3.5}$$

Proof. We prove this by an argument similar to the one used to prove Theorem 2.3. The only difference is that in the $RFSA$, no job is scheduled to start prior to its nominal start time. In terms of the potential function defined in §2.2.3, the $RFSA$ does not schedule any job that has strictly negative potential. A careful examination of all of the results of §2.2.3 reveals that none of the proofs depends on how the algorithm treats jobs with strictly negative potential. In particular, all of the results hold if a machine idles whenever all of the jobs queued for service have strictly negative potential. ■

The following theorem is obvious from the definition of $RFSA$.

Theorem 3.2 *Let $DC_{i,k}(n)$ be the completion time of the n^{th} (i,k) job in the discrete schedule computed by RFSA, and let $FC_{i,k}(n)$ be its completion time in the fluid relaxation. Then,*

$$DS_{i,k}(n) \geq FC_{i,k}(n-1). \tag{3.6}$$

In particular,

$$DC_{i,k}(n) \geq FC_{i,k}(n-1). \tag{3.7}$$

Proof. By definition,

$$\begin{aligned} DS_{i,k}(n) &\geq NS_{i,k}(n) \\ &\geq NS_{i,k}(n-1) + \frac{C_{\max}}{n_i} \\ &\geq FS_{i,k}(n-1) + \frac{C_{\max}}{n_i} \\ &= FC_{i,k}(n-1). \end{aligned}$$

■

In discretizing an optimal fluid solution to obtain an asymptotically optimal schedule for the problem of minimizing holding costs, we will make use of the *RFSA* repeatedly. As mentioned earlier, we will consider the following asymptotics, which we state explicitly as Assumption 1.

- **Assumption 1:** The number of initial jobs of type i is $\alpha_i \cdot N$. In our asymptotics, we let $N \to \infty$. Thus we consider a sequence of job-shop problems in which N varies, but all other quantities remain the same.

Without loss of generality, we also assume that the queue-lengths at the breakpoints are integral. If this assumption is violated in a given optimal fluid solution, we obtain a near-optimal fluid solution by rounding the queue-lengths down; such a rounding procedure affects at most one job of each class in each piece, resulting in $O(1)$ extra cost, which is asymptotically negligible.

Let $Z_{\mathrm{F}}(N)$ denote the cost of the given optimal fluid solution when the number of initial jobs of type i is $\alpha_i \cdot N$; similarly, let $Z_{\mathrm{JS}}(N)$ denote the cost of the optimal solution to the corresponding discrete job-shop problem.

A lower bound. We first establish a useful relationship between $Z_{\mathrm{F}}(N)$ and $Z_{\mathrm{JS}}(N)$. Ideally we would like to establish that $Z_{\mathrm{F}}(N)$ is a lower bound on the cost of an optimal job-shop schedule for the discrete network. While we have been unable to establish this result in general, we can prove the following theorem.

Theorem 3.3

(a) $Z_{\mathrm{F}}(N) = C\,N^2$.

(b) $Z_{\mathrm{JS}}(N) \geq Z_{\mathrm{F}}(N) - O(N)$.

Proof:

(a) This follows immediately from the formulation of the fluid relaxation. More formally, suppose we have a solution to the fluid relaxation for $N = 1$ (i.e. $n_i = \alpha_i$). This solution consists of the "allocation" variables $T^1_{i,k}(t)$, with the corresponding "queue-length" variables $x^1_{i,k}(t)$. We can use these to find a solution to the fluid relaxation when $n_i = \alpha_i \cdot N$ as follows. We set

$$T^N_{i,k}(N \cdot t) = N\,T^1_{i,k}(t),$$

$$x^N_{i,k}(N \cdot t) = N\,x^1_{i,k}(t).$$

(b) To prove this part, we "fluidize" the optimal solution to the job shop problem. Consider any feasible schedule to the discrete job-shop problem. We can "convert" this schedule into a schedule for the fluid network by processing the job "continuously." This is illustrated in Figure 3-1.

The feasibility of this schedule is immediate from the feasibility of the schedule for the discrete network. The extra cost incurred is

$$\sum_{i=1}^{I}\sum_{k=1}^{J_i} p_{i,k}\frac{(w_{i,k+1} - w_{i,k})}{2}\,\alpha_i N \ .$$

■

For the special case in which all of the weights for a particular job-type are equal (i.e. $w_{i,k}$ is independent of k), Z_F is indeed a lower bound for Z_{JS}. (We conjecture that this is true even for arbitrary weights.) In the rest of this paper we drop the "N" and use Z_{F} and Z_{JS} instead; we emphasize however that Z_{F} and Z_{JS} depend on N.

Description of Algorithm FSA-HC. We provide a complete description of algorithm $FSA - HC$; in doing so, we shall also introduce some notation that will be used in our analysis.

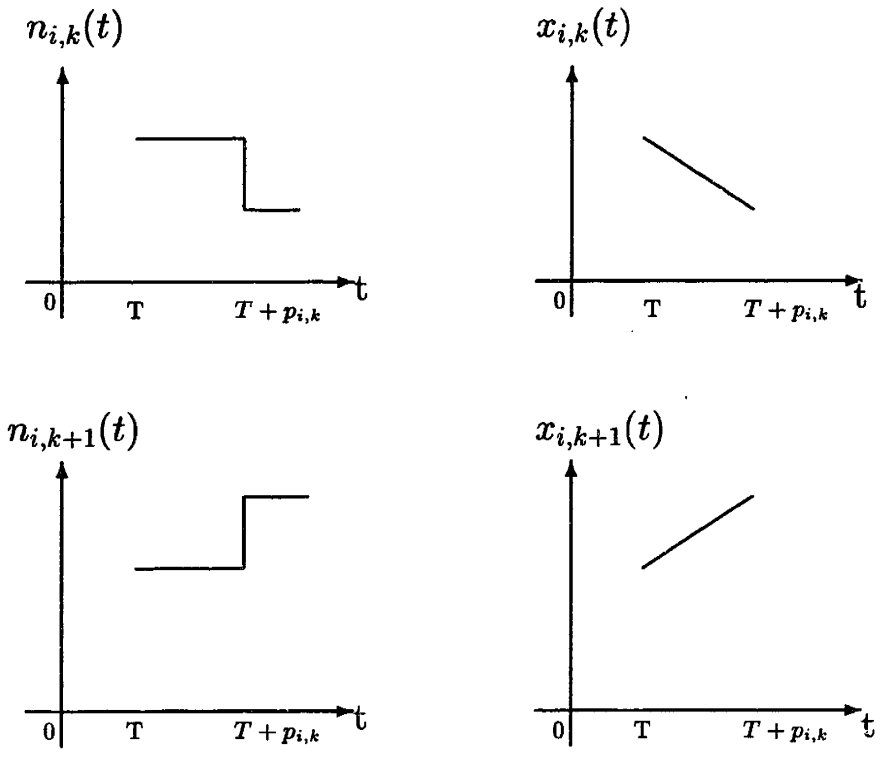

Figure 3-1: "Fluidizing" a discrete job shop schedule

Note that by Proposition 3.1, there is always an optimal fluid solution such that $T_{i,k}(t)$ is piecewise linear, and has a finite number of pieces. For this solution, we define

$$u_{i,k}(t) = \frac{d\, T_{i,k}(t)}{dt}. \tag{3.8}$$

Since $T_{i,k}(t)$ is piecewise linear, Eq. (3.8) does not determine $u_{i,k}(t)$ at the (finitely many) breakpoints; at each of these breakpoints, we set

$$u_{i,k}(t) = u_{i,k}(t^+).$$

Clearly, $u_{i,k}(t)$ can be interpreted as the instantaneous fraction of effort allocated to class (i,k) jobs by machine σ^i_k at time t. We shall find it convenient to work with $u_{i,k}(t)$ instead of $T_{i,k}(t)$.

The main idea of algorithm FSA-HC is the following: Suppose the optimal fluid solution has R pieces. Let T^i denote the time at which piece i ends (also the time at which piece $(i+1)$ begins), and let $T^0 = 0$ be the time origin. Thus, piece i starts at time T^{i-1} and ends at time T^i, for $1 \leq i \leq R$. Also, the vector of (fluid) queue-lengths at T^i is an integer for

all i. We discretize each piece separately using the $RFSA$ described earlier in this section: That is, we find times $\hat{T}^0 = 0, \hat{T}^1, \hat{T}^2, \ldots, \hat{T}^R$ such that the vector of queue-lengths at $\hat{T}^i$ in the discrete network is exactly the same as the vector of queue-lengths at T^i in the fluid relaxation. We now evaluate and compare the cost of each piece, and show that the discretization error accumulated over all the R pieces is negligible compared to the total fluid cost.

We now make a few definitions that will be useful in a formal description of our algorithm.

- **Length of piece r (L^r):** $L^r = T^r - T^{r-1}$.
- **Fluid queue-lengths ($x^r_{i,k}$):** $x^r_{i,k}$ is defined as the queue-length of (i,k) jobs in the fluid relaxation at the end of the r^{th} piece. In other words, $x^r_{i,k} = x_{i,k}(T^r)$.
- **Number of jobs processed in piece r ($y^r_{i,k}$):** $y^r_{i,k}$ is defined as the number of (i,k) jobs processed by the fluid relaxation in piece r; clearly, $y^r_{i,k} = \mu_{i,k} u^r_{i,k} L^r$.

We need an additional definition before we can describe algorithm FSA-HC. Recall that in the makespan objective, the fluid solution was constant, and that all of the jobs required to be processed were in their corresponding first stages. The latter property is true for the first piece in the holding costs objective, but may be violated for the subsequent pieces. Hence, we need to enhance our definition of "job types." This naturally leads to the definition of auxiliary variables discussed next.

We define variables z^r_{ikl} as the number of type i jobs that move from stage k to stage l during the r^{th} piece of the fluid relaxation; We use z^r_{ikE} to denote the number of type i jobs that start at stage k, but depart the network during piece r of the fluid relaxation. Since we assume that each machine processes jobs of each class in FCFS manner, the z^r_{ikl} are uniquely defined based on the fluid queue-lengths of type i jobs at time T^{r-1} and time T^r. We first illustrate this on a small example, followed by an algorithm to compute z^r_{ikl}.

Consider the following example (see Figure 3-2): There are four machines and two types of jobs. Type 1 jobs 1 require service at machines 1, 2, 3 and 4 in that order; Type 2 jobs require service at machines 4, 3, 2 and 1 in that order. The processing requirements and the holding cost rates at the various stages for each job-type are shown in Table 3.1. Suppose we have 250 jobs of type 1 and 500 jobs of type 2 initially. The fluid solution shown in Table 3.2,

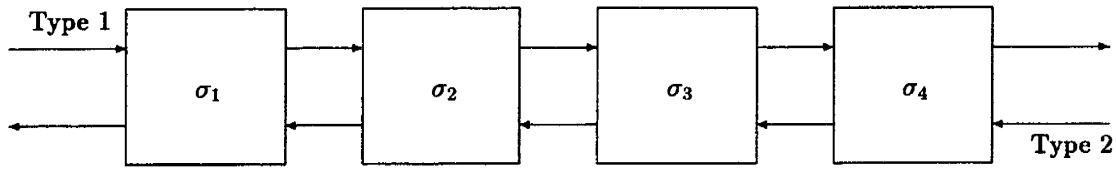

Figure 3-2: A four station network.

	Type 1	Type 2
Holding costs	(4, 1, 2, 1)	(4, 1, 2, 1)
Processing times	(1, 8, 4, 2)	(2, 4, 1, 8)

Table 3.1: Holding costs and processing times

while not optimal, has objective function value close to the optimal fluid cost; moreover the vector of queue-lengths at the end-points of each piece is integral. The auxiliary variables associated with this fluid solution are shown in Table 3.3. In Table 3.3, the entry E refers to the external environment: this just indicates that the corresponding jobs leave the network.

The auxiliary variables define the requirements for each piece and captures exactly the dynamics of the r^{th} piece of the fluid solution. An algorithm to compute the auxiliary variables z^r_{ikl} using the vector of fluid queue-lengths at time T^{r-1} and T^r is shown in Figure 3-3.

The discretization algorithm for holding costs can thus be described as follows.

Type 1	Type 2
(250, 0, 0, 0)	(500,0,0,0)
(0, 218, 0, 32)	(375, 125, 0, 0)
(0, 136, 0, 114)	(0, 406, 0, 0)
(0, 104, 0, 0)	(0, 370, 0, 0)
(0, 0, 0, 0)	(0, 250, 0, 0)
(0, 0, 0, 0)	(0, 0, 0, 0)

Table 3.2: A near-optimal fluid solution

	Type	Origin stage	Destination stage	Number of jobs
Piece 1	1	1	2	218
	1	1	4	32
	2	1	2	125
Piece 2	1	2	4	82
	2	1	2	375
	2	2	E	94
Piece 3	1	2	E	32
	1	4	E	114
	2	2	E	36
Piece 4	1	2	E	104
	2	2	E	120
Piece 5	2	2	E	250

Table 3.3: Auxiliary variables for Fluid solution of Table 3.2

For $i = 1, 2, \ldots, I$:

For $k = 1, 2, \ldots, J_i$:

$$outflow(i,k;r) \;=\; \min\left\{\sum_{p=1}^{k}(x_{i,p}^{r-1} - x_{i,p}^{r}), x_{i,k}^{r-1}\right\}.$$

$$inflow(i,k;r) \;=\; x_{i,k}^{r} - x_{i,k}^{r-1} + outflow(i,k;r).$$

$$outflow(i,E;r) \;=\; 0;\;\; inflow(i,E;r) = \sum_{p=1}^{J_i}(x_{i,p}^{r-1} - x_{i,p}^{r}).$$

For $k = 1, 2, \ldots, J_i$:

For $l = k+1, k+2, \ldots, J_i, E$:

$$z_{ikl}^{r} \;=\; \min\Big\{outflow(i,k;r), inflow(i,l;r)\Big\}.$$

$$outflow(i,k;r) := outflow(i,k;r) - z_{ikl}^{r};$$

$$inflow(i,k;r) := inflow(i,k;r) - z_{ikl}^{r};$$

Figure 3-3: Computing values of auxiliary variables

For $r = 1, 2, \ldots R$:

Compute the auxiliary variables z^r_{ikl} for piece r;

Discretize the r^{th} piece using $RFSA$, with input $\{z^r_{ikl}, L^r\}$.

Discretization Algorithm for Holding Costs (FSA-HC)

As discussed before, this algorithm discretizes each piece separately. While discretizing piece r, L^r plays the role of $C_{\max}$, job types are indexed by ikl, and the auxiliary variables z^r_{ikl} play the role of the n_i.

Remark. We note that the jobs in the discretized solution may no longer be processed in FCFS order. To see this, consider a solution in which $z_{ikl} \gg z_{ik'l}$ for some k, k' such that $k < k' < l$. In this case, in our interpretation of the fluid solution, type ikl jobs are processed at a much faster rate than the type $ik'l$ jobs, and so it is possible for some job at stage k to reach the destination l prior to some job at stage k'.

Analysis and Results. We now calculate the cost of the discrete schedule and compare it to that of the fluid relaxation. Our analysis will be on a job-by-job basis. The outline of the analysis is as follows.

1. Focus on (i, k, l) jobs in piece r; evaluate the cost of these jobs in the discrete network and in the fluid relaxation.
2. Find an expression for an upper bound on the error in the r^{th} piece.
3. Argue that the total error, summed over all pieces, is negligible compared to the cost of the fluid solution.

Since we have established that the cost of an optimal fluid solution is an asymptotic lower bound on the optimal achievable cost of the discrete network (Theorem 3.3), we are done.

For convenience, we define z^r_{ikk} to be the number of type i jobs that remain at stage k during piece r. Clearly,

$$z^r_{ikk} = x^{r-1}_{i,k} - outflow(i,k;r),$$

where $outflow(i,k;r)$ is defined in Figure 3-3.

Lemma 3.1 *The cost of the r^{th} piece of the fluid relaxation is given by*

$$C^r_f = \sum_{i=1}^{I}\sum_{k=1}^{J_i}\sum_{l=k}^{J_i}\left(\frac{w_{i,k}z^r_{ikl}L^r}{2} + \frac{w_{i,l}z^r_{ikl}L^r}{2}\right). \tag{3.9}$$

To evaluate the cost of the r^{th} piece in the fluid network, observe that

- The inventory level of (i,k,l) jobs at stage k *decreases linearly* from z^r_{ikl} to zero.
- The inventory level of (i,k,l) jobs at stage l *increases linearly* from zero to z^r_{ikl}.
- All of the intermediate stages (if any) have zero inventory level for (i,k,l) jobs.

Thus, in the r^{th} piece of the fluid solution, the cost incurred by (ikl,k) jobs is

$$\frac{w_{i,k}z^r_{ikl}L^r}{2},$$

and the cost incurred by (ikl,l) jobs is

$$\frac{w_{i,l}z^r_{ikl}L^r}{2}.$$

Observing that jobs of type ikl do not incur cost at any other stage, we see that the cost of type ikl jobs in the r^{th} piece of the fluid relaxation is

$$\left(\frac{w_{i,k}z^r_{ikl}L^r}{2} + \frac{w_{i,l}z^r_{ikl}L^r}{2}\right). \tag{3.10}$$

Summing Eq. (3.10) over all possible job types, we obtain Eq. (3.9). ∎

We now evaluate the cost of type i jobs in the discretized solution corresponding to piece r. For convenience, we shift the origin so that $T^{r-1} = 0$, and so $T^r = L^r$.

Lemma 3.2 *The cost of the r^{th} piece in the discrete network is given by*

$$C^r_d = \sum_{i=1}^{I}\sum_{k=1}^{J_i}\sum_{l=k}^{J_i} C^r_d(ikl), \tag{3.11}$$

where

$$\begin{aligned} C_d^r(ikl) &= w_{i,k}L^r\left(\frac{z_{ikl}^r+1}{2}\right) + w_{i,k}z_{ikl}^r(2\ P_{\max}+U_{\max}) \\ &\quad + w_{i,l}L^r\left(\frac{z_{ikl}^r+1}{2}\right) + w_{i,l}z_{ikl}^r J_{\max}(2\ P_{\max}+U_{\max}) \\ &\quad + \sum_{n=1}^{z_{ikl}^r}\sum_{p=k+1}^{l-1} w_{i,p}\left(\frac{L^r}{z_{ikl}^r} + (p-k+1)\ (2P_{\max}+U_{\max})\right). \end{aligned} \tag{3.12}$$

Proof: The cost of the r^{th} piece in the discrete network can be computed as follows. We focus on jobs of type ikl. For convenience, we renumber these jobs, if necessary, so that the jobs of type ikl in piece r are numbered $1, 2, \ldots, z_{ikl}^r$. For $k \le p \le l$, recall that $DC_{ikl,p}(n)$ is the completion of the n^{th} type ikl job at stage p. Clearly, the cost of type ikl jobs in piece r is given by

$$\sum_{n=1}^{z_{ikl}^r} w_{i,k}DC_{ikl,k}(n) + \sum_{n=1}^{z_{ikl}^r} w_{i,l}(\hat{L}_r - DC_{ikl,l-1}(n)) + \sum_{n=1}^{z_{ikl}^r}\sum_{p=k+1}^{l-1} w_{i,p}(DC_{ikl,p}(n) - DC_{ikl,p-1}(n)). \tag{3.13}$$

We will now simplify Eq. (3.13). To that end, we evaluate each of the three terms in Eq. (3.13) separately. First, consider the last term in Eq. (3.13). Using Theorem 2.3 for type ikl jobs in the r^{th} piece, we conclude that

$$DC_{ikl,p}(n) \ \le\ FC_{ikl,p}(n) + (p-k+1)\ (2\ P_{\max}+U_{\max}), \tag{3.14}$$

for $k \le p \le l$. Also, by definition, for any p such that $k \le p \le l$,

$$FC_{ikl,p}(n) \ =\ n\frac{L^r}{z_{ikl}^r}. \tag{3.15}$$

Combining Eqs. (3.14) and (3.15), we obtain

$$DC_{ikl,p}(n) \ \le\ n\frac{L^r}{z_{ikl}^r} + (p-k+1)\ (2\ P_{\max}+U_{\max}), \qquad k \le p \le l. \tag{3.16}$$

From Theorem 3.2, we obtain

$$DC_{ikl,p}(n) \ \ge\ (n-1)\frac{L^r}{z_{ikl}^r}. \tag{3.17}$$

Using Eqs. (3.16) and (3.17), we obtain

$$\begin{aligned}\sum_{n=1}^{z_{ikl}^r}\sum_{p=k+1}^{l-1} w_{i,p}(DC_{ikl,p}(n) - DC_{ikl,p-1}(n)) &\leq \sum_{n=1}^{z_{ikl}^r}\sum_{p=k+1}^{l-1} w_{i,p}\Big(FC_{ikl,p}(n) - FC_{ikl,p-1}(n-1) \\ &\qquad +(p-k+1)\,(2P_{\max}+U_{\max})\Big) \\ &= \sum_{n=1}^{z_{ikl}^r}\sum_{p=k+1}^{l-1} w_{i,p}\Big(\frac{L^r}{z_{ikl}^r}+(p-k+1)\,(2P_{\max}+U_{\max})\Big).\end{aligned} \tag{3.18}$$

We next consider the second term of Eq. (3.13). From Theorem 2.4, we know that the discretization of the r^{th} piece finishes at time $\hat{L}^r$, such that

$$\hat{L}^r \leq L^r + J_{\max}(2\,P_{\max}+U_{\max}). \tag{3.19}$$

Using Eqs. (3.19) and (3.17), we obtain

$$\begin{aligned}\sum_{n=1}^{z_{ikl}^r} w_{i,l}(\hat{L}_r - DC_{ikl,i-1}(n)) &\leq \sum_{n=1}^{z_{ikl}^r} w_{i,l}\Big(L_r + J_{\max}(2\,P_{\max}+U_{\max}) - \frac{(n-1)L^r}{z_{ikl}^r}\Big) \\ &= w_{i,l}z_{ikl}^r L^r + w_{i,l}z_{ikl}^r J_{\max}(2\,P_{\max}+U_{\max}) - w_{i,l}\frac{L^r}{z_{ikl}^r}\sum_{n=1}^{z_{ikl}^r}(n-1) \\ &= w_{i,l}L^r\Big(\frac{z_{ikl}^r+1}{2}\Big) + w_{i,l}z_{ikl}^r J_{\max}(2\,P_{\max}+U_{\max}).\end{aligned} \tag{3.20}$$

Finally, we consider the first term of Eq. (3.13). Using Eq. (3.16), we obtain

$$\begin{aligned}\sum_{n=1}^{z_{ikl}^r} w_{i,k}DC_{ikl,k}(n) &\leq \sum_{n=1}^{z_{ikl}^r} w_{i,k}(FC_{ikl,k}(n) + (2P_{\max}+U_{\max})) \\ &= \sum_{n=1}^{z_{ikl}^r} w_{i,k}\Big(\frac{nL^r}{z_{ikl}^r}+(2P_{\max}+U_{\max})\Big) \\ &= w_{i,k}L^r\Big(\frac{z_{ikl}^r+1}{2}\Big) + w_{i,k}z_{ikl}^r(2\,P_{\max}+U_{\max}).\end{aligned} \tag{3.21}$$

The cost of type ikl jobs in the r^{th} piece in the discrete network is obtained by adding Eqs. (3.18)-(3.21), which yields Eq. (3.12). ■

We are now ready to prove that algorithm FSA-HC yields an asymptotically optimal schedule.

Theorem 3.4 *Consider a job shop scheduling problem with I job types and J machines $\sigma_1, \sigma_2, \ldots, \sigma_J$. Given initially N α_i jobs of type $i = 1, 2, \ldots, I$, the algorithm $FSA - HC$ produces a schedule with cost $Z_{\mathrm{D}}(N)$ such that*

$$Z_{\mathrm{D}}(N) \leq Z_{\mathrm{F}}(n) + O(N). \tag{3.22}$$

In particular,

$$\frac{Z_{\mathrm{D}}(N)}{Z_{\mathrm{JS}}(N)} \to 1, \tag{3.23}$$

as

$$N \to \infty.$$

Proof: From Eqs. (3.9) and (3.11), we have

$$\begin{aligned} C_d^r - C_f^r \leq \sum_{i=1}^{I} \sum_{k=1}^{J_i} \sum_{l=k}^{J_i} \Bigg\{ & \frac{w_{i,k} L^r}{2} + w_{i,k} z_{ikl}^r (2\ P_{\max} + U_{\max}) \\ & + \frac{w_{i,l} L^r}{2} + w_{i,l} z_{ikl}^r J_{\max} (2\ P_{\max} + U_{\max}) \\ & + \sum_{n=1}^{z_{ikl}^r} \sum_{p=k+1}^{l-1} w_{i,p} \left(\frac{L^r}{z_{ikl}^r} + (p - k + 1)\ (2P_{\max} + U_{\max}) \right) \Bigg\}. \end{aligned}$$

From the proof of part (a) of Theorem 3.3, the terms z_{ikl}^r, and L^r all vary linearly with N. Thus,

$$C_d^r - C_f^r \leq AN,$$

for some (large enough) constant A. Thus,

$$\begin{aligned} Z_{\mathrm{D}} - Z_{\mathrm{F}} &= \sum_{r=1}^{R} C_d^r - C_f^r \\ &\leq ARN, \end{aligned}$$

which establishes $Z_{\mathrm{D}} = Z_{\mathrm{F}} + O(N)$, since R is also a constant. From part (b) of Theorem 3.3, we have $Z_{\mathrm{F}} \leq Z_{\mathrm{JS}} + O(N)$. Thus,

$$\frac{Z_{\mathrm{D}}}{Z_{\mathrm{JS}}} \leq \frac{Z_{\mathrm{D}}}{Z_{\mathrm{F}}} \frac{Z_{\mathrm{F}}}{Z_{\mathrm{JS}}}$$

$$\leq \frac{Z_{\mathrm{F}} + O(N)}{Z_{\mathrm{F}}} \frac{Z_{\mathrm{JS}} + O(N)}{Z_{\mathrm{JS}}},$$

from which the result follows. ■

Remark: We note that any algorithm that uses the fluid relaxation will incur $O(N)$ error in the worst case. For example, consider a single machine with N jobs, with $w = p = 1$. The cost of an optimal discrete schedule is $N(N+1)/2$, but the optimal fluid cost is $N^2/2$.

3.3 Holding costs with Deterministic Arrivals

This section generalizes the results of §3.2 to a model in which external arrivals are permitted over a (suitably restricted) finite horizon $[0, T^*]$. The objective is to minimize the holding costs incurred in processing all the initial jobs plus the jobs that arrive during $[0, T^*]$. This section is structured as follows: In §3.3.1, we formally define the model considered; The associated fluid relaxation and its solution is discussed in §3.3.2. In §3.3.3, we prove that the *fluid synchronization algorithm* (FSA) yields a schedule that is asymptotically optimal.

3.3.1 Problem Formulation

The model considered here is identical to that of §3.2.2, except that external arrivals are permitted. We assume that type i jobs arrive to the network in a deterministic fashion at rate λ_i. The traffic intensity at machine σ_j, denoted by ρ_j, is defined as

$$\rho_j = \sum_{(i,k)\in\sigma_j} p_{i,k}\lambda_i. \tag{3.24}$$

Our objective is to minimize the holding costs incurred in processing all the initial jobs, plus the jobs that arrive in the interval $[0, T^*]$, chosen suitably.

Let $n_{i,k}(t)$ be the number of (i,k) jobs at machine σ_k^i at time t. Let $w_{i,k}$ be non-negative integer holding cost rates associated with (i,k)-jobs. Our objective is to find a scheduling policy that minimizes

$$\int_{t=0}^{\infty} \sum_{i=1}^{I} \sum_{k=1}^{J_i} w_{i,k} n_{i,k}(t).$$

As before, we consider a continuous relaxation of this problem, in which the number of (i,k) jobs is allowed to assume arbitrary non-negative real values, and machines are allowed

to work simultaneously on several types of jobs. This relaxation is called the fluid job-shop scheduling problem.

3.3.2 The Fluid Job shop scheduling problem

The fluid relaxation associated with the model of §3.3.1 is as follows.

$$x_{i,1}(t) = x_i + \lambda_i \min(t, T^*) - \mu_{i,1} T_{i,1}(t), \qquad i = 1, 2, \ldots, I, \ t \geq 0 \tag{3.25}$$

$$x_{i,k}(t) = \mu_{i,k-1} T_{i,k-1}(t) - \mu_{i,k} T_{i,k}(t), \qquad k = 2, \ldots, J_i, \ i = 1, 2, \ldots, I, \ t \geq 0, \tag{3.26}$$

$$0 \leq \sum_{(i,k) \in \sigma_j} (T_{i,k}(t_2) - T_{i,k}(t_1)) \leq t_2 - t_1, \qquad \forall\ t_2 > t_1, \ t_1, t_2 \geq 0, \ j = 1, 2, \ldots, J, \tag{3.27}$$

$$x_{i,k}(t) \geq 0, \quad T_{i,k}(t) \geq 0. \tag{3.28}$$

These constraints can be interpreted in exactly the same manner as before. Our objective function for the fluid job shop is

$$\sum_{i=1}^{I} \sum_{k=1}^{J_i} \int_0^\infty w_{i,k} x_{i,k}(t) dt.$$

Proposition 3.1 holds for this model as well, which guarantees the existence of an optimal fluid solution with piecewise constant control. This property will help us find a schedule that is asymptotically optimal.

3.3.3 The fluid synchronization algorithm

In this section we describe an algorithm to discretize an optimal fluid solution for the holding costs objective. Our discretization algorithm is based on repeated applications of a variant of the algorithm FSA developed for the makespan objective. We make the following two assumptions.

- **Assumption 0:** For $N = 1$, the horizon T^* is chosen to be the first time at which the fluid relaxation empties in an optimal solution.
- **Assumption 1:** The number of initial jobs of type i is $\alpha_i \cdot N$, and the horizon is NT^*. In our asymptotics, we let $N \to \infty$. Thus we consider a sequence of job-shop problems in which N varies, but all other quantities remain the same.

We make assumption 0 so as to get a non-trivial lower bound from the fluid relaxation. If T^* is chosen to be much larger than the time at which the fluid relaxation empties, the fluid relaxation does not yield a good lower bound. This is immediate from the fact if the fluid relaxation empties at time T, an optimal fluid solution will keep it empty for all $t > T$, and so will incur no extra cost. We use $Z_{\mathrm{F}}(N)$ to denote the cost of the optimal fluid solution when the number of initial jobs of type i is $\alpha_i \cdot N$; similarly, we use $Z_{\mathrm{JS}}(N)$ to denote the cost of the optimal solution to the corresponding discrete job-shop problem.

It is easy to see that the fluid relaxation yields an asymptotic lower bound; in other words, Theorem 3.3 holds for this model. (The proof is similar to the model without arrivals, and is hence omitted.) The discretization algorithm used is also quite similar; we highlight only the important differences.

Description of Algorithm FSA-HC:

The main idea of algorithm FSA-HC is the following: Suppose the optimal fluid solution has R pieces. Let T^i denote the time at which piece i ends (also the time at which piece $(i+1)$ begins), and let $T^0 = 0$ be the time origin. Thus, piece i starts at time T^{i-1} and ends at time T^i, for $1 \leq i \leq R$. As before, we discretize each piece separately using algorithm *RFSA* of §3.2.2. That is, we find times $\hat{T}^0 = 0, \hat{T}^1, \hat{T}^2, \ldots, \hat{T}^R$ such that the vector of queue-lengths at $\hat{T}^i$ in the discrete network is exactly the same as the vector of queue-lengths at T^i in the fluid relaxation. We now evaluate and compare the cost of each piece, and show that the discretization error accumulated over all the R pieces is negligible compared to the total fluid cost. The key difference in this model is that arrivals occur during the interval $[0, T^R]$. In our algorithm, therefore, all the arrivals that occur during the interval $[T^r, \hat{T}^r]$ for $r = 1, \ldots R$. are processed at the very end. Since the length of such an interval is at most $r\ J_{\max}(2\ P_{\max} + U_{\max})$, the number of arrivals that occur during such intervals is $O(1)$; these arrivals can be post-processed in time $O(N)$, which preserves the asymptotic optimality of the schedule provided by algorithm FSA-HC.

3.4 Computational Results

In this section we report computational results on the proposed algorithms. In §3.4.1, we consider the case of deterministic processing times; §3.4.2 considers the case of stochastic processing times.

3.4.1 Weighted completion time: deterministic processing times

We now turn our attention to the objective of minimizing weighted completion times. This is the special case of the holding cost objective in the weights are all 1, i.e., $w_{i,k} = 1$ for all i, k. For our computational study, we chose a subset of 20 instances from the OR library (http://mscmga.ms.ic.ac.uk/info.html); the results shown on these instances are representative of the results obtained for our algorithm in general. The results reported in Table 3.4 are for these 20 benchmarks. The number of machines ranged from 6 to 20, and the number of job types ranged from 6 to 50. For each benchmark, we report results for $N = 1$, $N = 2$, $N = 5$, $N = 10$, $N = 100$, and $N = 500$. The lower bound based on the fluid relaxation, Z_{F}, is shown in the second column, and is valid for $N = 1$; the lower bound for $N = n$ is $n^2\ Z_F$. The subsequent columns report the value of the relative error, $(Z_D - Z_F)/Z_F$. Again, we see that the relative error goes to zero.

3.4.2 Weighted completion time: stochastic processing times

We now turn our attention to the case in which the processing times are stochastic. We present these results for only one benchmark; the results for the remaining instances follow the same pattern. For our experiments, we chose the problem instance *ft20*, which has twenty job classes, each with five stages. We assume that the processing times are geometrically distributed with mean $p_{i,k}$, where $p_{i,k}$ is the processing time in the deterministic case. The distributions of the cost of algorithm FSA is displayed in Figure 3-4.

3.5 Conclusions

In this chapter we considered the job shop problem with objective of minimizing holding costs. We presented an algorithm to compute an asymptotically optimal solution, based on rounding an optimal fluid solution. Essentially, we show that rounding an optimal fluid solution appropriately results in a schedule that incurs $O(N)$ extra cost compared to the optimal fluid cost, which is $O(N^2)$. Again, our results imply that as the number of jobs increases, the combinatorial structure of the problem is increasingly less important, and as a result, a fluid approximation of the problem becomes increasingly exact. The results of this chapter also imply that a continuous approximation to the problem is asymptotically exact. We then discussed an extension of our algorithm to the case in which deterministic

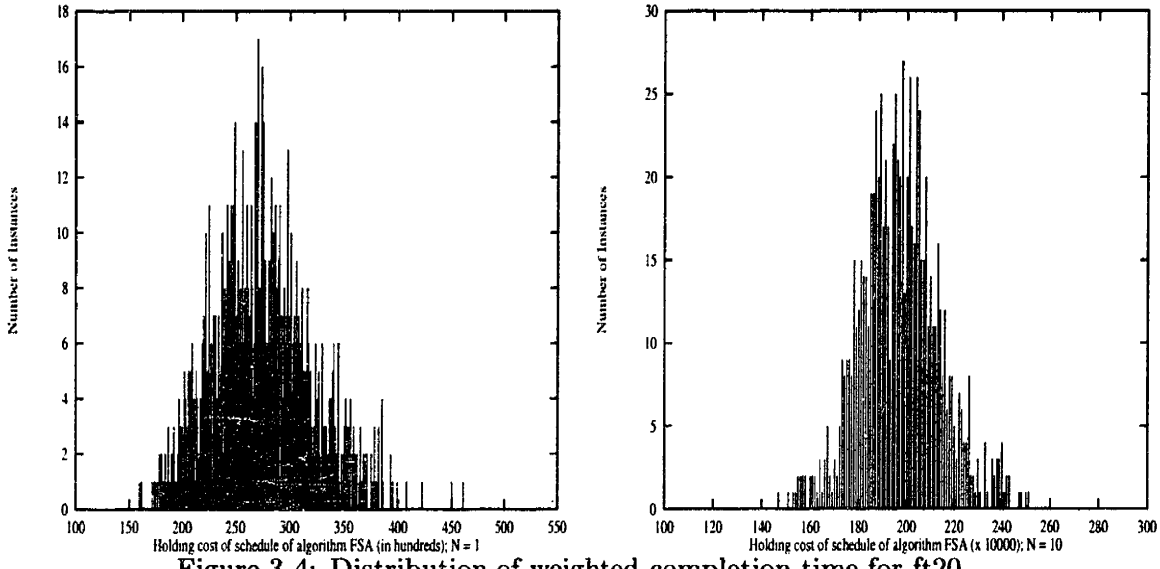

Figure 3-4: Distribution of weighted completion time for ft20

Benchmark	Z_F (N = 1)	$\frac{Z_D - N^2 Z_F}{N^2 Z_F}$ N = 1	N = 2	N = 5	N = 10	N = 100	N = 500
abz5	4154.54	1.731	1.663	1.302	0.876	0.087	0.014
abz6	3116.64	1.689	1.437	1.101	0.825	0.093	0.011
ft06	109.06	2.111	1.813	1.763	1.106	0.147	0.025
ft10	2740.45	2.117	1.987	1.671	1.037	0.436	0.022
ft20	9493.73	1.989	1.700	1.481	1.002	0.210	0.019
la01	2837.45	1.965	1.573	1.320	1.129	0.313	0.016
la02	2802.26	1.270	1.113	0.912	0.614	0.128	0.023
la03	2471.49	1.961	1.672	1.475	1.131	0.254	0.014
la04	2473.30	2.114	1.842	1.386	1.141	0.195	0.014
la05	2501.91	1.320	1.219	1.214	1.067	0.411	0.016
la06	5732.63	2.630	2.315	2.059	1.254	0.193	0.008
la10	5998.61	1.767	1.645	1.323	1.006	0.255	0.011
la11	10000.16	2.749	2.119	1.346	1.043	0.197	0.009
la13	9715.28	2.643	2.216	1.414	1.095	0.351	0.011
la15	10097.26	2.891	2.148	1.730	1.533	0.471	0.021
la17	2983.00	2.653	2.351	1.985	1.438	0.336	0.021
la19	3072.54	2.717	2.185	1.754	1.324	0.372	0.019
orb01	3013.75	2.018	1.811	1.439	1.007	0.221	0.007
orb03	2831.91	2.005	1.837	1.601	1.105	0.119	0.016
orb05	2719.82	1.882	1.473	1.338	0.903	0.143	0.009

Table 3.4: Job Shop instances in OR-Library—Weighted completion time

arrivals over a (suitably restricted) finite horizon. Essentially, we show that our results hold as long as the horizon is in the "fluid regime." This restriction is similar to the criterion of "fluid scale asymptotic optimality" used by Maglaras [53]. Finally, we demonstrated the efficacy of our algorithm on a few chosen benchmarks from the OR library.

Among the many open problems that remain, the most interesting ones deal with deriving good scheduling control policies for multiclass queueing networks in steady-state. We hope some of the ideas outlined here will help in achieving this lofty goal, which was the chief inspiration for our work.

Chapter 4

Semidefinite relaxations for optimizing multiclass queueing networks

4.1 Introduction

In this chapter we turn to the problem of optimizing multiclass queueing networks. A key difficulty in extending an approach based on fluid relaxations is that, being transient in nature, the fluid relaxation does not readily yield a lower bound for the steady-state problem. Thus, we are led naturally to consider alternative methods to derive good lower bounds. To this end, we study the effectiveness of semidefinite relaxations, first proposed by Bertsimas and Niño-Mora [12]. Interestingly, one of the shortcomings of the semidefinite relaxations is that there is no natural way to derive policies based on them. Thus, we use semidefinite relaxations to compute lower bounds, and derive policies based on a heuristic interpretation of optimal fluid solutions.

Multiclass queueing networks. Multiclass queueing networks (MQNETs) provide a rich range of models for complex service systems in application areas that include manufacturing (see Buzacott and Shanthikumar [17]) and computer-communication systems (see Gelenbe and Mitrani [32]). The practical needs to evaluate and improve the performance of such systems have motivated extensive research efforts on the analysis, optimization and

stability of MQNETs.

Most relevant MQNET models have not yielded an exact *performance analysis* (evaluating the system performance under a scheduling policy). This has only been achieved in a restricted range of models, such as product-form MQNETs (see Kelly [44]), and certain single-server priority and polling systems (see Levy and Sidi [49]). A more feasible research objective for those seemingly intractable MQNETs is to obtain *performance bounds* which can be efficiently computed. These bounds may be used to approximate the performance of a given scheduling policy, and to assess its sub-optimality gap with respect to a performance objective. The *performance optimization* problem (computing the optimal system performance under a range of scheduling policies, and finding a policy that achieves it) also appears computationally intractable in most MQNET models, as shown by Papadimitriou and Tsitsiklis [59]. Exact results have only been achieved in a range of systems that satisfy certain *work conservation* laws: for them simple priority-index policies have been shown to optimize linear performance objectives (see Bertsimas and Niño-Mora [10]). In more complex MQNETs researchers have focused their efforts on designing *heuristic* scheduling policies that exhibit a good empirical performance (see, e.g., Wein [72]). Motivated by these considerations, Bertsimas and Niño-Mora have explored the *achievable region approach*, with the objective of developing a systematic method for computing performance bounds and designing scheduling policies that nearly optimize performance objectives. In this chapter we study the effectiveness of this approach in addressing a variety of problems arising in performance evaluation and optimization of multiclass queueing networks.

Structure of the chapter. The rest of this chapter is structured as follows: the remainder of this section is devoted to a brief overview of the the achievable region approach. In §4.2 we describe the basic model we consider and provide a detailed review of the constraints derived from the achievable region approach; our treatment in this chapter draws heavily on the work of Bertsimas and Niño-Mora [11, 12]; the interested reader is referred to these papers for further details. In §4.3 we present detailed computational results for a variety of problems. We conclude this chapter in §4.4, where we identify some problems for future research.

The achievable region approach. The achievable region approach to performance optimization, surveyed in Bertsimas [7], was introduced by Coffman and Mitrani [42]. It draws on the mathematical programming approach to optimization, as it seeks to characterize the *performance region* achievable by a system performance measure under a class of *admissible* scheduling policies. The goal is to formulate explicitly this region by means of equality and inequality constraints. Since it may not be possible to formulate the exact performance region, we may have to settle for constructing a *relaxation* that contains it.

Coffman and Mitrani [42] first addressed with this approach the problem of minimizing the class-weighted mean delay in a multiclass $M/M/1$ queue. They formulated exactly the system performance region as a polyhedron, and showed that the known optimality of priority-index policies (the $c\mu$-rule) follows from structural properties of this underlying polyhedron. The scope of the approach has since been extended to tackle a range of increasingly more complex systems. Drawing on earlier work by Federgruen and Groenevelt [28] and Shanthikumar and Yao [69], Bertsimas and Niño-Mora [10] developed a unified approach for formulating the exact performance region in a wide variety of MQNETs that satisfy work conservation laws. They established that the strong structural properties of these performance optimization problems (optimality of priority-index policies) are a consequence of corresponding properties of their underlying polyhedral performance regions.

Researchers have sought recently to extend further the scope of the achievable region approach, with the aim of solving computationally hard performance optimization problems: restless bandits (see Bertsimas and Niño-Mora [13]) and MQNETs (see Bertsimas, Paschalidis and Tsitsiklis [14, 15], and Kumar and Kumar [46]).

The two critical problems the achievable region approach needs to overcome when tackling a performance optimization problem are (a) generating constraints on the performance region, and (b) designing effective policies from the solution of the corresponding relaxations. Regarding the first problem, an idea that has proven fruitful is to generate constraints by formulating stochastic *equilibrium relations* satisfied by the system. The kinds of equilibrium relations that have been so far used in the literature include (i) *Work conservation laws* (see Bertsimas and Niño-Mora [10]), (ii) *Work decomposition laws* (see Bertsimas and Xu [16], and Bertsimas and Niño-Mora [10]), and (iii) *Potential function recursions*, as developed by Bertsimas, Paschalidis and Tsitsiklis [14, 15], and by Kumar and Kumar [46]. The problem of designing in a systematic way effective scheduling policies for intractable

MQNETs from the solution of the relaxations remains an open challenge. Previous work in this direction includes the dual-index policy proposed in Bertsimas and Niño-Mora [13] for the restless bandit problem, and the policies for polling systems proposed in Bertsimas and Xu [16] and in Bertsimas and Niño-Mora [10]. In this chapter we rely on the fluid relaxation: in most cases we use an appropriate interpretation of an optimal fluid solution to construct a heuristic policy.

4.2 Semidefinite Relaxations for Stochastic Optimization

The development of semidefinite relaxations represents an important advance in discrete optimization. In this section, we review a theory for deriving semidefinite relaxations for classical stochastic optimization problems. The idea of deriving semidefinite relaxations for this class of problems is due to Bertsimas [7]. Our development in this chapter follows Bertsimas and Niño-Mora [11, 12]; the interested reader is referred to these papers for further details. We demonstrate the central ideas for the problem of optimizing a multiclass queueing network that represents a stochastic and dynamic generalization of a job shop.

4.2.1 Model description

We consider a network of queues composed of K single-server stations and populated by N job classes. The set of job classes $\mathcal{N} = \{1, \ldots, N\}$ is partitioned into subsets $\mathcal{C}_1, \ldots, \mathcal{C}_K$, so that station $m \in \mathcal{K} = \{1, \ldots, K\}$ only serves classes in its *constituency* $\mathcal{C}_m$. We refer to jobs of class i as i-jobs, and we let $s(i)$ be the station that serves i-jobs. The network is *open*, so that jobs arrive from outside, follow a Markovian route through the network (i-jobs wait for service at the *i-queue*) and eventually exit. External arrivals of i-jobs follow a Poisson process with rate α_i (if class i does not have external arrivals $\alpha_i = 0$). The service times of i-jobs are independent and identically distributed, having an exponential distribution with mean $\beta_i = 1/\mu_i$. Upon completion of service at station $s(i)$, an i-job becomes a j-job (and hence is routed to the j-queue), with probability p_{ij}, or leaves the system, with probability $p_{i0} = 1 - \sum_{j \in \mathcal{N}} p_{ij}$. We assume that the routing matrix $\mathbf{P} = (p_{ij})_{i,j \in \mathcal{N}}$ is such that a single job moving through the network eventually exits, i.e., the matrix $\mathbf{I} - \mathbf{P}$ is invertible. We further assume that all service times and arrival processes are mutually independent.

The network is controlled by a *scheduling policy*, which specifies dynamically how each

server is allocated to waiting jobs. Scheduling policies can be either *dynamic* or *static*. In a *dynamic* policy, scheduling decisions may depend on the current or past states of all queues; in a *static* policy, the scheduling decisions of each server are independent of the queue lengths of the job classes. A scheduling policy is *stable* if the queue-length vector process has an equilibrium distribution with finite mean. We allow policies to be *preemptive*, i.e., a job's service may be interrupted and resumed later. Finally, a scheduling policy is *non-idling* if a server cannot idle whenever there is a job waiting for service at that station.

Next, we define other model parameters of interest. The *effective arrival rate* of j-jobs, denoted by λ_j, is the total rate at which both external and internal jobs arrive to the j-queue. The λ_j's are computed by solving the system

$$\lambda_j = \alpha_j + \sum_{i \in \mathcal{N}} p_{ij} \lambda_i, \qquad \text{for } j \in \mathcal{N}.$$

The *traffic intensity* of j-jobs, denoted by $\rho_j = \lambda_j \beta_j$, is the time-stationary probability that a j-job is in service. The *total traffic intensity* at station m is $\rho(\mathcal{C}_m) = \sum_{j \in \mathcal{C}_m} \rho_j$, and is the time-stationary probability that server m is busy. We note that the condition

$$\rho(\mathcal{C}_m) < 1, \qquad \text{for } m \in \mathcal{K}$$

is necessary but not sufficient for guaranteeing the stability of any non-idling policy.

We assume that the system operates in a steady-state regime (under a stable policy), and introduce the following variables:

- $L_i(t)$ = number of i-jobs in system at time t.
- $B_i(t) = 1$ if an i-job is in service at time t; 0 otherwise.
- $B^m(t) = 1$ if server m is busy at time t; 0 otherwise; notice that $B^m(t) = \sum_{i \in \mathcal{C}_m} B_i(t)$.

In what follows we write, for convenience of notation, $L_i = L_i(0)$, $B_i = B_i(0)$ and $B^m = B^m(0)$.

4.2.2 The performance optimization problem

The *performance measures* we are interested in are $\mathbf{x} = (x_j)_{j \in \mathcal{N}}$, and $\mathbf{y} = (y_j)_{j \in \mathcal{N}}$, where

$$x_j = E[L_j], \qquad y_j = E\left[L_j^2\right], \qquad \text{for } j \in \mathcal{N},$$

i.e., the vectors whose components are the time-stationary mean and second moment of the number of jobs from each class in the system.

Given a *performance cost function* $\mathbf{c}'\mathbf{x} + \mathbf{d}'\mathbf{y}$, we investigate the following *performance optimization problem*: compute a lower bound $\underline{Z} \leq \mathbf{c}'\mathbf{x} + \mathbf{d}'\mathbf{y}$ that is valid under a given class of admissible policies, and design a policy which nearly minimizes the cost $\mathbf{c}'\mathbf{x} + \mathbf{d}'\mathbf{y}$. For our purposes, any preemptive, non-idling policy is admissible. In this chapter, we restrict our attention to the question of computing strong lower bounds. As we mentioned in the Introduction, the design of optimal policies is $EXPTIME$-hard (Papadimitriou and Tsitsiklis [59]), i.e., it is provably requires exponential time, as $P \neq EXPTIME$. In recent years, progress has been made in designing near-optimal scheduling policies based on the idea of fluid control, in which discrete jobs are replaced by the flow of a fluid; we refer the interested reader to the papers by Avram, Bertsimas and Ricard [5], Weiss [73], Luo and Bertsimas [52], and the references cited therein for details.

We study the problem of computing good lower bounds via the *achievable region* approach. The achievable region (equivalently, performance region) $\mathcal{X}$ is defined as the set of all performance vectors $(\mathbf{x}, \mathbf{y})$ that can be achieved under admissible policies. Our goal is to derive constraints on the performance vector $(\mathbf{x}, \mathbf{y})$ that define a relaxation of performance region $\mathcal{X}$. Since it is not obvious how to derive such constraints directly, we pursue the following plan: (a) identify system *equilibrium relations* and formulate them as constraints involving *auxiliary performance variables*; (b) formulate additional constraints (both *linear* and *positive semidefinite*) on the auxiliary performance variables; (c) formulate constraints that express the original performance vector, $(\mathbf{x}, \mathbf{y})$, in terms of the auxiliary variables.

Notice that this approach is fairly standard in the mathematical programming literature and has a clear geometric interpretation: It corresponds to constructing a relaxation of the performance region of the natural variables, $(\mathbf{x}, \mathbf{y})$, by (a) *lifting* this region into a higher dimensional space, by means of auxiliary variables, (b) bounding the lifted region through constraints on the auxiliary variables, and (c) *projecting* back into the original space. *Lift and project* techniques have proven powerful tools for constructing tight relaxations for hard discrete optimization problems (see, e.g., Lovász and Schrijver [50]). We have summarized the performance measures considered in this chapter (including auxiliary ones) in Table 4.1.

The rest of this section is organized as follows. In § 4.2.3, we include linear constraints that relate the natural performance measures in terms of auxiliary performance variables.

Performance variables	Interpretation
x_j; $\mathbf{x} = (x_j)_{j\in\mathcal{N}}$	$E[L_j]$
x_j^i; $\mathbf{X} = (x_j^i)_{i,j\in\mathcal{N}}$; $\mathbf{x}^i = (x_j^i)_{j\in\mathcal{N}}$	$E[L_j \mid B_i = 1]$
x_j^{0m}; $\mathbf{X}^0 = (x_j^{0m})_{m\in\mathcal{K},j\in\mathcal{N}}$; $\mathbf{x}^{0m} = (x_j^{0m})_{j\in\mathcal{N}}$	$E[L_j \mid B^m = 0]$
r_{ij}; $\mathbf{R} = (r_{ij})_{i,j\in\mathcal{N}}$	$E[B_iB_j]$
r_{ij}^k; $\mathbf{R}^k = (r_{ij}^k)_{i,j\in\mathcal{N}}$	$E[B_iB_j \mid B_k = 1]$
r_{ij}^{0m}; $\mathbf{R}^{0m} = (r_{ij}^{0m})_{i,j\in\mathcal{N}}$	$E[B_iB_j \mid B^m = 0]$
y_{ij}; $\mathbf{Y} = (y_{ij})_{i,j\in\mathcal{N}}$	$E[L_iL_j]$
y_{ij}^k; $\mathbf{Y}^k = (y_{ij}^k)_{i,j\in\mathcal{N}}$	$E[L_iL_j \mid B_k = 1]$
y_{ij}^{0m}; $\mathbf{Y}^{0m} = (y_{ij}^{0m})_{i,j\in\mathcal{N}}$	$E[L_iL_j \mid B^m = 0]$

Table 4.1: Network performance measures.

Using the fact that our performance measures are expectations of random variables, we describe a set of positive semidefinite constraints in § 4.2.4.

4.2.3 Linear constraints

In this section, we present several sets of linear constraints that express natural performance measures in terms of auxiliary ones. The first set of constraints describes constraints that follow from elementary arguments.

Theorem 4.1 (Elementary constraints) *Under any stable policy, the following equations hold:*

(a) *Projection Constraints:*

$$x_j = \sum_{i\in\mathcal{C}_m} \rho_i x_j^i + (1-\rho(\mathcal{C}_m))\, x_j^{0m}, \qquad j\in\mathcal{N},\ m\in\mathcal{K}, \tag{4.1}$$

$$r_{ij} = \sum_{k\in\mathcal{C}_m} \rho_k r_{ij}^k + (1-\rho(\mathcal{C}_m))\, r_{ij}^{0m}, \qquad i,j\in\mathcal{N},\ m\in\mathcal{K}, \tag{4.2}$$

$$y_{ij} = \sum_{k\in\mathcal{C}_m} \rho_k y_{ij}^k + (1-\rho(\mathcal{C}_m))\, y_{ij}^{0m}, \qquad i,j\in\mathcal{N},\ m\in\mathcal{K}. \tag{4.3}$$

(b) *Definitional Constraints:*

$$r_{ij} = \rho_j r_{ii}^j, \qquad i,j\in\mathcal{N}, \tag{4.4}$$

$$r_{ii} = \rho_i,\ r_{ii}^i = 1, \qquad i\in\mathcal{N}, \tag{4.5}$$

$$r_{ij} = 0,\ r_{ij}^k = 0, \qquad i,j\in\mathcal{C}_m, \tag{4.6}$$

$$r_{ij}^k = 0, \qquad i,k\in\mathcal{C}_m,\ \text{or}\ j,k\in\mathcal{C}_m, \tag{4.7}$$

$$r_{ij}^{0m} = 0, \qquad i \text{ or } j \in \mathcal{C}_m. \tag{4.8}$$

(c) *Lower bound constraints:*

$$r_{ij} \geq \max(0, \rho_i + \rho_j - 1), \qquad i,j \in \mathcal{N}, \tag{4.9}$$

$$x_j^i \geq \frac{r_{ij}}{\rho_i}, \qquad i,j \in \mathcal{N}, \tag{4.10}$$

$$x_j^{0m} \geq \max\left(0, \frac{\rho_j - \rho(\mathcal{C}_m)}{1 - \rho(\mathcal{C}_m)}\right), \qquad m \in \mathcal{K},\ j \in \mathcal{N}, \tag{4.11}$$

$$r_{ij}^k \geq \max\left(0, \frac{r_{ki} + r_{kj}}{\rho_k} - 1\right), \qquad i,j,k \in \mathcal{N}, \tag{4.12}$$

$$r_{ij}^{0m} \geq \max\left(0, \frac{\max(0, \rho_i - \rho(\mathcal{C}_m)) + \max(0, \rho_j - \rho(\mathcal{C}_m))}{1 - \rho(\mathcal{C}_m))} - 1\right), \quad i,j \in \mathcal{N},\ m \in \mathcal{K}, \tag{4.13}$$

$$y_{ij} \geq r_{ij}, \qquad i,j \in \mathcal{N}, \tag{4.14}$$

$$y_{ij}^k \geq r_{ij}^k, \qquad i,j,k \in \mathcal{N}, \tag{4.15}$$

$$y_{ij}^{0m} \geq r_{ij}^{0m}, \quad i,j \in \mathcal{N},\ m \in \mathcal{K}. \tag{4.16}$$

Proof

The constraints in (a) follow by a simple conditioning argument, by noticing that at each time instant, a server is either serving some job class in its constituency or idling. The constraints in (b), (c) follow from elementary arguments. ■

Flow conservation constraints

We next present a set of linear constraints on performance measures using the classical *flow conservation law* of queueing theory, $L^- = L^+$. We first provide a brief discussion of flow conservation in stochastic systems, and then show how to use these ideas to derive linear relations between time-stationary moments of queue lengths.

The classical flow conservation law of queueing systems states that, under mild restrictions, the stationary state probabilities of the number in system at arrival epochs and that at departure epochs are equal. The key assumption is that jobs arrive to the system and depart from the system one at a time, so that the queue size can change only by unit steps.

Consider a multiclass queueing network operating in a steady state regime, with the number in system process $\{L(t)\}$. We assume that the process $\{L(t)\}$ has right-continuous sample paths, and we use $L(t^-)$ to denote the left limit of the process at time t. The

corresponding right limit $L(t^+) = L(t)$ because of right-continuity of sample paths. Let $A = \{\tau_k^a\}$ and $D = \{\tau_k^d\}$ be the sequences of arrival and departure epochs of jobs respectively. Let $L(\tau_k^{a-})$ be the number of jobs in the system seen by the k^{th} arriving job just before its arrival; similarly, let $L(\tau_k^d)$ be the number of jobs in the system seen by the k^{th} departing job just after its departure. We define

$$L^- = L(\tau_0^{a-}),$$

and

$$L^+ = L(\tau_0^d).$$

Since we assumed the system to be in steady-state, L^- may be interpreted as the number of jobs in the system seen by a typical arrival, while L^+ may be interpreted as the number of jobs in the system seen by a typical departure. By considering any realization, we see that for every upward transition for the number in system from i to $(i+1)$, there is a corresponding downward transition from $(i+1)$ to i; thus every $L(\tau_k^{a-})$ is equal to a distinct $L(\tau_k^d)$ in a sample-path sense. In particular, we have $L^- = L^+$, yielding the following theorem.

Theorem 4.2 (Flow Conservation Law) *If jobs enter and leave the system one at a time, then*

$$L^- = L^+$$

holds in distribution.

In what follows, we apply the law $L^- = L^+$ to a family of queues obtained by aggregating job classes, as explained next. Let $S \subseteq \mathcal{N}$.

Definition 1 (S-queue) *The S-queue is the queueing system obtained by aggregating job classes in S. The number in system at time t in the S-queue is denoted by $L_S(t) = \sum_{j \in S} L_j(t)$.*

As usual we write $L_S = L_S(0)$, $L_S^- = L_S(0-)$, $L_S^+ = L_S(0+) = L_S(0)$. For convenience of notation we also write

$$p(i, S) = \sum_{j \in S} p_{ij}$$

and

$$\alpha(S) = \sum_{j \in S} \alpha_j.$$

The next theorem formulates the law $L^- = L^+$ as it applies to the S-queue.

Theorem 4.3 (The law $L^- = L^+$ in MQNETs) *Under any dynamic stable policy, and for any subset of job classes $S \subseteq \mathcal{N}$ and nonnegative integer l:*

$$\alpha(S)P(L_S = l) + \sum_{i \in S^c} \lambda_i p(i,S) P(L_S = l \mid B_i = 1) = \sum_{i \in S} \lambda_i (1 - p(i,S)) P(L_S = l+1 \mid B_i = 1). \tag{4.17}$$

Proof

By applying Theorem 4.2 to the S-queue, we have that

$$P(L_S^- = l) = P(L_S^+ = l).$$

An arrival epoch to the S-queue is either an arrival from the outside world (external arrival) that happens with rate $\alpha(S)$, or an internal movement from a class i in S^c to a class in S (internal arrival) that happens only if $B_i = 1$, for $i \in S^c$ with rate

$$\mu_i p(i,S) P(B_i = 1) = \mu_i p(i,S) \rho_i = \lambda_i p(i,S).$$

The total arrival rate to S-queue is

$$\lambda_S = \alpha(S) + \sum_{i \in S^c} \lambda_i p(i,S).$$

Therefore,

$$P\left(L_S^- = l\right) = \frac{\alpha(S)}{\lambda_S} P(L_S = l) + \sum_{i \in S^c} \frac{\lambda_i p(i,S)}{\lambda_S} P(L_S = l \mid B_i = 1).$$

A departure epoch from S-queue happens with rate

$$\mu_i (1 - p(i,S)) P(B_i = 1) = \lambda_i (1 - p(i,S),$$

for all $i \in S$. The total departure rate is:

$$\mu_S = \sum_{i \in S} \lambda_i (1 - p(i, S)).$$

It can be easily checked that the total arrival rate to the S-queue and the total departure rate from the S-queue are equal, i.e., $\lambda_S = \mu_S$. Therefore,

$$P\left(L_S^+ = l\right) = \sum_{i \in S} \frac{\lambda_i \left(1 - p(i, S)\right)}{\mu_S} P\left(L_S = l + 1 \mid B_i = 1\right).$$

By applying $P(L_S^- = l) = P(L_S^+ = l)$, Eq. (4.17) follows. ■

Taking expectations in identity (4.17) we obtain:

Corollary 4.1 *Under any stable policy, and for any subset of job classes $S \subseteq \mathcal{N}$ and positive integer p for which $E\left[(L_1 + \cdots + L_N)^p\right] < \infty$,*

$$\alpha(S) E\left[L_S^p\right] + \sum_{i \in S^c} \lambda_i p(i, S) E\left[L_S^p \mid B_i = 1\right] = \sum_{i \in S} \lambda_i \left(1 - p(i, S)\right) E\left[(L_S - 1)^p \mid B_i = 1\right]. \tag{4.18}$$

Note that Corollary 4.1 formulates a linear relation between time-stationary moments of queue lengths. The equilibrium equations in Corollary 4.1 corresponding to $p = 1, 2$ and $S = \{i\}, \{i, j\}$, for $i, j \in \mathcal{N}$, yield directly the system of linear constraints on performance variables shown next. Let $\mathbf{\Lambda} = \mathrm{Diag}(\boldsymbol{\lambda})$.

Corollary 4.2 (Flow conservation constraints) *Under any dynamic stable policy, the following linear constraints hold:*

(a)

$$-\boldsymbol{\alpha}\mathbf{x}' - \mathbf{x}\boldsymbol{\alpha}' + (\mathbf{I} - \mathbf{P})'\mathbf{\Lambda}\mathbf{X} + \mathbf{X}'\mathbf{\Lambda}(\mathbf{I} - \mathbf{P}) = (\mathbf{I} - \mathbf{P})'\mathbf{\Lambda} + \mathbf{\Lambda}(\mathbf{I} - \mathbf{P}). \tag{4.19}$$

(b) *If $E\left[(L_1 + \cdots + L_N)^2\right] < \infty$, then*

$$\alpha_j y_{jj} + \sum_{r \in \mathcal{N}} \lambda_r p_{rj} y_{jj}^r - \lambda_j y_{jj}^j + 2\lambda_j (1 - p_{jj}) x_j^j = \lambda_j (1 - p_{jj}), \quad j \in \mathcal{N}. \tag{4.20}$$

$$\alpha_i y_{jj} + \alpha_j y_{ii} + 2(\alpha_i + \alpha_j) y_{ij} + \sum_{r \in \mathcal{N}} \lambda_r p_{ri} y_{jj}^r + \sum_{r \in \mathcal{N}} \lambda_r p_{rj} y_{ii}^r + \sum_{r \in \mathcal{N}} 2\lambda_r (p_{ri} + p_{rj}) y_{ij}^r - \lambda_i y_{jj}^i - \lambda_j y_{ii}^j - 2\lambda_i y_{ij}^i - 2\lambda_j y_{ij}^j$$

$$
\begin{aligned}
&-2\lambda_i p_{ij} x_i^i - 2\lambda_j p_{ji} x_j^j \\
+2\lambda_i(1-p_{ii}-p_{ij})x_j^i + 2\lambda_j(1-p_{ji}-p_{jj})x_i^j &= -\lambda_i p_{ij} - \lambda_j p_{ji}, \quad (4.21)\\
&\qquad i,j \in \mathcal{N}.
\end{aligned}
$$

The equilibrium equations in Corollary 4.1 corresponding to $p = 2$ and $S = \{i,j,k\}$, for $i,j,k \in \mathcal{N}$, yield the following linear constraints on performance variables.

Corollary 4.3 (Flow conservation constraints (continued)) *Let* $A_{ijk} = \alpha_i+\alpha_j+\alpha_k$, *and* $Y_{ijk} = y_{ii} + y_{jj} + y_{kk} + 2y_{ik} + 2y_{jk} + 2y_{ik}$. *Also, for* $r \in \mathcal{N}$, *let* $Y_{ijk}^r = y_{ii}^r + y_{jj}^r + y_{kk}^r + 2y_{ij}^r + 2y_{jk}^r + 2y_{ik}^r$, $W_{ijk}^r = x_i^r + x_j^r + x_k^r$, *and* $P_{ijk}^r = p_{ri} + p_{rj} + p_{rk}$. *Under any dynamic stable policy, the following linear constraints hold:*

$$
\begin{aligned}
&A_{ijk}Y_{ijk} + \sum_{r\in\mathcal{N}} \lambda_r P_{ijk}^r Y_{ijk}^r - \lambda_i Y_{ijk}^i - \lambda_j Y_{ijk}^j - \lambda_k Y_{ijk}^k \\
+2\lambda_i(1-P_{ijk}^i)W_{ijk}^i + 2\lambda_j(1-P_{ijk}^j)W_{ijk}^j + 2\lambda_k(1-P_{ijk}^k)W_{ijk}^k &= \lambda_i(1-P_{ijk}^i) + \lambda_j(1-P_{ijk}^j) \\
&\quad +\lambda_k(1-P_{ijk}^k), \qquad (4.22)\\
&\qquad i,j,k \in \mathcal{N}.
\end{aligned}
$$

Remark: We note that for a fixed p in Corollary 4.1, it is enough to consider subsets of classes, S, such that $|S| \leq (p+1)$. Thus, the constraints we present in Corollaries 4.2 and 4.3 suffice for our purposes because we shall restrict our performance variables to first and second moments only.

The flow conservation constraints were first derived for multi-station MQNETs by Bertsimas, Paschalidis and Tsitsiklis [14], and by Kumar and Kumar [46], using a potential function approach. The derivation we presented is from Bertsimas and Niño-Mora [12].

4.2.4 Positive semidefinite constraints

We present in this section, a set of *positive semidefinite constraints* that strengthen the formulations obtained through equilibrium relations. Recall that the performance measures $\mathbf{x}, \mathbf{y}$ are moments of random variables. Recall also that the covariance matrix of a random vector is positive semidefinite. Applying this observation to the performance variables introduced in Table 4.1 yields directly the following result.

Theorem 4.4 *Under any dynamic stable policy, the following semidefinite constraints hold:*

(a) *Let* $\mathbf{r}^k = (r_{ii}^k)_{i\in\mathcal{N}}$ *and* $\mathbf{r}^{0m} = (r_{ii}^{0m})_{i\in\mathcal{N}}$.

$$\begin{bmatrix} 1 & \boldsymbol{\rho}' \\ \boldsymbol{\rho} & \mathbf{R} \end{bmatrix} \succeq \mathbf{0}, \tag{4.23}$$

$$\begin{bmatrix} 1 & \mathbf{r}^{k\prime} \\ \mathbf{r}^k & \mathbf{R}^k \end{bmatrix} \succeq \mathbf{0}, \qquad k \in \mathcal{N}, \tag{4.24}$$

$$\begin{bmatrix} 1 & \mathbf{r}^{0m\prime} \\ \mathbf{r}^{0m} & \mathbf{R}^{0m} \end{bmatrix} \succeq \mathbf{0}, \qquad m \in \mathcal{K}. \tag{4.25}$$

(b) *If* $E\left[\left(\sum_{j\in\mathcal{N}} L_j\right)^2\right] < \infty$, *then*

$$\begin{bmatrix} 1 & \mathbf{x}' \\ \mathbf{x} & \mathbf{Y} \end{bmatrix} \succeq \mathbf{0}, \tag{4.26}$$

$$\begin{bmatrix} 1 & \mathbf{x}^{k\prime} \\ \mathbf{x}^k & \mathbf{Y}^k \end{bmatrix} \succeq \mathbf{0}, \qquad k \in \mathcal{N}, \tag{4.27}$$

$$\begin{bmatrix} 1 & \mathbf{x}^{0m\prime} \\ \mathbf{x}^{0m} & \mathbf{Y}^{0m} \end{bmatrix} \succeq \mathbf{0}, \qquad m \in \mathcal{K}. \tag{4.28}$$

4.3 Computational Results

In this section we consider several networks and study the performance of both the LP relaxations and semidefinite relaxations for these problems. We also simulate policies for these networks, which are derived based on interpreting an optimal fluid solution, and assess their proximity to optimality by comparing them to the lower bounds derived from these relaxations. We solve the semidefinite relaxations using the package SDPA developed by Fujisawa, Kojima and Nakata [31], and the LP relaxations using CPLEX. The semidefinite relaxations were solved in less than one minute by SDPA on a Pentium II workstation as the SDP relaxation has a simple block structure; The corresponding LP relaxations took very few seconds to solve.

4.3.1 Open networks: first-moment objectives

Our objective in this section is to compare computationally the linear and semidefinite relaxations of the multiclass queueing network performance optimization problem. The linear programming relaxation is defined as follows:

$$
\begin{aligned}
Z_{LP} = \text{minimize} \quad & \mathbf{c}'\mathbf{x} \\
\text{subject to} \quad & \text{Projection constraints : (4.1) } - \text{ (4.3)}, \\
& \text{Definitional constraints : (4.4), (4.5), (4.6), (4.7), (4.8)}, \\
& \text{Lower bound constraints :(4.9) } - \text{ (4.16)}, \\
& \text{Flow } - \text{ conservation constraints : (4.19), (4.20), (4.21), (4.22)}, \\
& \mathbf{x} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0}.
\end{aligned}
$$

The semidefinite relaxation Z_{SD} is obtained by adding the constraints (4.23), (4.24), (4.25) (4.26), (4.27), (4.28).

Rybko-Stolyar network

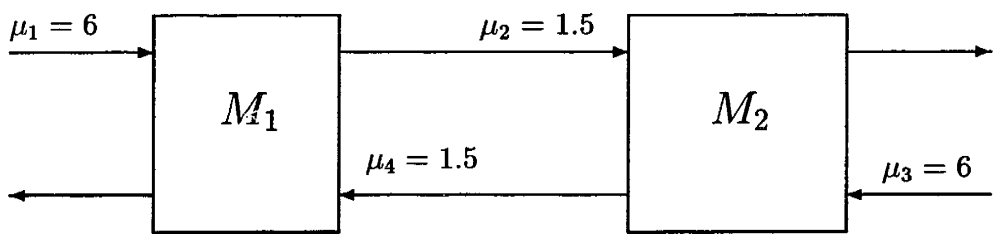

Figure 4-1: Rybko-Stolyar network.

We consider the network of Figure 4-1. In this network external arrivals come into either class 1 or class 3, and so $\alpha_2 = \alpha_4 = 0$. In our computations we fix the service times as shown in the figure, and vary only the arrival rates. We maintain the symmetry between classes, and so we set $\alpha_1 = \alpha_3 = \alpha$, where α varies from 0.1 to 1.18. We select $c_i = 1$ and $d_i = 0$, i.e., we are interested in minimizing the expected number of jobs in the system in steady-state. We present below the optimal values Z_{LP} and Z_{SD}.

For comparison purposes, we also report simulation results for a particular policy that was derived from fluid optimal control (see Avram et. al. [5]): When both $L_4(t)$, $L_2(t) > B$, the first station gives preemptive priority to class 4 and the second station gives preemptive

ρ	Z_{LP}	Z_{SD}	$E[Z_{LBFS-B}]$	Best B
0.083	0.170	0.170	0.179 ± 0.001	0
0.167	0.347	0.347	0.390 ± 0.002	0
0.250	0.538	0.538	0.643 ± 0.003	0
0.333	0.793	0.794	0.951 ± 0.004	1
0.417	1.113	1.113	1.339 ± 0.006	1
0.500	1.530	1.530	1.842 ± 0.008	1
0.583	2.102	2.103	2.527 ± 0.012	1
0.667	2.947	2.976	3.526 ± 0.017	1
0.750	4.360	4.416	5.131 ± 0.027	1
0.833	7.167	7.220	8.203 ± 0.045	2
0.875	9.930	9.980	11.227 ± 0.065	2
0.917	15.413	15.497	17.078 ± 0.106	2
0.958	31.777	31.832	34.414 ± 0.246	2
0.983	80.766	81.093	85.139 ± 0.893	3

Table 4.2: Relaxations and Policies for the network of Figure 4-1.

priority to class 2. When $L_4(t) \leq B$, class 3 has preemptive priority over class 2. Similarly, when $L_2(t) \leq B$, class 1 has preemptive priority over class 4. We call this policy last-buffer-first-served with a threshold B, denoted by $LBFS - B$. We let $E[Z_{LBFS-B}]$ denote the expected number of jobs under this policy. We select the value of B optimally using simulation.

In Table 4.2, we report the values Z_{LP}, Z_{SD}, the simulation value $E[Z_{LBFS-B}]$, and the value of the threshold B that gives the optimal performance.

Ou-Wein networks

We now report results on the networks considered by Ou and Wein [58]. Results for the networks of Figures 4-2, 4-3 and 4-4 are shown in Tables 4.3, 4.4 and 4.5 respectively. In all of these cases, a fluid-based policy was simulated. For example, for the network of Figure 4-2, the optimal fluid policy gives priority to class 3 over class 1. We simulated a policy in which class 3 has priority unless the number of class 2 customers (at station 2) falls below a certain threshold B; the optimal such B was found by simulation. For the network of Figure 4-3, we used a last-buffer first-serve policy, which gives priority to classes that are closer to exiting the network; also, type 1 jobs were given priority over type 2 jobs. For the network of Figure 4-4, we used a policy that gives priority to jobs with the shortest expected remaining processing time.

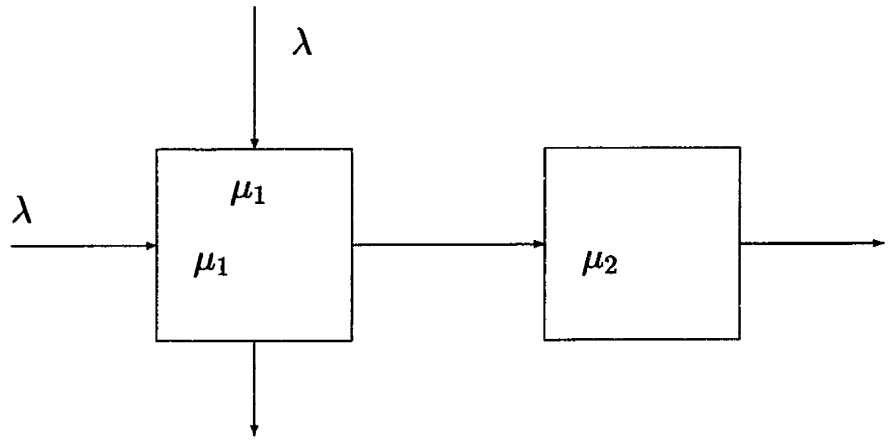

Figure 4-2: The Ou-Wein Network
(Balanced: $\mu_1 = 2, \mu_2 = 1$; Imbalanced: $\mu_1 = 2, \mu_2 = 1.5$)
Light: $\lambda = 0.3$; Medium: $\lambda = 0.6$; Heavy: $\lambda = 0.9$; Very Heavy: $\lambda = 0.99$

Load	Z_{LP}	Z_{SD}	$E[Z_P]$
Balanced Light	0.729	0.729	0.764 ± 0.008
Balanced Medium	2.100	2.100	2.193 ± 0.015
Balanced Heavy	9.900	10.883	13.310 ± 0.783
Balanced Very Heavy	99.990	103.730	107.610 ± 1.177
Imbalanced Light	0.629	0.629	0.657 ± 0.005
Imbalanced Medium	1.900	1.900	2.003 ± 0.011
Imbalanced Heavy	9.600	9.668	9.997 ± 0.376
Imbalanced Very Heavy	99.660	99.905	101.601 ± 0.969

Table 4.3: Bounds for network of Figure 4-2.

Load	Z_{LP}	Z_{SD}	$E[Z_P]$
Balanced Light	0.709	0.709	0.729 ± 0.006
Balanced Medium	1.939	2.022	2.107 ± 0.017
Balanced Heavy	8.209	8.739	9.129 ± 0.091
Balanced Very Heavy	72.943	77.176	80.176 ± 1.037
Imbalanced Light	0.624	0.624	0.660 ± 0.007
Imbalanced Medium	1.756	1.769	1.934 ± 0.021
Imbalanced Heavy	7.629	7.790	8.312 ± 0.187
Imbalanced Very Heavy	72.033	72.910	73.147 ± 0.593

Table 4.4: Bounds for network of Figure 4-3.

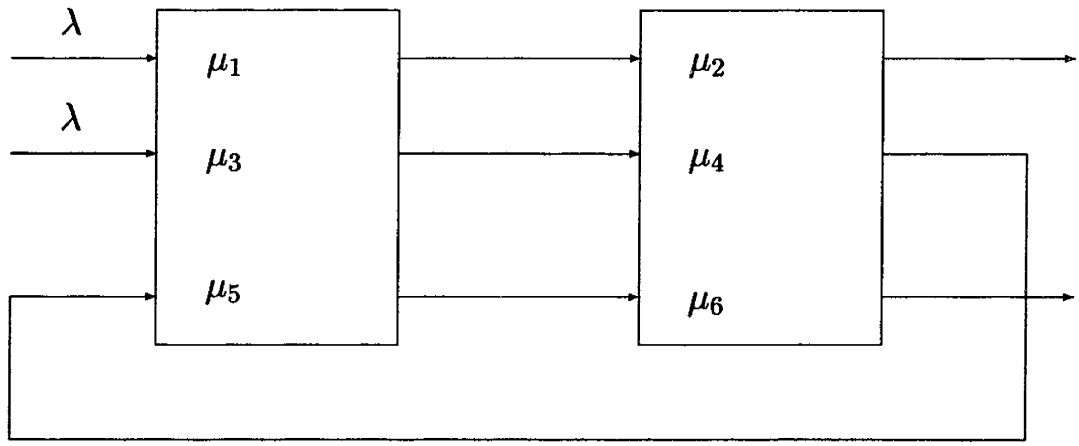

Figure 4-3: The Ou-Wein Network
(Balanced: $\mu_1 = 1/4, \mu_2 = 1, \mu_3 = 1/8, \mu_4 = 1/6, \mu_5 = 1/2, \mu_6 = 1/7$.
Imbalanced: $\mu_1 = 1/4, \mu_2 = 2/3, \mu_3 = 1/8, \mu_4 = 1/4, \mu_5 = 1/2, \mu_6 = 3/14$).
Light: $\lambda = 3/140$; Medium: $\lambda = 6/140$; Heavy: $\lambda = 9/140$; Very Heavy: $\lambda = 99/1400$

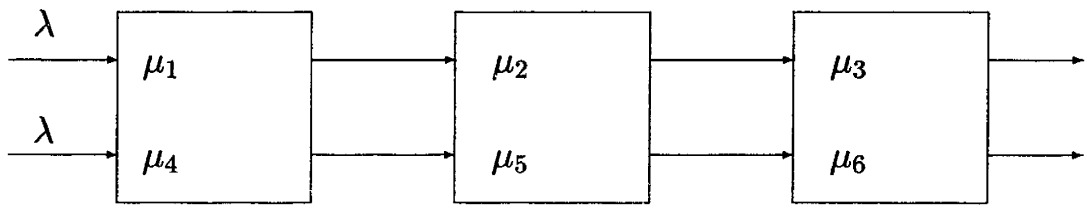

Figure 4-4: Ou-Wein Network
(Balanced: $\mu_1 = 1/2, \mu_2 = 1/4, \mu_3 = 1/6, \mu_4 = 1/7, \mu_5 = 1/5, \mu_6 = 1/3$.
Imbalanced: $\mu_1 = 1/2, \mu_2 = 1/2, \mu_3 = 1, \mu_4 = 1/7, \mu_5 = 1/4, \mu_6 = 1/2$.)
Light: $\lambda = 1/30$; Medium: $\lambda = 2/30$; Heavy: $\lambda = 3/30$; Very Heavy: $\lambda = 11/100$

Load	Z_{LP}	Z_{SD}	$E[Z_P]$
Balanced Light	1.012	1.012	1.211 ± 0.007
Balanced Medium	2.648	2.890	3.138 ± 0.029
Balanced Heavy	11.559	11.753	18.108 ± 0.945
Balanced Very Heavy	118.419	121.45	137.891 ± 1.732
Imbalanced Light	0.712	0.712	0.761 ± 0.008
Imbalanced Medium	1.972	1.972	2.101 ± 0.016
Imbalanced Heavy	8.721	8.937	9.798 ± 0.671
Imbalanced Very Heavy	84.645	87.395	91.213 ± 1.164

Table 4.5: Bounds for network of Figure 4-4.

These computations show that there is no significant benefit to solving the more complicated semidefinite programming relaxation if our objective is to minimize weighted queue-lengths in steady state. In all of these examples, we also observe that the optimal cost of the policy is quite close to the lower bounds; we have observed this on several other small networks (3 stations, up to 9 classes). This establishes both the quality of the bounds as well as the proximity to optimality of the policy simulated. On all the examples we have tested, it appears that the flow conservation laws identified are quite successful in capturing the interactions between various classes in the network.

4.3.2 Open networks: second-moment objective functions

Now we consider objective functions involving second moments of queue-lengths. The linear programming relaxation is defined as follows:

$$
\begin{aligned}
Z_{LP} = \text{minimize} \quad & \mathbf{c}'\mathbf{x} + \mathbf{d}'\mathbf{y} \\
\text{subject to} \quad & \text{Projection constraints : (4.1) } - \text{ (4.3)}, \\
& \text{Definitional constraints : (4.4), (4.5), (4.6), (4.7), (4.8)}, \\
& \text{Lower bound constraints :(4.9) } - \text{ (4.16)}, \\
& \text{Flow} - \text{conservation constraints : (4.19), (4.20), (4.21), (4.22)}, \\
& \mathbf{x} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0}.
\end{aligned}
$$

The semidefinite relaxation Z_{SD} is obtained by adding the constraints (4.23), (4.24), (4.25) (4.26), (4.27), (4.28).

We consider a single station network with four classes. Our objective here is to minimize $\sum_{i=1}^{4} x_i + y_{ii}$. For the case that we do not include terms involving y_{ii} in the objective function, the LP relaxation is exact (see Bertsimas and Niño-Mora [10]).

We assume that the arrival rate for each class is the same, and that the mean service times for the job classes are 0.05, 0.1, 0.2, and 0.4 respectively. The results of the LP and SDP relaxations are tabulated in Table 4.6.

For comparison purposes we have simulated the following dynamic priority policy P: At every service completion time t, we give priority to the class that has the highest index $\mu_i L_i(t)$. The policy was derived from fluid optimal control (see Avram et. al. [5]).

The computational results suggest that the semidefinite relaxation substantially im-

ρ	Z_{LP}	Z_{SD}	$E[Z_P]$
0.075	0.162	0.162	0.165 ± 0.001
0.150	0.352	0.358	0.368 ± 0.002
0.225	0.578	0.598	0.620 ± 0.005
0.300	0.854	0.901	0.946 ± 0.008
0.375	1.198	1.302	1.379 ± 0.013
0.450	1.639	1.857	1.982 ± 0.022
0.525	2.227	2.676	2.869 ± 0.038
0.600	3.047	3.982	4.255 ± 0.066
0.675	4.270	6.287	6.671 ± 0.129
0.750	6.269	10.991	11.588 ± 0.289
0.825	10.072	22.314	24.495 ± 0.267
0.900	19.811	60.948	75.429 ± 0.394
0.975	89.332	725.855	1203.778 ± 12.864

Table 4.6: Comparison of LP and SDP relaxations for a multiclass queue.

proves the linear programming relaxation. The improvement is more substantial as the traffic intensity ρ increases.

Based on our simulations, we conjecture that the simulated policy is, in fact, optimal. A simple interchange argument establishes this fact when we are given a fixed set of jobs, and there are no external arrivals; the processing times are stochastic, and, in this case, our objective is to minimize the expected total cost incurred in processing these jobs, where the instantaneous cost at time t is $\sum_{i=1}^{n} c_i L_i(t)^2$. While there may be indirect methods of establishing optimality of this policy, an exact performance analysis of this policy appears to be extremely difficult. We have been able to do this when the number of classes is two, with $c_1\mu_1 = c_2\mu_2$ and $h_1\mu_1 = h_2\mu_2$. Under these conditions, the index policy translates to serving the longer queue.

Serving the longer queue: We now present an analysis of a two class single server queue, in which the scheduling policy is to always serve the longer queue. Whenever the queue-lengths are the same, we assume that priority is given to queue 1. (The case when ties are broken by giving priority to queue 2 can be analyzed in an analogous manner.)

Let p_{n_1,n_2} be the probability that the system contains n_1 jobs of class 1 and n_2 jobs of class 2 in steady state. Recall that inter-arrival times for class i are independent and exponentially distributed with mean $1/\lambda_i$; service times are independent and exponentially distributed with mean $1/\mu_i$; all the relevant arrival and service time of the classes processes are mutually independent. Hence our model is Markov. As we shall see, we will not be

able to compute the steady-state distribution of the queue-lengths, p_{n_1,n_2}, explicitly. Our interest, however, is in computing the first and second moments of the queue-lengths of the two classes, and we shall exhibit a method to compute these quantities indirectly. To that end, it will be convenient to work with *differences* between the two queues. Let

$$q_d = \sum_{i=0}^{\infty} p_{i+d,i}, \qquad d = 0, \pm 1, \pm 2, \ldots .$$

We define some useful probability generating functions in terms of q_d. Let

$$F_d(x,y) = \sum_{i=0}^{\infty} p_{i+d,i} x^{i+d} y^d,$$

and

$$G_d(x,y) = \sum_{i=0}^{\infty} p_{i,i+d} x^i y^{i+d}.$$

Letting $\theta = xy$, we have $F_d(x,y) = x^d \sum_{i=0}^{\infty} p_{i+d,i}\theta^i$, and $G_d(x,y) = y^d \sum_{i=0}^{\infty} p_{i,i+d}\theta^i$.

Remark: Since we assume that queue 1 has preemptive priority over queue 2 if the queue-lengths are identical, we may assume that G_d is defined only for $d \geq 1$.

Assuming the steady-state probability distribution exists, an immediate consequence of the Chapman-Kolmogorov forward equations is that F_d and G_d satisfy

$$\frac{(\mu_1 + \theta\lambda_2)}{x} F_{d+1}(x,y) + \lambda_1 \not{x} F_{d-1}(x,y) = (\lambda_1 + \lambda_2 + \mu_1) F_d(x,y), \qquad d = 1, 2, \ldots , \tag{4.29}$$

$$\frac{(\mu_2 + \theta\lambda_1)}{y} G_{d+1}(x,y) + \lambda_2 y G_{d-1}(x,y) = (\lambda_1 + \lambda_2 + \mu_2) G_d(x,y), \qquad d = 2, 3, \ldots . \tag{4.30}$$

The corresponding "boundary" conditions are given by

$$\frac{(\mu_1 + \theta\lambda_2)}{x} F_1(x,y) + \frac{(\mu_2 + \theta\lambda_1)}{y} G_1(x,y) = (\lambda_1 + \lambda_2 + \mu_1) F_0(x,y) - \mu_1 p_{0,0}, \tag{4.31}$$

$$\frac{(\mu_2 + \theta\lambda_1)}{y} G_2(x,y) + \frac{(\mu_1 + \theta\lambda_2)}{x} F_0(x,y) = (\lambda_1 + \lambda_2 + \mu_2) G_0(x,y) + \frac{\mu_1}{x} p_{0,0}. \tag{4.32}$$

We rely on the theory of second order difference equations to solve the system of equations (4.29) - (4.32). Observe that $F_d(1,1) = q_d$, and $G_d(1,1) = q_{-d}$. In particular, both $F_d(1,1)$ and $G_d(1,1)$ are less than one. Therefore the general solution for $F_d(x,y)$ can be expressed as:

$$F_d(x,y) \;=\; C_1(\theta)(xa(\theta))^d, \qquad d = 0,1,2,\ldots. \tag{4.33}$$

In equation (4.33), $a(\theta)$ is *smaller* root of $(\mu_1+\theta\lambda_2)a(\theta)^2-(\lambda_1+\lambda_2+\mu_1)a(\theta)+\lambda_1 \;=\; 0$. (The larger root will, in general, lead to a solution that *does not* make F_d a probability generating function.)

Likewise, $G_d(x,y)$ can be expressed as:

$$G_d(x,y) \;=\; C_2(\theta)(yb(\theta))^d, \qquad d = 1,2,\ldots, \tag{4.34}$$

where $b(\theta)$ is the smaller root of $(\mu_2+\theta\lambda_1)b(\theta)^2-(\lambda_1+\lambda_2+\mu_2)b(\theta)+\lambda_2 \;=\; 0$.

We also note that $C_1(\theta)$ and $C_2(\theta)$ can be computed by solving a system of two linear equations obtained by substituting the conjectured form for G_d and F_d in equations (4.31) and (4.32). For $\theta = 1$, this system becomes singular; we compute $C_1(1)$ and $C_2(1)$ by taking the limit of the corresponding quantities $C_1(\theta)$ and $C_2(\theta)$ as $\theta \to 1$.

Let $F(x,y) := \sum_{d=0}^{\infty} F_d(x,y)$ and $G(x,y) := \sum_{d=1}^{\infty} G_d(x,y)$; let $H(x,y) := F(x,y) + G(x,y)$.

For $i = 1,2$, let $E[L_i]$ denote the expected queue-length of class i, and $E[L_i^2]$ denote the second moment of the queue-length of class i. Clearly, $E[L_i]$ and $E[L_i^2]$ can be computed from the quantities

$$\frac{dF}{dx}, \frac{dF}{dy}, \frac{d^2F}{dx^2}, \frac{d^2F}{dy^2}, \frac{dG}{dx}, \frac{dG}{dy}, \frac{d^2G}{dx^2}, \frac{d^2G}{dy^2},$$

which, in turn, can be computed with a knowledge of the following quantities, evaluated at $x = y = 1$.

$$a(xy), b(xy), C_1(xy), C_2(xy),$$

$$\frac{d^kC_i(xy)}{dx^k}, \frac{d^kC_i(xy)}{dy^k}, \quad k = 1,2;\;\; i = 1,2,$$

$$\frac{d^ka(xy)}{dx^k}, \frac{d^kb(xy)}{dx^k}, \quad k = 1,2,$$

$$\frac{d^ka(xy)}{dy^k}, \frac{d^kb(xy)}{dy^k}, \quad k = 1,2.$$

All our computations were done using the computer algebra package Maple. We present only the final result of our computations.

> $E[L_1] = -\lambda_1(\mu_1\mu_2^2\lambda_1 - \mu_1\lambda_2\mu_2\lambda_1 + 2\mu_1\lambda_2^2\mu_2 + \mu_1\mu_2^3 - 2\mu_1\mu_2^2\lambda_2 - \mu_2^3\lambda_1 - \lambda_1^2\mu_2^2 + \mu_2^3\lambda_2 - \lambda_2^2\mu_2^2 + 2\lambda_2\lambda_1\mu_2^2 + \lambda_2^2\mu_1^2)/((\mu_1 - \lambda_1 + \lambda_2)(\mu_2 + \lambda_1 - \lambda_2)\mu_2(-\mu_1\mu_2 + \mu_2\lambda_1 + \mu_1\lambda_2))$.

> $E[L_2] = -\lambda_2(-\mu_1\lambda_1^2 + \mu_1\lambda_2\lambda_1 - \mu_1\mu_2\lambda_1 + \mu_2^2\lambda_1 - 2\lambda_2\mu_2\lambda_1 + \mu_2\lambda_1^2 + \mu_1^2\lambda_1 - \mu_1\lambda_2^2 - \mu_1^2\lambda_2 + \mu_1\lambda_2\mu_2 + \mu_1^2\mu_2)/((\mu_1 - \lambda_1 + \lambda_2)(\mu_2 + \lambda_1 - \lambda_2)(-\mu_1\mu_2 + \mu_2\lambda_1 + \mu_1\lambda_2))$.

> $E[L_1^2] - E[L_1] = 2\lambda_1^2(-2\lambda_2^4\mu_1^5 - \mu_2^6\lambda_1^3 - 3\mu_2^4\lambda_1^5 + 2\mu_1^4\mu_2^2\lambda_1\lambda_2^2 + 32\mu_1\lambda_2^2\mu_2^5\lambda_1 - 8\mu_1^4\lambda_2^3\mu_2\lambda_1 + \mu_1^4\lambda_1^2\lambda_2^2\mu_2 - 21\mu_1\mu_2^3\lambda_1^4\lambda_2 - 42\mu_1\lambda_2\mu_2^4\lambda_1^3 + 76\mu_1\lambda_2^2\mu_2^4\lambda_1^2 + 59\mu_1\lambda_2^2\mu_2^3\lambda_1^3 + 12\mu_1\lambda_2^2\mu_2^2\lambda_1^4 - 6\mu_1\lambda_2\mu_2^6\lambda_1 - 13\mu_1\lambda_2^5\mu_2^3 + 4\mu_1\lambda_2^6\mu_2^2 + 3\mu_1\lambda_2^2\mu_2^6 + 5\mu_1^4\mu_2\lambda_2^4 - \mu_1^4\mu_2^2\lambda_2^3 + \mu_1^4\mu_2^3\lambda_2^2 + 54\mu_1\lambda_1\lambda_2^4\mu_2^3 - 61\mu_1\lambda_2^3\mu_2^4\lambda_1 - 29\mu_1\lambda_2\mu_2^5\lambda_1^2 - 5\mu_1^4\lambda_2^5 + 3\mu_1^3\lambda_1^2\mu_2^4 + 9\mu_1\mu_2^4\lambda_1^4 + 9\mu_1\mu_2^5\lambda_1^3 + 3\mu_1\mu_2^6\lambda_1^2 + 17\mu_1\lambda_2^4\mu_2^4 - 11\mu_1\lambda_2^3\mu_2^5 + 3\mu_1\mu_2^3\lambda_1^5 + \mu_1^3\lambda_1^3\mu_2^3 - 2\mu_1^3\lambda_2\mu_2^2\lambda_1^3 - 2\mu_1\lambda_2\lambda_1^5\mu_2^2 - 30\mu_1\lambda_1^3\mu_2^2\lambda_2^3 - 82\mu_1\lambda_1^2\mu_2^3\lambda_2^3 + 36\mu_1\lambda_1^2\mu_2^2\lambda_2^4 - 20\mu_1\lambda_1\lambda_2^5\mu_2^2 - 3\mu_1^5\lambda_1^4 + 18\mu_1^3\lambda_2^2\mu_2^3\lambda_1 + 6\mu_1^3\lambda_2^2\mu_2^2\lambda_1^2 - \mu_1^3\lambda_2^2\mu_2\lambda_1^3 - 9\mu_1^3\lambda_2\mu_2^3\lambda_1^2 - 12\mu_1^3\lambda_2\lambda_1\mu_2^4 + 2\mu_1^3\lambda_2^4\mu_2^2 + 2\mu_1^3\lambda_2^3\lambda_1^3 - 9\mu_1^3\lambda_2^3\mu_2^3 + 11\mu_1^3\lambda_2^2\mu_2^4 - 5\mu_1^3\lambda_2\mu_2^5 + 3\mu_1^3\mu_2^5\lambda_1 - 8\mu_1^3\lambda_2^3\mu_2^2\lambda_1 - 4\mu_1^3\lambda_2^6 + \mu_1^3\mu_2^6 + 22\mu_1^2\mu_2^5\lambda_2\lambda_1 - 4\mu_1^2\lambda_2^4\mu_2\lambda_1^2 - 11\mu_1^3\lambda_2^4\mu_2\lambda_1 - 4\mu_1^2\lambda_2^6\mu_2 + 3\mu_1^2\mu_2^6\lambda_2 + 10\mu_1^3\lambda_2^5\lambda_1 + 4\mu_1^3\lambda_2^5\mu_2 - 8\mu_1^3\lambda_2^4\lambda_1^2 - 3\mu_1^2\mu_2^6\lambda_1 + 4\mu_1^2\lambda_1^4\mu_2^2\lambda_2 - 20\mu_1^2\lambda_1^3\mu_2^2\lambda_2^2 + 39\mu_1^2\lambda_1^2\mu_2^4\lambda_2 - 63\mu_1^2\lambda_1^2\mu_2^3\lambda_2^2 + 43\mu_1^2\lambda_1^2\mu_2^2\lambda_2^3 + 30\mu_1^2\lambda_2^3\mu_2^4 - 14\mu_1^2\lambda_2^2\mu_2^5 - 32\mu_1^2\lambda_2^4\mu_2^3 + 17\mu_1^2\lambda_2^5\mu_2^2 - 9\mu_1^2\lambda_1^2\mu_2^5 - 3\mu_1^2\mu_2^3\lambda_1^4 + \mu_2^6\lambda_2^3 + 8\mu_1^2\lambda_1\lambda_2^5\mu_2 - 3\lambda_2^4\mu_2^5 + \lambda_2^3\lambda_1\mu_1^5 - 3\lambda_2^3\lambda_1^2\mu_1^4 + 8\lambda_2^4\lambda_1\mu_1^4 + 2\lambda_2^3\mu_1^5\mu_2 + 3\lambda_2^5\mu_2^4 - \lambda_2^6\mu_2^3 + 8\lambda_2^3\lambda_1^2\mu_1^3\mu_2 - 44\mu_1^2\lambda_1\lambda_2^4\mu_2^2 - 57\mu_1^2\lambda_1\lambda_2^2\mu_2^4 + 72\mu_1^2\lambda_1\lambda_2^3\mu_2^3 + 24\mu_1^2\lambda_1^3\mu_2^3\lambda_2 - 9\mu_1^2\lambda_1^3\mu_2^4 + 20\lambda_2^3\mu_2^3\lambda_1^3 + 30\lambda_2^3\mu_2^4\lambda_1^2 - 15\lambda_2^4\mu_2^3\lambda_1^2 + 6\lambda_2^5\mu_2^3\lambda_1 - 18\lambda_2^2\mu_2^5\lambda_1^2 - 30\lambda_2^2\mu_2^4\lambda_1^3 + 12\lambda_2^3\mu_2^5\lambda_1 - 15\lambda_2^2\mu_2^3\lambda_1^4 - 15\lambda_2^4\mu_2^4\lambda_1 - 3\lambda_2^2\mu_2^6\lambda_1 + 15\lambda_2\mu_2^4\lambda_1^4 + 3\lambda_2\mu_2^6\lambda_1^2 + 12\lambda_2\mu_2^5\lambda_1^3 + 6\mu_2^3\lambda_1^5\lambda_2 - \mu_2^3\lambda_1^6)/((\mu_1 - \lambda_1 + \lambda_2)^3(\mu_2 + \lambda_1 - \lambda_2)^3(-\mu_1\mu_2 + \mu_2\lambda_1 + \mu_1\lambda_2)^2\mu_2)$.

> $E[L_2^2] - E[L_2] = 2\lambda_2^2(-15\mu_1^4\lambda_1^2\lambda_2^2 + 11\mu_1^4\lambda_2^3\lambda_1 + \mu_1\lambda_2^2\mu_2^4\lambda_1 - 26\mu_1^4\mu_2\lambda_1\lambda_2^2 - 6\mu_1^5\lambda_2\lambda_1\mu_2 - 3\mu_1^5\lambda_2\lambda_1^2 + 26\mu_1^4\lambda_2\mu_2\lambda_1^2 + 19\mu_1^4\lambda_2\mu_2^2\lambda_1 + 11\mu_1^4\lambda_1^3\lambda_2 - 3\mu_1\lambda_1\lambda_2^5\mu_2 + 15\mu_1\lambda_1^2\mu_2\lambda_2^4 - 34\mu_1\lambda_1^2\mu_2^2\lambda_2^3 + 8\mu_2^2\lambda_1^5\lambda_2 + 10\lambda_2\mu_2^4\lambda_1^3 - 26\mu_1\lambda_1^3\mu_2\lambda_2^3 - 11\mu_1\lambda_2\lambda_1^5\mu_2 - 43\mu_1^3\lambda_2\mu_2\lambda_1^3 - 10\mu_1\lambda_2\mu_2^4\lambda_1^2 - 5\mu_1\lambda_2^3\mu_2^3\lambda_1 + 7\mu_1\lambda_1\lambda_2^4\mu_2^2 + \lambda_2\mu_2^5\lambda_1^2 + 16\lambda_2\mu_2^3\lambda_1^4 - 16\lambda_2^2\mu_2^2\lambda_1^4 - 18\lambda_2^2\mu_2^3\lambda_1^3 + 22\mu_1\lambda_2^2\mu_2\lambda_1^4 - 4\lambda_2^2\mu_2^4\lambda_1^2 - 3\lambda_2^4\mu_2^2\lambda_1^2 + 6\lambda_2^3\mu_2^3\lambda_1^2 + 12\lambda_2^3\mu_2^2\lambda_1^3 - 10\mu_1^2\lambda_1^3\mu_2^3 + 3\mu_1^3\lambda_1^5 + 14\mu_1^3\mu_2\lambda_1^4 - 10\mu_1^2\mu_2\lambda_1^5 + 3\lambda_1^6\mu_2\mu_1 - \lambda_1^6\mu_1^2 + 3\mu_1^5\lambda_2^2\mu_2 - 3\mu_1^5\lambda_2\mu_2^2 - \lambda_2^3\mu_1^5 - 18\mu_1^2\mu_2^2\lambda_1^4 - \mu_1^2\lambda_1^2\mu_2^4 + 3\mu_1^2\lambda_2^5\mu_2 - 3\mu_1^2\lambda_2^4\mu_2^2 + \mu_1^2\lambda_2^3\mu_2^3 - 3\mu_1^3\lambda_2^5 + 52\mu_1\lambda_2^2\mu_2^2\lambda_1^3 - \mu_1^2\lambda_2^6 + \mu_1^3\mu_2^4\lambda_1 + 3\mu_1^3\lambda_2^2\mu_2^3 - 9\mu_1^3\lambda_2^3\mu_2^2 + 9\mu_1^3\lambda_2^4\mu_2 + 27\mu_1\lambda_2^2\mu_2^3\lambda_1^2 + 20\mu_1^3\lambda_1^3\mu_2^2 + 7\mu_1\mu_2^2\lambda_1^5 + 2\mu_1\mu_2^5\lambda_1^2 + 5\mu_1\mu_2^4\lambda_1^3 + 7\mu_1\mu_2^3\lambda_1^4 + 10\mu_1^3\lambda_1^2\mu_2^3 - 9\mu_1^4\mu_2^2\lambda_2^2 + 9\mu_1^4\mu_2\lambda_2^3 - 3\mu_1^4\lambda_2^4 + 3\mu_1^4\lambda_2\mu_2^3 + \mu_1^5\lambda_1^3 + 3\mu_1^5\mu_2\lambda_1^2 - 27\mu_1\lambda_2\mu_2^3\lambda_1^3 - 11\mu_1^4\mu_2^2\lambda_1^2 - 4\mu_1^4\mu_2^3\lambda_1 + 3\mu_1^5\mu_2^2\lambda_1 - 3\mu_1^4\lambda_1^4 - 10\mu_1^4\lambda_1^3\mu_2 - 32\mu_1\mu_2^2\lambda_1^4\lambda_2 - 13\mu_1^3\lambda_2\lambda_1\mu_2^3 - 47\mu_1^3\lambda_2\mu_2^2\lambda_1^2 + 22\mu_1^3\lambda_2^2\lambda_1^3 + 59\mu_1^3\lambda_2^2\mu_2\lambda_1^2 + 36\mu_1^3\lambda_2^2\mu_2^2\lambda_1 + 13\mu_1^3\lambda_2^4\lambda_1 - 9\mu_1^2\lambda_2^4\lambda_1^2 + 2\mu_1^2\mu_2^4\lambda_2\lambda_1 - 37\mu_1^3\lambda_1^3\mu_2\lambda_1 - 14\mu_1^2\lambda_1\lambda_2^2\mu_2^3 - 20\mu_1^2\lambda_1\lambda_2^4\mu_2 - 22\lambda_2^3\lambda_1^2\mu_1^3 + 10\lambda_1^3\lambda_2^3\mu_1^2 + 5\lambda_1^5\lambda_2\mu_1^2 + 5\mu_1^2\lambda_1\lambda_2^5 + 46\mu_1^2\lambda_1^2\mu_2\lambda_2^3 - 57\mu_1^2\lambda_1^2\mu_2^2\lambda_2^2 + 21\mu_1^2\lambda_1^2\mu_2^3\lambda_2 - 52\mu_1^2\lambda_1^3\mu_2\lambda_2^2 + 33\mu_1^2\lambda_1^4\mu_2\lambda_2 - 13\mu_1^3\lambda_1^4\lambda_2 - 9\mu_1^2\lambda_1^4\lambda_2^2 + 3\lambda_1\mu_1^5\lambda_2^2 + 53\mu_1^2\lambda_1^3\mu_2^2\lambda_2 + 27\mu_1^2\lambda_1\lambda_2^3\mu_2^2 - \mu_2^2\lambda_1^6 - 5\mu_2^4\lambda_1^4 - 4\mu_2^3\lambda_1^5 - 2\mu_2^5\lambda_1^3 + \mu_1^5\mu_2^3)/((\mu_1 - \lambda_1 + \lambda_2)^3(\mu_2 + \lambda_1 - \lambda_2)^3(-\mu_1\mu_2 + \mu_2\lambda_1 + \mu_1\lambda_2)^2)$.

ρ	Z_{SD}	$E[Z_{P_1}]$	$E[Z_{P_2}]$
0.075	0.144731	0.147810	0.147998
0.150	0.340062	0.352268	0.353768
0.225	0.609888	0.637061	0.642124
0.300	0.993275	1.040898	1.052898
0.375	1.556770	1.629688	1.653125
0.450	2.419670	2.521624	2.562124
0.525	3.810332	3.943209	4.007522
0.600	6.205499	6.368000	6.464000
0.675	10.725960	10.910872	11.047560
0.750	20.500000	20.687500	20.875000
0.825	42.563250	47.129823	47.379385
0.900	123.853900	160.692000	161.016000
0.975	1659.171000	2859.700180	2860.112122

Table 4.7: Comparison of SDP relaxations and policies for a two class queue.

We consider a single station network with two classes. Our objective here is to minimize $\sum_{i=1}^{2} h_i y_{ii}$. We assume that the arrival rate for each class is the same ($= \lambda$); the mean service rates for the two classes are $\mu_1 = 2$ and $\mu_2 = 4$ respectively, with the corresponding h_i being $h_1 = 2$ and $h_2 = 1$. We assume that $c_1 = c_2 = 0$. Since $h_1\mu_1 = h_2\mu_2$ and $c_1\mu_1 = c_2 = \mu_2$, serving the longer queue is optimal. We evaluate the optimal cost for the two policies P_1 and P_2, where P_i gives priority to class i when the queue-lengths are the same. (Both policies serve the longer queue when the queue-lengths are different.) The results of SDP relaxation, and the cost of these two policies are tabulated in Table 4.7.

The computational results suggest that the semidefinite relaxation is good in light and moderate traffic, but can be improved in heavy traffic, assuming our conjecture about the optimality of the proposed policy is true. Thus, it appears that the semidefinite relaxation we consider is *not* exact. Attempts to strengthen the semidefinite relaxation in this special case may lead to new classes of constraints that are useful in other settings as well; for that reason, it would be interesting to find an exact relaxation for this special case.

Objectives involving second moments of system time: We also point out that this framework can be used to derive bounds for problems in which the objective is to minimize

$$\sum_{i=1}^{n} c_i E[S_i^2], \tag{4.35}$$

where S_i is a random variable indicating the time spent by class i jobs in the system (both

ρ	Z_{SD}	$E[Z_{P_1}]$	$E[Z_{P_2}]$
0.750	5.22	7.61 ± 0.14	6.33 ± 0.16
0.825	10.82	15.12 ± 0.35	12.81 ± 0.39
0.900	30.08	45.76 ± 1.53	41.57 ± 1.84
0.975	383.72	693.32 ± 42.32	677.21 ± 48.86

Table 4.8: Minimizing second moment of system time for a multiclass queue.

in queue and in service). By using distributional Little's law, and by noting that the system is "overtake-free," [9] we know that

$$E[L_i(L_i - 1)] = \lambda_i^2 E[S_i^2].$$

Using this identity, we can re-write the objective function in terms of L_i and L_i^2.

Optimizing multiclass single-server systems for objective functions involving convex functions of system times has been addressed by Jan van Mieghem [71]; in this paper, van Mieghem proves that a dynamic index policy, called the "generalized $c\mu$" rule, is asymptotically optimal as $\rho \to 1$. When specialized to the objective of Eq.(4.35), the proposed dynamic index can be described as follows: assume that jobs of each class wait in a separate queues in the order of arrival. The index of class i at time t is $c_i\mu_i a_i(t)$, where $a_i(t)$ is the "age" (i.e. time spent in the system by time t) of the class i job at the head of the queue. (This can be thought of as a "greedy" policy, appropriately generalized for the objective of Eq. (4.35).) In contrast, the policy we propose for this objective defines the index of class i at time t as $c_i\lambda_i^2\mu_i L_i(t)$. We revisit the single station multiclass queue with four classes, and consider the objective of minimizing Eq. (4.35), with $c_i = 1$ for all i. The results are tabulated in Table 4.8, where Z_{P_1} is the performance of van Mieghem's policy, and Z_{P_2} is the performance of our index policy.

From van Mieghem's asymptotic optimality result, it appears that the SDP relaxation can indeed be strengthened. It also strengthens our conjecture that our policy is, in fact, optimal. Interestingly, the deterministic version of this problem, in which the objective is to minimize a weighted sum of squared completion times of a given set of N jobs, is $\mathcal{NP}$-hard.

4.3.3 Performance analysis of priority policies

We consider the Rybko-Stolyar network, and study the problem of performance evaluation of priority policies. In a multiclass setting, evaluating the performance of priority policies

is not an easy problem, even under the usual Markovian restrictions. We shall use linear and semidefinite relaxations to provide bounds on the performance of priority policies. Whenever an exact analysis is possible, we shall compare the bounds to the true values; in all other cases, we shall compute the "true" cost of the policy using simulation, and compare compare the bounds to these results.

Recall that the relaxations Z_{LP} and Z_{SD} are valid for all admissible scheduling policies. In analyzing priority policies, we can add policy-specific constrains to strengthen Z_{LP} and Z_{SD}. For example,

$$x_i^k = 0, \quad \text{if class } i \text{ has higher priority than class } k, \text{ and } i,k \in \mathcal{C}_m, \quad \text{for some } m.$$

For convenience, we say that $i >_\pi j$ if class i has higher priority than class j under priority policy π. We always assume that priorities are imposed in a preemptive manner. Also, whenever the policy π is evident from the context, we drop the subscript π, and simply say $i > j$. The complete set of constraints that can be added for a priority policy π is as follows:

- **Priority constraints:**

$$\begin{aligned} x_i^k &= 0, \quad i,k \in \mathcal{N}, \quad i >_\pi k, \quad i,k \in \mathcal{C}_m, \quad \text{for some } m \\ y_{ij}^k &= 0, \quad i,j,k \in \mathcal{N}, \quad i >_\pi k, \quad i,k \in \mathcal{C}_m, \quad \text{for some } m. \end{aligned}$$

- **Non-idling constraints:**

$$\begin{aligned} x_i^{0m} &= 0, \quad i \in \mathcal{N}, m \in \mathcal{M}, i \in \mathcal{C}_m, \\ y_{ij}^{0m} &= 0, \quad i,j \in \mathcal{N}, m \in \mathcal{M}, i \in \mathcal{C}_m. \end{aligned}$$

For our first experiment, we consider the Rybko-Stolyar network of Figure 4-1. In our computations we fix the service times as shown in the figure, and vary only the arrival rates. We maintain the symmetry between classes, and so we set $\alpha_1 = \alpha_3 = \alpha$, where α varies from 0.1 to 1.18. We select $c_i = 1$ and $d_i = 0$, i.e., we are interested in minimizing the expected number of jobs in the system in steady-state.

We now consider all possible priority policies for this network; for each priority policy, we present upper and lower bounds on performance based on both linear and semidefinite

	Z_{LP}		Z_{SDP}		
ρ	(LB)	(UB)	(LB)	(UB)	$E[Z_P]$
0.083	0.174	0.189	0.174	0.189	0.184 ± 0.001
0.167	0.362	0.436	0.362	0.436	0.416 ± 0.002
0.250	0.574	0.774	0.574	0.774	0.715 ± 0.004
0.333	0.848	1.260	0.848	1.259	1.119 ± 0.006
0.417	1.194	2.004	1.194	2.004	1.693 ± 0.010
0.500	1.647	3.239	1.647	3.239	2.560 ± 0.016
0.583	2.274	5.502	2.274	5.502	3.972 ± 0.027
0.667	3.205	10.238	3.211	9.755	6.508 ± 0.044
0.750	5.290	22.221	5.303	19.095	11.583 ± 0.072
0.833	10.240	63.601	10.240	46.748	24.250 ± 0.111
0.875	15.785	120.255	15.785	84.842	38.670 ± 0.128
0.917	27.798	343.719	28.224	185.980	70.249 ± 0.245
0.958	66.537	1647.618	67.862	621.351	170.97 ± 0.408
0.983	187.388	11621.881	189.5	2703.6	476.512 ± 4.180

Table 4.9: Bounds for priority policy class 1 > class 4, class 2 > class 3.

relaxations.

We first consider the policy in which station 1 gives preemptive priority to class 1, and station 2 gives preemptive priority to class 2. The results are shown in Table 4.9. From Table 4.9, we see that the "gap" between the upper and lower bounds for the semidefinite relaxation is substantially smaller than that of the linear programming relaxation in heavy traffic. The optimal cost of this policy, obtained by simulation, is shown in the last column. By definition, the LP/SDP bounds yield intervals in which the actual cost lies; from Table 4.9, we see that the (approximate) actual cost obtained by simulation is typically closer to the upper bound in light traffic, and closer to the lower bound in heavy traffic.

By symmetry, we observe that the results of Table 4.9 hold good for the priority policy in which station 1 gives preemptive priority to class 4, and station 2 gives preemptive priority to class 3.

We next consider the priority policy in which station 1 gives preemptive priority to class 1, but station 2 gives preemptive priority to class 3. The results for this policy are shown in Table 4.10. Here we see that the LP relaxation yields a fairly narrow interval for the optimal cost. The lower bounds obtained from the SDP relaxation are identical to those of the LP; however, the SDP relaxation successfully reduces the upper bound in (very) heavy traffic. As before, in light traffic, the optimal cost is closer to the upper bound of the relaxations; in heavy traffic, the optimal cost is closer to the lower bound.

ρ	Z_{LP} (LB)	Z_{LP} (UB)	Z_{SDP} (LB)	Z_{SDP} (UB)	$E[Z_P]$
0.083	0.170	0.182	0.170	0.182	0.179 ± 0.0007
0.167	0.347	0.403	0.347	0.403	0.391 ± 0.002
0.250	0.540	0.674	0.540	0.674	0.644 ± 0.003
0.333	0.814	1.014	0.814	1.014	0.955 ± 0.004
0.417	1.169	1.455	1.169	1.455	1.349 ± 0.006
0.500	1.644	2.044	1.644	2.044	1.866 ± 0.009
0.583	2.314	2.874	2.314	2.874	2.578 ± 0.013
0.667	3.323	4.123	3.323	4.123	3.635 ± 0.019
0.750	5.012	6.212	5.012	6.212	5.367 ± 0.029
0.833	8.400	10.400	8.400	10.400	8.759 ± 0.050
0.875	11.794	14.594	11.794	14.594	12.195 ± 0.074
0.917	18.588	22.988	18.588	22.350	18.957 ± 0.123
0.958	38.982	48.182	38.982	44.487	39.299 ± 0.289
0.983	100.182	123.784	100.182	108.931	100.412 ± 0.839

Table 4.10: Bounds for priority policy class 1 > class 4, class 3 > class 2.

Finally, we analyze the priority policy in which station 1 gives preemptive priority to class 4 and station 2 gives preemptive priority to class 2. The results for this policy are shown in Table 4.11. It is well known that the network is unstable under this priority policy for $\rho > 0.625$. We should point out that all of our relaxations are *valid* only for stable policies. Thus, even though the LP and SDP bounds indicate instability for $\rho > 0.625$, this should not be construed as a check of stability. Strictly speaking, our relaxations require that the network be stable, which should be checked independently. The bounds appear to be weakest in this setting.

From the above examples we saw that the bounds obtained from the SDP relaxations sometimes (but not always) strictly improve on the LP bounds. We also saw that the bounds were worse when a single job type had higher priority (i.e. class 1 > class 4, and class 2 > class 3; or when class 4 > class 1, and class 3 > class 2.) One reason for this is that if the priorities of the job types are the same at various stations, intuitively, certain queue-lengths belonging to different job types become less correlated. Since none of our relaxations capture this "independence" between job classes, the bounds are conservative, and are worse when compared to the actual costs.

To better test this observation, we next considered the same network with a different set of parameters, shown in Figure 4-5. Specifically, we let the service times of all of the job classes served at a single station be identical. We analyze the policy in which classes 1 and

ρ	Z_{LP}		Z_{SDP}		$E[Z_P]$
	(LB)	(UB)	(LB)	(UB)	
0.083	0.179	0.193	0.179	0.193	0.190 ± 0.0009
0.167	0.389	0.461	0.389	0.461	0.444 ± 0.002
0.250	0.640	0.857	0.640	0.857	0.806 ± 0.005
0.333	0.947	1.511	0.947	1.511	1.368 ± 0.009
0.417	1.333	2.800	1.333	2.800	2.377 ± 0.019
0.500	1.840	6.571	1.840	6.571	4.716 ± 0.470
0.583	2.539	238.030	2.539	238.030	16.149 ± 0.255
0.667	-	-	-	-	∞
0.750	-	-	-	-	∞
0.833	-	-	-	-	∞
0.875	-	-	-	-	∞
0.917	-	-	-	-	∞
0.958	-	-	-	-	∞
0.983	-	-	-	-	∞

Table 4.11: Bounds for priority policy class 4 > class 1, class 2 > class 3.

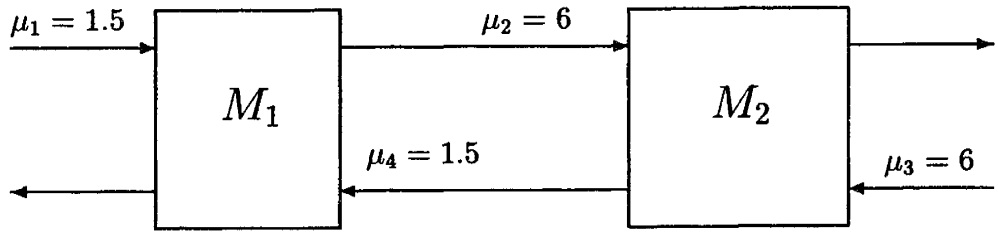

Figure 4-5: Rybko-Stolyar network.

2 have preemptive priority over classes 4 and 3 respectively. As before, we vary only the arrival rates, while maintaining the symmetry between classes; and so we set $\alpha_1 = \alpha_3 = \alpha$, where α varies from 0.05 to 0.74. We select $c_i = 1$ and $d_i = 0$, i.e., we are interested in minimizing the expected number of jobs in the system in steady-state. The results for this experiment is displayed in Table 4.12. Except at very heavy loads, the LP and SDP bounds are identical. For this special case, it is possible to analyze the performance of this priority policy explicitly — this follows from the observation that the random vector (L_1, L_2) has a joint distribution that is product-form; similarly, the random vector $(L_1 + L_4, L_2 + L_3)$ has a joint distribution that is product-form. In fact, it is easy to see that $\rho(\mathcal{C}_m) = \sum_{j \in \mathcal{C}_m} \rho_j$,

$$E\Big[\sum_{i=1}^{4} L_i\Big] = \frac{\rho(\mathcal{C}_1)}{1-\rho(\mathcal{C}_1)} + \frac{\rho(\mathcal{C}_2)}{1-\rho(\mathcal{C}_2)},$$

where $\rho(\mathcal{C}_i)$ is the traffic intensity at station i. The exact value of the performance is shown in the last column of Table 4.12; this was first communicated to us by Philippe Afeche [1]. As before, we see that the actual value lies closer to the upper bound in light traffic, and lies closer to the lower bound in heavy traffic. Even for this simple example we observe a gap in the values between the SDP bounds and the actual optimal cost. In this case, the gap can be explained by the following observation: in actuality, the random variables L_1 and L_2 are independent; similarly, the random variables $(L_1 + L_4)$ and $(L_2 + L_3)$ are independent. The LP and SDP relaxations, however, do not take advantage of this fact. We can add the following constraints to "strengthen" the relaxations in this special case.

$$\begin{aligned} E[L_1|B_2 = 1] &= E[L_1], \\ E[L_2|B_1 = 1] &= E[L_2], \\ E[L_1 + L_4|B^2 = 0] &= E[L_1 + L_4], \\ E[L_2 + L_3|B^1 = 0] &= E[L_2 + L_3]. \end{aligned}$$

The strengthened relaxation is indeed exact for this example.

4.3.4 Extensions

We now review briefly three extensions that can be incorporated into the basic framework: closed networks, probabilistic routing, and distributions that have rational Laplace trans-

α	Z_{LP} (LB)	Z_{LP} (UB)	Z_{SDP} (LB)	Z_{SDP} (UB)	$E[Z_P]$
0.05	0.086	0.089	0.086	0.089	0.088
0.10	0.182	0.192	0.182	0.192	0.188
0.15	0.285	0.312	0.285	0.312	0.303
0.20	0.401	0.454	0.401	0.454	0.435
0.25	0.546	0.624	0.546	0.624	0.591
0.30	0.720	0.826	0.720	0.826	0.778
0.35	0.937	1.075	0.937	1.075	1.007
0.40	1.214	1.393	1.214	1.393	1.297
0.45	1.581	1.814	1.581	1.814	1.676
0.50	2.091	2.398	2.091	2.398	2.2
0.55	2.851	3.268	2.851	3.268	2.974
0.60	4.111	4.709	4.111	4.709	4.250
0.65	6.622	7.580	6.622	7.580	6.777
0.70	14.132	16.165	14.132	15.737	14.304
0.74	74.143	84.751	74.152	77.803	74.327

Table 4.12: Bounds for priority policy class 1 > class 4, class 2 > class 3.

forms. We address each of these in the following sections.

Closed networks

By using additional variables we can model compute bounds on the throughput achievable in closed networks. A summary of results for the closed re-entrant line of figure 4-6 appears in Table 4.13.

N	Z_{LP} Lower bound	Z_{LP} Upper Bound	Z_{SD} Lower Bound	Z_{SD} Upper Bound
20	0.195	0.240	0.204	0.239
100	0.237	0.248	0.239	0.248

Table 4.13: Bounds for network of figure 4-6.

Probabilistic routing

We can incorporate models in which the routing is subject to optimization as well. Again, we can compute lower bounds on performance for this class of models, which is accomplished by enlarging the space of variables. We omit the details here. For illustrative purposes, we

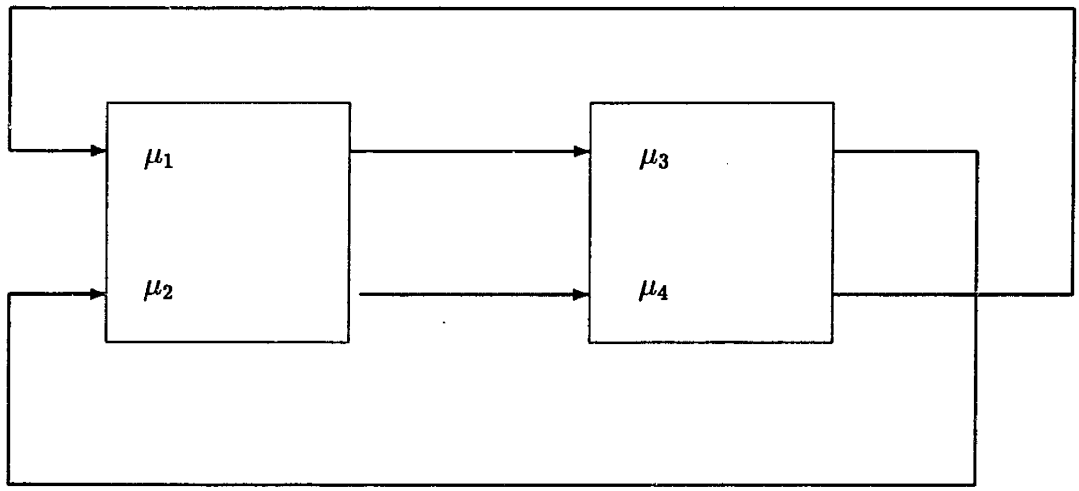

Figure 4-6: A balanced closed re-entrant line
$\mu_1 = 1/2, \mu_2 = 2/7, \mu_3 = 1, \mu_4 = 2$

λ	Z_{LP}	Z_{SD}	$E[Z_{SQ}]$
1.5	1.00	1.03	1.53 ± 0.02
2.1	2.33	2.39	2.91 ± 0.09
2.7	9.00	9.13	9.73 ± 0.74

Table 4.14: Bounds for probabilistic routing example

present results on an two station example with routing. Customers arrive in a Poisson manner at rate λ and have to be routed to one of two machines. Service times are exponentially distributed with rate $\mu = 1.5$ at both stations. Our objective is to find a (probabilistic) routing policy that minimizes the expected number of customers in the system. We consider three regimes: Light traffic ($\lambda = 1.5$), medium traffic ($\lambda = 2.1$) and heavy traffic ($\lambda = 2.7$). As noted by Bertsimas, Paschalidis and Tsitsiklis [14], the linear program can be solved in closed form for λ in this range; in fact, the optimum LP value is

$$Z_{LP} = \frac{\lambda}{2\mu - \lambda}.$$

From Table 4.14, we see that the semidefinite relaxation improves the lower bound only marginally. For comparison purposes, we simulated the dynamic routing policy in which arriving customers join the shortest queue immediately upon arrival. Foschini and Salz [30] have shown that this policy achieves the lower bound asymptotically, as $\rho \to 1$.

ρ	Z_{LP}	Z_{SD}	$E[Z_{LBFS-B}]$	Best B
0.083	0.169	0.169	0.176 ± 0.001	0
0.167	0.345	0.345	0.377 ± 0.001	0
0.250	0.531	0.532	0.611 ± 0.002	1
0.333	0.787	0.787	0.887 ± 0.003	1
0.417	1.091	1.107	1.223 ± 0.005	1
0.500	1.391	1.403	1.647 ± 0.006	1
0.583	1.887	1.913	2.205 ± 0.008	1
0.667	2.389	2.393	2.988 ± 0.012	1
0.750	3.481	3.498	4.212 ± 0.018	1
0.833	5.377	5.419	6.532 ± 0.031	1
0.875	7.397	7.673	8.790 ± 0.045	1
0.917	11.348	11.754	13.192 ± 0.076	1
0.958	22.853	22.961	26.222 ± 0.180	2
0.983	57.247	57.981	64.696 ± 0.532	2

Table 4.15: Results for the network of Figure 4-1 with Erlang(2) service times.

General distributions

We can extend the above framework to arrival/service distributions that have rational Laplace transforms. This is easily done using the method of stages (see Kleinrock [45]): we can use appropriate subnetworks (see Kumar and Kumar [46]) to model customers that are in the process of arriving or in the process of being served. We impose additional constraints that limit the number of customers in each subnetwork to be at most 1. For illustrative purposes we present results for the Rybko-Stolyar network of Figure 4-1, in which the service distributions are Erlang of order two (with the same mean). Since this decreases variability, we expect the average number of customers in the system to be smaller. Indeed we observe this effect in the results shown in Table 4.15.

4.4 Concluding Remarks

In this chapter we have explored the effectiveness of semidefinite relaxations for stochastic optimization problems. From our study we conclude that semidefinite relaxations improve the lower bounds marginally for objectives involving first moments, but lead to substantially better lower bounds for objective functions involving second moments. Based on our results we believe that these relaxations can be further strengthened: as a touchstone for this objective, we pose the problem of finding an exact relaxation to minimize average squared

system time in a single station multiclass queue. Proving the optimality of the generalized index rule we identified is also of independent interest.

Whenever applicable we derived heuristic policies based on the fluid relaxation. While there have been attempts to formalize this "translation," most of the known results guarantee the "goodness" of the translation only in a transient regime (see, for example, Maglaras [53]). An important and interesting open problem is to prove the effectiveness of such translation procedures (or find good translation procedures) for steady-state problems.

In closing, we believe that the combination of fluid optimal control methods to generate near optimal policies for large scale problems (see Luo and Bertsimas [52]), and linear/semidefinite relaxations to provide near optimal bounds is a promising methodology to address the multiclass queueing network optimization problem.

Chapter 5

Optimal Control of Fluid Tandem Networks

5.1 Introduction

In chapters 2 through 4 we have explored various ways in which the optimal solution to a fluid relaxation can be used to design effective scheduling policies for a variety of models. For a class of approximate models to be successful, however, we need to address an additional issue: the complexity of solving the approximate model. In this chapter we address this issue for a simple class of models, namely, single-class tandem networks. Essentially, we consider a push model of a manufacturing system with the objective of minimizing holding costs. We provide polynomial-time algorithms for the special case in which the buffer holding costs are non-decreasing along the route. An interesting feature of our approach is that we do not use time-discretization explicitly.

Structure of the chapter. The rest of this chapter is organized as follows. In §5.2, we provide a formal description of our model. For the class of networks we consider, we provide a simpler proof of a structural property in §5.3; this property was originally proved by Luo [51]. We use this structural property in §5.4 to formulate our problem as an optimization problem in which the objective function is convex and piecewise quadratic. In §5.5, we show how such a formulation gives rise to an efficient algorithm for some special cases. Specifically, we show that tandem networks with non-decreasing cost structures can be solved using a single convex quadratic program. Furthermore, we show that this convex programming

problem can be solved as a shortest path problem on a suitably defined graph. We conclude the chapter by proving some additional properties in §5.6; our hope is that these ideas will lead to an efficient algorithm for the general fluid tandem network.

5.2 Model

We consider the optimal control of a fluid tandem network with n single-server stations (see Figure 5-1).

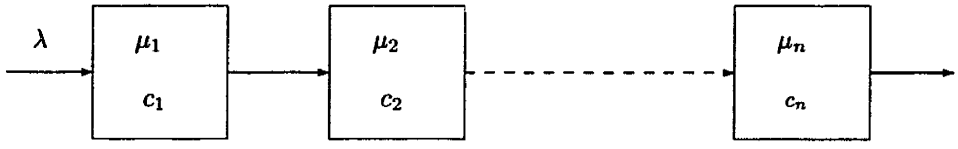

Figure 5-1: An n station tandem network

In a fluid network, jobs are continuous and hence arbitrarily divisible. Also, arrivals and service completions occur continuously, at a deterministic rate. Since there is no competition, we use the terms "station" and "class" interchangeably; also, we use the terms "fluid" and "jobs" interchangeably. The stations are conveniently numbered 1 through n; external arrivals enter the network at station one and depart after being processed at stations $1, 2, \ldots, n$ (in that order). For ease of exposition, we assume the existence of two dummy stations: station 0 from which all the arrivals come to station 1, and station $n+1$ which collects all the departures from station n. The *capacity* of station i is μ_i: this is the rate at which fluid gets cleared from station i if the server at station i processes fluid at full capacity. The arrival rate into station 1 is λ; for convenience (and consistent with our naming convention for job classes) we will use μ_0 and λ interchangeably. Each station has an infinite capacity buffer to hold jobs waiting to be processed. Holding costs are incurred for storing fluid in the buffers; the cost per unit per unit time at buffer i is c_i. We set $c_0 = c_{n+1} = 0$. The buffer level of station i at time t is denoted by $x_i(t)$. For each station i, we are given an initial buffer level that is strictly positive; The initial buffer level at station i, $x_i(0)$, is also denoted by x_i. As a convention, we set $x_{n+1} = 0$. Our objective is to

minimize the total cost incurred

$$V = \int_0^T \sum_{i=1}^{n} c_i x_i(s) ds \tag{5.1}$$

until the first time T at which all classes become empty. For the network to reach a state in which all of the classes are empty, it is necessary for the stations to have sufficient capacity to clear the incoming demand (i.e., the traffic intensity at each station is strictly smaller than one). In the rest of this chapter, we will assume that this condition always holds: thus, $\mu_i > \lambda$ for $1 \leq i \leq n$. An immediate consequence of this assumption is that the cost incurred beyond T by an optimal fluid control is zero. So we can use ∞ instead of T in equation (5.1) without loss of generality, which we do, whenever convenient, in the rest of this chapter.

We can formulate our problem as a continuous linear program in the following manner. For $1 \leq i \leq n$, let $x_i(t)$ be the buffer level of station i at time t, and $u_i(t)$ be the rate at which station i is operated at time t. These are the *decision variables.* For convenience, we also introduce the decision variable $u_0(t)$, which is always set to λ. The problem of optimal control of an n-station fluid tandem network can then be formulated as:

$$\text{(TQ)} \qquad \text{Minimize} \quad \int_0^\infty \sum_{i=1}^{n} c_i x_i(t)\, dt$$

$$\text{subject to}$$

$$\begin{aligned}
\dot{x}_i(t) &= u_{i-1}(t) - u_i(t), \quad i = 1, 2, \ldots, n, \\
x(0) &\quad \text{given}, \\
u_i(t) &\leq \mu_i, \quad i = 1, 2, \ldots, n, \\
u_i(t), x_i(t) &\geq 0, \quad i = 1, 2, \ldots, n.
\end{aligned}$$

As discussed before, problem (TQ) is a special case of a continuous linear program (CLP). In fact, (TQ) belongs to a well studied special class of CLP called *separated continuous linear programs* (SCLP). In this chapter we will briefly review some properties of (SCLP) that are most relevant for our purposes; for a thorough discussion of SCLPs we refer the reader to the book by Anderson and Nash [3], and the papers by Pullan [63].

Separated Continuous Linear Programs (SCLPs). Separated continuous linear programs were introduced by Anderson [2] in an attempt to model job shop scheduling problems. The general form of SCLPs is as follows:

$$\begin{aligned} \text{SCLP:} \quad \text{Minimize} \quad & \int_0^T c(t)'u(t)\,dt \\ \text{subject to} \quad & \int_0^t G\,u(s)ds \;+\; x(t) = a(t) \\ & Hu(t) + z(t) = b(t) \\ & x(t), u(t), z(t) \geq 0, \;\; t \in [0,T] \end{aligned}$$

Here, $u(t)$, $z(t)$, $b(t)$ and $c(t)$ are bounded measurable functions and $x(t)$ and $a(t)$ are continuous functions. These problems are called *separated* because the constraints are naturally separated into two parts—the instantaneous constraints and the integral constraints. An alternate formulation of SCLP can be obtained by integrating the dynamics, and using integration by parts. This is the natural formulation as a linear optimal control problem with state positivity constraints, and is given by:

$$\begin{aligned} \text{CONTROL:} \quad \text{Minimize} \quad & \int_0^T (\alpha(t)'u(t) + \beta(t)'x(t))\,dt \\ \text{subject to} \quad & \dot{x}(t) = \dot{a}(t) - Gu(t) \\ & Hu(t) \leq b(t) \\ & x(0) = a(0) \\ & x(t), u(t) \geq 0, \;\; t \in [0,T] \end{aligned}$$

Here, we leave $\alpha(t)$ and $\beta(t)$ unspecified—the point to note is that we can eliminate $x(t)$ from the objective function in CONTROL (replace $x(t)$ by $\int_0^t \dot{x}(t)\,dt$, and use the constraint on $\dot{x}(t)$ to eliminate any term involving $x(t)$ from the objective function). So, the objective function in both the problems involve only $u(t)$, and the constraints are identical—this establishes the equivalence of CONTROL and SCLP. The following three propositions are

all we need about SCLPs; these are proved in Anderson and Nash [3] (pages 135-142).

Proposition 5.1 *If the feasible region of an SCLP is nonempty and bounded, then there exists an optimal extreme point solution for SCLP.*

Proposition 5.2 *A feasible solution* $\omega(t) = (x(t), u(t), z(t))$ *for SCLP is an extreme point solution if and only if the columns of the matrix*

$$\begin{pmatrix} G & I & 0 \\ H & 0 & I \end{pmatrix}$$

corresponding to the support of $\omega(t)$ *(i.e.,* i *such that* $\omega_i(t) > 0$*) are linearly independent for almost all* $t \in [0, T]$.

Proposition 5.3 *Suppose that* $a(t)$ *and* $c(t)$ *are piecewise linear,* $a(t)$ *is also continuous, and* $b(t)$ *is piecewise constant on* $[0, T]$. *Suppose also that the feasible region of SCLP is nonempty and bounded. Then there exists an optimal solution for SCLP with a piecewise constant* $x(t)$ *on* $[0, T]$.

We also note that in recent years, several algorithms have been proposed to solve SCLPs (see the papers of Pullan [63], and Luo & Bertsimas [52]). While these algorithms are not theoretically efficient, they perform extremely well in practice. These algorithms, however, are not directly relevant to our goal in this chapter, which is to find a polynomial-time algorithm for the optimal control of a fluid tandem network.

Related work: Fluid models have received a lot of attention in recent years, mostly in connection with proving global stability results for multiclass queueing networks. The complexity of optimal control of fluid networks has been addressed in very few papers. To our knowledge, only two other classes of fluid networks are known to have polynomial-time algorithms: single-station fluid networks [5] and two-station re-entrant lines [75]. The papers of Avram, Bertsimas and Ricard [5], and Weiss [73, 74] consider particular examples, and provide complete solutions. Chen and Yao [21] consider a push model with holding cost objectives, and provide an algorithm that computes the optimal fluid solution whenever there is a *unique globally optimal* solution. Such a form of optimality is too strong, and in fact, they show simple examples that do not admit globally optimal allocations. We

also note that optimal draining in *minimum time* is somewhat simpler, and can be solved explicitly (see Weiss [73], Bertsimas and Gamarnik [8]). Finally, Hajek and Ogier [35] address a single-source, single-destination version of the problem with *unit* costs, and provide polynomial-time algorithms to compute an optimal control.

The work closest in spirit to ours is the paper of Perkins and Kumar [62], in which the authors consider a *pull* model of a manufacturing system. Their model is similar to ours, except that: (i) there is an infinite supply of parts at buffer one, whose holding cost rate is zero (ii) there is a constant external demand d for material from the last buffer (hence the term "pull" manufacturing system) and (iii) demand not met immediately is back-ordered. Their objective is to minimize the infinite horizon total costs incurred; costs here include holding costs (at all stations) plus shortfall costs at the last station. Just as in our problem, the system will eventually empty as long as the system has sufficient capacity to meet demand, and so the optimal cost incurred is finite. Perkins and Kumar [62] show that if holding costs are increasing along the route, the problem reduces to a series of quadratic programming problems with linear inequality constraints. In contrast to their work, we consider a *push* model of a manufacturing system. We show that if the buffer holding costs are non-decreasing along the route, an optimal fluid solution can be computed by solving a single *convex* quadratic programming problem; we also show that this convex quadratic program can be solved much more efficiently as a shortest path problem on an associated graph. Finally, we prove some additional properties which might be helpful in resolving the case of general costs.

5.3 A structural property

We have seen that problem *(TQ)* admits a natural formulation as a separated continuous linear programming problem *(SCLP)*. It is easy to verify that the data for problem (TQ) satisfies the hypotheses of Proposition 5.3; this readily shows that problem (TQ) always admits an optimal basic feasible solution with a piecewise constant control. In this section we shall prove that we can always find an optimal control satisfying an interesting structural property, which enables us to infer the existence of an optimal control with a *small* number of pieces. This property was originally proved by Luo [51]; we provide a simpler proof.

In what follows, we use (i, j) to refer to the set of stations set $\{i+1, i+2, \ldots, j-1\}$.

Recall that at time t, station i has inventory level $x_i(t)$ and has instantaneous rate of operation $u_i(t)$. Also, the storage costs for the buffers at station i are c_i, and the costs do not vary with time.

We first classify the stations in the tandem network into *static* and *dynamic* stations: station i is static if $c_i > c_{i+1}$, and is dynamic otherwise.

Our main objective in this section is to show that we can always find an optimal solution to problem TQ satisfying the following property:

Theorem 5.1

- *If i is a static station, station i works at its maximum processing rate μ_i until the time T_i, when class i becomes empty. After this time, the processing rate of station i is determined by the flow conservation equations—station i works at the rate required to maintain its buffer level at zero.*

- *For each dynamic station i, there exists times s_i, T_i such that $u_i(t) = 0$ for $t \in [0, s_i)$, $x_i(t)$ remains positive for $t \in [0, T_i)$, and $x_i(t)$ is zero in $[T_i, \infty)$. Furthermore, $u_i(t)$ can be easily determined for $t \in [s_i, T_i]$, as a function of the capacities of the other stations in the system.*

Remark: The times s_i and T_i in the statement of Theorem 5.1 will also be referred to as the *start* time and the *depletion* time for station i; we use these terms for static stations as well, with the understanding that s_i is zero if i is a static station.

We now prove Theorem 5.1 in the rest of this section. To that end, we introduce a few additional definitions. We define a dynamic relation N among stations by letting $N(i,t)$ denote the first station following station i that has a *non-empty* buffer at time t. If, at time t, none of the stations after station i has any inventory, then there is no next positive inventory station for i; in this case we say $N(i,t) = (n+1)$. We also introduce a decomposition of the machines into "sections," $B_1 = \{1, 2, \ldots b_1\}$, $B_2 = \{b_1 + 1, \ldots b_2\}$, $B_3 = \{b_2 + 1, \ldots b_3\}, \ldots B_r = \{b_{r-1} + 1, \ldots b_r = n\}$. Notice that the sections are determined once we determine the stations $b_1, b_2, \ldots b_m = n$. The b_i are defined recursively as follows:

$$b_{i-1} = \max\{0 \leq j < b_i : \mu_j < \mu_{b_i} \text{ or } c_j \neq \min_{k \in [j, b_i]} c_k\}$$

The B_i are called sections. In any section B_i, the first station is called its head and the last station is called its tail.

This decomposition will be useful in proving Theorem 5.1. (Perkins and Kumar [62] adopt a similar approach for a different problem.)

Lemma 5.1 *There always exists an optimal basic feasible solution to problem (TQ) with a piecewise constant optimal control $u(t)$. Furthermore, if $(u(t), x(t))$ is such a solution, then the following property holds for almost all $t \in [0, T]$.*

- *If $x_i(t) > 0$ and $N(i,t) = j$, then either $u_i(t) = 0$ or $u_i(t) = \min_{k \in [i, j-1]} \mu_k$.*

Proof: First we note that the data for our problem satisfies the hypotheses of proposition 5.3, and so we can conclude that there is always an optimal solution that has a piecewise constant control. For the second part of the lemma, observe that feasibility implies $u_i(t) \leq \min_{k \in [i, j-1]} \mu_k$. The fact that one can always find an optimal *extreme point* solution proves $u_i(t) = \min_{k \in [i, j-1]} \mu_k$. For otherwise, by increasing and decreasing $u_i(t)$ in this *piece*, we can construct two feasible solutions, which average to u, contradicting the fact that u is an extreme point solution. ■

Lemma 5.2 *Let $(u(t), x(t))$ be an optimal basic feasible solution to (TQ), with $u(t)$ piecewise constant. Suppose $x_i(t) > 0$, and $j = N(i,t)$. If $c_i > \min_{k \in [i,j]} c_k$, then $u_i(t) > 0$.*

Let us interpret the lemma before attempting a proof. Suppose we are operating the system in an optimal fashion, and suppose stations i and j have positive inventory, while none of the stations in (i, j) has any inventory. Under what conditions will we let i idle? The lemma says that for i to idle, it must necessarily be the cheapest station in $[i, j]$ (i.e. storage costs in b_i are the smallest among the buffers $\{b_i, b_{i+1}, \ldots, b_j\}$). Otherwise, we can let i work and pump some material to the cheapest station in $[i, j]$.

Proof: Suppose the contrary. Thus, $i < j$, $c_i > \min_{k \in [i,j]} c_k$, and $u(t) = 0$. So, we can find times t_1, t_2, such that $0 \leq t_1 < t_2 \leq T$, $x_i(t_1) > 0$, but $u_i(t) = 0$ for all $t \in [t_1, t_2]$. Let j' be the cheapest station in $[i, j]$. We construct a new feasible solution $(u'(t), x'(t))$, which is less expensive compared to the current solution. Let $u'_k(t) = u_k(t)$ for all $k \notin [i, j' - 1]$. Let

$\delta = \min\{x_i(t_1), (t_2 - t_1)\min_{k\in[i,j'-1]}\mu_k\}$. For station i, we define:

$$x_i'(t) = \begin{cases} x_i(t) & \text{for } t \le t_1; \\ x_i(t_1) - \delta(t - t_1)/(t_2 - t_1) & \text{for } t \in (t_1, t_2); \\ \max\{x_i(t) - \delta, 0\} & \text{for } t \ge t_2 \end{cases}$$

Clearly, $x_i'(t)$ uniquely determines a control $u_i'(t)$, and $u_i'(t) \le u_i(t)$ for all $t > t_2$. Obviously, control variables obviously change only for stations in the section (i, j). Thus, for $k \in [i+1, j'-1]$, we define $u_k'(t)$ as:

$$u_k'(t) = \begin{cases} u_k(t) & \text{for } t \le t_1; \\ \delta/(t_2 - t_1) & \text{for } t \in (t_1, t_2); \\ u_k(t) & \text{for } t \ge t_2,\ x_k'(t) > 0 \\ \max\{u_{k-1}'(t) - u_{k-1}(t) + u_k(t), 0\} & \text{for } t \ge t_2, x_k'(t) = 0 \end{cases}$$

This definition clearly implies $0 \le u_k'(t) \le u_k(t)$ for all $t \ge t_2$, and $x_k'(t) \le x_k(t)$ for all t and $k \in [i, j'-1]$. The fact that the total material in the section $[i, j']$ does not change (i.e., $\sum_{k\in[i,j']} x_k(t) = \sum_{k\in[i,j']} x_k'(t)$) shows that the new solution is strictly better than the old one, which is a contradiction. ■

Lemma 5.3 *Let $(u(t), x(t))$ be an optimal basic feasible solution to (TQ), with $u(t)$ piecewise constant. Suppose $x_i(t) > 0$, and $j = N(i,t)$. If $c_i < c_j$ and $\mu_i > \mu_j$, then $u_i(t) = 0$.*

Proof: The proof of this lemma is similar to the proof of Lemma 5.2: we suppose the contrary, and exhibit a solution with strictly smaller cost. ■

Lemma 5.4 *Let $(u(t), x(t))$ be an optimal basic feasible solution to (TQ) such that $u(t)$ is piecewise constant. Let t_i' and t_i'' be such that $0 \le t_i' < t_i'' \le T$. If $u_i(t) = 0$ in the interval (t_i', t_i'') and $x_i(t_i') > 0$, then $u(t) = 0$ for all $t \in [0, t_i')$.*

Proof: Suppose not (i.e., station i was busy before time t_i'), and let $j = N(i, t_i')$. Using Lemma 5.2, we may conclude that $c_i = \min_{k\in[i,j]} c_k$. Since station i has been busy before time t_i', some material would have been sent from i before t_i'. Since j is the next positive inventory station at time t_i', (without loss of generality) some of the current material at station j was sent by i. Think of a new control in which an amount δ, which is currently at

station j, was withheld at station i itself. Clearly, the only change resulting from this is that δ amount of fluid spends more time at station i and less at station j. Since i is the cheapest station in this section $[i, j]$, the new solution is cheaper, contradicting the optimality of the old solution. ■

Lemma 5.5 *There is an optimal solution* $(u(t), x(t))$ *to problem (TQ) such that if* $x_1(t') = 0$, *then* $x_1(t) = 0$ *for all* $t \in [t', T]$.

Proof: Let t' be the first time instant at which $x_1(t') = 0$. By Lemma 5.4, station 1 always works at a positive rate, which by Lemma 5.1 cannot be smaller than λ (because $\mu_i > \lambda$ for all $i \geq 1$). ■

Next we extend Lemma 5.5 to the other stations in the network.

Lemma 5.6 *There is an optimal solution* $(u(t), x(t))$ *to problem (TQ) such that if* $x_i(t') = 0$, *then* $x_i(t) = 0$ *for all* $t \in [t', T]$, *for* $1 \leq i \leq n$.

Proof: To prove this lemma, we make use of the following observations, which are immediate from the definition of "sections."

- Within each section B_i, the holding costs increase. In other words, the "cheapest" station in any section is its head.
- Within each section B_i, all stations are faster than b_i. In other words, the tail station in any section is its slowest.
- Let h_i, h_{i+1} be the head stations of two consecutive sections and let b_i, b_{i+1} be the corresponding tails. Then, b_i is either more expensive than h_{i+1} or slower than b_{i+1}. (Notice that we do not claim anything about the relationship between h_i and stations in section B_{i+1}.)

The first two observations, together with Lemma 5.3, suggest that within a section, the controls follow a very simple pattern. The maximum rate at which material can get cleared from a section is determined by its tail station. Since the holding costs increase from head to tail, an optimal control exists in which no station processes any material until all the stations ahead of it in its section have cleared their inventory. Processing any earlier would

mean incurring storage costs at more expensive stations without a corresponding gain in the *time* taken to clear this material.

Suppose there is some station which clears its inventory and has positive inventory at a later time. There is a first time at which this happens (say t) and let the station at which this happens be j. By the above decomposition, j is in some section B_i. Clearly, j has to be the head of its section ($j = b_{i-1} + 1$)—otherwise by the first two observations, we would have been better off storing this material at $b_{i-1} + 1$ instead of processing it. The third observation helps us prove that this cannot happen to any head station. By the third observation, either B_{i-1} is slower than B_i (i.e. $\mu_{b_{i-1}} < \mu_{b_i}$) or $c_{b_{i-1}} > c_{b_{i-1}+1}$. If B_{i-1} is slower than B_i, it is impossible for stations in B_i to have positive inventory again—stations in B_i are capable of processing any material sent by B_{i-1} without incurring any storage. Also, if b_{i-1} were more expensive and faster than $b_{i-1} + 1$, then Lemma 5.2 shows that section B_i could never have become empty in the first place (assuming section B_{i-1} had some material left over). Thus the only possibility is if section B_{i-1} also went from zero to positive inventory some time prior to time t. But by assumption, t was the first time and B_i was the first section to have positive inventory again, a contradiction. The proof is complete by observing that the head of the first section cannot have positive inventory again (Lemma 5.5). ■

Proof of Theorem 5.1: If i is a static station, by definition $c_i > c_{i+1}$. So the hypotheses of Lemma 5.2 remain satisfied until station i depletes its inventory, and so $u_i(t) > 0$ for all t. By Lemma 5.1, $u_i(t) = \min_{k \in [i, N(i,t)-1]} \mu_k$. Since station i works at rate μ_i initially, we see that any station k with $\mu_k < \mu_i$ cannot belong to the interval $[i, N(i,t) - 1]$ at any t. So station i works at rate μ_i until it depletes its inventory. If i is a dynamic station, Lemmas 5.4 and 5.6 establish the existence of times s_i and T_i respectively. ■

Theorem 5.1 has several useful corollaries, two of which we explicitly state next.

Corollary 5.1 *There exists an optimal solution $(u(t), x(t))$ to problem (TQ) such that $u(t)$ has at most $2n$ breakpoints.*

Proof: Consider any basic feasible solution that is optimal for (TQ). Let station i first start work at time t_i. Lemma 5.4 shows that after time t_i, station i never idles whenever it has positive inventory. Suppose station i first clears its inventory at time t_i'. Lemma 5.6 shows that inventory at station i is always zero after t_i'. Thus, after a station clears its inventory,

its working rate is always controlled (rather determined) by its incoming flow rate. For any $t \in (t_i, t_i')$, let $j = N(i,t)$. Lemma 5.1 shows that $u_i(t) = \min_{k\in[i,j-1]}\mu_k$. All these statements show that the working rate of any station changes only when a station starts work or clears its inventory. Since each station starts work exactly once and clears inventory exactly once, we have an optimal basic feasible solution with at most $2n$ breakpoints. ■

Corollary 5.2 *There is always an optimal solution to problem (TQ) such that station i works at a constant rate*

$$\hat{\mu}_i \;=\; \min_{k\in[i,N^s(i))}\mu_k$$

in the interval $[s_i, T_i)$.

rProof: If i is a static station, $N^s(i) \;=\; (i+1)$, and so $\hat{\mu}_i = \mu_i$; and the first part of Theorem 5.1 proves the corollary. If i is a dynamic station, the rate at which station i works at time s_i is at most $\min_{k\in[i,N^s(i))}\mu_k$ (as all of the station with zero inventory have to remain at zero level); since $u_i(t) > 0$ for $t \geq s_i$, by Lemma 5.1 $u_i(t)$ *equals* $\hat{\mu}_i \;=\; \min_{k\in[i,N^s(i))}\mu_k$. Since station i works at rate $\hat{\mu}_i$, any station with smaller capacity than $\hat{\mu}_i$ can never be $N(i,t)$ for $t \in [s_i, T_i)$. ■

Using Theorem 5.1, we can reduce our problem to optimally choosing the vector $\hat{s}$ of delays (one component for each dynamic station). Corollary 5.1, coupled with the strong duality results known for SCLPs, provides a short certificate to verify both optimality and non-optimality of any feasible solution. Thus, we see that the optimal control problem of a fluid tandem network belongs to the complexity class $NP \cap co-NP$, a strong evidence that this problem belongs to P, the class of problems that have polynomial time algorithms. Motivated by these observations, we have tried to find a polynomial-time algorithm for this problem, and what follows is a summary of our findings.

5.4 Formulation

In this section we exploit the structural property proved in §5.3 and reformulate our problem as a piecewise convex quadratic optimization problem. To do this, it will be convenient to define a few additional quantities, which we do below.

We define a dynamic relation N among buffers by letting $N(i,t)$ denote the first non-empty buffer following buffer i at time t. From Theorem 5.1, we know that there is always

an optimal policy such that a class never has positive inventory after it empties for the first time; such a class does not contribute to the cost after it empties, and may be crossed out from the network for all purposes. The role of the dynamic relation N is precisely to keep track of the classes present in the network, over time. We also introduce a dynamic relation P which is the inverse of N; $N(i,t) = j$ is the same as saying $P(j,t) = i$. Just as N keeps track of the next non-empty class, P keeps track of the previous non-empty class. The values of $N(i,t)$ for $t = s_i$ and for $t = T_i$ will be used often in our analysis; for this reason we let $N^d(j) = N(j,T_j)$, and $N^s(j) = N(j,s_j)$. In case of ties, we choose the farthest station while determining $N^d(\cdot)$ and $N^s(\cdot)$, i.e., $N^d(j) = N(j,T_j^+)$ and $N^s(j) = N(j,s_j^+)$.

We now express the cost function in a convenient form. We first do this for the case of uncontrolled tandem networks (i.e., networks for which $s_i = 0$ for all i). This arises, for example. when all stations are static. Of course, in such a case, there is no optimization to be performed, but the development of the formula for the cost function in this simple setting will help us understand the more general case easily.

5.4.1 Uncontrolled tandem networks

We will use a graphical device that was first provided by Weiss [73] to describe the dynamics of re-entrant lines (see Figure 5-2). A tandem network is a special case of a re-entrant line, and so the same ideas can be used. In this figure, we plot the $n+1$ lines

$$\sum_{i=k}^{n} x_i + \int_0^t u_{k-1}(s)\mu_{k-1}ds, \ k = 1,2,\ldots n,n+1.$$

Recall that x_{n+1} is assumed to be zero and so the line corresponding to $k = n+1$ is exactly

$$\int_0^t u_n(s)\mu_n ds.$$

For convenience, we number the lines from top to bottom, starting from zero (see Figure 5-2. Thus, the line labeled OO' is line zero, and the line labeled f_4t_4 is line 3. The k^{th} line at time t ($k = 0,1,\ldots,n$) represents the sum of the total inflow into the subsystem consisting of the stations $k+1, k+2, \ldots, n$ and the initial amount of fluid present in that subsystem. The differences in heights between lines k and $k+1$ ($0 \leq k < n$) at time t represents the value of $x_{k+1}(t)$. Clearly, the lines are monotone non-decreasing ($u_i(t) \geq 0$ for

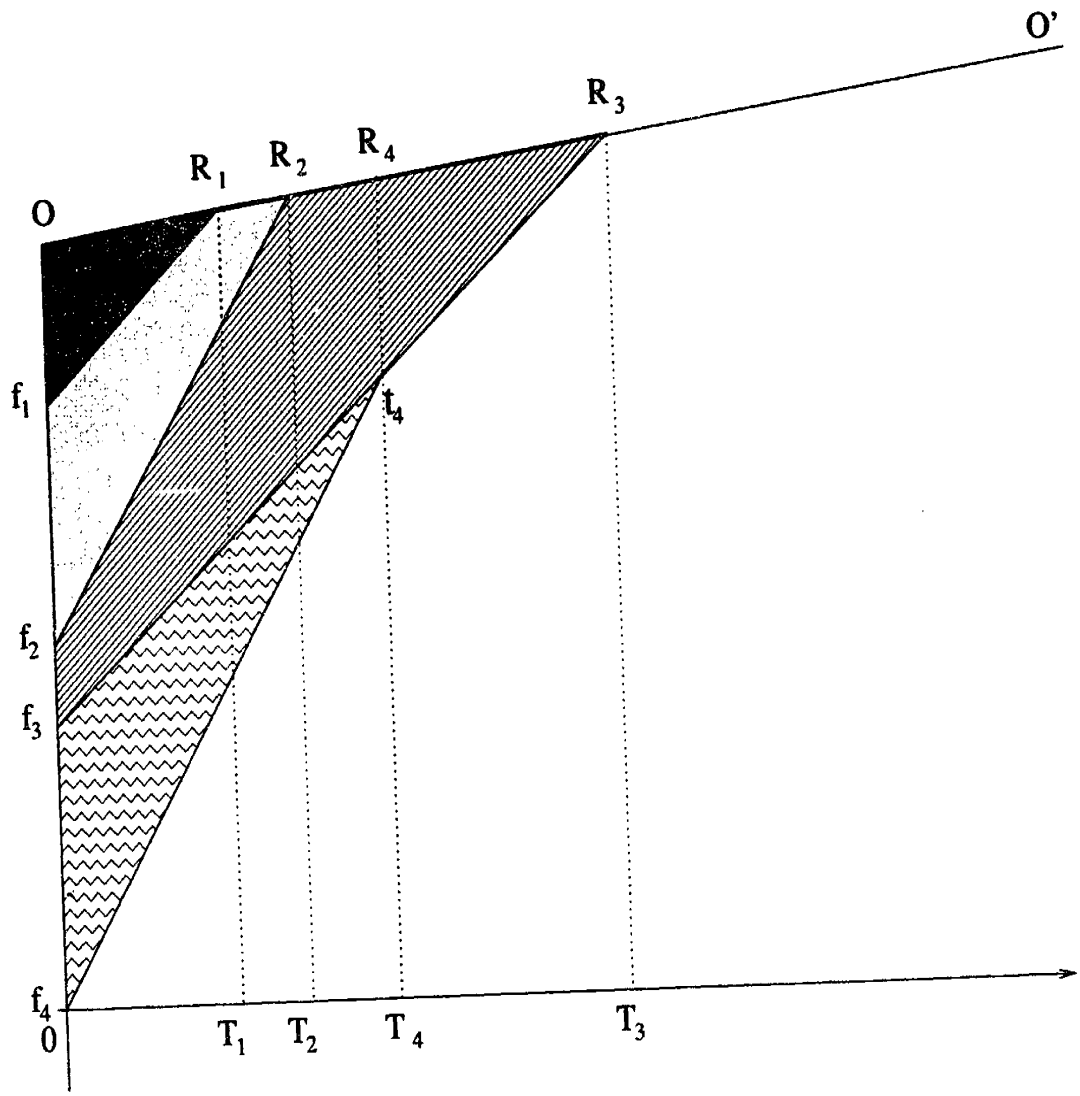

Figure 5-2: A four station uncontrolled tandem network

all i, $t \geq 0$) and do not cross ($x_i(t) \geq 0$ for all i, and $t \geq 0$). The point marked O represents the point $(0, \sum_{i=1}^{n} x_i)$; points on the line OO' contain points of the form $(t, \sum_{i=1}^{n} x_i + \lambda t)$. In this representation, the abscissa represents time, and the ordinate represents the total amount of fluid that has entered the system at time t. All of the results in this section can be directly interpreted and proved in terms of Figure 5-2.

For $i < j$, we say line j *meets* line i if lines i and j intersect. Essentially, this means that $P(j, T_j) = i$, i.e., just before its depletion, the immediate predecessor of station j with a positive buffer level is station i. For convenience, we also say $P^d(j) = i$ if line j meets line i. We break ties according to rules discussed earlier in this section. So, for any $1 \leq j < n$, there is a *unique* line i, $0 \leq i < j$, such that line j meets line i; in such a case we define the *workload* of class j, w_j, is:

$$w_j = \sum_{k:k=i+1}^{j} x_k.$$

For instance, for the network of Figure 5-2:

- line 1 meets line 0; so $w_1 = x_1$.
- line 2 meets line 0; so $w_2 = x_1 + x_2$.
- line 3 meets line 0; so $w_3 = x_1 + x_2 + x_3$.
- line 4 meets line 3; so $w_4 = x_4$.

We can think of w_i as initial amounts which would yield the same termination times, if the evolution of each class were to always proceed at its last speed before emptying. Let $A'_i = w_i T_i / 2,\ i = 1, 2, \ldots, n$.

Lemma 5.7

- *Suppose line j meets line i ($0 \leq i < j$) in an uncontrolled tandem network. Then, the aggregate inventory of class j, A_j, is:*

$$A_j = (A'_j - \sum_{k:k=i+1}^{j-1} A_k), \qquad \forall j \leq n. \tag{5.2}$$

- *The total cost may be written as*

$$V = \sum_{j=1}^{n} A'_j(c_j - c_{N^d(j)}). \tag{5.3}$$

Proof: Consider the subsystem, $(i, j]$, consisting of the stations $(i+1)$ through j. Since line j meets line i, station j is the last station with positive inventory in this subsystem: once station j depletes its inventory, the total inventory in the subsystem becomes zero, and, by Theorem 5.1, stays at zero thereafter. Thus, until its depletion, the subsystem $(i, j]$ has a constant inflow rate μ_i and a constant outflow rate μ_j; so, the aggregate inventory of the subsystem $(i, j]$ is A'_j. We get the aggregate inventory of class j by subtracting the contributions of the classes $(i+1)$ through $(j-1)$, which proves the first part of the lemma. The total cost, by definition, is:

$$V = \sum_{j=1}^{n} c_j A_j \tag{5.4}$$

$$= \sum_{j=1}^{n} c_j (A'_j - \sum_{k:k=P^d(j)+1}^{j-1} A_k). \tag{5.5}$$

To prove the second part of the lemma, we just need to show that

$$\sum_{j=1}^{n} \sum_{k:k=P^d(j)+1}^{j-1} c_j A_k = \sum_{j=1}^{n} c_{N^d(j)} A'_j.$$

Consider the term $\sum_{j=1}^{n} c_{N^d(j)} A'_j$. Here, A'_j is weighted by a factor c_k where k is the unique station with the property $N(j, T_j) = k$. Observe, however, that $\sum_{j:N^d(j)=k} A'_j = \sum_{l:l=P^d(k)+1}^{j-1} A_l$, which proves the lemma. ■

For instance, let us again consider the example of Figure 5-2. Here, A'_1 is the area of the triangle Of_1R_1, which is also A_1. For class 2, the total inventory cost is given by the area of the quadrilateral $f_1R_1R_2f_2$ (area = A_2), which is the difference in the areas of the triangles Of_2R_2 (area = A'_2) and Of_1R_1 (area = A_1). For class 3, the total inventory cost is given by the area of the quadrilateral $f_2R_2R_3f_3$, which is the difference in the areas of the triangles Of_3R_3 (area = A'_3) and Of_2R_2 (area = $A_1 + A_2$). Finally, for class 4, the total inventory cost is the area of the triangle $f_3f_4t_4$ (area = A_4), which is A'_4. Also, in Figure 5-2, $N^d(1) = 2$, $N^d(2) = 3$, $N^d(3) = N^d(4) = 5$; and the second part of Lemma 5.7 can be verified easily.

In the next section we study the effect of introducing "delays", and use it to provide a

similar decomposition for controlled tandem networks.

5.4.2 Controlled tandem networks

We consider now the general case of a tandem network with n classes out of which m are dynamic. Let D be the set of dynamic classes. By Corollary 5.2, station i works at a constant rate

$$\hat{\mu}_i = \min_{k \in \{i, N^s(i)\}} \mu_k$$

until depletion; in the rest of this section we will work with $\hat{\mu}_i$ instead of μ_i.

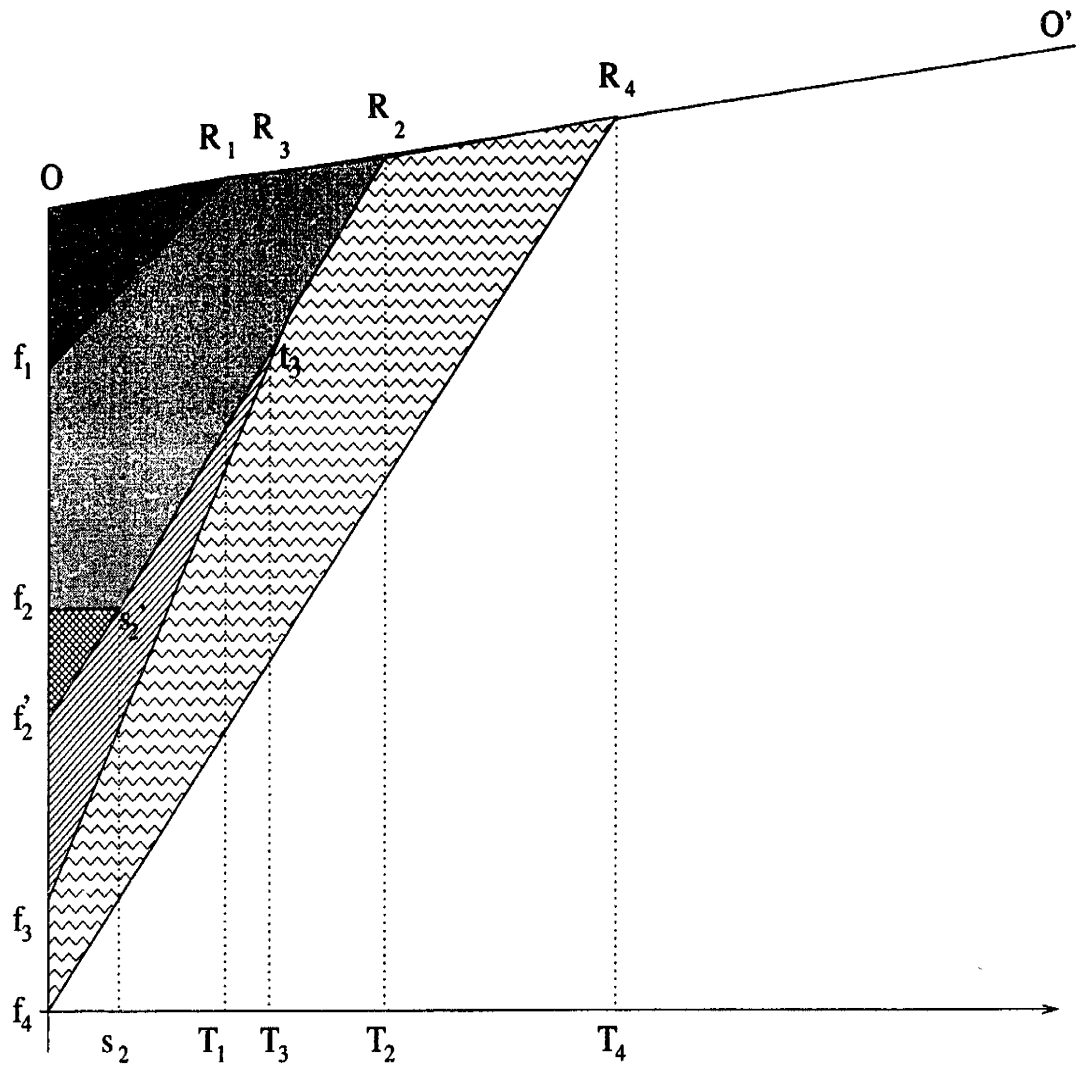

Figure 5-3: A four station controlled tandem network

We use a similar graphical device for the controlled tandem networks as well (see Figure 5-3). The only difference is that if station i is dynamic, then it waits until time s_i before

processing material. Again, for any $1 \leq j < n$, there is a *unique* line i, $0 \leq i < j$, such that line j meets line i; in such a case we define the *workload* of class j, w_j, is:

$$w_j = \sum_{k:k=i+1}^{j} x_k + s_j\hat{\mu}_j - 1(s_i \leq T_j)s_i\hat{\mu}_i. \tag{5.6}$$

In equation 5.6, if j is a static station, s_j is assumed to be zero. As before, we express the cost function in a convenient form, in terms of variables A'_i, $i = 1, 2, \ldots, n$, and B'_i, $i \in D$. The A'_i are defined as before: $A'_i = w_i T_i/2$, and $B'_i = \hat{\mu}_i s_i^2/2$, $i \in D$.

Lemma 5.8

- *Suppose line j meets line i $(0 \leq i < j)$ in a controlled tandem network. Then, the aggregate inventory of class j, A_j, is:*

$$A_j = A'_j - \sum_{k:k=i+1}^{j-1} A_k - B'_j + 1(s_i \leq T_j)B'_i \qquad \forall j \leq n. \tag{5.7}$$

- *The total cost may be written as*

$$V = \sum_{i=1}^{n} A'_i(c_i - c_{N^d(i)}) + \sum_{i \in D} B'_i(c_{N^s(i)} - c_i). \tag{5.8}$$

Proof: The proof is similar to the proof of Lemma 5.7, and is hence omitted. ■

For instance, let us consider the example of Figure 5-3. Here, A'_1 is the area of the triangle Of_1R_1, which is also A_1. For class 2, the aggregate inventory is given by the area of the polygon $f_1R_1R_2s'_2f_2$ (area $= A_2$), which is the difference between the area of the triangle Of'_2R_2 (area $= A'_2$) and the areas of the two triangles Of_1R_1 (area $= A_1$) and $f_2s'_2f'_2$ (area B'_2). For class 3, the aggregate inventory is given by the area of the polygon $f_2s'_2t_3f_3$, which is the sum of the areas of the triangles $f'_2t_3f_3$ (area $= A'_3$) and $f_2s'_2f'_2$ (area B'_2). Finally, for class 4, the aggregate inventory is the area of the polygon $f_3t_3R_2R_4f_4$ (area $= A_4$), which is the difference in areas between the triangle Of_4R_4 (area $= A'_4$) and the aggregate inventory of all of the other classes $(A_1 + A_2 + A_3)$. Also, in Figure 5-3, $N^d(1) = 2$, $N^d(2) = N^d(3) = 4$, $N^d(4) = 5$, $N^s(2) = 3$; and the second part of Lemma 5.8 can be verified easily.

We now prove the following lemma, which guarantees the existence of an optimal delay vector satisfying some nice properties.

Lemma 5.9

(a) Consider a two station tandem network, with $c_1 \leq c_2$. *The optimal value of* s_1 *(the only decision variable in this problem) is such that* $s_1 \leq T_2 < T_1$.

(b) Suppose we have stations i *and* j *with* $j > i$, $x_i(t) > 0$, *and* $N(i,t) = j$, *at some* t *in an optimal solution to problem* (TQ). *Then,*

$$c_i < c_j,\ c_i > \max\{c_{j+1}, c_{j+2}, \ldots, c_n\} \Longrightarrow s_i \leq T_j.$$

(c) Suppose we have stations i *and* j *with* $j > i$, $x_i(t) > 0$, *and* $N(i,t) = j$, *at some* t *in an optimal solution to problem* (TQ). *Then,*

$$c_i < c_j \Longrightarrow T_i > T_j.$$

(d) There is always an optimal solution to problem (TQ) *such that if* i *is a static station*

$$c_{N^d(i)} \leq c_i.$$

Proof: Part (a) follows from the explicit formula known for s_1 (see, for example, [5]). Part (b) is immediate from Lemma 5.2. To prove part (c), we shift the origin to time t, omit stations with empty buffer levels, and set the initial buffer levels at the remaining stations to be the corresponding values in the optimal solution at time t. By Bellman's principle of optimality, any portion of an optimal trajectory is optimal as well: If all of the stations after j get deleted in this process, part (c) is immediate from part (a). The presence of stations beyond j can only delay the start time of station i, and so can only increase T_i. We prove part (d) by showing that for a static station i, $c_i \geq c_{N(i,t)}$ for $0 \leq t \leq T_i$. Certainly, this inequality is true at time zero, and will remain true until the station $N(i,0)$ depletes. For this inequality to become false, there should be stations j, k such that $i < j < k$, $N(i, T_j^-) = j$, $N(i, T_j^+) = k$, $c_i > c_j$, and $c_i < c_k$. However, such a station has to deplete before station j: this is because $N(j, T_j^-) = k$, and $c_j < c_k$, which by part (c) implies $T_j > T_k$, a contradiction. ■

Any vector of delays s determines $N^d(i)$ and $N^s(i)$ for all i. From Lemma 5.9, we conclude that for an optimal vector of delays $\hat{s}$, $c_{N^d(i)} \leq c_i \leq c_{N^s(i)}$. Thus, we can restrict

our search for an optimal delay vector to the following set $S = \{(s_i)_{i \in D} \mid c_{N^d(i)} \leq c_i \leq c_{N^s(i)}\}$ of D dimensional vectors. We emphasize that $c_{N^d(i)}$ and $c_{N^s(i)}$ are functions of a delay vector; a delay vector s belongs to S if and only if the $N^d(\cdot)$ and $N^s(\cdot)$ that it determines satisfies $c_{N^d(i)} \leq c_i \leq c_{N^s(i)}$ for all dynamic stations i. By part (d) of Lemma 5.9, we see that $c_{N^d(i)} \leq c_i$ if i is a static station; In fact, the proof shows that this is a consequence of imposing the condition $c_{N^d(i)} \leq c_i \leq c_{N^s(i)}$ on all dynamic stations i. If we restrict our search to the set S of delay vectors, we are guaranteed to find an optimal solution. Furthermore, for any element of S, the objective function is a sum of squares with non-negative coefficients, and hence is a convex function. This proves that our objective function is piecewise convex quadratic.

To better understand and exploit the characterization obtained, we consider the following question: Given an arbitrary tandem network, and a dynamic station i, what are the *potential candidates* for $N^s(i)$? A station k is a potential candidate for $N^s(i)$ if, for some instance of the problem, $N^s(i) = k$ for *every* optimal delay vector. In other words, there is an instance of the problem for which *the* optimal $N^s(i)$ is k. Clearly, we can restrict our attention to potential candidates of $N^s(i)$, if only we can find them. We now provide a succinct characterization for a station k to be a potential candidate for $N^s(i)$.

Lemma 5.10

For a dynamic station i, k is a potential candidate for $N^s(i)$ if and only if (i) $c_k \geq c_i$, and (ii) $c_k \leq c_j$ for $i < j < k$.

Proof: If k satisfies the conditions of the lemma, we can easily construct an instance of the problem where every optimal delay vector has $N^s(i) = k$. This is done by making $x_i, x_{i+1}, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n$ extremely small, and by making x_k large. Since all of the stations in (i, k) have costs higher than c_k, by making x_k large enough, we can ensure that every station in (i, k) and $(k, n]$ empties before station k empties. Also, since $c_i < c_k$ and since station k has a huge inventory, by making x_i small enough, we can ensure that it is optimal for station i to start only when station k brings down its inventory below a certain threshold. Thus, we can find an instance of the problem for which $N^s(i) = k$ in every optimal solution.

We now argue that none of the other stations can be potential candidates. Suppose there is a station k which violates the conditions of the lemma, and yet is a candidate for

$N^s(i)$. Clearly, $c_k \geq c_i$ because we know that there is always an optimal solution satisfying $c_{N^s(i)} \geq c_i$. So the only way the conditions of Lemma 5.10 can be violated for a potential candidate is that $c_k > c_j$ for some station $j \in (i,k)$. If there are many such stations in (i,k), we pick the station closest to k, i.e., we pick a station j such that $c_j < c_k$ and every r in (j,k) is such that $c_r \geq c_k$. Repeated application of part (c) of Lemma 5.9 shows that $T_j > T_k$, contradicting the fact that k is a potential candidate for $N^s(i)$. ■

Let PC(i) be the set of all potential candidates for $N^s(i)$. Lemma 5.10 provides an alternative characterization of the natural domain S, which is summarized as Lemma 5.11.

Lemma 5.11

- *The natural domain S can also be characterized as:*

$$S = \{(s_i)_{i \in D} \mid 0 \leq s_i \leq \max_{j \in \mathrm{PC}(i)} T_j \leq T_i\}.$$

Proof: Immediate from the definitions. ■

To summarize, we have represented the objective function as a sum of squares with positive coefficients over a natural domain. A natural approach would be to show that S is a convex set, and that the objective function is globally convex over S (i.e., the first derivatives of the function match at the "boundaries" between the pieces). If we do, our problem is an instance of a convex optimization problem, for which efficient algorithms are known. Unfortunately, the following example shows that S is not convex.

Example 1: Consider the following four station tandem network. The holding costs rates are given by $c_1 = 10$, $c_2 = 50$, $c_3 = 99$, and $c_4 = 49$, with the initial inventory levels being $x_1 = 150$, $x_2 = 5$, $x_3 = 15$ and $x_4 = 22$. Let $\lambda = 0.5$, $\mu_1 = 1$, $\mu_2 = 2$, $\mu_3 = 3$ and $\mu_4 = 4$. Stations 1 and 2 are dynamic and so have associated decision variables s_1 and s_2; also, PC(1) $= \{2,4\}$ and PC(2) $= \{3\}$. The natural domain is characterized by two conditions: (i) $s_1 \leq \max(T_2, T_4) \leq T_1$ and (ii) $s_2 \leq T_3 \leq T_2$. A plot of the natural domain is shown in Figure 5-4, which is non-convex.

Therefore, in the next section, we focus our attention on the special case of tandem networks with non-decreasing costs, and show that this characterization leads to simple, efficient algorithms. We revisit the general case in §5.6 and make some observations, which might prove useful in extending our approach.

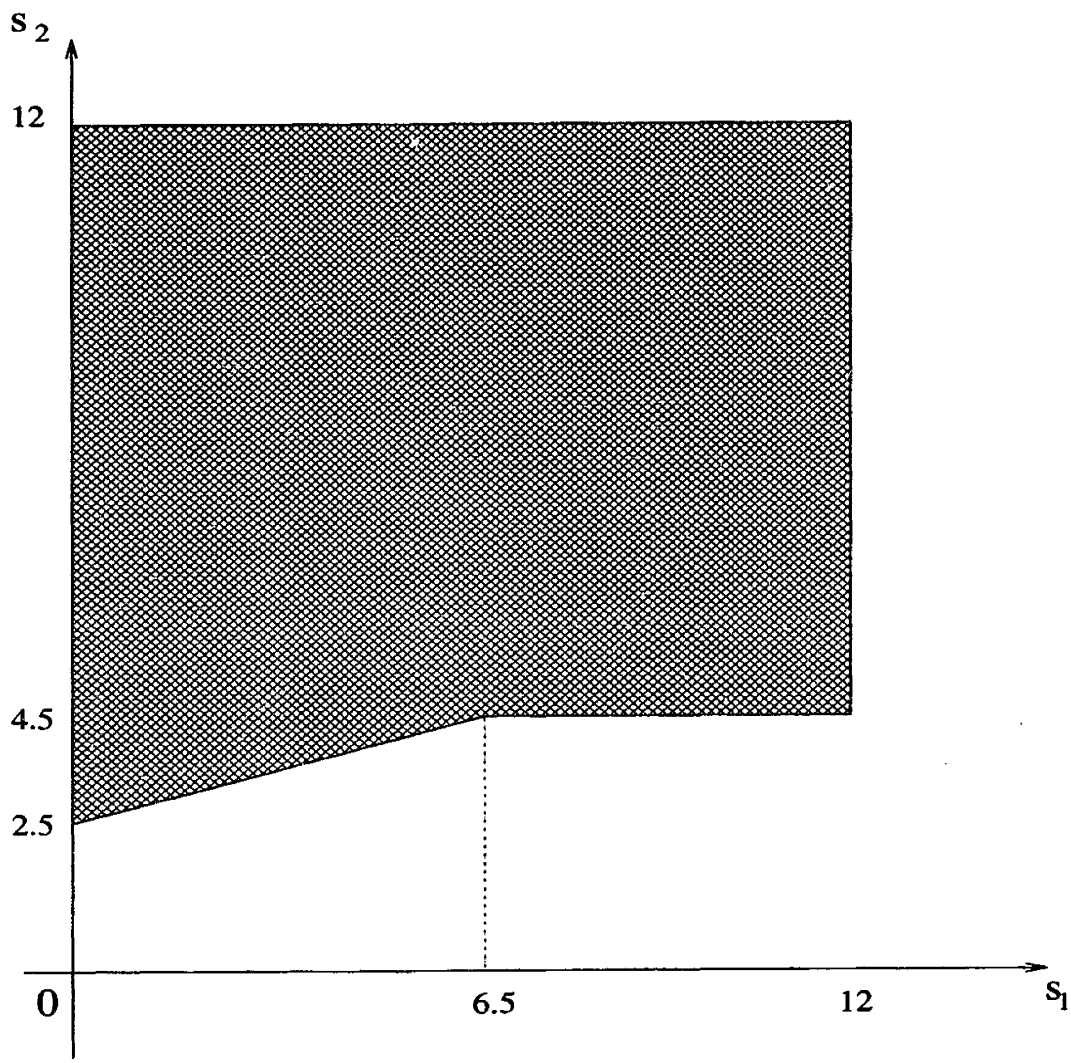

Figure 5-4: A plot of S for Example 1

5.5 Tandem networks with non-decreasing costs

In this section, we consider the special case of a tandem network in which the costs are non-decreasing: $c_1 \leq c_2 \leq \ldots \leq c_n$.

To make the essential ideas clear, we first assume that both the *holding cost rates* and *capacities* are *strictly increasing.* i.e., $c_1 < c_2 < \ldots < c_n$, and $\mu_1 < \mu_2 < \ldots < \mu_n$; towards the end of this section we will discuss how to get rid of these assumptions.

5.5.1 Strictly increasing costs and capacities

Since the costs are strictly increasing, only class n is static. Using the characterization of potential candidates obtained in Lemma 5.10, we find that for $i \in (0, n)$, $\mathrm{PC}(i) = \{i+1\}$, and therefore $N^s(i) = i+1$. Moreover, since $c_{N^d(i)} \leq c_i$, the only possibility is that $N^d(i) = n+1$ for any i. Thus, station i starts work on or before the time at which station $(i+1)$ depletes, and depletes before station $(i-1)$ depletes. These observations lead to the following theorem.

Theorem 5.2

(a) For any $s \in S$ the depletion times are given by:

$$T_i = \frac{x_i + s_i\mu_i - s_{i-1}\mu_{i-1}}{\mu_i - \mu_{i-1}}, \tag{5.9}$$

where $s_0 = s_n = 0$.

(b) The domain S is convex.

(c) The total cost may be also represented as

$$V = \sum_{i=1}^{n} c_i \frac{x_i^2}{2(\mu_i - \mu_{i-1})} + \sum_{i=1}^{n} \frac{h_i(s_{i-1} - s_i)^2}{2} + \sum_{i=1}^{n} \frac{c_i x_i(s_i\mu_i - s_{i-1}\mu_{i-1})}{\mu_i - \mu_{i-1}}, \tag{5.10}$$

where

$$h_i = c_i \frac{\mu_i\mu_{i-1}}{\mu_i - \mu_{i-1}}.$$

Proof: We know that $s_{i-1} \leq T_i$ and since $T_{i-1} > T_i > T_j$ for any $j > i$. Let us focus our attention on stations $i-1$ and i. There are two possibilities: either $s_{i-1} \leq s_i$ or $s_{i-1} > s_i$. In either case, station i receives no input in the interval $[0, s_{i-1})$, and receives input at rate μ_{i-1} thereafter until depletion. Also, the outflow from station i is zero in the interval $[0, s_i)$ and at rate μ_i thereafter until depletion. First, let us suppose that $s_{i-1} \leq s_i$; in this case, T_i can be computed as follows:

$$\begin{aligned} T_i &= s_i + \frac{x_i + (s_i - s_{i-1})\mu_{i-1}}{(\mu_i - \mu_{i-1})} \\ &= \frac{x_i + s_i\mu_i - s_{i-1}\mu_{i-1}}{\mu_i - \mu_{i-1}}. \end{aligned}$$

Now, we suppose that $s_{i-1} > s_i$; in this case,

$$\begin{aligned} T_i &= s_{i-1} + \frac{x_i - (s_{i-1} - s_i)\mu_i}{(\mu_i - \mu_{i-1})} \\ &= \frac{x_i + s_i\mu_i - s_{i-1}\mu_{i-1}}{\mu_i - \mu_{i-1}}. \end{aligned}$$

The expression for T_i in both cases is identical, proving part (a). We note that part (a) can be directly proved by interpreting the workload variables defined in §5.4.2: the workload of class i was defined as the initial amount of material that would yield the same termination time, if the evolution of class i proceeded exactly at the same rate just before depletion. In this case, line i meets line $(i-1)$, and $s_{i-1} \leq T_i$; the inventory of class i decreases at rate $(\mu_i - \mu_{i-1})$ at time T_i, and so $T_i = \frac{w_i}{(\mu_i - \mu_{i-1})}$. From equation (5.6), $w_i = x_i + s_i\mu_i - s_{i-1}\mu_{i-1}$, which proves part (a) as well. For part (b), we observe that

$$S = \{(s_i)_{i \in D} \mid 0 \leq s_i \leq T_{i+1} \leq T_i\}.$$

Since the T_i are linear in the s_i, S is just an intersection of linear inequalities, which is convex. Note that $c_{N^d(i)} = 0$ and $c_{N^s(i)} = c_{i+1}$ for all i. Using the expressions for T_i from part (a) and w_i from equation (5.6), and simplifying equation (5.8) proves part (c). ■

Theorem 5.2 shows that the optimal delay vector can be obtained by solving a convex quadratic programming problem. One potential solution is obtained by setting the partial derivatives of V with respect to s_i in equation (5.10) to zero; this defines a system of linear equations in the s_i that have a unique solution: If the s vector thus obtained belongs to

S (i.e., satisfies the constraints), we have obtained an optimal solution. We discuss this computation next.

Explicit solution: For $1 \leq i \leq n$, we set

$$g_i = \frac{c_{i+1}x_{i+1}}{\mu_{i+1} - \mu_i} - \frac{c_i x_i}{\mu_i - \mu_{i-1}}.$$

As always, we set s_0 and s_n to be zero. The partial derivative of V with respect to s_i is

$$\frac{\partial V}{\partial s_i} = h_i(s_i - s_{i-1}) - h_{i+1}(s_{i+1} - s_i) - \mu_i g_i, \quad i = 1, 2, \ldots, n-1. \tag{5.11}$$

The system of $n-1$ equations defined by equations (5.11) is a tri-diagonal system that has a unique solution, given by:

$$s_i = F_i + H_i(\sum_{k=i+1}^{n-1} \mu_k g_k - \frac{F_{n-1}}{H_n}), \tag{5.12}$$

where $H_i = \sum_{k=1}^{i} h_k^{-1}$, and $F_i = \sum_{k=1}^{i} \mu_k H_k g_k$.

If the s_i obtained from equation (5.12) are non-negative, we can show that they satisfy the system of equations $s_i < T_{i+1} < T_i$, $(1 \leq i \leq n-1)$ as well, and in that case we have explicit expressions for optimal delay. We now use this to design a dynamic programming algorithm (DP) to obtain an optimal delay vector.

DP algorithm: We show that we can find an optimal delay vector by solving a shortest path problem on a graph G, which we define now. The graph G is a complete graph on $n+1$ nodes $\{0, 1, 2, \ldots, n\}$; associated with edge (p, q) $(p < q)$ is a distance d_{pq} computed as follows: We pretend that only stations p through q exist, $s_p = s_q = 0$, and solve for $s_{p+1}, s_{p+2}, \ldots s_{q-1}$ explicitly using the applicable portion of equations (5.12). If the s_k, $(p < k < q)$ so obtained are positive and satisfy the constraints $s_k < T_{k+1} < T_k$ for all $(p \leq k < q)$, then we define d_{pq} to be the associated cost of clearing stations $p+1$ through q. If not, we define d_{pq} to be infinity.

Theorem 5.3 *A shortest path between the nodes 0 and n in the graph G yields an optimal solution.*

Proof: In any optimal solution to the fluid tandem network with increasing costs and capacities, a (possibly empty) subset of the s_i, $1 \leq i < n$, are zero. Let $s_{a_1}, s_{a_2}, \ldots s_{a_k}$

be the subset of s_i that are zero, and let the cost of the optimal solution be V^*. We assume without loss of generality that $s_{a_1} = s_{a_2} = \ldots = s_{a_k} = 0$, with $0 < a_1 < a_2 < \ldots < a_k < n$; also, we assume that all of the other s_i are strictly positive. By the optimality of the solution, $T_{a_1} \geq T_{a_2} \geq \ldots \geq T_{a_k}$. and so the path in G given by $(0, a_1) - (a_1, a_2) - \cdots - (a_{k-1}, a_k) - (a_k, n)$ has cost V^*. Similarly, we can read off an optimal solution from a shortest path in G: if (a_i, a_j) is an edge in the path, then $s_{a_i} = s_{a_j} = 0$ and the s_k for $a_i < k < a_j$ are obtained using equations (5.12). ■

Remark: The mapping of solutions to the fluid tandem network to paths in G may not preserve the cost of non-optimal feasible solutions; it certainly preserves the cost of optimal solutions, which suffices for our purposes.

5.5.2 Weakly increasing costs & arbitrary capacities

The results of §5.5.1 were obtained under the assumption that both costs and capacities were strictly increasing. We now discuss how this assumption can be relaxed.

Consider stations i and $i+1$. We know that $c_i \leq c_{i+1}$, and we do not know anything about μ_i and μ_{i+1}. There are four possibilities: (i) $c_i < c_{i+1}$, $\mu_i < \mu_{i+1}$, (ii) $c_i < c_{i+1}$, $\mu_i \geq \mu_{i+1}$, (iii) $c_i = c_{i+1}$, $\mu_i < \mu_{i+1}$, and (iv) $c_i = c_{i+1}$, $\mu_i \geq \mu_{i+1}$.

In §5.5.1 we dealt with the case when all of the stations satisfy case (i). We now consider the possibility that some stations may fall into cases (ii)-(iv).

Case (ii): Suppose there is a station i such that $c_i < c_{i+1}$ and $\mu_i \geq \mu_{i+1}$. In this case the problem decomposes: we can set $s_i = T_{i+1}$, and operate station i at rate $\hat{\mu}_i$ instead of μ_i (see Corollary 5.2). In particular, station i will never work at rate faster than μ_{i+1}.

Case (iii): Suppose there is a station i such that $c_i = c_{i+1}$ and $\mu_i < \mu_{i+1}$. We set $s_i = 0$ as that is clearly optimal (the proof is immediate).

Case (iv): Suppose there is a station i such that $c_i = c_{i+1}$ and $\mu_i \geq \mu_{i+1}$. In this case, we move the initial inventory of station i to station $(i+1)$ (i.e. set $x_{i+1}(0) := x_{i+1}(0) + x_i(0)$), and eliminate station i from the problem. This is because station $(i+1)$ cannot process material faster than rate μ_{i+1}, and stations i and $(i+1)$ have the same cost.

These changes can be incorporated into our convex programming problem or into the DP algorithm easily.

5.6 Convex Sub-domains

We now discuss some ideas to extend our results to the case of mixed costs. We know (see Figure 5-4) that the natural domain S is not always a convex set. We now show that by fixing $N^s(i)$ for all the dynamic stations i, we get a sub-domain that is convex. (Notice that we do not do not restrict the value of $N^d(\cdot)$. In particular, there could be many sequences of events consistent with a given $N^d(\cdot)$, and our optimization problem is that of finding the best possible delay vector consistent with the given $N^s(\cdot)$.)

Recall that the natural domain S was defined as

$$S = \{(s_i)_{i \in D} \mid 0 \leq s_i \leq \max_{j \in \mathbf{PC}(i)} T_j \leq T_i\}.$$

The difficult portion of the constraints defining S is

$$s_i \leq \max_{j \in \mathbf{PC}(i)}.$$

Fixing $N^s(i)$ for every dynamic station i simplifies the natural domain S as follows. Suppose $N^s(i) = j$. By definition, every station in (i, j) depletes *before* j does, which depletes *before* station i. Also, station j works until a constant rate $\hat{\mu}_j$ until depletion, which is known explicitly if we know $N^s(j)$. We can thus determine T_j explicitly as a linear function of s_i and s_j. Thus, all of the relevant T_i can be determined as a function of the s vector, which results in a description of S by linear inequalities.

Since the sub-domain determined by fixing $N^s(\cdot)$ is convex, we might be able to solve the corresponding optimization problem efficiently; however, we still have not proved that our objective function is convex over this sub-domain. All we know is that we have a piecewise convex quadratic objective function. A simple computation of derivatives at the boundaries of the pieces (i.e. values of s at which $N^d(\cdot)$ changes) shows that our objective function is convex over this sub-domain, and hence we can solve the associated optimization problem efficiently.

To show that this results in a polynomial-time algorithm, we need to show that there are only polynomially many ways to fix $N^s(\cdot)$. Apriori, the bound on the number of convex programs we need to solve is

$$\Pi_{i \in D} |PC(i)|,$$

which could be exponential in the number of stations. The following observation might prove useful in reducing the possibilities: *For any two dynamic stations i and j, with $i < j$, no station in* $\mathrm{PC}(i)$ *occurs between two stations in* $\mathrm{PC}(j)$.

In fact, we may even be able to "decompose" the problem in an intelligent way, and bound the number of convex programs needed by

$$\sum_{i \in D} |\mathrm{PC}(i)|.$$

5.7 Conclusions

In this chapter we studied the complexity of finding an optimal solution to the fluid tandem network. We discovered a simple algorithm to solve the special case in which the costs are non-decreasing along the route. In addition we identified some potentially useful properties that may lead to a resolution of the general case.

A major open problem is the question of resolving the complexity of general fluid networks. It is not even known if optimal solutions exist for general fluid networks with polynomially many pieces; experiments with a variety of small networks using the algorithm of Luo and Bertsimas [52] suggest that this is indeed the case. An affirmative answer to this question would make the problem of discovering polynomial-time algorithms for general fluid networks interesting and perhaps more tractable.

Chapter 6

Concluding remarks

We conclude with a brief summary of the results presented in this dissertation, followed by some directions for future research.

6.1 Summary of results

The work presented in this dissertation was motivated by the question of designing good scheduling policies for optimizing multiclass queueing networks. Dynamic control problems for multiclass queueing networks are known to be intractable; hence an approach that has emerged during the last twenty years is based on studying tractable relaxations. (This is very much in the spirit of research in the discrete optimization community, where relaxations have a long and distinguished history in coping with $\mathcal{NP}$-hard optimization problems.) Fluid relaxations, a class of approximate models studied in connection with queueing networks, have been the focus of much attention during the last decade, primarily due to its success in resolving global stability questions. Motivated by this success, we explored the role of fluid relaxations in addressing the natural problem of optimizing performance of queueing networks. A summary of our results follows:

1. **Minimizing makespan in job shops:** We discussed an efficient algorithm to round an optimal fluid solution such that the resulting schedule is asymptotically optimal. In contrast to several methods proposed for this problem, our algorithm is simple and practical. We demonstrated the efficacy of our algorithm on a wide variety of benchmark instances from the OR library. We discussed extensions of our results to

the model in which deterministic arrivals occur over a finite horizon.

2. **Minimizing holding costs in job shops:** We presented an efficient algorithm to round an optimal fluid solution such that the resulting schedule is asymptotically optimal. It is well known that this version of the job shop problem does not admit polynomial time approximation schemes. Experiments on benchmarks from the OR library show that our algorithm performs quite well. We discussed extensions of these results to a (suitably restricted) model in which deterministic arrivals occur over a finite horizon. These results can be viewed as a first step toward solving similar problems in a stochastic, dynamic setting (multiclass queueing network model).

3. **Optimal control of multiclass queueing networks:** We studied the effectiveness of the *achievable region* approach in addressing a variety of problems arising in performance evaluation and optimization of multiclass queueing networks. We performed an extensive computational investigation of a class of lower bounds, developed by Bertsimas and Niño-Mora [12], for a variety of models including open networks, closed networks, models with routing, etc. For all of these problems we derived policies based on a heuristic interpretation of optimal solutions to the associated fluid relaxation. Our results show that the combination of fluid optimal control methods to generate near optimal policies for large scale problems and linear/semidefinite relaxations to provide near optimal bounds is a promising methodology to address the multiclass queueing network optimization problem.

4. **Optimal control of fluid tandem networks:** We presented a polynomial time algorithm for the problem of optimal control of fluid tandem networks in which the holding costs are nondecreasing along the route. Our approach is *algebraic* and does not use time-discretization explicitly: We showed that an optimal control for a fluid tandem network with non-decreasing holding costs can be computed by solving a *single* convex programming problem. Furthermore, we showed that this convex program can be solved even more efficiently as a shortest-path problem on a suitably defined graph.

6.2 Directions for future research

We now briefly summarize problems and directions for future research.

- **Complexity of solving fluid networks:** A major open problem is to resolve the complexity of computing optimal solutions to general fluid networks. It is not even known if optimal solutions exist for general fluid networks with polynomially many pieces. Our experiments on a variety of small networks suggest that this is indeed the case; a proof of this fact would require identification of strong structural properties of an optimal fluid solution. (For e.g., the "queue-length" of a job class goes from positive to zero at most k times, for some small k.) At the time of writing, we know of only two classes of non-trivial networks for which a proof of the existence of a polynomial-sized solution exists: two-station re-entrant lines with unit weights, and tandem networks. Identifying other special network topologies with this property seems to be a natural intermediate step.

- **Optimal control of multiclass queueing networks:** In chapter 4 we discussed several examples in which an appropriate interpretation of an optimal fluid solution resulted in a good policy for the underlying stochastic network. An important open problem is a mechanization of this procedure. Building on the work of Maglaras [53], we have shown that the most natural translation of an optimal fluid control could result in an *unstable* policy for the stochastic network [54]. A natural approach to this problem could involve a discrete-review policy [38] that uses the *fluid synchronization algorithm* as a subroutine. It would be interesting to compare such an approach with the results of Maglaras [53].

- **Asymptotically optimal solutions for job shops:** Apart from the obvious questions involving improving our bounds, there are many problems that could serve as a touchstone for the usefulness of fluid-based approaches. First, it is not clear how natural generalizations such as release dates, precedence delays, set-up costs, set-up times etc. can be incorporated into this framework. Second, it would be interesting to address more complicated scheduling problems using this approach: for example, project scheduling problems, and scheduling problems involving simultaneous possession of processors, multiple resources, etc.

Several other problems for study were identified in the individual chapters. We hope that the methods proposed in this dissertation will play an important role in solving some of these problems.

Bibliography

[1] P. Aféche. Personal communication, July 1999.

[2] E. J. Anderson. *A continuous model for job-shop scheduling.* PhD thesis, University of Cambridge, 1978.

[3] E. J. Anderson and P. Nash. *Linear Programming in Infinite-Dimensional Spaces.* John Wiley & Sons, New York, 1987.

[4] D. Atkins and H. Chen. Performance evaluation of scheduling control of queueing networks: fluid model heuristics. *Queueing Systems and Applications*, 21:391–413, 1995.

[5] F. Avram, D. Bertsimas, and M. Ricard. Fluid models of sequencing problems in open queueing networks: an optimal control approach. In F. P. Kelly and R. J. Williams, editors, *Stochastic Networks*, volume 71 of *Proceedings of the International Mathematics Association*, pages 199–234. Springer-Verlag, New York, 1995.

[6] I. S. Belov and Ya. N. Stolin. An algorithm in a single path operations scheduling problem. *Mathematical Economics and Functional Analysis*, pages 248–257, 1974. (in Russian).

[7] D. Bertsimas. The achievable region method in the optimal control of queueing systems: formulations, bounds and policies. *Queueing Systems and Applications*, 21:337–389, 1995.

[8] D. Bertsimas and D. Gamarnik. Asymptotically optimal algorithms for job shop scheduling and packet routing. Technical Report RC 21094, IBM Research, January 1998.

[9] D. Bertsimas and G. Mourtzinou. A unified method to analyze overtake free queueing systems. *Advances in Applied Probability*, 28(2):588–625, 1996.

[10] D. Bertsimas and J. Ni no Mora. Conservation laws, extended polymatroids and multi-armed bandit problems; a polyhedral approach to indexable systems. *Mathematics of Operations Research*, 21:257–306, 1996.

[11] D. Bertsimas and J. Ni no Mora. Optimization of multiclass queueing networks with changeover times via the achievable region approach: Part i, the single-station case. *Mathematics of Operations Research*, 24(2):306–330, 1999.

[12] D. Bertsimas and J. Ni no Mora. Optimization of multiclass queueing networks with changeover times via the achievable region approach: Part ii, the multi-station case. *Mathematics of Operations Research*, 24(2):331–361, 1999.

[13] D. Bertsimas and J. Ni no Mora. Restless bandits, linear programming relaxations and a primal-dual heuristic. *Operations Research*, to appear.

[14] D. Bertsimas, I. Ch. Paschalidis, and J. Tsitsiklis. Optimization of multiclass queueing networks: Polyhedral and nonlinear characterizations of achievable performance. *Annals of Applied Probability*, 4:43–75, 1994.

[15] D. Bertsimas, I. Ch. Paschalidis, and J. N. Tsitsiklis. Branching bandits and klimov's problem: Achievable region and side constraints. *IEEE Transactions on Automatic Control*, 40(12):2063–2075, dec 1995.

[16] D. Bertsimas and H. Xu. Optimization of polling systems and dynamic vehicle routing problems on networks. Technical report, Operations Research Center, MIT, 1993.

[17] J. A. Buzacott and J. G. Shanthikumar. *Stochastic Models of Manufacturing Systems.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[18] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35:164–176, 1985.

[19] H. Chen. Fluid approximations and stability of multiclass queueing networks: work-conserving policies. *Annals of Applied Probability*, 5:637–655, 1995.

[20] H. Chen and A. Mandelbaum. Discrete flow networks: Bottleneck analysis and fluid approximations. *Mathematics of Operations Research*, 16(2):408–446, 1991.

[21] H. Chen and D. Yao. Dynamic scheduling of a multiclass fluid network. *Operations Research*, 41(6):1104–1115, 1993.

[22] J. G. Dai. On positive harris recurrence of multiclass queueing networks: A unified approach via fluid limit models. *Annals of Applied Probability*, 5:49–77, 1995.

[23] J. G. Dai and S. Meyn. Stability and convergence of moments for multiclass queueing networks via fluid limit models. *IEEE Transactions on Automatic Control*, 40(11):1889–1904, November 1995.

[24] J. G. Dai and G. Weiss. Stability and instability of fluid models for certain re-entrant lines. *Mathematics of Operations Research*, 21:115–134, 1996.

[25] J. G. Dai and G. Weiss. A fluid heuristic for minimizing makespan in job-shops. Technical report, School of Industrial and Systems Engineering, Georgia Institute of Technology, April 1999.

[26] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Internetworking Research and Experience*, 1, 1990.

[27] D. Eng, J. Humphrey, and S. P. Meyn. Fluid network models: Linear programs for control and performance bounds. In 13^{th} *World Congress of International Federation of Automatic Control*. San Francisco, 1996.

[28] A. Federgruen and H. Groenevelt. Characterization and optimization of achievable performance in general queueing systems. *Operations Research*, 36(5):733–741, 1988.

[29] T. Fiala. Kozelíto algorithmus a három gép problémára. *Alkalmazott Matematikai Lapok*, 3:389–398, 1977.

[30] G. J. Foschini and J. Salz. A basic dynamic routing problem and diffusion. *IEEE Transactions on Communications*, 26:320–327, 1978.

[31] K. Fujisawa, M. Kojima, and K. Nakata. *SDPA (Semidefinite Programming Algorithm) User's Manual, Version 4.10.* Tokyo Institute of Technology, Tokyo, Japan, 1998. Research Report on Mathematical and Computing Sciences.

[32] E. Gelenbe and I. Mitrani. *Analysis and Synthesis of Computer Systems.* Academic Press, London, 1980.

[33] T. Gonzalez and S. Sahni. Flowshop and jobshop schedules: complexity and approximation. *Operations Research*, 26:36–52, 1978.

[34] A. G. Greenberg and N. Madras. How fair is fair queueing? *Journal of the Association for Computing Machinery*, 39:568–598, 1992.

[35] B. Hajek and R. G. Ogier. Optimal dynamic routing in communication networks with continuous traffic. *Networks*, 14:457–487, 1984.

[36] L. Hall. Approximation algorithms for scheduling. In D. Hochbaum, editor, *Approximation Algorithms for $\mathcal{NP}$-hard problems.* PWS Publishing company, 1997.

[37] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22(3):513–544, 1997.

[38] J. M. Harrison. The bigstep approach to flow management in stochastic processing networks. In F. P. Kelly, S. Zachary, and I. Ziedins, editors, *Stochastic Networks: Theory and Applications*, pages 57–90. Oxford University Press, 1996.

[39] H. Hoogeveen, P. Schuurman, and G. Woeginger. Non-approximability results for scheduling problems with minsum criteria. In R.E. Bixby, E.A. Boyd, and R.Z. Rios-Mercado, editors, *Integer Programming and Combinatorial Optimization (IPCO-VI proceedings)*, Lecture Notes in Computer Science, **1412**, pages 353–366. Springer-Verlag, 1998.

[40] K. Jansen, R. Solis-Oba, and M. Sviridenko. A linear time approximation scheme for job shop scheduling problems. In *Proceedings of APPROX '99*, 1999. (to appear).

[41] K. Jansen, R. Solis-Oba, and M. Sviridenko. Makespan minimization in job shops: a polynomial time approximation scheme. In *Proceedings of the Symposium on Theory of Computing*, pages 394–399, 1999.

[42] E. G. Coffman Jr. and I. Mitrani. A characterization of waiting time performance realizable by single server queues. *Operations Research*, 28:810–821, 1980.

[43] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. In *CRC Handbook on Algorithms*, 1997.

[44] F. P. Kelly. *Reversibility and Stochastic Networks.* Wiley, New York, NY, 1979.

[45] G. P. Klimov. Time sharing service systems i. *Theory Probab. Appl.*, 19:532–551, 1974.

[46] S. Kumar and P. R. Kumar. Performance bounds for queueing networks and scheduling policies. *IEEE Transactions on Automatic Control*, 39:1600–1611, 1994.

[47] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan, and D. B. Shmoys. Sequencing and scheduling: algorithms and complexity. In S. C. Graves, A. H. G. Rinnoy Kan, and P. H. Zipkin, editors, *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445–522. North-Holland, 1993.

[48] F. T. Leighton, B. Maggs, and S. Rao. Universal packet routing algorithms. In *Proceedings of the* 29^{th} *Annual Symposium on Foundations of Computer Science*, pages 256–269, 1988.

[49] H. Levy and M. Sidi. Polling systems: Applications, modeling, and optimization. *IEEE Transactions on Communications*, 38:1750–1760, 1990.

[50] L. Lovász and A. Schrijver. Cones of matrices and set-functions and $0-1$ optimization. *SIAM Journal on Optimization*, 1:166–190, 1991.

[51] X. Luo. *Continuous linear programming: theory, algorithms, and applications.* PhD thesis, Operations Research Center, MIT, 1995.

[52] X. Luo and D. Bertsimas. A new algorithm for state-constrained separated continuous linear programs. *SIAM Journal on control and optimization*, 37(1):177–210, 1999.

[53] C. Maglaras. Discrete-review policies for scheduling stochastic networks: Trajectory tracking and fluid-scale asymptotic optimality. submitted to *Annals of Applied Probability*, 1997.

[54] C. Maglaras and J. Sethuraman. A note on fluid model heuristics for optimal control of queueing networks. Unpublished manuscript, April 1999.

[55] S. P. Meyn. The policy improvement algorithm for markov decision processes with general state space. *IEEE Transactions on Automatic Control*, 42(12):1663–1680, 1997.

[56] S. P. Meyn. Stability and optimization of queueing networks and their fluid models. In G. G. Yin and Q. Zhang, editors, *Mathematics of Stochastic Manufacturing Systems*, volume 33 of *Lectures in Applied Mathematics*, pages 175–200. American Mathematical Society, 1997.

[57] J. F. Muth and G. L. Thompson, editors. *Industrial Scheduling*, Englewood Cliffs, NJ, 1963. Prentice-Hall.

[58] J. Ou and L. M. Wein. Performance bounds for scheduling queueing networks. *Annals of Applied Probability*, 2(2):460–480, 1992.

[59] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of optimal queueing network control. *Mathematics of Operations Research*, 24(2):293–305, 1999.

[60] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Transactions on Networking*, 1:344–357, 1993.

[61] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple-node case. *IEEE/ACM Transactions on Networking*, 2:137–150, 1994.

[62] J. R. Perkins and P.R. Kumar. Optimal control of pull manufacturing systems. *IEEE Transactions on Automatic Control*, 40(12):2040–2051, 1995.

[63] M. C. Pullan. An algorithm for a class of continuous linear programs. *SIAM Journal on Control and Optimization*, 31(6):1558–1577, November 1993.

[64] M. Queyranne and M. Sviridenko. Approximation algorithms for shop scheduling problems with minsum criteria. Technical report, Faculty of Commerce, University of British Columbia, April 1999.

[65] A. N. Rybko and A. L. Stolyar. Ergodicity of stochastic processes describing the operations of open queueing networks. *Problems of Information Transimission*, 28:199–220, 1992.

[66] S. V. Sevast'yanov. On an asymptotic approach to some problems in scheduling theory. In *Abstracts of papers at 3rd All-Union Conference of Problems of Theoretical Cybernetics*, pages 67–69, Novosibirsk, 1974. Inst. Mat. Sibirsk. Otdel. Akad. Nauk SSSR.

[67] S. V. Sevast'yanov. Efficient construction of schedules close to optimal for the cases of arbitrary and alternative routes of parts. *Soviet Math. Dokl.*, 29(3):447–450, 1984.

[68] S. V. Sevast'yanov. Bounding algorithm for the routing problem with arbitrary paths and alternative servers. *Kibernetika*, 22(6):74–79, 1986. Translation in *Cybernetics*, 22:773–780.

[69] J. G. Shanthikumar and D. D. Yao. Multiclass queueing systems: Polymatroidal structure and optimal scheduling control. *Operations Research*, 40:S293–S299, 1992.

[70] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing*, 23(3):617–632, 1994.

[71] J. A. van Mieghem. Dynamic scheduling with convex delay costs: The generalized $c\mu$ rule. *Annals of Applied Probability*, 5(3):809–833, 1995.

[72] L. M. Wein. Scheduling networks of queues: Heavy traffic analysis of a two-station network with controllable inputs. *Operations Research*, 38:1065–1078, 1990.

[73] G. Weiss. On optimal draining of fluid re-entrant lines. In F. P. Kelly and R. J. Williams, editors, *Stochastic Networks*, volume 71 of *Proceedings of the International Mathematics Association*, pages 91–103. Springer-Verlag, New York, 1995.

[74] G. Weiss. Optimal draining of fluid re-entrant lines: some solved examples. In F. P. Kelly, S. Zachary, and I. Ziedins, editors, *Stochastic Networks: Theory and Applications*, pages 19–34. Oxford University Press, 1996.

[75] G. Weiss. An algorithm for minimum weight draining of two station fluid re-entrant lines. Technical report, Department of Statistics, University of Haifa, 1998.

[76] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, October 1995.

THESIS PROCESSING SLIP

FIXED FIELD: ill. _____ name _____

index _____ biblio _____

▶ COPIES: (Archives) Aero (Dewey) Eng Hum
Lindgren Music Rotch Science

TITLE VARIES: ▶ ☐ _____

_____

_____

NAME VARIES: ▶ ☐ _____

_____

IMPRINT: (COPYRIGHT) _____

_____

▶ COLLATION: 158 P

_____

▶ ADD: DEGREE: _____ ▶ DEPT.: _____

SUPERVISORS: _____

_____

_____

_____

_____

_____

_____

_____

NOTES:

cat'r: date:

page: 550

▶ DEPT: O.R.

▶ YEAR: 1999 ▶ DEGREE: Ph.D.

▶ NAME: SETHURAMAN,
Jayachandran