Efficient Algorithms for Buffer Allocation

by

James E. Schor

Bachelor of Science,
University of Massachusetts
Amherst, Massachusetts (1987)

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

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1995

Author ....
Department of Electrical Engineering and Computer Science
May 26, 1995

Certified by ....
Stanley B. Gershwin
Senior Research Scientist, Department of Mechanical Engineering
Thesis Supervisor

Accepted by ....
F. R. Morgenthaler
Chair, Department Committee on Graduate Students

Efficient Algorithms for Buffer Allocation

by

James E. Schor

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 1995, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

This thesis describes four efficient algorithms for allocating *buffer space* in a *transfer line* where the machines are specified. These algorithms, which are based on gradient methods, also calculate the optimal number of *kanbans* for tandem systems that produce a single part type. The objectives of these algorithms are to (1) minimize total buffer space while meeting a production rate constraint, (2) maximize production rate for a specified total buffer space, (3) maximize the profitability of a transfer line for a specified total buffer space and (4) maximize the profitability of a transfer line with no total buffer space constraint. The first two problem formulations ignore inventory costs while the third and fourth formulations include inventory costs. These problems are solved using the deterministic processing time and the continuous material flow transfer line models. The DDX and ADDX algorithms are used to evaluate the production rate and average inventory for these models. These evaluation routines are very efficient, enabling our optimization algorithms to quickly calculate buffer sizes for transfer lines consisting of twenty or more machines.

Thesis Supervisor: Stanley B. Gershwin
Title: Senior Research Scientist, Department of Mechanical Engineering

Efficient Algorithms for Buffer Allocation

by

James E. Schor

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 1995, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

This thesis describes four efficient algorithms for allocating *buffer space* in a *transfer line* where the machines are specified. These algorithms, which are based on gradient methods, also calculate the optimal number of *kanbans* for tandem systems that produce a single part type. The objectives of these algorithms are to (1) minimize total buffer space while meeting a production rate constraint, (2) maximize production rate for a specified total buffer space, (3) maximize the profitability of a transfer line for a specified total buffer space and (4) maximize the profitability of a transfer line with no total buffer space constraint. The first two problem formulations ignore inventory costs while the third and fourth formulations include inventory costs. These problems are solved using the deterministic processing time and the continuous material flow transfer line models. The DDX and ADDX algorithms are used to evaluate the production rate and average inventory for these models. These evaluation routines are very efficient, enabling our optimization algorithms to quickly calculate buffer sizes for transfer lines consisting of twenty or more machines.

Thesis Supervisor: Stanley B. Gershwin
Title: Senior Research Scientist, Department of Mechanical Engineering

Acknowledgments

First I would like to thank my advisor, Stan Gershwin, for being an excellent advisor and a good friend. I also thank him for the financial support I received.

I would like to thank Mitchell Burman twice: First for reinvigorating Stan and motivating additional study in the area of transfer lines and second for doing significant research in this area and making the fruits of his research available for me to use in this thesis.

I would also like to thank Asbjoern Bonvik for always taking the time to listen and for his thoughtful responses. I want to give special thanks to Augusto and Maria for making our office a very fun place at all times but especially in the final days of preparing this thesis. The camaraderie in the office and with our advisor was certainly the most enjoyable part of my M.I.T. experience. Also I'd like to thank Al Drake because he is a great guy.

I also thank my mother for being so accepting, supportive and loving during my formative years, 22 to 30. Last, though first in many respects, I want to thank Sharon. Her encouragement and love made it easy.

In closing I would like to acknowledge my sister Julie, her husband Prasannan and their son Dasartha. She acknowledged me in her book for no reason so I thought I'd do the same here but one up her by acknowledging the rest of her family. Maybe she'll say something nice about me in her next book.

Research reported in this paper has been supported by NSF Grant DDM-9216358.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem description

Manufacturers frequently build dedicated production systems when producing large quantities of a single part type. These systems are often characterized by machines or work centers connected in series and separated by buffers. This arrangement is usually referred to as a *flow line* or a *transfer line*. Material flow along the line may be disrupted by machine failures or variable processing times. Buffers are inserted between machines to limit the propagation of disruptions in material flow. Consequently, the average production rate of the line is higher than a similar line configuration without buffers. Inclusion of buffers in the line requires additional capital investment and floor space, which may be expensive. Buffering also increases the amount of in-process inventory. If buffers are too large, the work-in-process inventory and capital costs incurred will outweigh the benefit of increased productivity. If buffers are too small the line will be underutilized. The problem of how to choose buffer sizes to improve the performance of a transfer line is explored in this thesis.

A considerable amount of work has been done to analyze the effect of machine unreliability and buffer sizes on system performance. Though many papers characterize and describe optimal buffer distributions, significantly less effort has been expended to develop *methods for optimizing* or improving the allocation of buffer space in a production system. The purpose of the this thesis is to construct and describe efficient algorithms for transfer line design. These algorithms will help designers determine how buffer space should be allocated.

1.2 Approach

The general problem is: given the machines of a transfer line, how should we allocate buffer space? To answer this problem certain characteristics of the system we are studying must be ascertained. Specifically, the appropriate transfer line model, constraints and cost function must be determined. In this thesis, we limit the scope of this problem by only considering systems that may be analyzed by employing the deterministic processing time model (Buzacott 1967, Gershwin 1994) or the continuous material flow model (Gershwin and Schick 1980, Burman 1995). We are only concerned with systems that process discrete material. We use the continuous material flow model to approximate the production rate, and average buffer levels, of discrete manufacturing systems consisting of machines with different processing rates.

Within this framework, we begin by considering two formulations of this problem. In these formulations we consider inventory costs to be negligible. The goal of the first formulation, which is denoted the *primal* in this thesis, is to minimize the total buffer space. The specific question that is addressed is: What is the minimum total buffer space required for the line to meet or exceed a given average production rate and how should this space be distributed? The goal of the second formulation, designated the *dual*, is to maximize production rate. The specific question is: What is the maximum production rate achievable for a given total buffer space and how should the space be distributed to achieve this rate?

In Chapters 2 through 6 we describe two algorithms, denoted the *primal* and the *dual*, which answer these questions. The primal algorithm is based on the ability to solve the dual problem. The dual algorithm is constructed so that it is efficient when used to solve the primal problem.[1]

[1] In most cases significantly less computational effort will be required to solve the dual problem than the primal problem. Therefore, when constructing the dual algorithm, we are not concerned about minimizing the computational effort required to solve the dual as a stand-alone problem.

These algorithms are primarily intended as line design tools, but the algorithms may also be used to design production control policies, provided demand changes are relatively infrequent. If *kanbans* are used to authorize production, the primal algorithm determines the minimum number of kanbans needed to achieve a given production rate.

In Chapter 2, we describe the deterministic processing time and continuous material flow transfer line models and formalize the dual and primal problems. We then preview the dual and primal algorithms which we describe in detail in Chapters 3 and 5. These algorithms are based on the premise that we are able to evaluate the function $P(N_1, \ldots, N_{k-1})$, where N_i is the size of the i^{th} buffer, P is the production rate of the line, and k is the number of machines in the line. The function P is assumed to monotonic and concave. This allows us to use existing gradient methods to optimize the line. The justification for these assumptions is presented in Chapter 2.

We use the DDX (Dallery, David, and Xie 1988) algorithm to evaluate $P(N_1, \ldots, N_{k-1})$ when employing the deterministic processing time model. The ADDX algorithm (Burman 1995) is used when the system is modeled using the continuous material flow model. Both the DDX and ADDX algorithms use analytical methods and decomposition (Gershwin 1987) to determine an approximate solution for $P(N_1, \ldots, N_{k-1})$. The speed of the these algorithms, and their ability to analyze long lines, enables the primal and dual algorithms to solve practical problems. We present examples describing the performance of the dual and primal algorithms in Chapters 4 and 6.

In Chapters 7 we reformulate the buffer allocation problem to include inventory costs, but dispense with the production rate constraint. The line design criterion in this case is to maximize profit. In Chapter 8 we construct two algorithms to solve this problem. Chapter 9 describes the behavior of these algorithms. In Chapter 10, we use the algorithms to develop intuition about transfer lines. Chapter 11 discusses future research.

1.3 Literature review

There is a large number of papers concerned with analysis of many different transfer line models (Dallery and Gershwin 1992). The models studied are mainly distinguished by the following characteristics: the machines may be either reliable or unreliable; the processing time of individual machines may be either deterministic or variable; and the buffers between machines may be finite or infinite.

Most of this literature addresses the issue of how to determine expected system performance. One of the major reasons authors focused on system performance was to provide designers of production systems with tools to facilitate the design process. The ability to analyze the performance of transfer lines allowed the development of optimization algorithms for line design. In this thesis we develop four efficient buffer allocation algorithms. Therefore, in this review we are mainly concerned with either existing optimization methods or papers that discuss qualitative properties. We focus on the qualitative properties of transfer lines because these properties dictate what optimization techniques are appropriate.

1.3.1 Simulation

The time expended in performing the evaluation step in an optimization procedure may significantly impact the practical value of the algorithm. In our algorithms we have chosen to calculate production rate using the DDX and ADDX algorithms. Because these algorithms are based on analytical methods and decomposition, they are able to evaluate reasonably long lines very quickly. This allows us to use these algorithms to optimize transfer lines consisting of twenty or more machines. Unfortunately, these algorithms are only available for models that exhibit the following properties. Failure and repair times are either geometric or exponentially distributed random variables and process times are either deterministic or exponentially distributed. The DDX algorithm was developed in 1988 and the ADDX was developed in 1995. Before the development of these algorithms, simulation was the only method for determining production rates and average buffer levels for large systems.

An alternative method of evaluation is simulation. We have chosen not to use simulation as our evaluation method because it requires much more computation time than the DDX and ADDX algorithms. The benefit of simulation is that models evaluated by simulation may incorporate arbitrary failure, repair and processing time distributions. In addition, simulation can determine the variability and transient behavior of performance measurements.

One of the earliest papers describing the use of simulation to optimize inventory levels in a transfer line is Barten (1962). Barten simulates transfer lines of up to ten machines with normally distributed processing times. He varies the total buffer space but imposes the restriction that all buffers are the same size. He uses the simulation results to determine optimal buffer sizes by expressing mean delay time, and ultimately total cost, as a function of storage capacity. The strategy of using simulation to predict a performance measurement, in this case mean delay time, as a function of the line parameters has been extensively used. Anderson and Moodie (1969) also use simulation to predict operating costs as a function of line length and buffer sizes.

Both Chow (1987) and Liu and Lin (1994) use simulation to construct functions that predict the throughput and the CV of the interdeparture time of a reliable two-machine line. Liu and Lin (1994) improve upon Chow's approach by designing an experiment that requires fewer simulations to construct these functions. Liu and Lin (1994) also propose a method of extending their results to longer lines. This method uses an iterative procedure that eventually represents the long line as a single two-machine line. Each iteration replaces the first two machines in the line by a single machine with the same performance characteristics of the two-machines in isolation. Liu and Lin (1994) then use this evaluation method in a dynamic program to solve the production rate maximization problem. They propose solving the minimum total buffer space problem by using the solution to the production rate maximization problem in the following procedure: (1) Evaluate maximum throughput beginning with the zero buffer case. (2) Increment the total buffer space by one until the throughput constraint is satisfied.

We also propose using the algorithm we develop to solve the production maximization problem to solve the total buffer space minimization problem but employ a much more efficient procedure than that of Liu and Lin (1994). The experiments they perform are on lines of less than ten machines with a total buffer space of less than fifty. When evaluating longer lines they only consider bottleneck machines.

Other authors construct algorithms to optimize buffer allocation by doing a *perturbation analysis*. The output from a single simulation is used to construct a gradient of the production rate with respect to buffer sizes. Ho, Eyler, and Chien (1979) and (1983), and Caramanis (1987) use this approach to solve the production rate maximization problem. For a more detailed discussion of the use of simulation in optimizing buffer allocation see Gershwin and Goldis (1995).

1.3.2 Exact solutions

Transfer lines with unreliable machines and finite buffers are studied by Hillier and So (1991). Hillier, So, and Bolling (1993) study reliable machines with exponential processing times. Both papers evaluate all possible buffer combinations. From these results both the solution to the production rate maximization and total buffer space minimization problems may be extracted. They use an exact method to determine production rate. The difficulty with this approach is that exact solutions may only be obtained for small systems.[2] They observe that the optimal allocation for buffers, in a balanced transfer line, resembles an inverted bowl with the amount of buffer space increasing for buffers close to the center of the line. This important characteristic is denoted the *inverted bowl phenomenon*.

1.3.3 Approximate solutions

There are a number of papers that use decomposition as a method of evaluating production rates. Two papers that explicitly address the total buffer space minimization

[2]Hillier *et al.* analyze transfer lines consisting of less than ten machines and a total buffer space less than forty. In cases where the line is longer than five machines they usually restrict the total buffer space to be less than ten.

problem are Park (1993) and Gershwin and Goldis (1995).

Park (1993) develops a "two phase heuristic" to solve the total buffer space minimization problem. His algorithm uses a "dimension reduction strategy" and a "buffer utilization-based beam search" that is only guaranteed to find a near-optimal solution. Park uses in-process inventory levels to decide which buffer to increase. In certain cases this criterion chooses to increment a buffer which gives a smaller increase in production rate than the buffer that would be chosen using a gradient method. Consequently Park's algorithm does not always converge to an optimal or near optimal solution.

Gershwin and Goldis (1995) employ a gradient method to solve the total buffer space minimization problem. Their algorithm is based on the observation that if the production rate is expanded to first order the the problem may be formulated as an integer linear program. They construct three different algorithms to solve this problem. They guarantee that their solutions are optimal or near-optimal. Their algorithm searches using a procedure that is fundamentally different from the one presented here. Our algorithm converges more quickly in the majority of the cases we study.

Seong, Chang and Hong (1994a) use a gradient method to solve the production rate maximization problem for a transfer line with exponentially distributed failure, repair, and processing times. Seong, Chang, and Hong (1994b) also solve this problem and the profit maximization problem for a specified total buffer space for a continuous flow transfer line. We compare our results with their's in Chapter 9. Dogan and Altiok (1995) use decomposition to estimate the throughput gradient vector with respect to repair rates. Their ultimate goal is to optimize "repair rate allocation".

Jacobs and Meerkov (1993, 1994) employ the concept of *improvability* to determine how buffer space should be allocated. They propose a method for determining whether the individual buffer sizes may be changed, while holding the total buffer space constant, so that the production rate will increase. The transfer line model they study is different than the two models employed in this thesis. Kuo, Lim and Meerkov (1994) present an application of this work.

1.3.4 Qualitative properties

The algorithms we construct are based on steepest descent methods and incorporate existing, though slightly modified, line search procedures. Any guarantees we make concerning the optimality, or near optimality, of our primal and dual solutions are predicated on assumptions about the properties of $P(N_1, \ldots, N_{k-1})$. The following two properties are required to ensure the correctness of our algorithms. The first property is that production rate is a monotonically increasing function of individual buffer size, provided all other quantities are held constant. The second property is that production rate is a concave function of buffer sizes. If the second property holds, the feasible region for the dual formulation will be a concave set.

Glasserman and Yao (1992) demonstrate that production rate is a monotonically increasing function of buffer size for systems with general blocking and general service times. They formulate a transfer line model as a generalized semi-Markov process to derive this result. Even though they assume reliable servers this result is applicable to our system for the following reason.

The differences between flow lines with reliable machines and flow lines with unreliable machines has more to do with how the system is described than how the system performs. This is because any system with unreliable machines may be transformed into a system of reliable machines by changing the distribution of the service time. For example, Altiok and Stidham (1983) show that a machine with exponential service, failure and repair times may be represented as a reliable machine with a coxian server.

Many numerical experiments, some of which are presented in this thesis, support our conjecture about the concavity of the production rate function. We know of no results that conclusively show that the systems we study in this thesis exhibit the concavity property, but we feel this is a reasonable assumption given work done in similar systems. The previously published results we present as justification generally assume exponential reliable servers (machines). This is because models that have general service time distributions are not as tractable as Markovian models. Consequently, most of the published results are for systems with exponential service times. In the next paragraphs we discuss papers that show that throughput (production rate) is monotonic and concave for many systems that are similar to ours.

Shantikumar and Yao (1989) show that production rate is an increasing function of buffer size for a cyclic queuing network with exponential servers and finite buffers. Adan and Van der Wal (1989) demonstrate that production rate is a monotonic function of buffer size for an acyclic system.

Meester and Shantikumar (1990) show that production rate is an increasing concave function of buffer size for a tandem queueing system with exponential servers and finite buffers. Antharam and Tsoucas (1990) also demonstrate concavity for a system of queues with finite buffers and exponential servers. Dallery, Liu and Towsley (1994) generalize the results of Shantikumar and Yao (1989) and Mester and Shantikumar (1990).

We know of no published papers that discuss the qualitative properties of profit as a function of buffer sizes. Gershwin (1994) shows that average buffer level may be either a concave or convex function of buffer size for a two-machine line. The shape of this function depends upon the reliability parameters and processing rates of the machines. Consequently, if profit is a function of average buffer level, it may not be concave.

Chapter 2

Problem formulation and system description

2.1 Description of general problem

The goal of this thesis is the construction of algorithms which choose buffers sizes for a transfer line. The number of machines and the reliability parameters of each machine are assumed to be specified. The next five chapters explore the problems that arise from two different, but very closely related, buffer allocation criteria. The first problem, designated the *primal*, imposes a production rate constraint to guarantee a minimum level of customer service. The second problem, designated the *dual*, assumes that the total buffer space is specified. The goal in both cases is to determine the sizes of each buffer. The dual problem formulation may be used in line design or as subproblem when solving the primal. The type of system that is appropriate for each problem formulation is described below.

A solution to the primal problem is a buffer allocation that minimizes the total buffer space while enabling the line to meet or exceed a given production rate target. This formulation is appropriate if either floor space or buffering machinery is expensive, work in process inventory is inexpensive, and if an average production rate is mandated. The cost of buffer space or machinery is assumed to be directly proportional to buffer size and is equivalent for all buffers. The problem is formulated as a constrained minimization problem. The objective function and constraints are determined by capital costs and forecasted demand respectively.

The dual problem seeks a buffer allocation that maximizes the production rate for a given total buffer size. If a fixed amount of capital is budgeted for buffer space and demand is either unconstrained or unknown, the dual formulation is appropriate. When a limited amount of floor space is available the dual formulation may also be appropriate.

We state in Section 1.3 that some authors determine optimal buffer distributions by enumerating all possible buffer configurations. This approach is time-consuming and significantly limits the size of the problems that may be studied. By employing a gradient method we significantly reduce the number of different buffer distributions that must be evaluated.

Different methods for evaluating production rates also exist. We use an approximate decomposition method, developed by Gershwin (1987), to analytically determine production rate and average buffer levels. The algorithm we use to evaluate the performance measures for the deterministic processing time model was developed by Dallery, David, and Xie (1988). This algorithm is known as the DDX algorithm. It is quicker and converges more frequently than Gershwin's original algorithm. The algorithm we use to evaluate the performance measures for the continuous material flow model was developed by Burman (1995) and is known as the ADDX algorithm. The DDX and ADDX algorithms are much quicker than simulation and are able to calculate production rates for systems for which no analytical solution exists[1] or are too large to solve by brute force numerical techniques.

2.2 Models

The buffer allocation procedure we propose will find an optimal or near-optimal solution for any transfer line model provided production rate, as a function of buffer sizes

[1]Gershwin and Schick (1983) derived an analytical solution for a three-machine line with unreliable machines and small buffers. No solutions exist for longer lines.

and machines parameters, possesses the qualitative properties postulated in Section 2.6. Therefore, we solve both the dual and primal problems using two different transfer line models. The two models are the deterministic processing time long line and continuous material flow long line models. The deterministic long line model is chosen because it has has been widely studied. This allows us to compare our results with results reported by other authors. We use the continuous long line model because it is a more general model which allows us to optimize lines that include machines with different cycle times. One of the reasons the original continuous material flow model was constructed was to model the movement of discrete material in a transfer line, provided certain conditions are satisfied. In the rest of this section we describe these models.

2.2.1 Deterministic processing time model

In this section we describe the deterministic processing time model of Gershwin (1987) which is based on an earlier Buzacott model (1967). For a complete, detailed description see Gershwin (1994). The model assumes discrete parts and synchronous movement of material. All operations require one time unit. Consequently, all working machines have identical, constant processing times. The model assumes operation dependent failures, that is, that a machine cannot fail if it is either starved or blocked. Therefore, if buffer $i-1$ is empty or buffer i is full, machine i is not allowed to operate and may not fail. The probability that machine i fails, provided that it is working, is p_i. The number of operations performed by machine i between consecutive failures is a geometric random variable with mean $1/p_i$. If machine i is down the probability it is repaired in any given time unit is r_i. The time until machine i is repaired is also a geometric random variable with mean $1/r_i$.

Frequently used measures of reliability are the mean time to fail (MTTF) and the mean time to repair (MTTR). For our model MTTF $= 1/p_i$ and MTTR $= 1/r_i$. The MTTF is the average number of operations between failures, which is less than the average time between failures. This is because a machine may not fail when it is blocked or starved. Consequently, the MTTF does not include the idle time caused

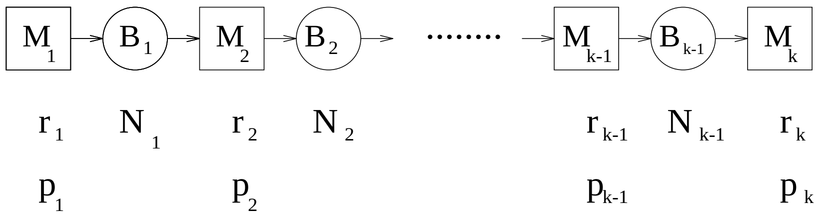

Figure 2-1: Deterministic processing time transfer line

by either blockage or starvation.

The size of the i^{th} buffer is denoted as N_i, the vector of buffer sizes is $\mathbf{N} = (N_1, \ldots, N_{k-1})$. The number of parts in the i^{th} buffer is n_i. The line is uniquely characterized by specifying $r_1, p_1, N_1, r_2, p_2, N_2, \ldots, r_{k-1}, p_{k-1}, N_{k-1}, r_k, p_k$. Figure 2-1 is a pictorial representation of this system.

The system may be modeled as a discrete time, discrete state Markov chain. Each state may be represented by the vector $(n_1, \ldots, n_{k-1}, \alpha_1, \ldots, \alpha_k)$ where α_i is a binary variable equal to 0 if machine i has failed and 1 if machine i is operational. Finite buffers require that $0 \leq n_i \leq N_i$. Consequently the number of states required to represent the line is $2^k \prod_{i=1}^{k-1} (N_i + 1)$.

We use the DDX algorithm to calculate the performance measures for a deterministic processing time transfer line.

2.2.2 Continuous material flow model

In this section we describe the continuous material flow model (Gershwin and Schick 1980). For a complete, detailed description see Burman (1995). We use this model to approximate discrete manufacturing systems consisting of machines with deterministic, but different, processing rates. If buffers are present, and not too small, the continuous material flow model will do a good job of predicting the production rate and average buffer levels of discrete systems. See Suri and Fu (1992) or Burman (1995) for a discussion of the accuracy of this approximation.

The processed material is treated as a continuous fluid. A machine may be thought

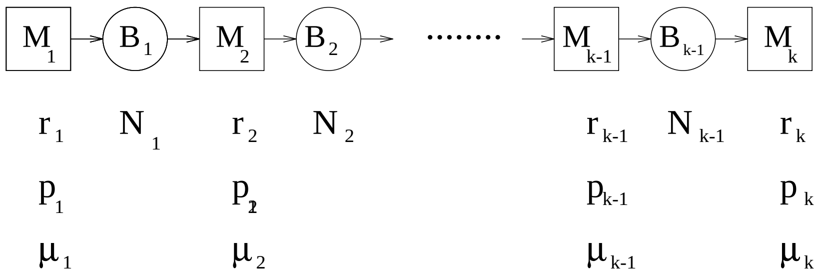

Figure 2-2: Continuous material flow transfer line

of as a valve that is open when the machine is up and closed when the machine is down. Buffers are storage tanks separating the valves.

Failures are operation dependent which means that a machine may only fail when it is working. The amount of working time until machine i fails is an exponential random variable with mean $1/p_i$. The time to repair machine i is also a exponential random variable with mean $1/r_i$. As in the deterministic processing time model, the MTTF $= 1/p_i$ and the MTTR $= 1/r_i$. In the deterministic processing time model r_i and p_i are probabilities. In this model r_i and p_i are rates. Unlike the deterministic model, we do not require that all machines process material at the same rate. Consequently, the additional parameter μ_i is needed to represent the maximum processing rate of machine i. We also define the quantity $\rho_i = \frac{\mu_i r_i}{r_i + p_i}$, which is the long term isolated processing rate of machine i. In this model maximum processing rates are constant . Machine i will process at μ_i provided buffer $i-1$ is not empty and buffer i is not full. Unlike the deterministic processing time model, machine i may continue to operate when buffer $i-1$ is empty or buffer i is full provided that machine $i-1$ is up and not starved or machine $i+1$ is up and not blocked respectively. If either of these conditions occur the processing rate and failure rate of machine i must be adjusted because machine i is no longer decoupled from the adjacent machine.

The size of the i^{th} buffer is still denoted as N_i and the vector of buffers sizes is $\mathbf{N}$. The amount of material in the i^{th} buffer is a continuous variable denoted n_i. The line is uniquely characterized by specifying $\mu_1, r_1, p_1, N_1, \ldots, \mu_{k-1}, r_{k-1}, p_{k-1}, N_{k-1}, \mu_k, r_k, p_k$. Figure 2-2 is a pictorial representation of this system.

The system may be modeled as a mixed state, continuous time Markov process. The state of the system may be represented by the vector $(n_1, \ldots, n_{k-1}, \alpha_1, \ldots, \alpha_k)$ where α_i continues to represent the machine state and finite buffers require that $0 \leq n_i \leq N_i$.

We use the ADDX algorithm to calculate performance measures for a continuous material flow line.

2.2.3 Additional notation

The performance measures we are concerned with are the *production rate* of the line and *average buffer levels* when the line is in steady state. The production rate is also the average number of parts produced per time unit and the efficiency of the line for the deterministic model. The production rate of the line is a function of the buffer sizes, reliability parameters, and machine processing times. In this thesis we express the production rate of both the deterministic processing time transfer line and the continuous material flow line as $P(N_1, \ldots, N_{k-1})$. This is because buffer sizes are the only quantities that are not fixed. The average inventory level of buffer i is $\bar{n}_i$. The target production rate for the primal problem is expressed as P^* and the total buffer space is $N^{total} = \sum_{i=1}^{k-1} N_i$. We also denote the minimum total buffer space that will achieve P^* as N^*.

The maximum average production rate, for a specified N^{total}, is $P_{max}(N^{total})$. That is $P_{max}(N^{total})$ is the production rate of the line when the buffers are allocated optimally and the allocation criterion is to maximize throughput. $P_{max}(N^{total})$ is also the value of the objective function of the solution to the dual problem. The function $P^{-1}_{max}(P)$ gives the minimum total buffer space which achieves production rate P. Therefore, $P^{-1}_{max}(P^*) = N^*$. N^{total} is a specified constant for each instance of the dual problem. N^{total} is a variable in the primal formulation.

2.3 Problem statements

In this section we formalize the primal and dual problem statements. We also introduce the one dimensional primal problem, which appears as part of the algorithm constructed for solving the primal problem.

2.3.1 Primal problem

Choose $N_1, \ldots, N_{k-1}$ to

$$\text{minimize} \;\; N^{total} = \sum_{i=1}^{k-1} N_i \tag{2.1}$$

subject to

$$P(N_1, \ldots, N_{k-1}) \geq P^*; \;\; P^* \;\text{specified}$$

N_i non-negative integer for the deterministic processing time model ($N_i \in Z^+$)

N_i non-negative real for the continuous material flow model ($N_i \in R^+$)

The solution to the primal problem minimizes N^{total} subject to the production rate being greater than or equal to P^*. The objective function is linear and the constraint is non-linear.

The solution to the primal problem may not be unique. It may be possible to allocate the same total buffer space in different ways and still satisfy the constraint. The total buffer space required for all solutions will be the same but the production rate achieved may differ.

The implementation of the deterministic long line model that is available to us enforces the additional constraint $N_i \geq 4$. This is a consequence of facilitating the coding of the DDX algorithm and is not inherent to the solution technique.

Though a significant amount of work has been done to determine buffer distributions only two papers explicitly address the problem as stated in problem (2.1): Park (1993) and Gershwin and Goldis (1995). The same problem, using a different model, is studied by Jacobs and Meerkov (1993, 1994) but the goal was a production control technique, not a line design tool.

2.3.2 Dual problem

Choose $N_1, \ldots, N_{k-1}$ to

$$\text{maximize} \;\; P(N_1, \ldots, N_{k-1}) \tag{2.2}$$

subject to

$$N^{total} = \sum_{i=1}^{k-1} N_i; \; N^{total} \, \text{specified} \tag{2.3}$$

N_i non-negative integer for the deterministic processing time model ($N_i \in Z^+$)

N_i non-negative real for the continuous material flow model ($N_i \in R^+$)

The goal is to maximize the production rate for a specified N^{total}. The total buffer space constraint is linear and the objective function is nonlinear. The solution to the dual, like the primal solution, may not be unique. Unlike the primal, all solutions to the dual give the same production rate.

We study the dual problem for two reasons. The dual formulation is appropriate for many cases of the line design problem. Second, determining a dual solution is a critical step in the algorithm we propose for solving the primal problem.

As in stated Section 1.3, the dual problem is widely studied. Ho, Eyler, and Chien (1979) use a simulation-based method to study the dual formulation for machines with deterministic processing times. We compare their results to ours in Section 4.3. Seong, Chang, and Hong (1994a) also address the dual problem as stated in (2.2). Their are two major differences between their work and ours. The first is that they employ a model with random, independent processing times while we consider machines that have deterministic processing times. The second is that they only consider feasible buffer distributions when searching for an optimum. We relax the constraint that buffer sizes be discrete in the early stages of the algorithm. This reduces the search time. The justification for this approach is discussed in Section 2.5.

Seong, Chang and Hong (1994b) solve a more general formulation of the dual problem for a continuous flow production system. Their formulation allows for the specification of additional linear constraints and assigns a cost for inventories. They solve one case which is identical to the dual problem and we discuss this case in Chapter 9.

2.4 Properties of $P(N_1, \ldots, N_{k-1})$

We discuss the differentiability, monotonicity, and concavity of $P(N_1)$ for the deterministic processing time two-machine line in Section 2.4.1. The production rate of a three-machine deterministic processing time line is explored in Section 2.4.2. We argue that the numerical experiments presented in this section and work done on similar systems supports the assertion that $P(N_1, \ldots, N_{k-1})$ is monotonically increasing and concave. We also argue that $P_{max}(N^{total})$ is monotonically increasing.

After stating that $P(N_1, \ldots, N_{k-1})$ is a monotonic and concave function we discuss how these properties are central to the development of both the primal and dual algorithms.

2.4.1 Two-machine line

Buzacott (1967) demonstrates that efficiency of a deterministic processing time two-machine line is a ratio of two polynomials of degree N. This is also true for our deterministic processing time model (Gershwin 1994). When the restriction $N_1 \in Z^+$ is relaxed to $N_1 \in R^+$, $P(N_1)$ becomes a differentiable, monotonically increasing, concave function. That is, if we permit N_1 to range over all non-negative reals, $P(N_1)$ has these properties.

Okamura and Yamashina (1977) do many numerical experiments demonstrating that $P(N_1)$ is monotonically increasing. They classify two-machine lines into three categories. The representative curves for all three categories are concave although the degree of curvature differs significantly. They also conclude that incrementing small buffers provides substantial increases in production rate but the improvement decreases when buffers become large. In the following paragraph we offer an intuitive explanation of this behavior.

Number the spaces in a buffer from 1 to N_i and consider that each buffer space provides a hedge against a failure. This hedge is used when a failure is at least as long as the number of the space. The first space hedges against all failures. The higher numbered spaces only hedge against very long failures. Therefore, increasing N_i by 1 when N_i is small will give a larger increase in P than increasing N_i by 1 when N_i is large.

For a detailed discussion of the two-machine line see Gershwin (1994) or Gershwin and Goldis (1995).

2.4.2 Three-machine line

In this section we investigate a three-machine line, consisting of machines with identical processing times and the failure and repair probabilities described in Table 2.1. The machines are the first three machines in the twelve-machine line studied by Park and described in Table 4.2. Figure 2-3 plots production rate vs. buffer sizes for this line. Production rate appears to be monotonically increasing in each N_i. Furthermore, if N_j is held constant $\forall j \neq i$ then as i increases, P approaches a finite limit. Figure 2-3 also supports the conjecture that production rate is close to a concave function of (N_1, N_2).

The purpose in establishing the second order properties of $P(N_1, N_2)$ and asserting that $P(N_1, \ldots, N_{k-1})$ also has these properties is to justify the method employed to solve the dual problem.

Glasserman and Yao (1992) showed that $P(N_1, \ldots, N_{k-1})$, for the models we study, is monotonically increasing in each N_i. Section 1.3 cites work done in similar, but not identical, systems which shows that $P(N_1, \ldots, N_{k-1})$ is concave. We present numerical evidence to argue that the production rate behaves similar to a concave function of buffer size for two-machine and three-machine lines. We therefore conjecture that $P(N_1, \ldots, N_{k-1})$ is also concave for $k \geq 3$.

If $P(N_1, \ldots, N_{k-1})$ is a concave function we can show that $P(N_1, \ldots, N_{k-2}, C - \sum_{i=1}^{k-2} N_i)$, where C is a constant, is also a concave function. The argument is as

Machine	r_i	p_i
1	0.35	0.037
2	0.15	0.015
3	0.4	0.02

Table 2.1: Three-machine line parameters

follows[2]:

1. The function $P(N_1, \ldots, N_{k-1})$ is concave function on the convex set $C = \mathbf{N} : \mathbf{N} \geq 0$.

2. The set of points, $D = \mathbf{N} : \sum_{i=1}^{k-1} N_i = d$, where d is a constant, is a convex set such that $D \subset C$.

3. Therefore $P(N_1, \ldots, N_{k-2}, d - \sum_{i=1}^{k-2} N_i)$ is also a convex function on a convex set.

A sufficient condition, for the algorithm we construct in Chapter 3 to work correctly is that $P(N_1, \ldots, N_{k-2}, d - \sum_{i=1}^{k-2} N_i)$ is concave function.

2.5 Solution approach

If the properties of monotonicity and concavity, exhibited by two-machine and three-machine lines, extend to longer lines then it is reasonable to use existing optimization algorithms and methods to solve the primal and dual problems.

The algorithm we develop to solve the dual, for the deterministic processing time model, is based on our ability to construct a differentiable function $\hat{P}(Y_1, \ldots, Y_{k-1})$, where $Y_i \in R^+ \;\; \forall \; i$. This function is increasing in each Y_i when $Y_j \;\; \forall \; j \neq i$ is non-decreasing and satisfies equation (2.4). If we are solving the dual problem for the continuous material flow model we do not require that N_i be discrete so the function $\hat{P}(N_1, \ldots, N_{k-1}) = P(N_1, \ldots, N_{k-1}) \; \forall \; \mathbf{N} \geq 0$. The following argument only applies to cases where we employ the deterministic processing time model. Assume that

[2]See Luenberger (1984) for a detailed, formal argument.

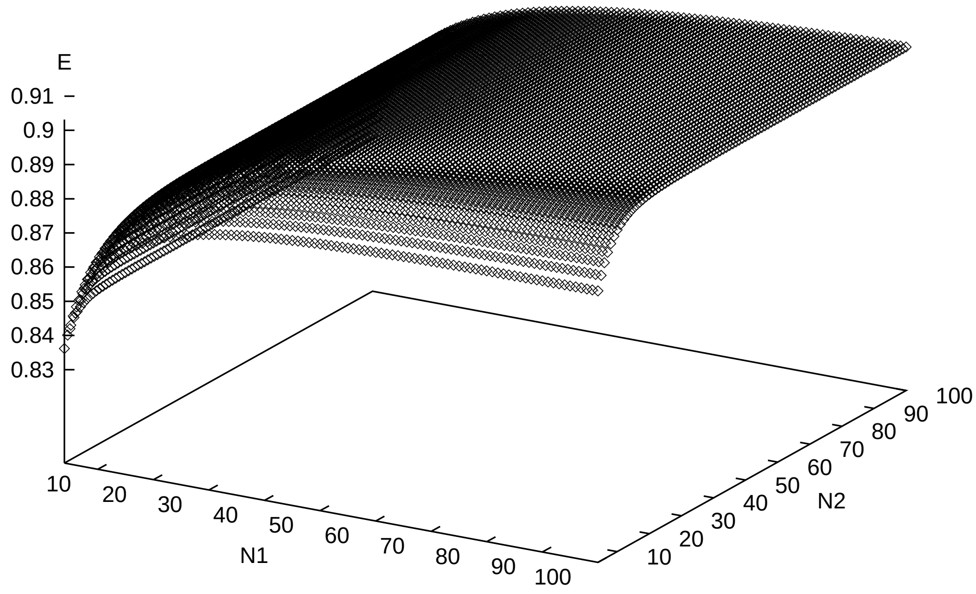

Figure 2-3: P vs. N_1 and N_2

$$\hat{P}(N_1, \ldots, N_{k-1}) = P(N_1, \ldots, N_{k-1}) \tag{2.4}$$

$$\forall\ (N_1, \ldots, N_{k-1}); \quad N_i \in Z^+ \ \ \forall\ i$$

This construction allows us to transform our original integer programming problem into a programming problem with continuous variables. Since $\hat{P}(Y_1, \ldots, Y_{k-1})$ is differentiable, a search algorithm based on steepest descent methods is appropriate. The transformed problem may be solved more easily than the original problem.

Approximating a discrete function by a continuous function is a common technique for solving optimization problems (Bradley *et al.* 1977). We propose solving both the dual and primal problems for the deterministic processing model in the following way:

1. Construct $\hat{P}$.

2. Replace $P(N_1, \ldots, N_{k-1})$ by $\hat{P}(Y_1, \ldots, Y_{k-1})$ in both the dual and primal formulations.

3. Replace the constraint $N_i \in Z^+$ with $Y_i \in R^+$ in both the dual and primal formulations.

4. Solve the transformed problem.

5. Evaluate discrete buffer distributions near the solution to the transformed problem to determine the solution to the original problem.

We construct a function of continuous variables for reasons of mathematical convenience and tractability. We are not solving a new problem. When employing the deterministic processing time model we are not considering a system that processes continuous material.[3] The underlying model is still a discrete material model.

[3] If we were considering a continuous material flow system we wouldn't need to construct $\hat{P}(Y_1, \ldots, Y_{k-1})$. We would only use the continuous material flow model.

The approach we propose is contingent upon the ability to construct $\hat{P}(Y_1, \ldots, Y_{k-1})$. This means that we need to be able to evaluate $\hat{P}(Y_1, \ldots, Y_{k-1})$ for [4] $Y_i \in R^+$.

When we are analyzing systems where all machines have the same cycle time we use the deterministic processing time model. The production rate of the deterministic two-machine line may be evaluated for non-integer buffer sizes. The result may not be meaningful, in a physical sense, but the solution does exhibit the property we require. That is, $\hat{P}(Y_1)$ is increasing in Y_1. The DDX algorithm for the deterministic processing time long line uses the two-machine line solution to evaluate production rates. Therefore, we are able to use the DDX algorithm to construct $\hat{P}(Y_1, \ldots, Y_{k-1})$.

We transform the dual and primal problems by dropping the restriction that N_i be discrete. We do this to reduce the computational complexity of the problem. We reimpose this constraint before determining a final solution. To simplify notation we will use $P(N_1, \ldots, N_{k-1})$ in place of $\hat{P}(Y_1, \ldots, Y_{k-1})$ for the remainder of this thesis.

2.5.1 One dimensional primal problem

We solve the primal problem by dividing the problem into two subproblems which we know how to solve. This approach is motivated by the observation that the solution to the dual problem for some N^{total}, if its cost is P^*, is also a solution to the primal problem when the specified production target is P^*. Once we are able to determine the value of N^*, which corresponds to P^* specified in the primal problem, we may choose $N_1, \ldots, N_{k-1}$ by solving the dual problem. The ability to evaluate $P_{max}(N^{total})$ permits us to solve the primal problem by solving the subproblem described in (2.5). We refer to this subproblem as the *one dimensional primal problem* (ODPP). The second subproblem, which is the evaluation of $P_{max}(N^{total})$, is the dual problem.

One dimensional primal problem

Choose N^{total} to

$$\text{minimize} \quad N^{total} \tag{2.5}$$

[4]As stated in the original problem statement the implementation of the primal and dual algorithms for the deterministic processing time model requires the additional constraint $Y_i \geq 4$ be imposed. For the remainder of this thesis we will assume that the reader is familiar with this detail of the implementation and will no longer explicitly address this constraint.

subject to $\qquad P_{max}(N^{total}) \geq P^*;\ N^{total} \in Z^+$

The method developed in Chapter 5 to solve ODPP requires that $P_{max}(N^{total})$ is a monotonically increasing function. To show this , let

$$P_{max}(N^{total}) = P(\tilde{N}_1, \ldots, \tilde{N}_i, \ldots, \tilde{N}_{k-1})$$

where $\sum_{i=1}^{k-1} \tilde{N}_i = N^{total}$ and $(\tilde{N}_1, \ldots, \tilde{N}_i, \ldots, \tilde{N}_{k-1})$ is the solution to the dual problem as stated in (2.2). Then

$$P_{max}(N^{total}) \leq P(\tilde{N}_1, \ldots, \tilde{N}_i + 1, \ldots, \tilde{N}_{k-1}) \leq P_{max}(N^{total} + 1)\ \ \forall\, i \tag{2.6}$$

because $P(N_1, \ldots, N_{k-1})$ is monotonically increasing.

Figure 2-4 is a graph of production rate vs. total buffer space optimally allocated, for the three-machine line described in Table 2.1. The shape of the curve is consistent with the assertion that $P_{max}(N^{total})$ is a monotonically increasing function. The function also approaches a limit which is the production rate of the line with infinite buffers. The graph exhibits the same saturating behavior as a two-machine line and coincides with the behavior that Okamura and Yamashina (1977) observe.

Figures 2-5 and 2-6 are graphs of near-constant P curves for the three-machine lines with reliability parameters described in Tables 2.1 and 2.2 respectively. All machines, in both lines, operate at the same rate. We refer to these curves as *Iso-P* curves because all points on a given curve achieve approximately equal production rates. The production rate of any buffer distribution on an Iso-P curve in both figures differs from the rate reported for that curve by no more than 0.001. When we generated these graphs we only evaluated discrete buffer sizes. Therefore, it is very unlikely that any two buffer distributions we evaluate will give the exact same production rate. Each Iso-P curve my be described as the collection of buffer size vectors which achieve approximately equal production rates. The nesting of the the Iso-P curves is consistent with the assertion that $P_{max}(N^{total})$ is monotonically increasing and that $P(N_1, \ldots, N_{k-1})$ is concave.

The *optimal curve*, shown in both figures, consists of all buffer distributions that

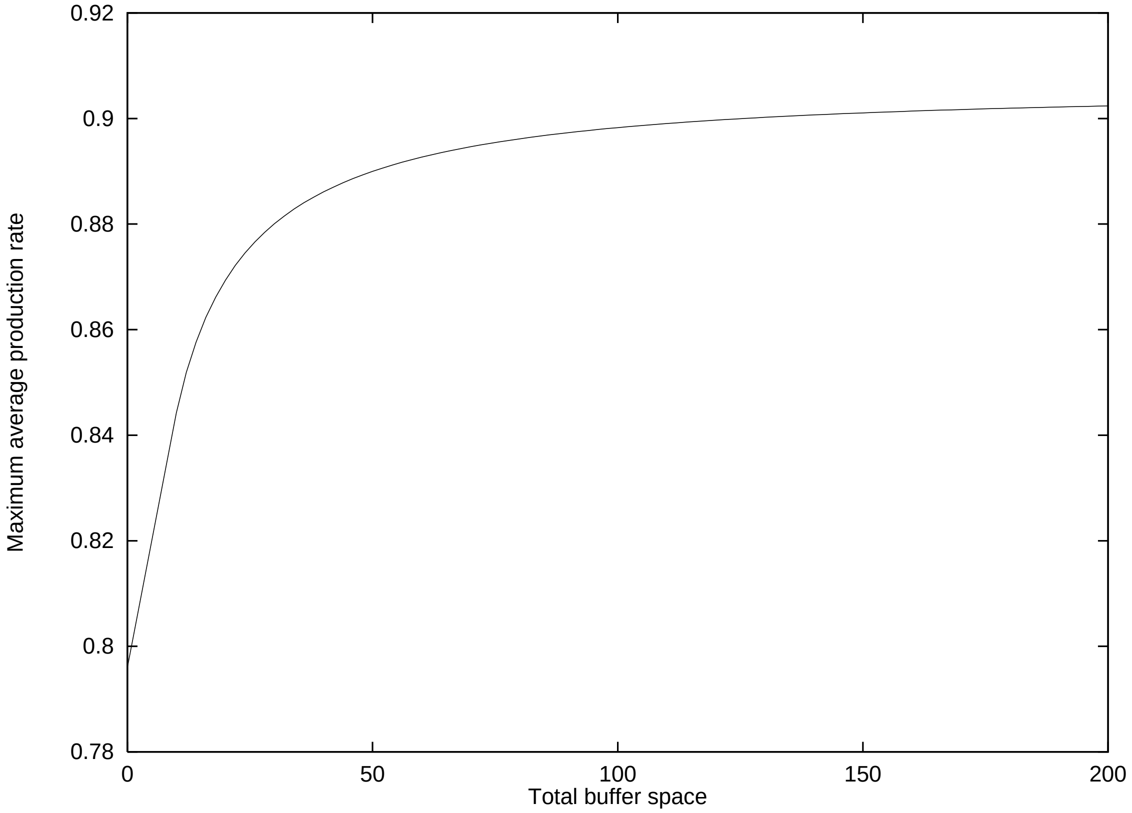

Figure 2-4: P_{max} vs. N^{total}

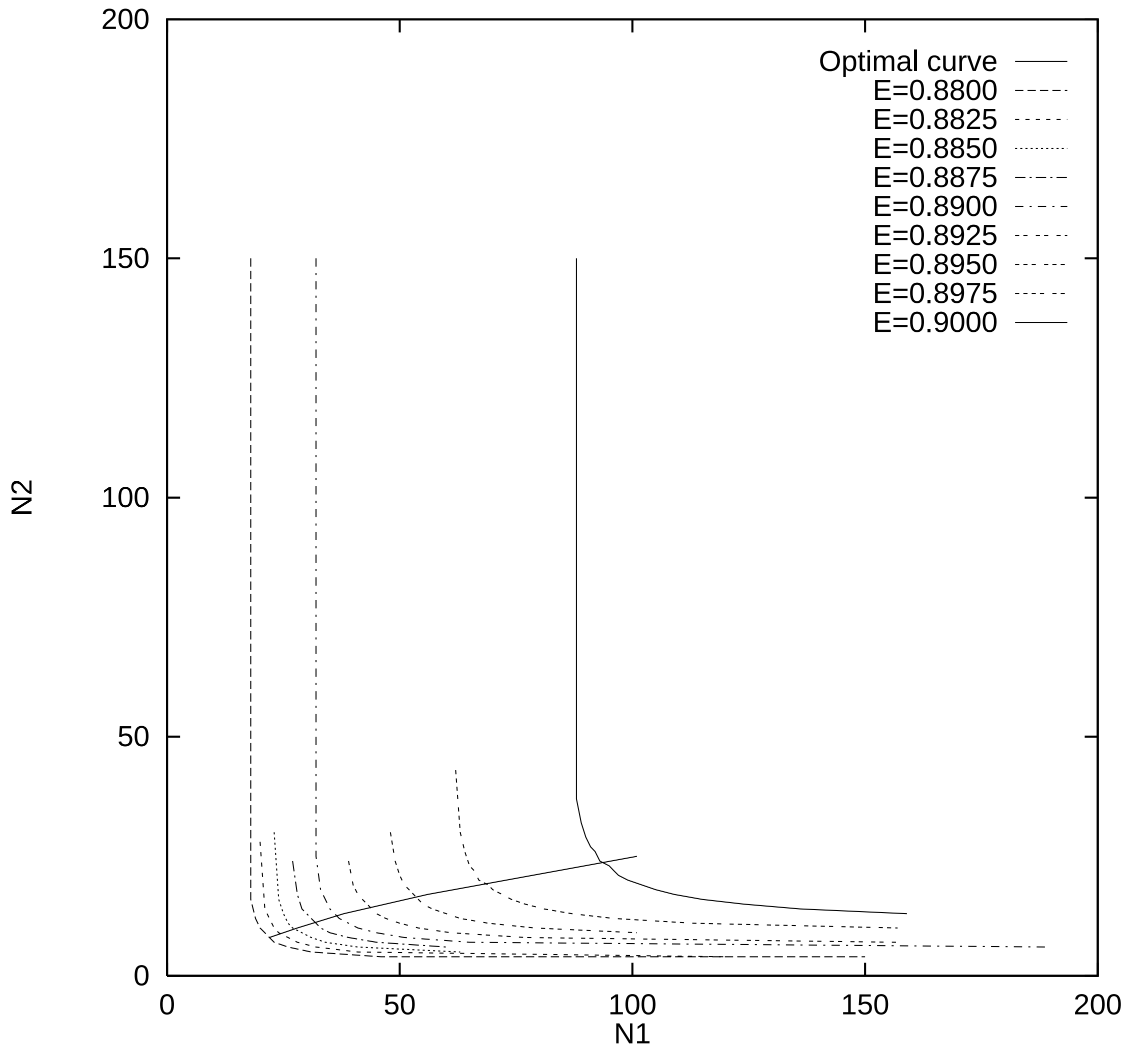

Figure 2-5: Iso-P curves for an three-machine line

are optimal for some P^*. The optimal curve is perpendicular to the ISO-P curves and appears to be continuous. This observation, and our ability to solve the dual problem, motivates the construction of the primal algorithm.

Figure 2-6 is a graph of the optimal curve and the Iso-P curves for a balanced three-machine line. A balanced transfer line is a transfer line for which all of the machines are identical. For this case the optimal curve is a straight line. The more balanced the transfer line, the straighter the optimal curve.

Machine	r_i	p_i
1	0.1	0.01
2	0.1	0.01
3	0.1	0.01

Table 2.2: Three-machine line parameters

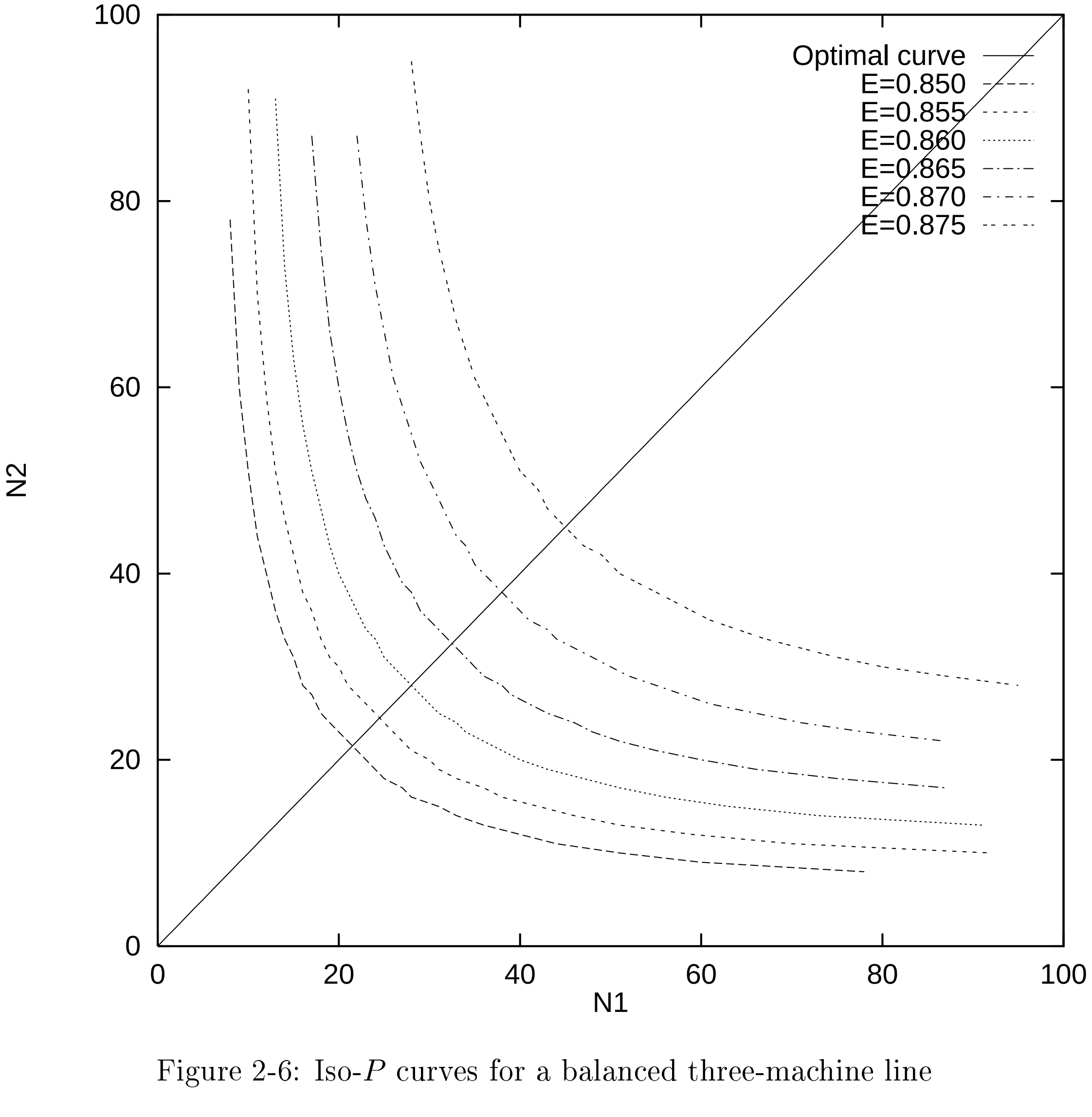

Figure 2-6: Iso-P curves for a balanced three-machine line

2.5.2 Optimality and uniqueness

Gradient methods guarantee that if a solution to the dual is found it will be optimal. This is a consequence of the concavity property of the objective function and is the reason we have been examining whether $P(N_1, \ldots, N_{k-1})$ is a concave function. In other words, if we locate a local maximum of $P(N_1, \ldots, N_{k-1})$ concavity guarantees that we have determined a global maximum.

Because we are still assuming $N_i \in R^+$, the solution to the dual is unique. We can determine the solution to the degree of precision that we require. When we reimpose the constraint $N_i \in Z^+$ the solution to this problem will be near the solution to the transformed problem. When $(N_1, \ldots, N_{k-1})$ is rounded the resultant vector should be optimal or very close to optimal.

After the integer constraint is added, the dual may no longer have a unique solution. An example of a dual formulation with a non-unique solution is a symmetric three-machine line with N^{total} odd. The solution satisfies either $N_1 = N_2 + 1$ or $N_2 = N_1 + 1$.

We relax the restriction $N_i \in Z^+$ to facilitate determination of the dual solution. Relaxing the restriction $N^{total} \in Z^+$ does not make the primal problem easier to solve. Therefore, we continue to require that total buffer space be discrete. Consequently, the solution to the primal problem is not necessarily unique. Multiple solutions may occur for two reasons. The first reason is that the solution to the dual problem for $N^{total} = N^*$ is not unique. The second reason is that, even if the dual solution is unique, a sub-optimal solution to the dual may still satisfy the primal constraint. The objective function of the primal does not require that we maximize $P(N_1, \ldots, N_{k-1})$.

2.6 Summary of assumptions

As stated in Section 1.3, considerable work has been done to demonstrate that systems similar to the transfer line model we consider are both monotonic and concave. Many numerical experiments, including the examples we present in Section 2.4, and work done in systems similar to ours, support the claim that throughput is a concave function for our models as well. Therefore we have based the algorithms developed in Chapters 3 and 5 on the following properties:

1. $P(N_1, \ldots, N_{k-1})$ is concave and monotonically increasing in each N_i. Further, if N_j is fixed for $j \neq i$, $\lim_{N_i \to \infty} P(N_1, \ldots, N_{k-1})$ is finite.

2. $P_{max}(N^{total})$ is monotonically increasing in N^{total} and $\lim_{N^{total} \to \infty} P_{max}(N^{total})$ is finite.

3. When the deterministic processing time model is used a function $\hat{P}(Y_1, \ldots, Y_{k-1})$ may be constructed with the following properties. $\hat{P}(Y_1, \ldots, Y_{k-1})$ is a differentiable, continuous, monotonic, bounded, concave function of continuous variables and satisfies (2.4). This construction is not necessary when using the continuous material flow model because it is assumed that $P(N_1, \ldots, N_{k-1})$ has these properties. In the rest of this thesis, we do not distinguish between $\hat{P}(Y_1, \ldots, Y_{k-1})$ and $P(N_1, \ldots, N_{k-1})$ when the deterministic processing time model is used. Consequently, we use the same notation for both transfer line models.

Chapter 3

Dual algorithm

The dual algorithm is a gradient algorithm (Luenberger 1984). It proceeds in a fashion similar to all gradient algorithms. First, a direction to move is determined. A linear search is then conducted in that direction until a maximum is encountered. A new direction is chosen and the process continues until no improvement is realized. The issues are how to determine a direction to search, what line search algorithm to use, and when to terminate the search. Figure 3-1 diagrams the important steps in the dual algorithm. The thesis section that describes each step is indicated.

3.1 Definition of the gradient $\mathbf{g}$

The goal of the dual is to maximize P. Therefore the first step is to determine $\mathbf{g}$, where $g_i = \frac{\partial P}{dN_i}(N_1, \ldots, N_{k-1})$. This is the direction in which P increases the most. Since we are unable to determine the gradient analytically we must approximate the gradient numerically.

The formula we use to calculate the gradient depends on which problem we are solving and the value of $P(N_1, \ldots, N_{k-1})$ for our most recent guess. The two formulas we use to calculate $\mathbf{g}$ are:

$$g_i = \frac{P(N_1, \ldots, N_i + \delta N, \ldots, N_{k-1}) - P(N_1, \ldots, N_i, \ldots, N_{k-1})}{\delta N} \tag{3.1}$$

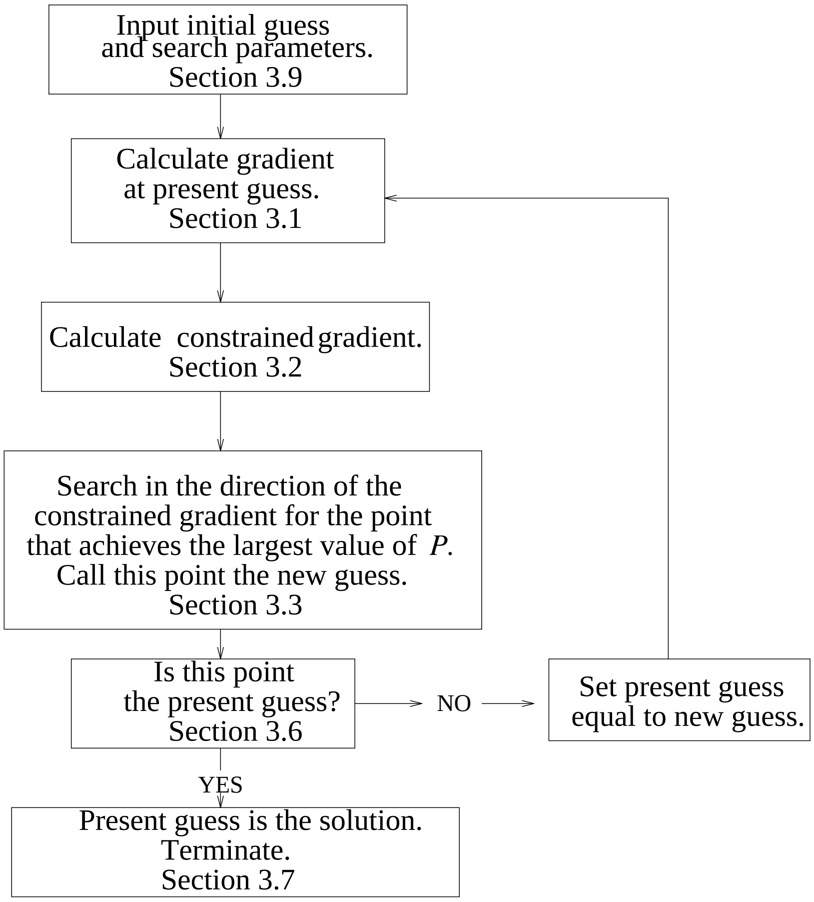

Figure 3-1: Block diagram of the dual algorithm

$$g_i = \frac{P(N_1, \ldots, N_i, \ldots, N_{k-1}) - P(N_1, \ldots, N_i - \delta N, \ldots, N_{k-1})}{\delta N} \qquad (3.2)$$

When the dual is called by the primal use (3.1) when $P(N_1, \ldots, N_{k-1}) < P^*$ and (3.2) when $P(N_1, \ldots, N_{k-1}) \geq P^*$. Calculating the gradient in this way takes the difference in the direction that $\mathbf{N}$ will eventually move. This is important in regions where $P(N_1, \ldots, N_{k-1})$ is nearly flat, because in this region the two methods do not produce the same answers. When the goal is only to solve the dual problem, use (3.1).

The size of δN must be chosen. If δN is too small the algorithm will be unable to distinguish between the change in P and the noise from the evaluation routine. We get good results with $0.1 \leq \delta N \leq 1$.

3.2 The constrained gradient p

Once we calculate the gradient, we know the direction in which the increase in P is the largest. The difficulty we face is that is we are unable to move (i.e. select $\delta N_i = \hat{N}_i - N_i$ where N_i is the most recent guess of the dual solution and $\hat{N}_i$ is the next guess) in the gradient direction and still remain feasible. Monotonicity guarantees that $g_i \geq 0 \; \forall \; i$. Therefore, all points in the direction of the gradient are infeasible. The remedy is to move in the direction of the vector $\mathbf{p}$, that is closest to $\mathbf{g}$, but still satisfies the total buffer space constraint. We construct $\mathbf{p}$ by projecting $\mathbf{g}$ onto the hyperplane $\sum_{i=1}^{k-1} \delta N_i = 0$. This is equivalent to projecting $\mathbf{g}$ onto the feasible region for the dual problem. The formulas for calculating the components of $\mathbf{p}$ are:

$$\begin{aligned} \bar{g} &= \frac{1}{N-1} \sum_{i=1}^{k-1} g_i \\ p_i &= g_i - \bar{g} \end{aligned} \qquad (3.3)$$

There are some cases when the current guess contains an N_q that is on the boundary of the feasible region. This occurs when $N_q = 0$. If $p_q \leq 0$ and N_q is on the boundary we are unable to move and remain feasible. When this occurs we need to calculate a new direction. We calculate the new direction by deleting the q_{th} component of $\mathbf{g}$ and setting $p_q = 0$. Then we recalculate the other components of $\mathbf{p}$ using (3.3), and the new $\mathbf{g}$. This process is continued until a feasible step may be taken or all components of $\mathbf{g}$ are deleted.

3.3 Line search method

We design the dual algorithm, and the line search procedure employed by this algorithm, to execute efficiently when invoked to solve a subproblem of the primal. We adopt this criterion because each primal problem will require solving many instances of the dual.

The goal of the line search is to find α such that $P(\mathbf{N^{start}} + \alpha)$ is maximized. Here $\mathbf{N^{start}}$ is the point at which the search is started and α is a scalar such that $\alpha \geq 0$ $\alpha \in R^+$. The line search procedure we construct is based on the following observation. When the dual algorithm is invoked as a procedure in the primal algorithm, the initial guess, provided as input to the dual, is generally close to the solution (see Section 5.1.1). The degree to which the transfer line is balanced will determine how good the initial guess will be. The only time the initial dual guess may not be close to the solution to the dual is the the first time the dual algorithm is invoked by the primal. The initial primal guess, which is also the initial dual guess for the first instance of the dual, is input by the user in the implementation of the primal algorithm developed for this thesis.

$[\mathbf{N^{start}}, \mathbf{N^{start}} + \alpha_{max}\mathbf{p}]$ is the line segment that is searched. We choose α_{max} so that all possible points in this interval are feasible. By enforcing the constraint that $\mathbf{N^{start}} + \alpha\mathbf{p} \geq 0$, which is equivalent to requiring that all candidate buffer distributions have non-negative buffer sizes.

Denote the minimum step β where $\beta > 0$ and $\beta \in R^+$. The method for determining β is described in Section 3.4. β is used to determine the degree of precision for which we calculate a dual solution. The fact that the maximum will generally occur near the starting guess suggests initially considering points that are close to $\mathbf{N^{start}}$. We take advantage of this fact when constructing the line search method. We search the line segment by evaluating the points $\mathbf{N^{start}} + 2^{l-1}\beta p$, for $l = 1, 2, \ldots$, until either of the following conditions occur:

case 1: $P(\mathbf{N^{start}} + 2^{l-1}\beta\mathbf{p}) > P(\mathbf{N^{start}} + 2^{l}\beta\mathbf{p})$ and $\alpha_{max} > 2^l\beta$.

case 2: $\alpha_{max} < 2^l\beta$.

Let $\mathbf{N^{start}} + 2^l\beta\mathbf{p} = \mathbf{N^l}$. If case 1 occurs, concavity ensures that the $\mathbf{N}$ that maximizes P must be contained in the line segment bounded by $\mathbf{N^{l-2}}$ and $\mathbf{N^l}$. If case 2 occurs the $\mathbf{N}$ that maximizes P is on the line segment bounded by $\mathbf{N^{l-2}}$ and $\mathbf{N^{start}} + \alpha_{max}\mathbf{p}$. This resulting line segment is searched using a binary search method. The termination conditions for the search are described in Section 3.5.

3.4 Step size

This section describes the procedure for calculating β and α_{max}. α_{max} determines the length of the line segment we are searching. β determines the step size used in the early stages of the line search, and the width of the interval in which the solution is contained.[1] The formulas used to calculate α_{max} and β are:

$$\alpha_{max} = \text{maximum } \alpha \text{ such that } N_i^{start} + \alpha p_i \geq 0 \; \forall \; i \tag{3.4}$$

$$\beta = \text{minimum } s/p_i \; \forall \; i \tag{3.5}$$

The value for s is input by the user. The default value, in our implementation, is $s = 1$. We have found experimentally that this value works well in the cases we have tried. This value for s guarantees that one component of $\mathbf{N}$ will increase by 1 and all other components will increase by less than 1.

[1] The interval which contains the solution is often refered to as the *interval of uncertainty* (Luenberger 1984).

In our implementation the user may reduce the interval of uncertainty by specifying s. If s is too small, the evaluation procedure will spend an excessive amount of time refining the answer or will be unable to differentiate between consecutive points. We have found that for most of the cases we study a reasonable lower bound on s is 0.25. When the transfer line is very long a smaller s may be warranted.

We have observed that the size of s impacts the final allocation of buffer space if N^{total} is large. Total buffer space, which is calculated by the primal algorithm, is less sensitive to s.

3.5 Terminating the line search

The line search terminates if either of two conditions are satisfied. The primary condition for termination is that the interval of uncertainty is less than the minimum step size. This occurs when the distance between two successive guesses is less than β.

To describe the second condition we need to introduce some additional notation. Designate the production rate of the j^{th} guess as P^j. If (a) the distance between guesses $j-1$ and j is less than β/s but greater than β and (b) $|P^j - P^{j-1}| \leq \epsilon$, where $\epsilon = 10^{-8}$, then the search also terminates. The reason for a second criterion is that when the transfer line consists of many machines and the total buffer space is large, the ADDX and DDX algorithms may have difficulty evaluating the difference in production rate between two similar, but not identical, buffer distributions. At this point the candidate distributions are practically equally good and there is no reason to continue the search.

3.6 Terminating the dual algorithm

The criterion for termination of the dual algorithm is

$$P(\mathbf{N^{start}}) \geq P(\mathbf{N^{start}} + \beta\mathbf{p}) \tag{3.6}$$

When (3.6) is satisfied $\mathbf{N^{start}}$, an endpoint of the line segment, achieves a higher production rate than $\mathbf{N^{start}}+\beta\mathbf{p}$ which is the next point that can be considered. Since $P(\mathbf{N^{start}} + \alpha\mathbf{p})$ is concave, the maximum production rate occurs in $[\mathbf{N^{start}}, \mathbf{N^{start}} + \beta\mathbf{p}]$. Therefore we have reached a maximum among the points we can consider. We are essentially unable to improve on the current guess.

A common criterion for termination in maximization problems is to stop when all components of the gradient vector $\mathbf{g}$ are equal (other than those that correspond to buffers whose size is zero). If this occurs $\mathbf{p} = 0$ and (3.6) is satisfied.

3.7 Rounding methodology

When the dual problem is solved as a stand-alone problem and the deterministic processing time model is used, the final values for N_i must be integer. Consequently, the last step in the dual algorithm is to round.

In addition, in our implementation the final solution is rounded even when the continuous material flow model is employed. For this case rounding should be considered as post processing and not as part of the algorithm since an integer solution is not required.

Before the rounding step the solution to the dual problem is the vector $\mathbf{N}^*$. The components of $\mathbf{N}^*$ are real numbers. At this time we need to impose the constraint $N_i^* \in Z^+$ without changing the total buffer space. In our implementation, for reasons of convenience, the total buffer space is always integer so the value of N^{total} specified in the dual is always integer. The method we use to round $\mathbf{N}^*$ is described in the remainder of this section.

First, rank order the buffers according to each of the criteria outlined in the following paragraph.

Rounding Methods

1. Minimum Euclidean distance

Let $D_i = N_i - \lfloor N_i \rfloor$ for each i. The components are ranked in order of increasing D_i.

2. Gradient

   The components are ranked in order of increasing g_i. We are rounding in the direction of the gradient.

3. Weighted gradient

   We use $D_i g_i$ to rank the buffers.

We construct a one to one function $R(i) = l$ where $1 \leq l \leq k-1$ for each of the three rounding methods. This function calculates the rank of each buffer i. For example, when $\mathbf{N}^*$ is being rounded in the direction of the gradient, and g_3 is the largest component of $\mathbf{g}$, then $R(3) = k-1$. If g_5 is the smallest component of $\mathbf{g}$, then $R(5) = 1$. The following steps are then performed to round $\mathbf{N}^*$.

1. Let $\mathbf{N^{rnd}} = \mathbf{N}^*$.

2. Choose i such that $R(i) = k-1$ and round N_i^{rnd} up. Each time a component is rounded the new vector is called $\mathbf{N^{rnd}}$. Let $first = k-2$ and $last = 1$.

3. Let $direction = N^{rnd} - N^*$ where $N^{rnd} = \sum_{i=1}^{k-1} N_i^{rnd}$ and $N^* = \sum_{i=1}^{k-1} N_i^*$.

4. If $direction \geq 0$, choose i such that $R(i) = last$. Round N_i^{rnd} down, and let $last = last + 1$.

5. If $direction < 0$, choose i such that $R(i) = first$. Round N_i^{rnd} up, and let $first = first - 1$.

6. If $last > first$ terminate.

7. Go to step 2.

When all components of $\mathbf{N}^*$ are rounded $N^{rnd} = N^*$.

An integer vector is calculated for each of the three different rounding criteria. The integer vector with the largest production rate is chosen.

There is an option to continue searching after rounding because, in cases where k is large, this rounding procedure may not find the optimal distribution. In the numerical chapters (4 and 6) we refer to this option as post-processing. The search procedure is similar to the method used in the later stages of the Gershwin and Goldis (1995) algorithm. The buffer with the largest gradient is incremented by 1 and the buffer with the smallest gradient is decremented by 1. This is continued until no improvement is realized.

This procedure does not always find the optimum because it limits the search space to points that may only be reached by increasing or decreasing a single buffer by 1. Therefore, a point that may only be reached by simultaneously exchanging two or more buffers may not be evaluated. However, this procedure will always find a solution that is very close to optimal.

3.8 Pseudo code for dual algorithm

In this section the dual algorithm is described using pseudo code.

1. Retrieve input and initialize the variables.

2. Calculate $\mathbf{g}$ and $\mathbf{p}$ for $\mathbf{N^{start}}$ using (3.1) and (3.3).

3. Calculate α_{max} and β using (3.4) and (3.5).

4. If (3.6), the termination condition, is satisfied go to step 7. Otherwise conduct a linear search as outlined in Section 3.3.

5. Let $\mathbf{N^{start}} = \mathbf{N^{current}}$, where $\mathbf{N^{current}}$ is the solution found by the linear search procedure.

6. Go to step 2.

7. If we are solving a dual instance of the primal problem return $\mathbf{N^{current}}$ as the solution to this instance of the dual. If we are solving the dual problem, as a stand-alone problem, round $\mathbf{N^{current}}$ using the method described in Section 3.7.

3.9 Implementation issues

Non-reinitialization of the DDX and ADDX We have chosen to modify the initialization procedure of the DDX and ADDX algorithms to improve performance. Instead of using the standard initialization procedure, start the algorithm with the final values from the last long line evaluation. we use the final value from the last evaluation. This decreases the number of iterations required for the algorithms to converge. For an in-depth discussion of why this improves performance see Gershwin and Goldis (1995).

User specified inputs The precision of the solution is determined by the step size, which may be modified by the user. The step size may be changed by changing the parameter s in (3.5). The default value is $s = 1$. This value guarantees that the interval of uncertainty is such that one N_i changes by 1 and all other components of $\mathbf{N}$ change by one or less.

The user must also supply an initial guess, $\mathbf{N^{start}}$.

Post processing There is an option to do additional searching, to improve the solution, after the rounding step. This option is controlled by the user.

Minimum buffer sizes If the deterministic long line processing time model is used the minimum buffer size permitted is 4. Therefore all constraints and equations that assume $N_i \geq 0$ must be modified to guarantee that $N_i \geq 4$. If the continuous material flow model is used, no constraints or equations need to be modified.

Chapter 4

Behavior of the dual algorithm

In this section we present examples to demonstrate that the dual algorithm is a practical line design tool. The total buffer space required, suggested buffer distributions, and execution times are reported. We report execution times as the number of two-machine or long-line evaluations required for the algorithm to converge. This is a reasonable measure to use for comparison because it should be unaffected by the computer used to perform the experiment. We begin this section with a discussion of the accuracy of the dual solutions.

4.1 Accuracy

We use the results of the combined algorithm of Gershwin and Goldis (1995) to verify the accuracy of both the dual and primal algorithms. Gershwin and Goldis establish the accuracy of their solutions by comparing the output of their algorithm to the results of an exhaustive search. The search was performed for several five-machine lines and individual buffer sizes of up to 20.

We claim that our solutions are either optimal or very close to optimal in all of the cases we study. The accuracy of our solution is impacted by the the decision to do post-processing (Section 3.7) after rounding, the value of s (Section 3.4), and rounding methodology (Section 3.7). Post-processing has marginally improved the final solution in certain cases. The longer the line the greater the likelihood that post-processing will help. The cost of the increased accuracy, attained with post processing, is the additional processing time required.

If post-processing is not done, the optimality of the solution is impacted by the step size, β, and rounding methodology. Choice of a smaller step size will usually improve the intermediate solution. The intermediate solution is the final buffer distribution before the rounding step. Because buffer sizes are continuous variables, a reduced step size will decrease the interval of uncertainty in which the answer lies. Therefore, a better intermediate solution will be found. The final solution, which is the rounded intermediate solution, is not guaranteed to improve because two different intermediate solutions may round to the same final solution. Since smaller step sizes require additional processing there is a cost to reducing the step size. See Section 3.4 for a discussion of step size.

When we round (Section 3.7) only three discrete points in the neighborhood of the non-integer solution are considered. The number of feasible candidates may be very large, especially when k is large. Therefore, it is not surprising that we do not always round to the optimal integer solution.

4.2 Dual examples

In this section we report execution times and buffer distributions for the dual algorithm. We do all examples using the deterministic processing time model. Examples are also done with and without post-processing to demonstrate both the benefit and cost of the additional optimization.

4.2.1 Example 1

Table 4.1 reports the buffer allocation and execution time for a balanced ten-machine line. The line consists of identical machines with $r_i = .095$ and $p_i = .007$. The constraint is $N^{total} = 346$. Cases 1 and 2 started with an initial guess of $\mathbf{N} = (266, 10, 10, 10, 10, 10, 10, 10, 10)$. Case 2 did additional processing in an attempt to improve the solution. In this case we were unable to improve the solution. Cases 3 and 4 started with an initial guess of $\mathbf{N} = (39, 39, 39, 39, 38, 38, 38, 38, 38)$. Case 4 also did additional processing after rounding. The post-processing step improved P from 0.880090 to 0.880098, a negligible amount in practical cases, by reallocating one buffer space.

Case 1 required twice as many two machines evaluations as case 3 but this is not surprising given the difference in initial guesses. Neither the solution nor the execution time appear to be overly sensitive to a bad initial guess. The solution reached in cases 1 and 3 would be identical if one space was switched from buffer 1 to buffer 9. The additional optimization after the rounding step increased the execution time by approximately 10% in case 1 and 50% in case 2. For these cases additional processing appears unwarranted.

	Buffer i									Two-machine	
Case	1	2	3	4	5	6	7	8	9	evaluations	P
1	26	38	43	44	44	44	42	39	26	103,776	0.880098
2	26	38	43	44	44	44	42	39	26	112,288	0.880098
3	27	38	43	44	44	44	42	38	26	53,456	0.880090
4	26	38	43	44	44	44	42	39	26	70,416	0.880098

Table 4.1: Buffer allocation for a balanced ten-machine line

4.2.2 Example 2

Table 4.3 reports buffer allocation and execution time for the line studied by Park and described in Table 4.2. Park reports a solution for the primal problem for this line. Our solution to the primal problem is compared with his results in Section 6.3. Park does not solve the dual problem.

The constraint is $N^{total} = 243$. Cases 1 and 2 started with an initial guess of $\mathbf{N} = (143, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10)$. Cases 3 and 4 started with an initial guess of $\mathbf{N} = (23, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22)$. Cases 2 and 4 did additional processing to optimize the solution. For all cases $s = 4$.

The solution reached in cases 1 and 3 would be identical if one space was switched from buffer 2 to buffer 10. The production rates in all cases are equal. The additional optimization after the rounding step increased the execution time by approximately

Machine	1	2	3	4	5	6	7	8	9	10	11	12
r_i	0.35	0.15	0.4	0.4	0.3	0.2	0.3	0.3	0.4	0.4	0.3	0.25
p_i	0.037	0.015	0.02	0.03	0.03	0.01	0.02	0.02	0.02	0.03	0.03	0.01

Table 4.2: Twelve-machine line (Park 1993)

16% in case 1 and 50% in case 2 without improving the solutions. Post-processing is unwarranted in these cases.

	Buffer i											Two-machine	
Case	1	2	3	4	5	6	7	8	9	10	11	evaluations	P
1	58	26	21	25	23	16	15	13	14	20	12	596,360	0.8951
2	58	27	21	25	23	16	15	12	14	20	12	693,760	0.8951
3	58	27	21	25	23	16	15	13	14	19	12	307,380	0.8951
3	58	27	21	25	23	16	15	12	14	20	12	461,470	0.8951

Table 4.3: Buffer allocation for a twelve-machine line

4.3 Comparison with the literature

In this section we compare our results with those of Ho, Eyler, and Chien (1979). The "optimal allocation algorithm" Ho *et al.* propose is an iterative algorithm which differs from ours in the way production rates and gradients are calculated. They use simulation, not analytical methods, to calculate production rates. They also have a novel method of evaluating gradients that enables them to determine the production rate and gradient by simulating one transfer line. Previously, simulating k different transfer lines was necessary to calculate the gradient.

They construct each successive guess in the following way. $\mathbf{p}$ is multiplied by an empirically determined step size and added to the previous guess of $\mathbf{N}$. Simulation is then used to estimate $P(N_1, \ldots, N_{k-1})$ and $\mathbf{g}$ for each guess. When no improvement is realized the algorithm stops.

Our results differ from those of Ho *et al.* This is due, in part, to disparities between the calculations of $P(N_1, \ldots, N_{k-1})$ and g_i performed by their simulation and the DDX algorithm. Differences may also be a consequence of the precision of their solution. The major contribution of the dual algorithm is not in the novelty of the solution

Machine	1	2	3	4	5
MTTR	11	19	12	7	7
MTTF	20	167	22	22	26

Table 4.4: Five-machine line (Ho *et al.* 1993)

technique but in the reduced computational effort required to determine a solution. The reduced effort is a consequence of using the DDX and ADDX algorithms and this reduction in computational effort allows us to use the dual algorithm as a procedure in solving the primal problem as well as the profit maximization problem we present in Chapter 8.

4.3.1 Example 3

This example uses the deterministic processing time model. We are solving for the optimal buffer allocation for the five-machine line described in Table 4.4. The MTTR and MTTF, rather than corresponding probabilities, are specified because Ho *et al.* use these values to describe the transfer line. The final buffer distributions are presented in Table 4.5. The production rates for both buffer distributions are calculated using the DDX algorithm and simulation. The simulation was run for 50 runs of 100,000 cycles each. These values were chosen because they are similar to the 50 runs of 50,000 parts Ho *et al.* use.

To simulate our final solution requires more than 5 minutes. To run the dual algorithm requires 0.3 seconds.[1]

	Buffer i						
Case	1	2	3	4	N^{total}	DDX production rate	Simulation production rate
Ho *et al.*	5	11	8	7	31	0.4914	0.4931
Dual	7	10	10	4	31	0.4943	0.4962

Table 4.5: Buffer allocation for a five-machine line

[1]This experiment was performed on a Sun Sparc station 2.

Machine	1	2	3	4	5	6	7
MTTR	450	760	460	270	270	650	320
MTTF	820	5700	870	830	970	1900	1100
Cycle time	40	34	39	38	37	40	43

Table 4.6: Seven-machine line (Ho *et al.* 1979)

4.3.2 Example 4

In this example, which uses the continuous material flow model, we compare our dual solution to results reported by Ho *et al.* The individual machines have different processing rates so we cannot use the deterministic processing time model. We are seeking the optimal buffer allocation for the seven-machine line described in Table 4.6. Notice that the production rate is less than 0.02. This is a consequence of the machine cycle times which are all on the order of 40. The final buffer distributions are presented in Table 4.7.

	Buffer i							
Case	1	2	3	4	5	6	N^{total}	ADDX production rate
Ho *et al.*	5	11	8	7	19	4	54	0.0126
Dual	8	10	13	10	9	4	54	0.0128

Table 4.7: Buffer allocation for a seven-machine line

The production rates reported in the last column of Table 4.7 are calculated using the ADDX algorithm. The buffer distribution calculated by the dual algorithm achieves a production rate that is approximately 2% larger than the distribution calculated by Ho *et al.*

Chapter 5

Primal algorithm

In this chapter we develop an algorithm to solve the primal problem as stated in (2.1). The algorithm we construct is an iterative algorithm based on gradient methods. It is based on the assumption that $P(N_1, \ldots, N_{k-1})$ exhibits the properties postulated in Section 2.6. We also assume that we are able to evaluate $P(N_1, \ldots, N_{k-1})$.

The dual algorithm calculates a projected gradient, $\mathbf{p}$ and searches in that direction. The linear constraint (2.3) allows us to guarantee that all guesses are feasible. Because the constraint is linear, we are able to use an efficient search method to determine a solution. We need a different approach to solve the primal problem. A block diagram of the procedure we employ to solve the primal problem is shown in Figure 5-1. The looping step is performed in cases where the solution may have been found but has not yet been verified. We do not evaluate $P_{max}(N^j + 1)$ or $P_{max}(N^j - 1)$ for each guess. In our implementation the termination condition is satisfied when the length of the interval of uncertainty has been reduced to 1. We explain the termination condition in Sections 5.1.1 and 5.1.3.

In the primal problem the constraint is not only non-linear, but we are unable to express the constraint in closed form. Therefore, even when we are able to determine a direction to move, we are not easily able to determine how far we should move.

In addition, even if we know a point on this surface $P(N_1, \ldots, N_{k-1}) = P^*$ we are unable to construct the next guess easily so that it will also be on the surface. For this reason, the issue of how to develop an efficient algorithm to search the function

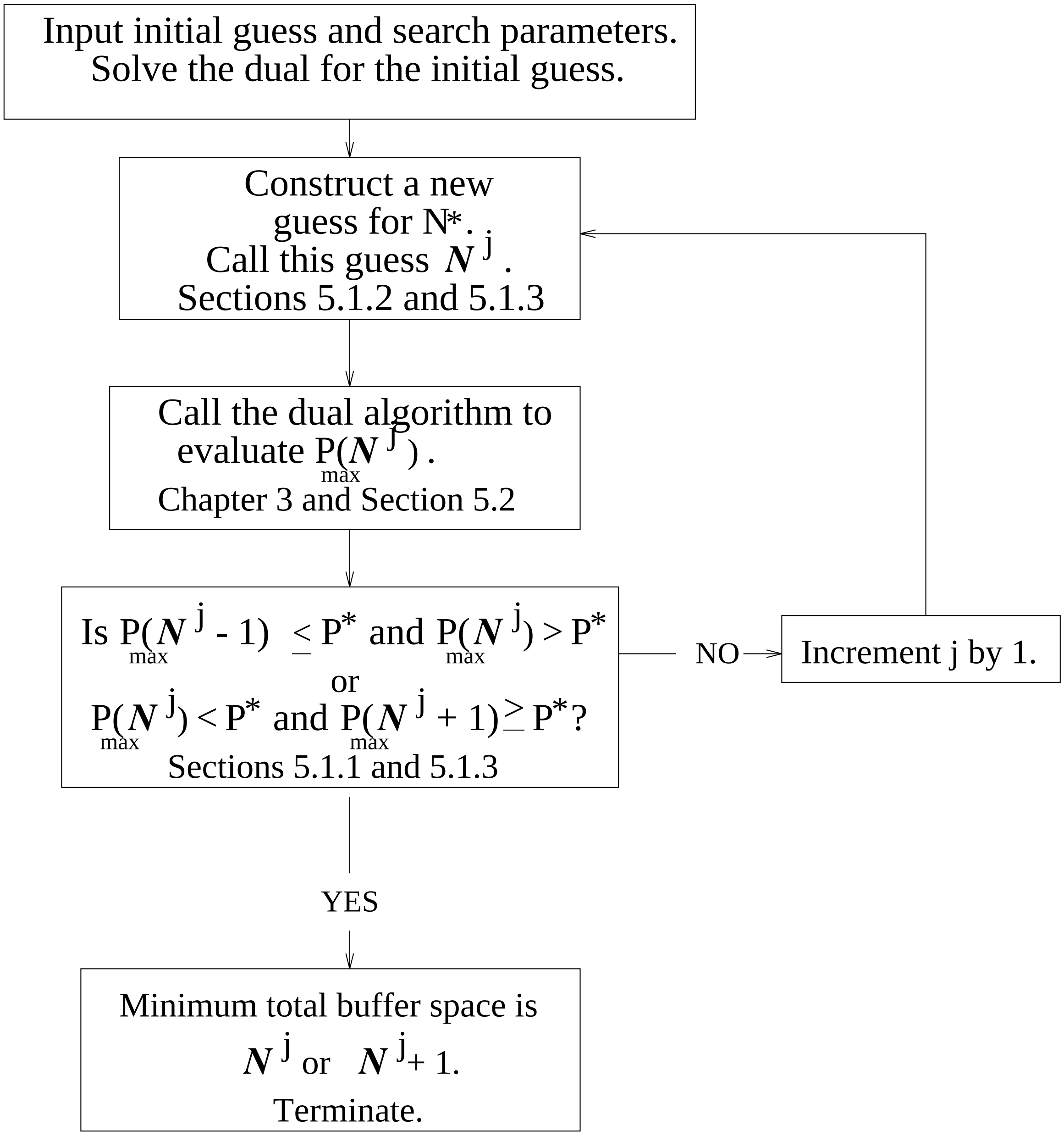

Figure 5-1: Block diagram of the primal algorithm

$P(N_1, \ldots, N_{k-1}) = P^*$ for a minimum N^{total} is problematic. We illustrate these difficulties with the help of the three-machine system described in Table 2.1. Figure 5-2 is a graph of Iso-P lines for this system. This figure is a modification of Figure 2-5. We reduced the number of Iso-P curves shown, included a portion of the feasible region for a dual formulation with $N^{total} = 100$, and included an arrow that points in the direction of $\mathbf{g}$ calculated at the point $(50, 50)$.

Assume that $P^* = 0.90$ and the initial guess is $(50, 50)$. The first step is to calculate $\mathbf{g}$. We then move in the direction of $\mathbf{g}$. We are confronted with the following question. How far should we move until we recalculate the gradient? If we move until we encounter the surface $P(N_1, \ldots, N_{k-1}) = P^*$ we may be far from the optimum (which is the case in Figure 5-2). Once we have reached the surface $P(N_1, \ldots, N_{k-1}) = P^*$ how do we remain there and move closer to a solution? Unfortunately, the only way we know of searching this surface is by taking small steps which usually takes an inordinate amount of computational effort.

An alternative is to calculate gradients more frequently, which is essentially what Gershwin and Goldis (1995) do. They develop a gradient algorithm that uses the gradient to determine which buffer to change and a multiple of the difference between the production rate achieved by the current guess and P^*, to calculate how much to change the buffer. The amount the buffer is changed corresponds to determining how far to move before recalculating a gradient. The differing distances between the three Iso-P curves in Figure 5-2 demonstrates why this approach can not work well in all cases. To increase the production rate from 0.880 to 0.890 requires approximately ten additional buffer spaces. To increase the production rate from 0.890 to 0.900 requires approximately sixty additional buffer spaces. Moving, or taking a step, is equivalent to changing the buffer sizes. Therefore, it is likely that a fixed step size will be too large in some cases and too small in others. In cases where the step size is too small, an excessive amount of time will be spent calculating gradients. If the step size is too large the surface $P(N_1, \ldots, N_{k-1}) = P^*$ may be encountered at a point that is far from the optimal point.

By using the dual algorithm as part of the solution of the primal problem we

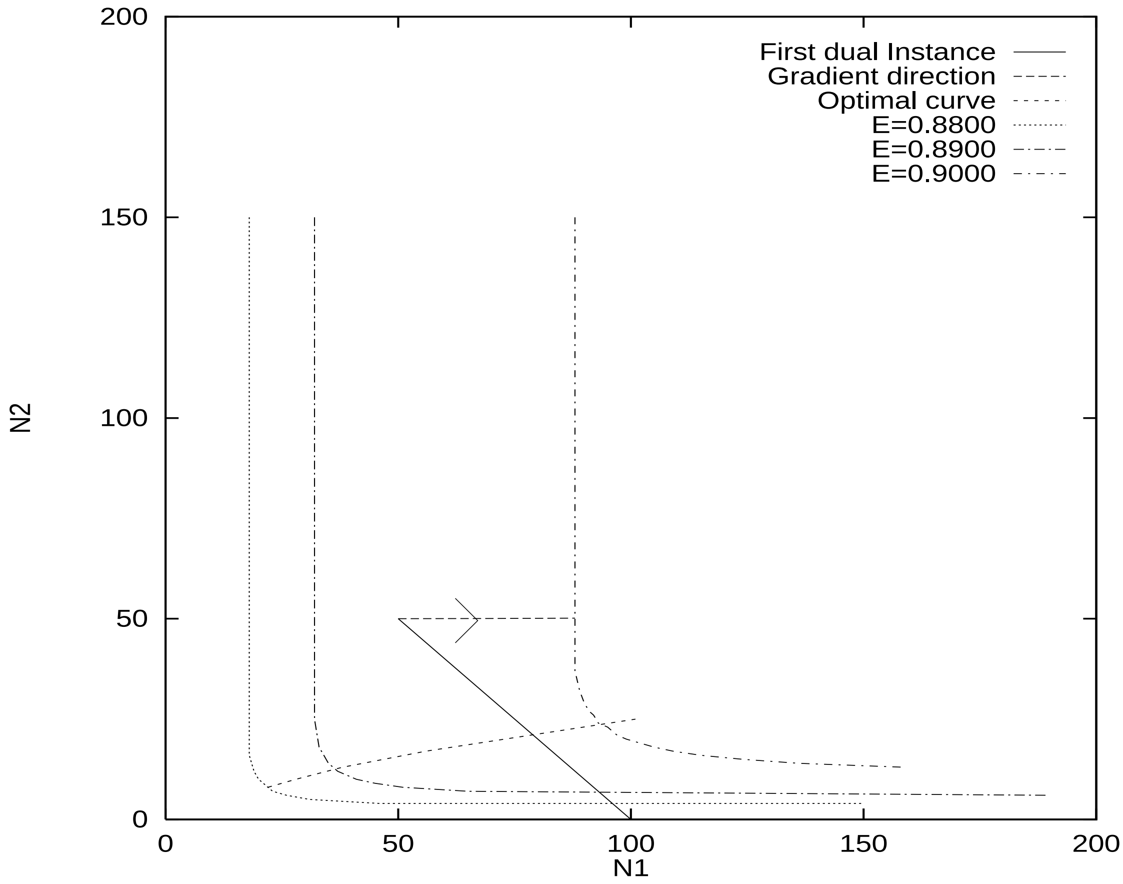

Figure 5-2: Iso-P lines for an unbalanced three-machine line

have avoided the necessity of developing a systematic method for searching the surface $P(N_1, \ldots, N_{k-1}) = P^*$ or determining a step size. We instead solve the primal problem by dividing it into two subproblems that we know how to solve. These two subproblems are the dual and ODPP. They are formulated in (2.2) and (2.5) respectively.

The primal problem is solved by searching the curve $P_{max}(N^{total})$. The dual algorithm is invoked to perform the evaluation step of this search. In principle, any algorithm that correctly evaluates $P_{max}(N^{total})$ can be called in place of the dual algorithm described in Chapter 3. Though the correctness of the primal solution is independent of the evaluation procedure, the computational effort required depends on the evaluation method. The linear search procedure, in the dual algorithm, will perform better when provided with an initial guess that is close to optimal. We use information about how the primal algorithm traverses the search space when designing the linear search routine in the dual algorithm. Specifically, we have taken advantage of the fact that, after solving for $P_{max}(N^1)$, $\mathbf{N^{initial}}$ will be close to the optimal curve. We discuss why $\mathbf{N^{initial}}$ is close to the dual solution in Section 5.1.1.

5.1 One dimensional primal problem (ODPP)

The solution to the ODPP may be found by performing a one dimensional linear search. The linear search method we develop is based on the property that $P_{max}(N^{total})$ can be evaluated and is monotonically increasing in N^{total}. We show in Section 2.5 that $P_{max}(N^{total})$ has this property . The dual algorithm not only evaluates $P_{max}(N^{total})$, but determines the buffer distribution that achieves $P_{max}(N^{total})$.

The goal is to find the minimum value of N^{total} such that $P_{max}(N^{total}) \geq P^*$. This value of N^{total} is N^*. The following paragraph is a preview of how we solve for N^*.

5.1.1 Primal overview

The algorithm requires an initial guess. Call it N^1. We evaluate $P_{max}(N^1)$ and $P_{max}(0)$, where $P_{max}(0)$ is the production rate of the line when all buffers are of size zero. $P_{max}(N^1)$ is evaluated by the dual algorithm and $P_{max}(0)$ is evaluated using the procedure outlined in Section 5.4. Using these two points we determine m and b to construct the line $N = mP + b$, then use this line to estimate N^*. Let this estimate of N^* be denoted N^j, where $j = 2$ and $N^j = mP^* + b$.[1] $P_{max}(N^j)$ is evaluated, j is incremented and then we iterate. Each iteration uses N^j and N^{j-1} to construct m and b for the next version of the line. The process continues until one of following three conditions is satisfied.

$$P_{max}(N^j) \geq P^* \quad \text{and } P_{max}(N^{j-1}) < P^* \quad \text{and } N^j = N^{j-1} + 1 \tag{5.1}$$

$$P_{max}(N^j) \geq P^* \quad \text{and } P_{max}(N^j - 1) < P^* \quad \text{and } N^j > N^{j-1} + 1 \tag{5.2}$$

$$P_{max}(N^j) > P^* \quad \text{and } P_{max}(N^j - 1) > P^* \quad \text{and } N^j > N^{j-1} + 1 \tag{5.3}$$

If condition (5.1) is satisfied the algorithm terminates. If the dual problem were solved with infinite precision condition (5.3) would not be encountered. In practice, condition (5.3) is encountered. If conditions (5.2) or (5.3) are satisfied a binary search is performed, on the interval $[N^{j-1}, N^j]$, until N^* is determined. The reason (5.3) is unlikely in most cases is explained in Section 5.1.2. The cases for which it is likely that condition (5.3) occurs are described in Section 5.1.3.

The new guess for $\mathbf{N}^*$, called $\mathbf{N}^j$, is constructed by calculating $\mathbf{g}$ at the point $\mathbf{N}^{j-1}$ and moving in the direction of $\mathbf{g}$ until $\sum_{i=1}^{k-1} N_i = N^j$. The shape of the optimal curve in Figure 2-5, illustrates why our new guess will generally not be an exact optimal distribution. We approximate the optimal curve as a straight line in the direction of the gradient. We then restrict our next guess to this straight line. Therefore our next guess is usually close to, but not exactly, an optimal distribution. For that reason, the dual is solved for each estimate of N^*. The distance from the new guess to an optimal distribution is determined by the shape of the optimal curve.

[1] The estimate of N^* is a real number. This number is rounded because no benefit is gained by allowing N^{total} to be a continuous variable when the deterministic processing time model is used. If this number is rounded down it will no longer be feasible because $P_{max}(N^{total})$ is monotonically increasing. Therefore, the number is rounded up to satisfy the production rate constraint.

Figure 2-6 demonstrates why the computational effort required to solve the primal problem depends upon the curvature of the optimal curve. For this case the optimal curve is a straight line. The more balanced the transfer line, the straighter the optimal curve. Therefore, our new guess should be closer to the optimum when the line is balanced. Since line balancing is a common line design goal we describe how the primal algorithm traverses the search space for a balanced line in Section 6.2.

5.1.2 Linearization

The main issue in solving the ODPP is how to search $P_{max}(N^{total})$ for N^*. We are not able to express $P(N_1, \ldots, N_{k-1})$ or $P_{max}(N^{total})$ in closed form for $k > 2$. Therefore, we can not directly solve for N^*. This necessitates employing a linear search technique to determine N^*.

As stated in Section 5.1.1, the search technique we use constructs the guess N^j from the known quantities $N^{j-1}, N^{j-2}, P_{max}(N^{j-1}), P_{max}(N^{j-2})$. The new guess is determined by approximating the curve $P_{max}^{-1}(P)$ as a line. $P_{max}^{-1}(P)$, which is the inverse of $P_{max}(N^{total})$, gives the minimum total buffer space which achieves production rate P. Therefore, $P_{max}^{-1}(P^*) = N^*$. When $P_{max}(N^{j-1}) < P^*$ the line $N = mP + b$ is used to construct the j^{th} guess for N^*. The slope and intercept of this line and the new guess for N^* are calculated from (5.4).

$$\begin{aligned} N^j &= mP^* + b \qquad (5.4) \\ m &= (N^{j-1} - N^{j-2})/(P_{max}(N^{j-1}) - P_{max}(N^{j-2})) \\ b &= N^{j-1} - m * P_{max}(N^{j-1}) \end{aligned}$$

In our implementation N^1 is provided by the user and $N^0 = 0$. If $P_{max}(N^{j-1}) > P^*$ the method described in Section 5.1.3 is used to calculate N^j.

In Section 5.1.1 we state that if $N^1 < N^*$, and the dual is solved using infinite precision, all guesses satisfy $N^j \leq N^* \; \forall \; j$ (i.e. all guesses will underestimate N^*). In the next paragraphs we explain why this is true and why it is not a good idea to estimate N^* using (5.4) when $P_{max}(N^j) \geq P^*$. To facilitate the explanation we use the

Assume $N^{j-2} < N^{j-1} < N^*$ and the dual is solved using infinite precision. The curve $N = P_{max}^{-1}(P)$ is approximated by the line $N = mP + b$. Since $P_{max}^{-1}(P)$ is a convex function, $\frac{dP_{max}^{-1}(P)}{dP} > m$ for $P > P_{max}(N^{j-1})$. Consequently, $mP^* + b < P_{max}^{-1}(P^*)$ which says that $N^j < N^*$.

The graph in Figure 5-3 illustrates why (5.4) underestimates N^*. This is a graph of the function $P_{max}^{-1}(P)$ for the system described in Table 2.1 and Figure 2-4. Assume we are solving the primal problem for this system with $P^* = 0.89$ and $N^1 = 12$. The line segments included in Figure 5-3 are portions of the lines constructed from successive guesses for N^*, calculated using (5.4). Each new guess for N^* is the point where the line $P = 0.89$ and the line drawn through the previous two guesses intersect. It is impossible to construct a line, which passes through two points where $P < P^*$, that will intersect the line $P^* = 0.89$ at a point where $N > N^*$, because $P_{max}^{-1}(P)$ is a convex function . This demonstrates geometrically that (5.3) cannot be satisfied, when we precisely determine $P_{max}(N^{total})$.

Figure 5-4, a modified version of 5-3, shows why it is not a good idea to use (5.4) if $N^j > N^*$. $N^j = 160$ is chosen so that $N^j \gg N^*$. A line through the points $(P_{max}(N^{j-1}) = 0.796, N^{j-1} = 0)$ and $(P_{max}(N^j) = 0.901, N^j = 160)$ is drawn. This line segment determines the value for $N^{j+1} = 143$, the estimate of N^*. The point $(P_{max}(N^{j+1}) = 0.9008, N^{j+1} = 143)$ is a convex combination of the points $(P_{max}(N^{j-1}), N^{j-1})$ and $(P_{max}(N^j), N^j)$. The fact that $P_{max}^{-1}(P)$ is a convex function guarantees that $N^{j+1} > N^*$. A line through the points $(P_{max}(N^j), N^j)$ and $(P_{max}(N^{j+1}), N^{j+1})$ is constructed. This line is very steep and $N^{j+1} \ll 0$.

To avoid this difficulty we could require that one of the points used to construct the line segment, which determines the next guess, be $(P_{max}(0), 0)$ or a point such that $N^{total} < N^*$. The drawback of this method is as we get closer to N^* we continually decrease the step size. The step size in this case is the difference between the last two estimates of N^*. The remedy is to insist that the initial guess satisfy $P_{max}(N^1) < P^*$. In addition, we choose to use a binary search technique, instead of (5.4), when we

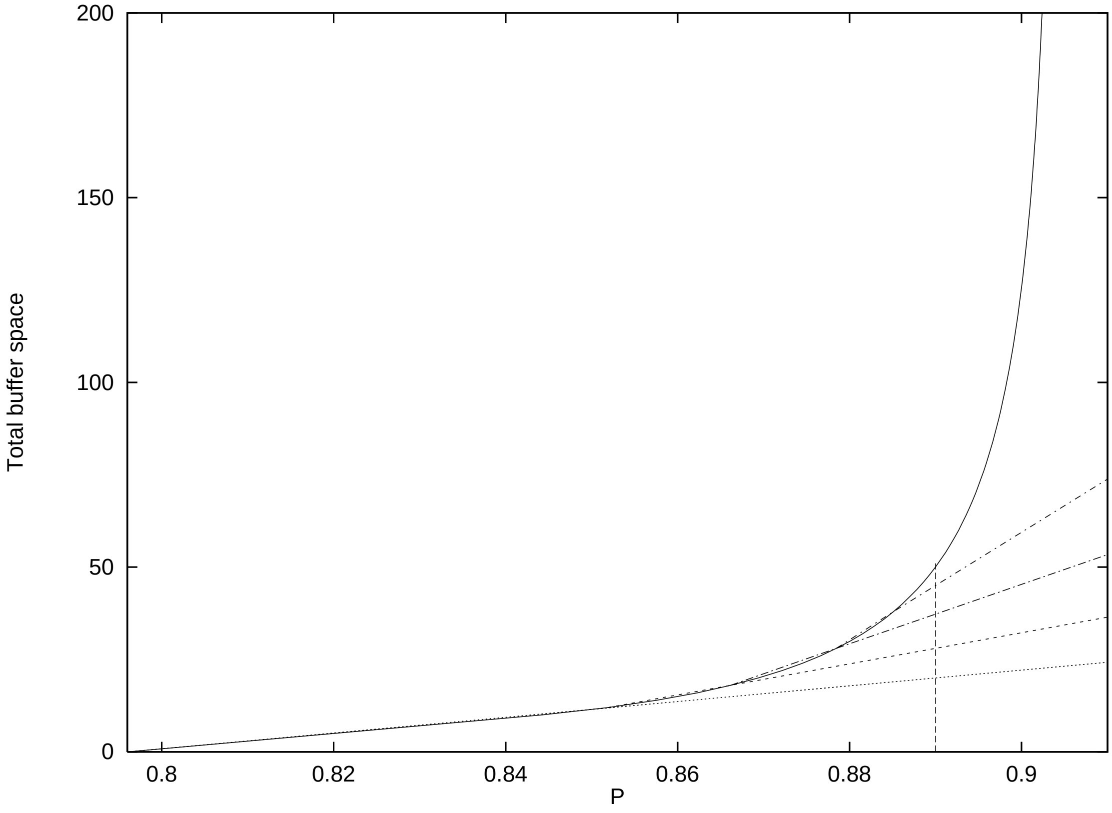

Figure 5-3: N^{total} vs. P_{max}

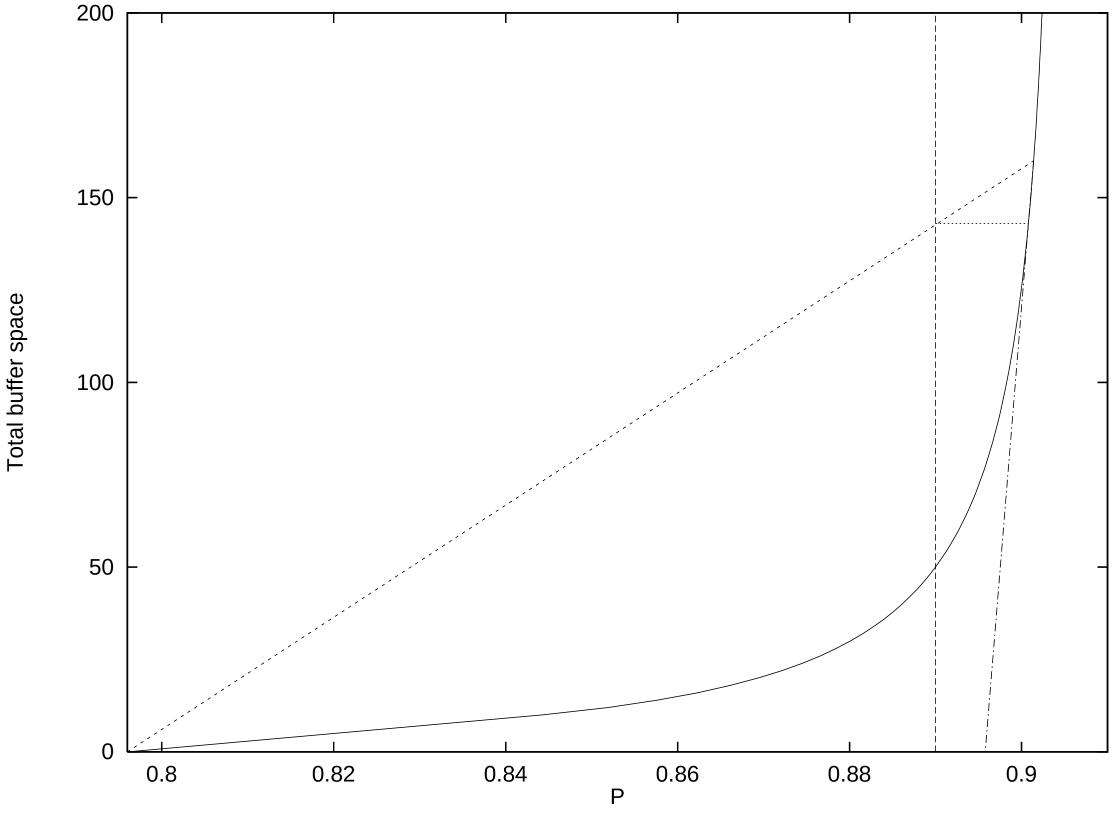

Figure 5-4: N^{total} vs. P_{max}

encounter an estimate such that $P_{max}(N^j) > P^*$.

Figure 5-4 also demonstrates how condition (5.3) may occur. Denote the dual solution calculated with infinite precision $P_{inf}(N^j)$. $P_{inf}(N^j) > P_{max}(N^j)$ because we cannot calculate the dual solution using infinite precision[2]. Consequently the points we use in (5.4) will all lie above the curve $N = P_{inf}^{-1}(P)$. If $P_{inf}(N^{j-1}) - P_{max}(N^{j-1})$ is large enough $N^j > N^*$. This can be demonstrated graphically. Let the point $(P_{max}(N^{j-2}), N^{j-2})$ lie very close to the curve $P_{inf}^{-1}(P)$. Assume $P_{inf}(N^{j-1}) - P_{max}(N^{j-1})$ is large so the point $(P_{max}(N^{j-1}), N^{j-1})$ lies significantly above the curve on the line constructed to approximate $P_{inf}^{-1}(P)$. When this happens, $N^j > N^*$. This scenario occurs in cases where P^* is very close to $P(\infty)$, the production rate of the transfer line when all buffers are infinite.

One of the techniques used for speeding up iterative algorithms is to limit the computational effort expended in the early stages of the algorithm. For the primal algorithm, this means solving the dual algorithm with less precision when $P^* \gg P_{max}(N^j)$. The difficulty with this approach is that it adversely impacts our estimation of N^* for the reason described in the previous paragraph. We have experimented with improving the algorithm using this technique and we discuss our findings in Section 11.2.1.

There is one additional benefit gained from linearizing $P_{max}^{-1}(P)$ to estimate N^*. We avoid the difficulty of determining g_i when N_i is very large. The difficulty is due to to the level of precision with which we are able calculate $P(N_1, \ldots, N_{k-1})$. When N_i is very large, g_i will be very small, and numerical errors introduced when calculating g_i may be the same order of magnitude as g_i. When we use (5.4) we will not encounter a case where $N^j \gg N^*$.

We are limiting the number of guesses we make in the region where the curve in Figure 5-3 is very steep. If P^* is close to $P(\infty)$, we need to perform evaluations in this region. But, by underestimating N^*, we limit how far we will venture into this steep region. Many other linear search techniques, including a binary search,

[2]The magnitude of $P_{inf}(N^j) - P_{max}(N^j)$ is affected by the size of β, which is the interval of uncertainty to which we determine a dual solution. See Section 3.4 for a discussion of β.

require knowing the interval in which the solution is contained. For this case this is the interval that contains N^*. To determine this interval requires guessing a value greater than N^* which, in certain cases, can make gradients difficult to calculate.

We only employ a binary search,when $P_{max}(N^j) \geq P^*$ and $N^j > N^{j-1} + 1$. This means that the last guess is greater than P^* and the total buffer space, of the last two guesses, differs by more than 1 buffer space. If the total buffer space, of the last two guesses, differs by 1 buffer space there is no need to do a binary search because N^j is the primal solution when N is integer.

5.1.3 Binary search

Condition (5.3) occurs when we overestimate N^*. We have found this is more likely to happen when k is very large or P^* is close to $P(\infty)$. When we overestimate N^*, we employ a binary search to avoid the difficulties described in 5.1.2. The endpoints of the line segment searched are the two previous guesses for N^*, which are N^j and N^{j-1}. The search keeps track of two variables. They are the minimum value of N^{total} found so far such that $P_{max}(N^{total}) > P^*$ and the maximum value of N^{total} found so far such that $P_{max}(N^{total}) < P^*$. These variables, denoted N^{over} and N^{under} respectively, are initialized to N^j and N^{j-1}. Then the new guess is given by rounding up (5.5).

$$(N^{over} + N^{under})/2 \tag{5.5}$$

In our implementation the search is terminated, when $N^{over} - N^{under} = 1$. The block in Figure 5-1 which says "construct a new guess" should now be broken up in to the three blocks shown in Figure 5-5 to include the methods for constructing the next guess of N^*.

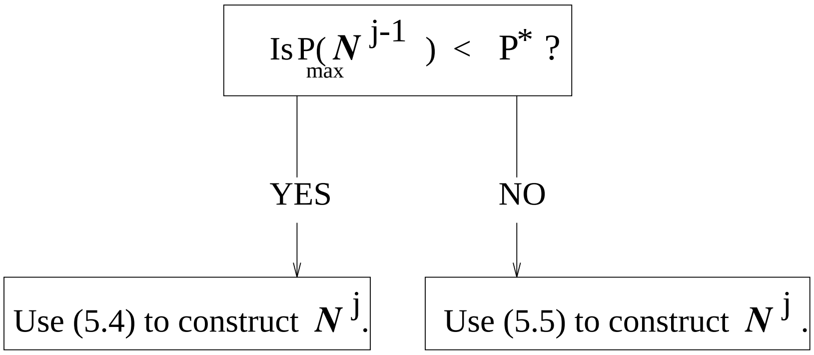

Figure 5-5: Block diagram for constructing the next guess

5.2 Initializing the dual

The dual algorithm requires an initial guess so that it can evaluate $P_{max}(N^j)$. In this section we describe how this initial guess, which is denoted as $\mathbf{N^{initial}}$, is calculated. To do this we need to introduce additional notation. Denote the guess provided as input to the primal algorithm as $\mathbf{N^{input}}$. In addition, the solution to the dual, corresponding to the j^{th} primal guess of N^*, is denoted $\mathbf{N^j}$ where $\sum_{i=1}^{k-1} N_i^j = N^j$. Therefore, $P(\mathbf{N^j}) = P_{max}(N^j)$. Finally, $\mathbf{g^j}$ is the gradient (3.1) calculated at $\mathbf{N^j}$.

If we are invoking the dual algorithm for the first time, we set $\mathbf{N^{initial}} = \mathbf{N^{input}}$. If we are invoking the dual for the j^{th} time, where $j > 1$, we choose $\mathbf{N^{initial}}$ and γ to satisfy

$$\begin{aligned} \mathbf{N^{initial}} &= \mathbf{N^{j-1}} + \gamma \mathbf{g^{j-1}} \\ N^{initial} &= N^j \end{aligned} \tag{5.6}$$

Using (5.6) to construct the initial dual guess moves $\mathbf{N}$ as close as possible to the dual solution. The last step of the $j-1^{st}$ instance of dual algorithm evaluated $\mathbf{g^{j-1}}$ to ensure that $\mathbf{N^{j-1}}$ was optimal. Therefore, no additional evaluations of $P(N_1, \ldots, N_{k-1})$ are required to construct $\mathbf{N^{initial}}$.

5.3 Pseudo code for the primal algorithm

In this section the pseudo code for the primal algorithm is described.

1. Retrieve line parameters, initial guess, and s. Evaluate $P_{max}(0)$.

2. Invoke the dual algorithm to evaluate $P_{max}(N^{input})$. If $P_{max}(N^{input}) > P^*$ terminate with an error message saying the initial guess is too large. Otherwise let $N^1 = N^{input}$ and set $j = 1$.

3. Increment j by 1. Calculate N^j, the new estimate of N^*. If $P_{max}(N^j) < P^*$ use (5.4) to calculate N^j. Otherwise use (5.5) and keep track of N^{under} and N^{over}.

4. Calculate $\mathbf{N^j}$ using (5.6).

5. Use the dual algorithm to determine $P_{max}(N^j)$ and $\mathbf{N^j}$.

6. If $P_{max}(N^j) \geq P^*$ and $P_{max}(N^{j-1}) < P^*$ and $N^j = N^{j-1} + 1$ or (5.5) is satisfied go to step 8.

7. Go to step 3.

8. Round $\mathbf{N^j}$ and call this vector $\mathbf{N^{final}}$. If $P(\mathbf{N^{final}}) \geq P^*$ then $\mathbf{N^{final}}$ is the answer, and terminate. Otherwise perform steps 4 and 5. Go to step 8.

5.4 Implementation issues

Feasibility Before we begin the primal algorithm we must insure that the problem is feasible. We check feasibility by requiring that the isolated production rate of each machine is greater than the production rate target. This is a consequence of the observation that the production rate of a line with infinite buffers is the production rate of the least efficient machine (Buzacott 1967). To enforce this condition we require that

$$P^* < P_i = \frac{r_i}{r_i + p_i} \;\; \forall \, i$$

for the deterministic processing time model and

$$P^* < P_i = \frac{\mu_i r_i}{r_i + p_i} \;\; \forall\, i$$

for the continuous material flow model.

Necessity We also want to determine if there is a need to insert buffers. If the production rate of the line with no buffers is greater than P^* there is no reason to run the algorithm. We are able to calculate the production rate of a line with zero buffers (Buzacott 1967). We use this result to determine the necessity of employing the algorithm. When the deterministic processing time model is used.

$$P(0) = \frac{1}{1 + \sum_{i=1}^{k} \frac{p_i}{r_i}}$$

If the processing rates of the machines are different we invoke the ADDX algorithm with $N_i = 0 \; \forall\, i$ to evaluate $P(0)$. If $P(0) \geq P^*$ the solution is $\mathbf{N} = 0$.

ADDX and DDX algorithm initialization The comments made about initializing the DDX algorithm and the ADDX algorithm in Section 3.9 are still applicable.

Initial Primal guess We have chosen to begin the algorithm by setting $N_i = 5 \;\forall\; i$ for most of the cases we study. Though a better initial guess reduces the execution time the improved performance does not seem to warrant expending energy in determining and inputting an initial guess. If the production rate achieved by the initial guess exceeds P^* we must restart the algorithm with an new guess.

Rounding In step 8 of the primal algorithm, we round $\mathbf{N^j}$ so that the solution is integer. The algorithm rounds according to the procedure outlined in Section 3.7. In some cases this may reduce the production rate enough so that $P(\mathbf{N^{final}}) < P^*$. The algorithm then increases the total buffer space by 1 and solves the dual problem for this new value of $N^{total} = N^{final} + 1$. In all of the experiments we have performed this has been sufficient to satisfy the production rate constraint. There is a provision in our implementation to iterate if the constraint is not satisfied.

Chapter 6

Behavior of the primal algorithm

In this chapter we present examples to demonstrate that the primal algorithm is a practical line design tool. We report the execution time of the primal algorithm, total buffer space required, and calculated buffer distributions.

We discuss the accuracy of our solutions in Section 6.1. Then we describe the behavior of the algorithm for balanced and unbalanced lines in Section 6.2. Next we compare the performance of the algorithm to results reported by Park (1993) and Gershwin and Goldis (1995) in Section 6.3.

The deterministic processing time model is used for all the examples in this chapter. In these examples no post-processing is done after the rounding step. We have initialized $N_i = 5 \ \forall\, i$ and $s = 1$, where s determine the precision of the dual solution (Section 3.4), for all examples unless we explicitly state otherwise.

6.1 Accuracy

We use the results of the Gershwin and Goldis (1995) algorithm to verify the accuracy of the primal algorithm. Gershwin and Goldis establish the accuracy of their solutions by comparing the output of their algorithm to the results of an exhaustive search. The search was performed for several five-machine lines and individual buffer sizes of up to 20.

We claim that our solutions are either optimal or very close to optimal. In all of the cases we study we achieve a N^{total} that differed by no more than 1 from the solutions determined by Gershwin and Goldis (1995). In these cases we are able to close this gap by changing s. The cost of increased accuracy is longer execution time. The other factor that impacts the optimality of the solution is the rounding methodology. When we round (Section 3.9) only three points in the neighborhood of our solution are considered. The number of feasible candidates may be very large, especially when the line is long. Therefore, it is not surprising that we do not always round to the optimal integer solution.

6.2 Examples

In this section we present examples of how the primal algorithm traverses the search space. We do this in detail for both a balanced and an unbalanced three-machine line. We also present the results of an experiment on balanced and unbalanced ten-machine lines. These results support claims made in Section 2.5.1 concerning the expected performance of the algorithm on balanced and unbalanced lines.

Before doing these examples, the unbalanced three-machine line described in Table 2.1 and Figure 2-3 is used to demonstrate how buffer allocation impacts production rate. We also discuss some simple buffer allocation rules presented in the literature.

Figure 2-3 illustrates that both total buffer space and buffer placement impact the production rate. Two buffer distributions that demonstrate this are $(4, 100)$ and $(45, 17)$. Their respective production rates and total buffer sizes are $P(4, 100) = 0.848$, $N^{total} = 104$ and $P(45, 17) = 0.896$, $N^{toatal} = 62$. The buffer sizes $(45, 17)$ achieve more than a four percent increase in production rate over $(4, 100)$ while using 50% less total buffer space.

This example demonstrates the importance of intelligently allocating buffer space. Some authors argue that a reasonable rule for allocating buffer space is to make all the buffers equal (Buzacott and Shantihkumar 1993). This policy is not optimal but may perform well for balanced lines. If $N^{total} = 62$, the production rate achieved using Buzacott's rule is $P(31, 31) = 0.892$. This production rate is 0.4% less than

$P(45, 17) = 0.896$ which is optimal. Examples that magnify the difference between near optimal distributions and equal buffer distributions may be constructed by increasing the length of the line and making the line less balanced.

Dallery and Gershwin (1992) argue that buffers should be placed where disruptions are greatest. The difficulty associated with this approach is that it may be hard to determine where disruptions are greatest for long unbalanced transfer lines.

6.2.1 Example 1

Figure 6-1 is a graph of Iso-P curves for the balanced three-machine line described in Table 6.1. We have included some tangent lines that satisfy the constant total buffer space constraint imposed in the dual problem (2.2). We use Figure 6-1 to illustrate how the primal algorithm arrives at a solution.

In Section 2.5.1 we reported the following observation. The line $N_1 + N_2 = C$, where C is a constant, is tangent to some Iso-P curve and the tangent point is the solution to the dual for the constraint $N_1+N_2 = C$. Because the constraint is linear it is much easier to move along the line $N_1+N_2 = C$ or the hyperplane $N^{total} = \sum_{i=1}^{k-1} N_i$ than it is to move along an Iso-P curve.

We claim that subsequent initial guesses for the dual should be close to optimal, as a consequence of moving in the direction of the gradient. For balanced two-machine lines we claim the dual should only need to be solved once because every future guess would lie on the line $N_2 = N_1$, which is both the optimal curve and the direction of the most recently calculated gradient. The sequence of points visited, by the primal algorithm, supports these claims.

We solve the primal problem for $P^* = .875$ and an initial guess of (23,43).[1] The algorithm proceeds by searching along the line $N_1 + N_2 = 65$ until the point (33,33) is found. Subsequently the line $N_2 = N_1$ is searched until the point (46,46) is encountered. When the primal algorithm is executed it guesses the sequence of points listed in Table 6.2. Additional points are also evaluated to determine which direction

[1] In this case an initial guess of $N_i = 5 \; \forall \; i$ would be very uninteresting.

Machine	r_i	p_i
1	.1	.01
2	.1	.01
3	.1	.01

Table 6.1: Three-machine line parameters

Primal iteration	Initial dual guesses	Subsequent guesses
1	(23,43)	(25,41) (29,37) (33,33)
2	(36.5,36.5)	
3	(44,44)	
4	(45.5,45.5)	
5	(46,46)	

Table 6.2: Sequence of guesses for example 1

to move and to verify the solution. This path shows that the dual only needs to be solved once, for $N^{total} = 66$. All subsequent initial dual guesses are optimal solutions to the current dual subproblem.

The equal distribution of optimally allocated buffer space is a consequence of the symmetry of the line. For longer, balanced[2] lines the buffers are allocated in the shape of a bowl with more space allocated to the center of the line than the ends. Examples that exhibit the bowl phenomenon (Hillier, So, and Boling 1993) are described later in this section.

6.2.2 Example 2

We expected that the results of example 1 could be extended to longer lines. We ran the following experiment to confirm our intuition. In case 1, no modifications to the primal algorithm were made. For case 2, we only solve the first dual instance. After solving the dual for $\mathbf{N^{input}}$ we execute the primal algorithm as described in Section 5.3 with the following modification. The call to the dual algorithm is skipped and $\mathbf{N^j} = \mathbf{N^{initial}}$. In addition, we set $\mathbf{g^j} = \mathbf{g^1}\ \forall\ j > 1$. In case 2, no gradients are calculated after the first iteration of the dual algorithm. We estimate the optimal curve as the line passing through the point $\mathbf{N^1}$, in the direction of $\mathbf{g}$, when $\mathbf{g}$ is

[2]All balanced lines are symmetric.

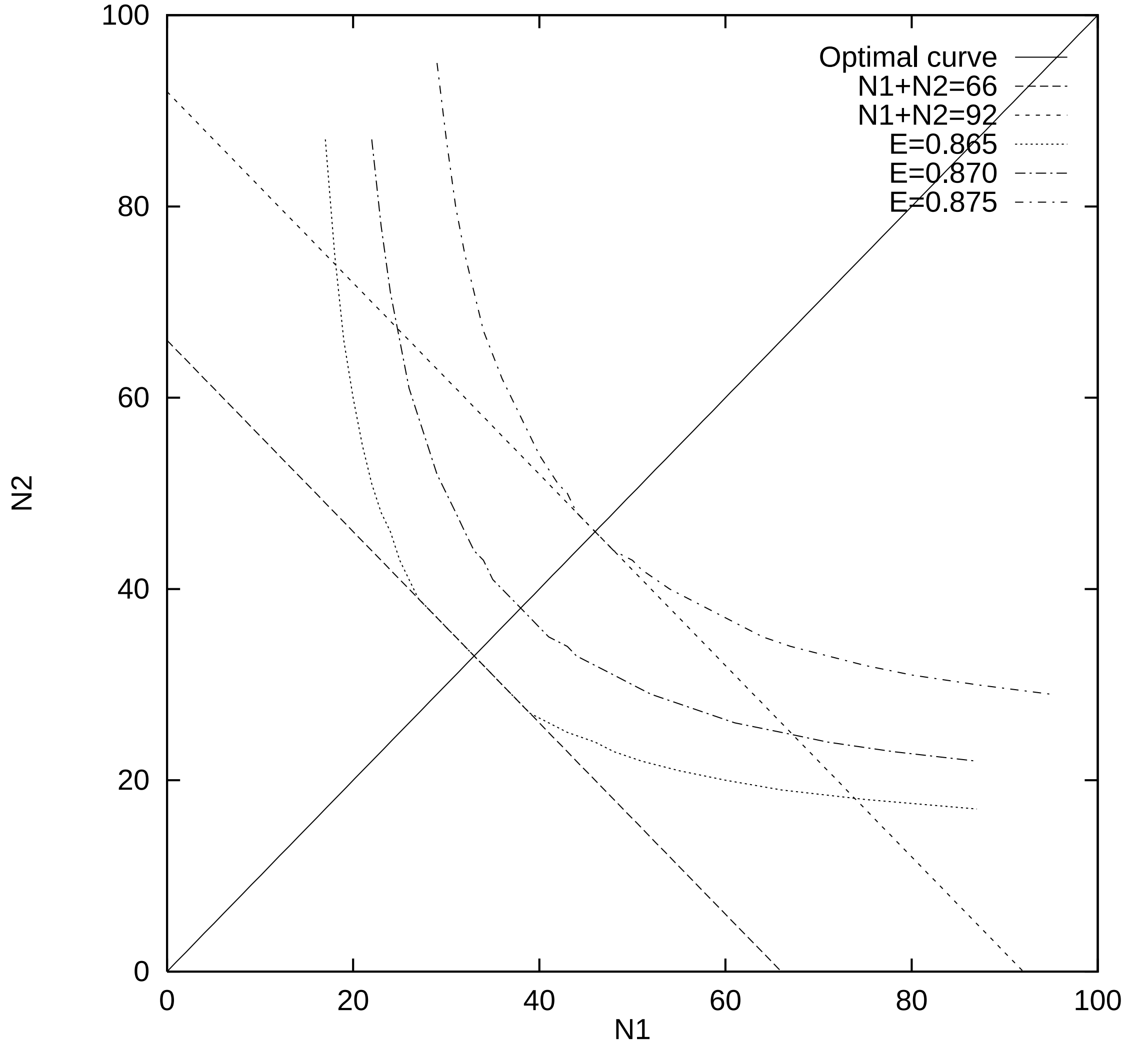

Figure 6-1: Iso-P curves with tangent lines for three-machine line

calculated at the point $\mathbf{N^1}$, the solution to the dual for the first guess of N^*. All future guesses are constructed so that they lie on this line. In future examples we refer to this solution method as option G^1. This designation is used to indicate that the gradient is not recalculated after the first iteration of the dual algorithm.

The system for this experiment is a balanced ten-machine deterministic processing time line with $r_i = 0.095$, $p_i = 0.007$, and $P^* = 0.88$. We begin both cases with $N_i = 10\ \forall\ i$, so that our limitation of $N_i \geq 4$ will not prevent us from determining a near optimal buffer allocation for our initial dual instance.

Case 2 requires two more buffer spaces than case 1. Case 1 also places additional buffer spaces closer to the center of the line. Both differences are most likely a consequence of the accuracy of the initial dual solution. The results are reported in Table 6.3. It is important to note that, while case 1 is an optimal integer solution of the primal problem it is not an optimal dual solution. This is due to the rounding procedure. The optimal dual solution is obtained by incrementing buffers N_2 and N_9 by one and decrementing buffers N_1 and N_{10} by one.

The number of two-machine evaluations and the production rates achieved for each case are presented in Table 6.3. Case 2 takes less than one fifth the number of two machine evaluations required for case 1. This is because most of the computer effort expended by the dual algorithm is to verify that a solution had been reached rather than searching for the solution.

	Buffer i										Two-machine	
Case	1	2	3	4	5	6	7	8	9	N^{total}	evals.	P
1	27	38	42	44	44	44	42	38	27	346	67,264	0.88009
2	32	38	41	42	43	42	41	37	32	348	8,896	0.88004

Table 6.3: Results for a balanced ten-machine line

6.2.3 Example 3

The sequence of points traversed, for the unbalanced three-machine line described in Table 2.1 and Figure 2-3, is listed in Table 6.4. For this example we use $P^* = 0.8955$ and $s = .5$. We reduce s to ensure that our rounding methodology does not prevent

Primal iteration	Initial dual guess	Subsequent dual guesses
1	(5,5)	(6,4)
2	(15.5,5.5)	(14.5,6.5)
3	(21.4,9.6)	
4	(30.12,13.88)	(31.12,12.8)
5	(38.26,15.74)	(39.26,14.74)
6	(43.27,16.73)	(43.77,16.23)
7	(45.8,16.2)	(45.2,16.8) rounded to (45,17)

Table 6.4: Sequence of guesses for example 3

us from reaching the optimal solution. Unlike example 1, it was necessary to solve the dual for subsequent guesses constructed by the primal. This is because, as shown in Figure 2-5, the optimal curve is not a straight line.

Notice that, even for an unbalanced transfer line, all dual solutions except iteration 3 were only one step away from the initial guess. For iteration 3 the initial primal guess was the solution.

6.2.4 Example 4

In this example we compare the primal algorithm, option G^1, and an additional variation of the primal algorithm. This variation is based on the results of the previous example. The transfer line we study consists of the first ten machines of the unbalanced line studied by Park and described Table 4.2. For the first case we solve the dual for every primal guess. In the second case we only solve the dual for the initial primal guess but recalculate the gradient every time we construct $\mathbf{N^j}$, the next primal guess. This updates our estimate of the optimal curve every time we take a step. In case 3 we use option G^1. Option G^1 only invokes the dual algorithm once and only approximates the optimal curve once. The initial guess is $N_i = 10 \;\; \forall \, i$ and $P^* = 0.89$. The results are shown in Table 6.5.

The large difference between the total buffer space required for cases 2 and 3 demonstrates that the optimal curve, for this example, is not a straight line. The strategy of only calculating gradients during the first iteration of the dual will therefore not work well for unbalanced lines. There is a significant difference in the number

Case	N^{total}	Two-machine evals.
1	162	125,696
2	167	45,328
3	315	9,184

Table 6.5: Comparison of solutions for example 4

of two-machine evaluations between cases 1 and 2 and a small difference in total buffer space.

The results of this and previous examples, and the observation that a gradient calculation accompanies each line segment search performed as part of the dual algorithm, suggests the following strategy for improving the performance of the primal algorithm.

When the production rate is significantly less than P^* only perform one line search when the dual is called. We would expect a substantial reduction in computer effort because the number of calls to the DDX algorithm to calculate a gradient is equal to the number of buffers in the line. When the line is long a majority of the computational effort is expended calculating gradients. If we only perform the line search once we will not need to calculate a gradient to verify the solution.

The drawback to this strategy is that the reducing the accuracy of the dual solution adversely impacts our estimate of N^*. This may ultimately require making additional calls to the dual algorithm, which would require additional gradient calculations.

6.3 Comparison with the literature

In this section we compare the performance of the primal algorithm to algorithms developed by Park (1993) and Gershwin and Goldis (1995). Park's algorithm uses a two-phase tree search method and does not guarantee an optimal solution. Gershwin and Goldis developed three algorithms. We use their combined algorithm for comparison because it is the fastest of the three. In this thesis we refer to the combined algorithm as the GG algorithm to avoid confusion. The GG algorithm uses a gradient method. Gershwin and Goldis (1995) claim the algorithm always reaches an optimal

		Park			GG			Primal		
Case	Target rate	Actual rate	Long line evals.	N^{total}	Actual rate	Evals. long line (two-mach.)	N^{total}	Actual rate	Evals. long line (two-mach.)	N^{total}
1	0.85	0.8396	739	84						
2	0.85	0.8420	1,107	84						
3	0.85	0.8505	1,659	93	0.8507	182	87	0.8507	114	87
4	0.85	0.8505	1,838	93		(125,923)			(79,140)	
5	0.895	0.8950	510	390	0.8950	1,173 (1,481,928)	242	0.8950	342 (534,820)	243

Table 6.6: Comparison of algorithms

or near optimal solution. Both the Park and GG algorithms solve Problem (1) for the deterministic processing time model. Both algorithms use the DDX algorithm to evaluate $P(N_1, \ldots, N_{k-1})$. Both papers present results for the twelve-machine line described in Table 4.2.

Table 6.6 compares the total buffer size, production rates, and number of long line evaluations for experiments with two different values of P^*. The initial guess for the primal and GG algorithms, in both cases, is $\mathbf{N} = (5,5,5,5,5,5,5,5,5,5,5)$. Park's algorithm does not require an initial guess. The disparity in Park's results in cases 1-4 are a consequence of specifying different search parameters. We also present the number of two-machine evaluations required. Park does not report this number.

For cases 1-4 the GG algorithm and the primal algorithm achieve identical solutions but the primal algorithm requires approximately 33% fewer two-machine evaluations. This difference in execution time is not completely reflected in the number of long line evaluations because, in the early iterations of the GG algorithm, long line evaluations are replaced by two-machine evaluations. Parks algorithm does not satisfy the production rate constraint in cases 1 or 2. In all cases where Park does satisfy the constraint, a larger N^{total} is calculated. When $P^* = 0.85$ Park's algorithm requires significantly more long line evaluations to arrive at a solution than the other two algorithms.

For $P^* = 0.895$, the solution of the primal algorithm has one more buffer space than the GG algorithm. If we set $s = 0.25$, this difference disappears. The primal algorithm requires the least computational effort in all cases.

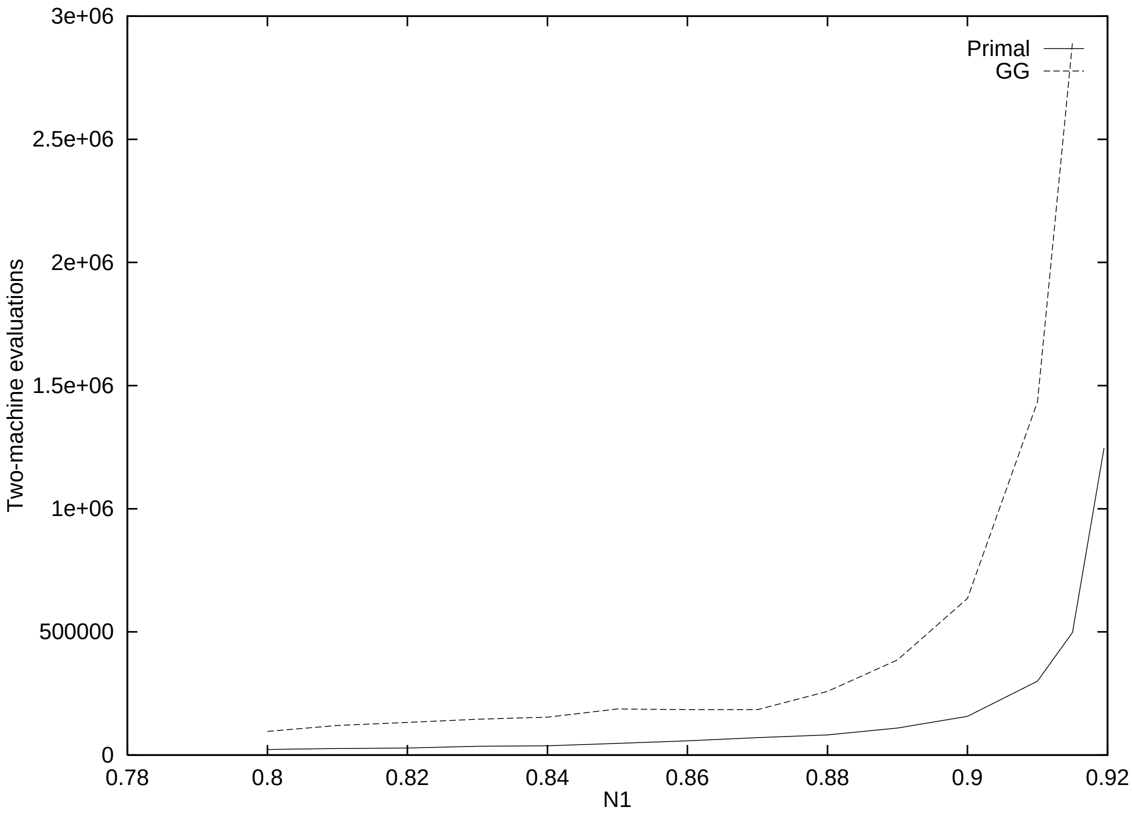

Figure 6-2: Two-machine evaluations as a function of P

6.3.1 Increasing P^*

Figure 6-2 illustrates how execution time is impacted by P^* for the primal and GG algorithms. The number of two-machine evaluations, for the line described in Table 6.7, is reported. When P^* approaches the infinite buffer production rate (0.92) the number of two-machine evaluations required, using either algorithm, increases significantly. The increase is much more dramatic for the GG algorithm. The primal and GG algorithms calculate the same value for N^{total} in all cases. The primal generally requires between 60% and 80% fewer two-machine evaluations. When P^* approaches 0.92 the differences in execution time are even more pronounced.

6.3.2 Long lines

Table 6.8 demonstrates how increasing the length of the line impacts computational effort. Each long line is composed of ten-machine blocks. Each block is the line

Machine	i									
	1	2	3	4	5	6	7	8	9	10
r_i	.094	.095	.045	.078	.069	.094	.095	.045	.078	.069
p_i	.007	.008	.003	.004	.006	.007	.008	.003	.004	.006
P_i	.93	.92	.94	.95	.92	.93	.92	.94	.95	.92

Table 6.7: Ten-machine line

described in Table 6.7. $P^* = 0.88$ for all the lines. We set $s = 0.25$ for the 20 and 30 machine lines. The significant increase in computational effort is a consequence of three factors. The first is the increased number of two-machine evaluations required for the DDX algorithm to converge. The second is that each gradient calculation requires an additional long line evaluation for each added machine. The third is that as the line increases the total buffer space increases and we must travel farther to reach a solution.

Number of machines	Two-machine evals.		N^{total}	
	Primal	GG	Primal	GG
10	82,176	258,661	433	443
20	1,814,868	6,867,878	995	995
30	15,955,968	47,157,058	1,556	1,557

Table 6.8: Long line computation times

Chapter 7

Profit maximization problem formulation

7.1 Description of profit maximization problem

In the next two chapters, we continue our construction of algorithms to facilitate the allocation of buffer space when designing a transfer line. Though the algorithm we construct is primarily intended as a line design tool, the suggested buffer sizes may also be used to implement a production control strategy. Maximization of profit, rather than target production rate, may be an alternative criterion for determining the buffer sizes or number of kanbans to use in a line.

For this formulation we consider the impact of generalizing the objective function to include inventory holding costs. We also dispense with the production rate constraint imposed in the primal formulation (2.1). We no longer require that the cost of a buffer i space be the same as the cost of a buffer j, $i \neq j$, to allow for the possibility that cost of buffer space may depend on a buffer's location in the line.

This is a more realistic model for the following reason. When a part is processed it may change in size, complexity, or durability. Consequently the stage, in the production process, where a part resides may affect either the amount of physical space or the complexity of facilities required to store the part.

We assume that a market exists for all of the parts we produce. Henceforth, we refer to this problem as the *PMP*. The PMP is formulated as an unconstrained maximization problem. Once again, the number of machines and the reliability parameters of each machine are specified. The independent variables continue to be the buffer sizes.

We assume that estimates for the profit per part and the expected lifetime of the product are available. Our calculation of the profitability of the line is straightforward. The total revenue is the number of parts manufactured over the product life cycle multiplied by the profit per part. The capital costs are due to the buffers and the inventory cost is the cost of the average work-in-process inventory. The profit is the revenue minus the capital and inventory costs.

This formulation is appropriate when we want to account for the cost of floor space or buffering machinery and work in process inventory, and when no production rate is mandated. The capital costs due to the machines is not included in the problem formulation because the machines have been specified.

The capital cost of buffer i, $a_i N_i$, is assumed to be directly proportional to the size of buffer i. There are no fixed costs associated with buffers. The cost of the average inventory in buffer i is $b_i \bar{n}_i$. The product of the expected product life time and profit per part is Φ. Therefore, the total revenue of the line is $\Phi P(N_1, \ldots, N_{k-1})$. We continue to use the DDX and ADDX algorithms for evaluating production rates and average buffer levels.

The approach we use to solve the PMP is very similar to the approach we use to solve the primal problem. This approach is described after some additional notation is introduced. Let the profit rate realized for a given buffer distribution be denoted $\Pi(N_1, \ldots, N_{k-1})$ and the maximum profit for a given total buffer space be denoted $\Pi_{max}(N^{total})$.

We search $\Pi_{max}(N^{total})$ for a maximum. Unlike $P_{max}(N^{total})$, $\Pi_{max}(N^{total})$ is not monotonically increasing. Therefore, we need to use a search procedure that is different from that of the primal algorithm. The search procedure we employ requires that $\Pi_{max}(N^{total})$ be unimodal[1]. Unlike $P(N_1, \ldots N_{k-1})$ for $\sum_{i=1}^{k-1} N_i = C$, $\Pi(N_1, \ldots N_{k-1}$

[1]Unimodal means have a single extreme point (Luenberger 1984).

for $\sum_{i=1}^{k-1} N_i = C$ is not a concave function. These issues are treated in Section 7.3.

As before, we formulate and solve two subproblems to arrive at a solution to the PMP. We denote these subproblems as the constrained profit maximization problem (CPMP) and the one dimensional profit maximization problem (ODPMP).

7.2 Problem statement

In this section we formalize the PMP, CPMP, and ODPMP.

7.2.1 Profit maximization problem (PMP)

Let $\Phi, a_i, b_i \; \forall \; i$ be non-negative constants.

Choose $N_1, N_2, \ldots, N_{k-1}$ to

$$\text{maximize} \;\; \Phi P(N_1, \ldots, N_{k-1}) - \sum_{i=1}^{k-1} a_i N_i - \sum_{i=1}^{k-1} b_i \bar{n}_i \tag{7.1}$$

subject to $\qquad N_i \in R^+$

Though a significant amount of work has been done to determine buffer distributions we know of no papers that address this problem as stated. Barten (1962) and Anderson and Moodie (1969) choose buffer sizes to maximize profit but they impose the additional constraint that all buffer sizes be equal.

7.2.2 Constrained profit maximization problem (CPMP)

Choose $N_1, N_2, \ldots, N_{k-1}$ to

$$\text{maximize} \;\; \Phi P(N_1, \ldots, N_{k-1}) - \sum_{i=1}^{k-1} a_i N_i - \sum_{i=1}^{k-1} b_i \bar{n}_i \tag{7.2}$$

subject to

$$N^{total} = \sum_{i=1}^{k-1} N_i; \;\; N_i \in R^+$$

The CPMP is a reasonable formulation when a space constraint exists. This formulation also accounts for the possibility that the size of the part will change when it is processed by permitting different cost coefficients for each N_i.

7.2.3 One dimensional profit maximization problem (ODPMP)

Choose N^{total} to

$$\text{maximize } \Pi_{max}(N^{total}) \tag{7.3}$$

7.3 Properties of $\Pi(N_1, \ldots, N_{k-1})$

In this section we describe the properties of $\Pi(N_1, \ldots, N_{k-1})$ for two-machine and three-machine lines. This section is a description of numerical experiments and observations.

The algorithm we construct in Chapter 8, to solve the CPMP problem, is based on gradient methods. The results of examples described in this section are presented as evidence that constructing an algorithm which use uses steepest descent methodology to solve the CPMP problem is warranted. Gradient methods are appropriate when the space being searched has a single maximum. Therefore, it is essential that the function $\Pi(N_1, \ldots, N_{k-1})$ have only one maximum point for the proposed optimization methods to work correctly. The examples we describe indicate that $\Pi(N_1, \ldots, N_{k-1})$ exhibits this property.

7.3.1 Two-machine lines

In this section we discuss the behavior of two-machine lines. We present an example to demonstrate that $\Pi(N_1)$ may be either a concave or convex function when $\Phi = 0$. From these results we know that $\Pi(N_1, \ldots, N_{k-1})$, unlike $P(N_1, \ldots, N_{k-1})$, is not necessarily concave.

Example 1

Figure 7-1 is a graph of average inventory vs. buffer size for three different continuous material two-machine lines. The reliability parameters are $r_i = 0.1$ and $p_i = 0.01$ for both machines in all three lines. In the balanced line $\mu_1 = \mu_2 = 1$. In the unbalanced lines the processing rate of the faster machine is $\mu = 1.05$ and the rate for the slower machine is $\mu = 1$. When the faster machine is M_1, $\bar{n}$ increases with increasing N. When the line is balanced $\bar{n} = N/2$ and and when M_1 is slower, $\bar{n}$ appears to approach a limit with increasing N. Figure 7-1 demonstrates that $\Pi(N_1)$ may be either concave or convex.[2] For this case, which is $\Phi = 0$, $a_1 = 0$, and $b_1 = 1$, $\Pi(N_1)$ would only have one maximum at the point $N_1 = 0$.

Notice that the two unbalanced lines are the reverse of each other. Therefore, the two unbalanced lines will have the same production rate and the line with the faster machine downstream is always more profitable if $b_1 > 0$, for any a_1,b_1 and Φ.

7.3.2 Three-machine line

Example 2

Figure 7-2 is a graph of the cost of inventory versus buffer sizes for a balanced three-machine line. In this case balanced refers only to the fact that the machines are identical. That is the cost coefficients and buffer sizes are not necessarily the same. For this example $\Phi = 0$, $a_i = 0$, and $b_i = 1$ for $i = 1, 2$. Consequently, $\Pi = -\sum b_i \bar{n}_i$ and "profit" consists only of inventory costs, and is always negative. The parameters of the line are $r_i = 0.1$, $p_i = 0.01$, and $\mu_i = 1$ for $i = 1, 2, 3$. The symmetry exhibited in previous graphs of production rate versus buffer sizes is not seen in this graph. This is because average buffer levels are not symmetric. If the line is reversed, the production rate does not change, but both the total average inventory and the individual average buffers levels will change. Therefore, cases where N_1 is larger than N_2 will have have a larger total inventory than the reversed line. Table 7.1 illustrates this phenomenon for two possible buffer distributions.

[2]There is also a discussion of the limiting behavior of time two-machine lines in Gershwin (1994).

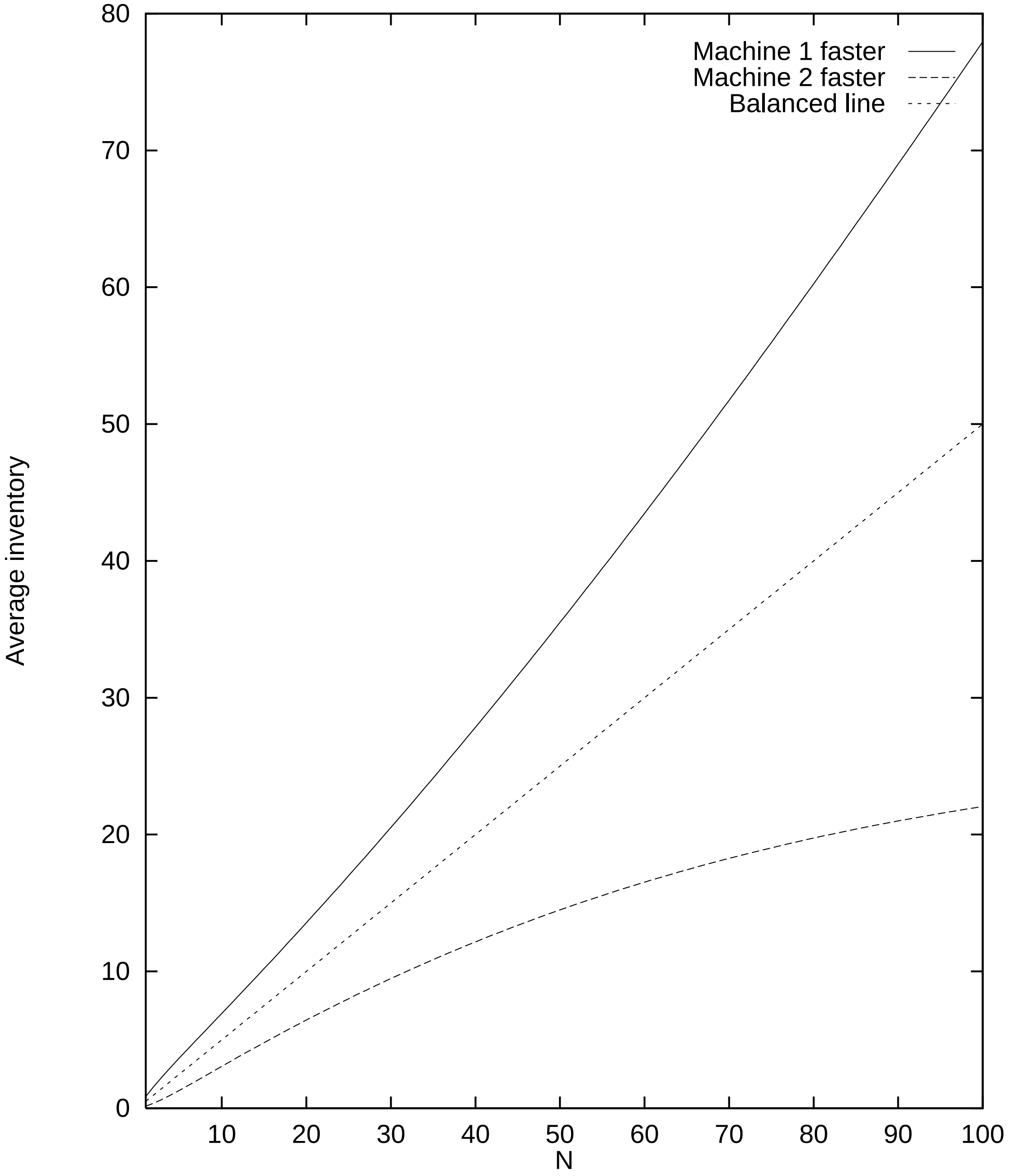

Figure 7-1: $\bar{n}_1$ vs. N_1

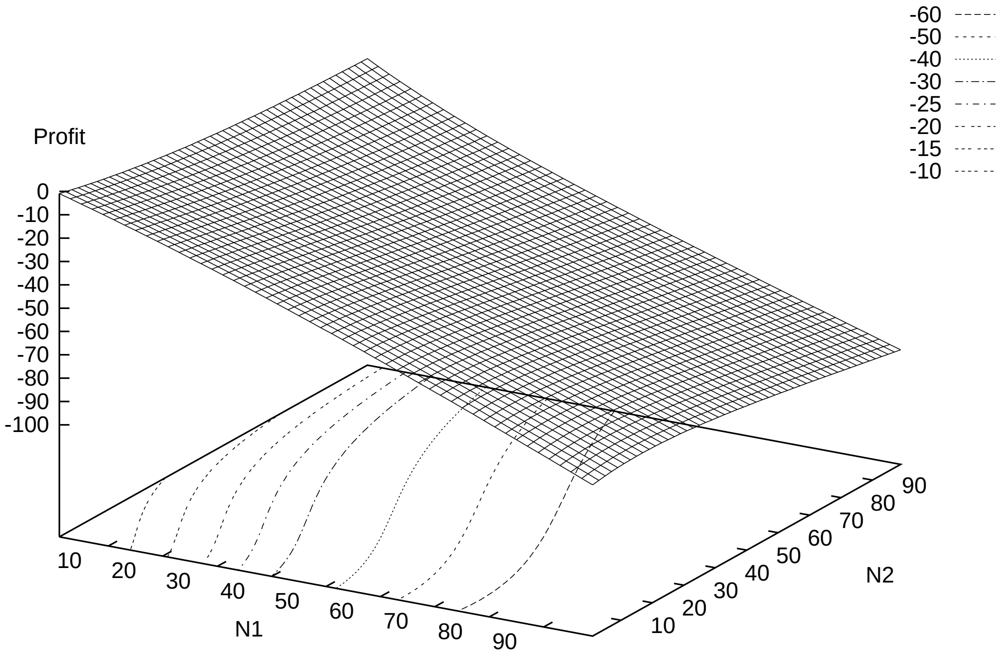

Figure 7-2: $-(\bar{n}_1 + \bar{n}_2)$ vs. N_1 and N_2

Case	N_1	N_2	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_1 + \bar{n}_2$	P
1	15	33	8.40	11.80	20.20	0.852
2	33	15	21.19	6.59	27.79	0.852

Table 7.1: Three-machine line with $r = .1$, $p = .01$, and $\mu = 1$

The shapes of the contour lines in Figure 7-2 demonstrate that $\Pi(N_1, \ldots, N_{k-1})$ is not a concave function. The contours also show that inventory costs increase in N_i. The reason the maximum is at $N_1 = N_2 = 0$ is that we have not included throughput in the objective function.

Example 3

Figure 7-3 is the same transfer line with $b_2 = 2$. We have increased b_2 to represent situations where value is added to the part as it is processed. As in the previous graph, inventory cost increases in each N_i. For this example the profit decreases more quickly when N_2 increases.

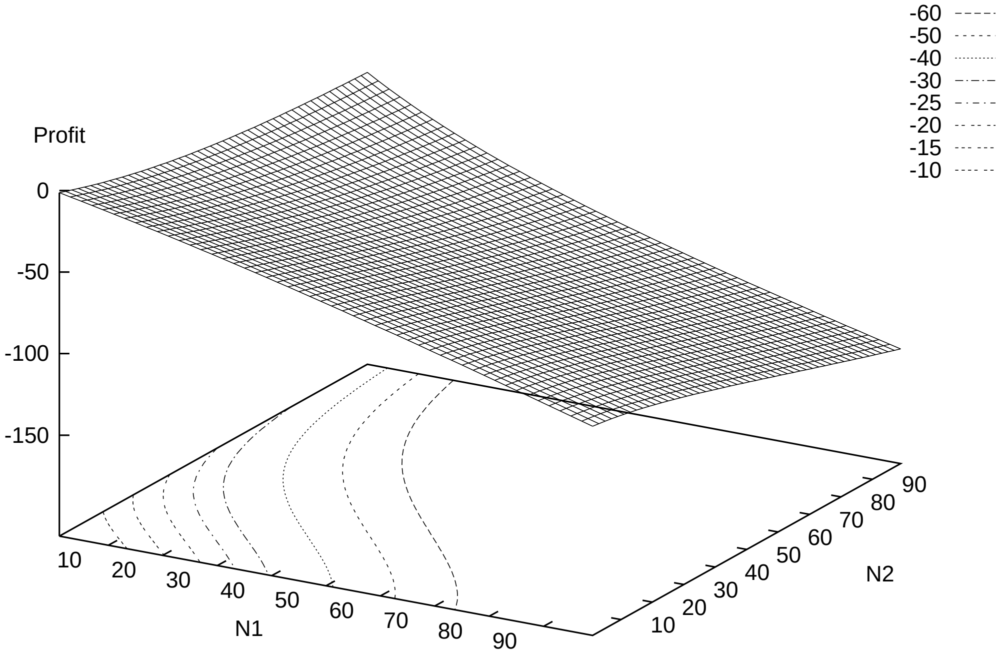

Figure 7-3: $-(\bar{n}_1 + 2\bar{n}_2)$ vs. N_1 and N_2

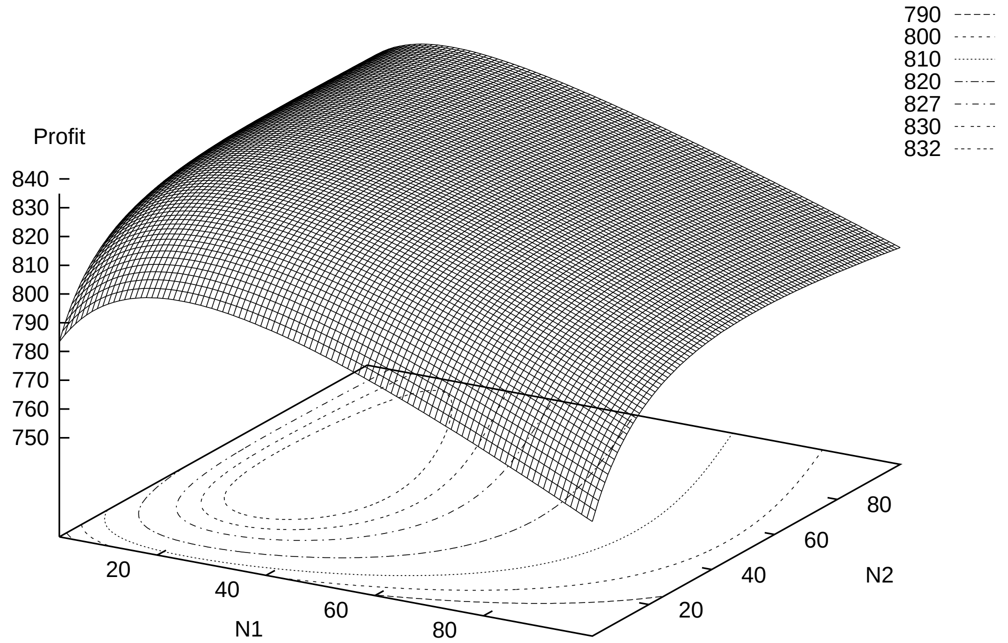

Figure 7-4: Profit vs. N_1 and N_2

Example 4

Figure 7-4 is a graph of $\Pi(N_1, N_2)$ vs. N_1 and N_2 for the balanced three-machine line described in example 2. The cost coefficients are $\Phi = 1,000$, $a_i = 0$, and $b_i = 1$ for $i = 1, 2$. Therefore $\Pi(N_1, N_2) = 1000P(N_1, N_2) - \bar{n}_1 - \bar{n}_2$. The contour lines indicate that the function $\Pi(N_1, N_2)$ has a global maximum and no other local maxima. The maximum profit for the specified transfer line is realized when $\mathbf{N} = (21, 49)$. Given the results of the previous example it is not surprising that the buffer sizes are not equal, even though the line is balanced.

Example 5

Figure 7-5 is a graph of profit vs. buffer sizes for the three-machine line described in Table 7-2, for $\Phi = 100,000$, $a_1 = a_2 = 100$, and $b_1 = b_2 = 1$.

Therefore $\Pi(N_1, N_2) = 100,000P(N_1, N_2) - 100N_1 - 100N_2 - \bar{n}_1 - \bar{n}_2$. In this example

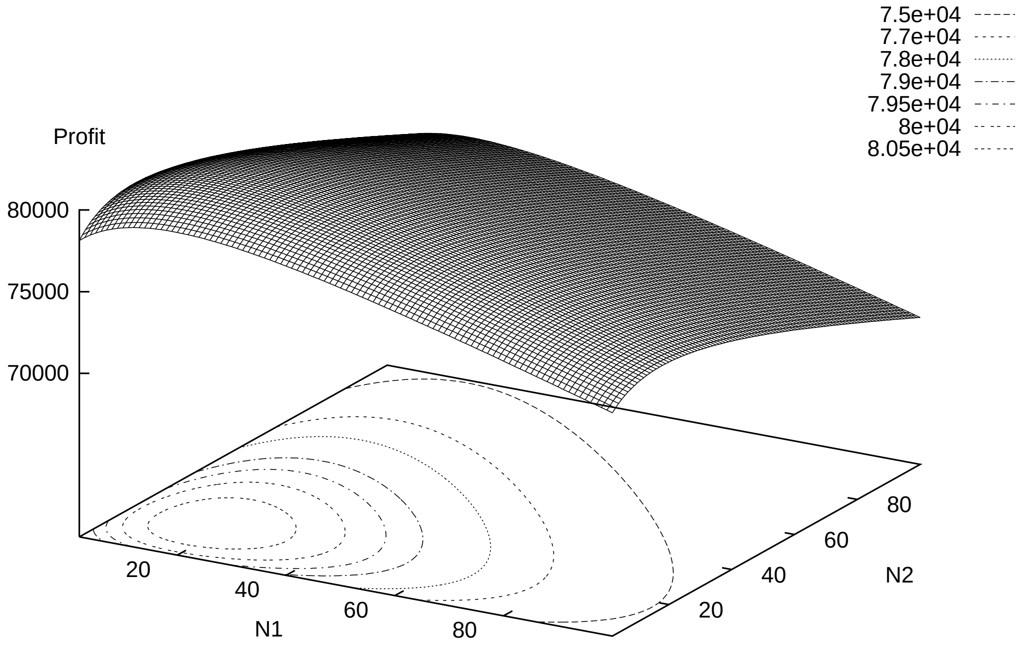

Figure 7-5: Profit vs. N_1 and N_2 for example 5

the cost of the buffers is included by setting $a_i = 100$. The contour lines indicate that the function $\Pi(N_1, \ldots, N_{k-1})$ also has only one global maximum, and no other local maxima. The maximum profit is realized at a buffer distribution of $(17, 18)$, which is almost symmetric. The production rate and buffer size coefficients are considerably larger than the cost of inventory. Consequently the buffer cost dominates the inventory cost. The solution is nearly symmetric because the production rate of any non-symmetric solution may be increased, without appreciably increasing the cost of buffer space, by allocating the buffers symmetrically.

Example 6

One of the graphs in Figure 7-6 is a graph of profit vs. N_1 for an unbalanced three-machine line with parameters described in Table 7.2 and $N_2 = N^{total} - N_1$ with $N^{total} = 100$. The other graph is the reversed line. For this example $\Phi = 1,000$. These graphs are included to demonstrate that the shape of $\Pi(N_1, \ldots, N_{k-1})$, for a specified

r_i	p_i	μ_i	a_i	b_i
.1	.01	1	0	1
.1	.0125	1	0	1
.1	.015	1		

Table 7.2: Parameters for a three-machine line

N^{total}, looks concave. The shapes of these curves also indicates that using a gradient algorithm to determine the $\Pi_{max}(N^{total})$ is reasonable. The difference between the two curves demonstrates that the optimal profit of the reversed line is larger than the optimal profit of the forward line. There are certain buffer distributions, where N_1 is small, for which the forward line is more profitable. This is because, in the reversed line, machine 1 is the bottleneck and with N_1 small machine 1 will block more frequently. The graph also be supports the claims made in Section 7.3.1, that faster machines should be located downstream when inventory costs are insignificant. The difference in maximum profit, for lines with faster upstream vs. downstream machines, is even more pronounced when the inventory cost coefficients, b_i, increase with increasing i.

7.4 Qualitative properties

In Section 7.3 we present numerical evidence to support the reasonableness of a gradient algorithm for solving the PMP, as stated in (7.1). The algorithm is based on the conjecture that, in most cases, $\Pi(N_1, \ldots, N_{k-1})$ has a single maximum. In the next paragraph we present an intuitive argument that demonstrates why this behavior should be expected. We make this argument for a two-machine line with the understanding that the argument may be extended to longer lines.

$\Pi(N)$ for a two-machine line, as stated in (7.1), is the sum of three functions. We discuss the behavior of each of these functions to show that $\Pi(N)$ only has a single maximum. In Chapter 2 we refer to results reported by Okamura and Yamashina (1977) concerning production rate increases gained by incrementing the size of the buffer. They demonstrate that the improvement in production rate realized, when N

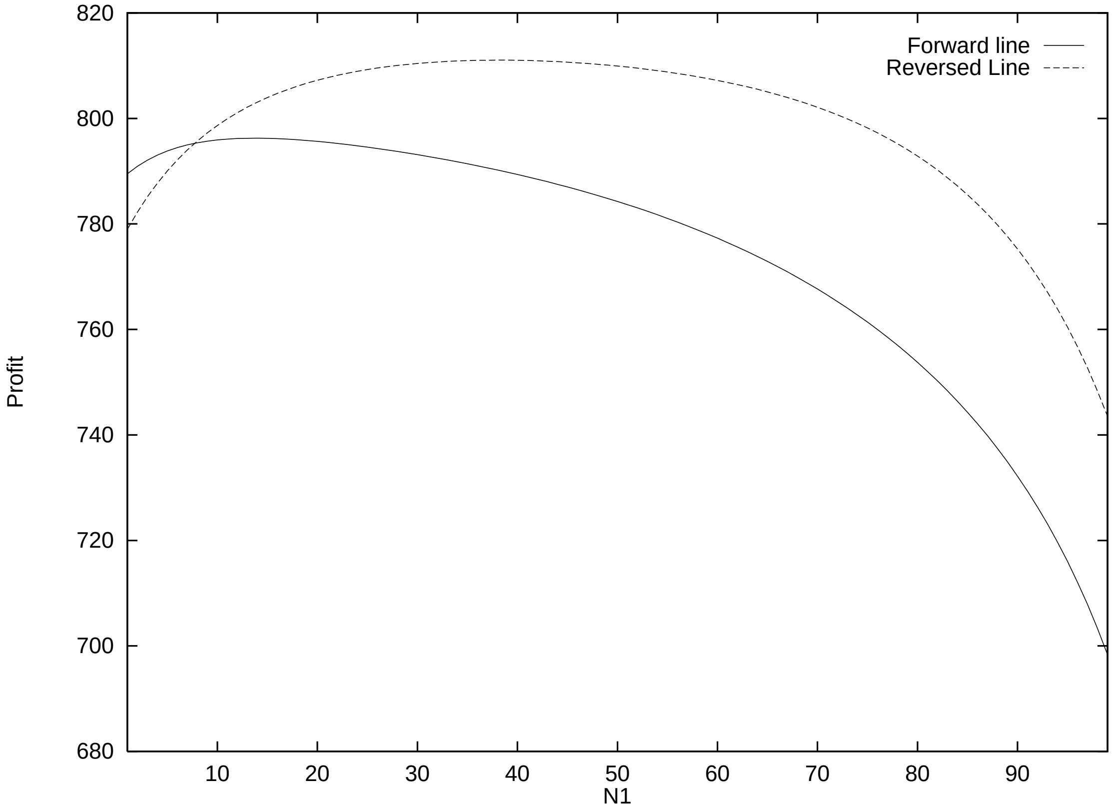

Figure 7-6: Profit vs. N_1 for $N_1 + N_2 = 100$ for lines described by Table 7.2

is incremented by 1, is larger for smaller values of N. Therefore, the increase realized in $\Phi P(N)$, corresponding to incrementing N by 1, decreases as N increases.

For cases where $a > 0$ the argument is as follows. When N is incremented by 1 the buffer cost will increase by a and the inventory cost increase depends on whether $\bar{n}$ approaches a limit or increases as N increases. Figure 7-1 illustrates the different possibilities for how $\bar{n}_1$ depends on N_1. Consequently, when N is incremented by one, the minimum increase in the resulting cost is a. Consider the largest value of N such that $\Phi P(N+1) - \Phi P(N) \leq a$. For all values of N greater than this value, it is no longer profitable to increase the buffer size. For all values of N less than this value it will be profitable to increase the buffer when

$$\Phi P(N + \delta N) - \Phi P(N) \quad \geq \quad a + b(\bar{n}(N + \delta N) - \bar{n}(N)) \tag{7.4}$$

If there is only one value of N_1 for which inequality (7.4) is an equality than $\Pi(N_1)$ will have a single maximum. Though we can not prove this all our numerical experiments indicate that $\Pi(N_1)$ will only have a single maximum.

Now we consider the three different cases discussed in Section 7.3.1. For the case where $\bar{n} \geq \frac{N}{2}$, each increase in N costs at least $\frac{b}{2}$, and the argument presented in the previous paragraph may be used to show that $\Pi(N)$ has a single maximum. The case where both $P(N)$ and $\bar{n}$ approach a limit as $N \to \infty$ is more difficult. To explain this case we define $\bar{n}(N)$ as the value of $\bar{n}$ when $N_1 = N$. If $P(N)$ approaches $P(\infty)$ faster than $\bar{n}(N)$ approaches $\bar{n}(\infty)$, $\Pi(N)$ will have a single maximum for all values of $a \geq 0$. An experiment performed on the two-machine line used in example 1, where $\mu_2 > \mu_1$, indicates that $P(N)$ approaches $P(\infty)$ faster than $\bar{n}(N)$. This is consistent with numerical experiments that we perform which show that $\Pi(N)$ has a single maximum.

If $P(N)$ approaches $P(\infty)$ slower than $\bar{n}(N)$ approaches $\bar{n}(\infty)$ there could be more than one maximum point for cases where $a > 0$ and $\Pi(N)$ would be unbounded for cases where $a = 0$. We have not encountered any cases that behave in this manner and we believe $\Pi(N)$ always has a single maximum.

For cases where $a > 0$, $\Pi(N)$ is always bounded. In addition, for practical cases there is a physical limitation on how large buffers may be.

The argument we use to show that $\Pi(N_1)$ has a single maximum may be extended to longer lines. The argument proceeds by assuming the present buffer allocation is the most profitable for a given N^{total}. The increase in revenue derived from increasing the buffer which maximizes revenue may be compared to the cost of increasing this buffer. At some point it would no longer be profitable to increase any of the buffers. This would be the buffer allocation that is most profitable, which is the single maximum of $\Pi(N_1, \ldots, N_{k-1})$.

7.5 Summary

In this chapter we formulate the profit maximization problem (PMP) and the constrained profit maximization problem (CPMP). In Section 7.3 we argue that using a gradient algorithm is warranted for this objective function if $\Pi(N_1, \ldots, N_{k-1})$ has a single maximum. We present numerical evidence as well as an intuitive argument why the assumption that $\Pi(N_1, \ldots, N_{k-1})$ has a single maximum is reasonable.

Chapter 8

PMP algorithm

In this chapter we describe an algorithm to solve the profit maximization problem as stated in (7.1). The algorithm, which we denote the PMP algorithm, is similar to the primal algorithm presented in Chapter 6. It is an iterative algorithm based on gradient methods. When constructing the PMP algorithm we assume that we can evaluate $\Pi(N_1, \ldots, N_{k-1})$ and that it has a single maximum.

Like the primal problem, the PMP is divided into two subproblems. These subproblems, which are denoted the CPMP and ODPMP, are formulated in (7.2) and (7.3). The algorithm which solves the CPMP problem evaluates $\Pi_{max}(N^{total})$. The CPMP algorithm performs the same task, in solving the PMP, as the dual algorithm performs when solving the primal problem. The major difference between the PMP and the primal algorithm is that the line search procedure used to search $\Pi_{max}(N^{total})$ for N^{opt}, where N^{opt} is the solution to the ODPMP, is different from the method used to search $P_{max}(N^{total})$ for N^*. The PMP algorithm first determines an interval which contains N^{opt}, then uses a binary search to find N^{opt}.

To describe the PMP algorithm in detail would require repeating a significant amount of the material presented in Chapters 3 and 5. Instead of reproducing this material we focus on the differences between the PMP and primal algorithms. A block diagram of the PMP algorithm is shown in Figure 8-1.

We begin by contrasting the dual and CPMP algorithms. Then we explain the method used to search the function $\Pi_{max}(N^{total})$.

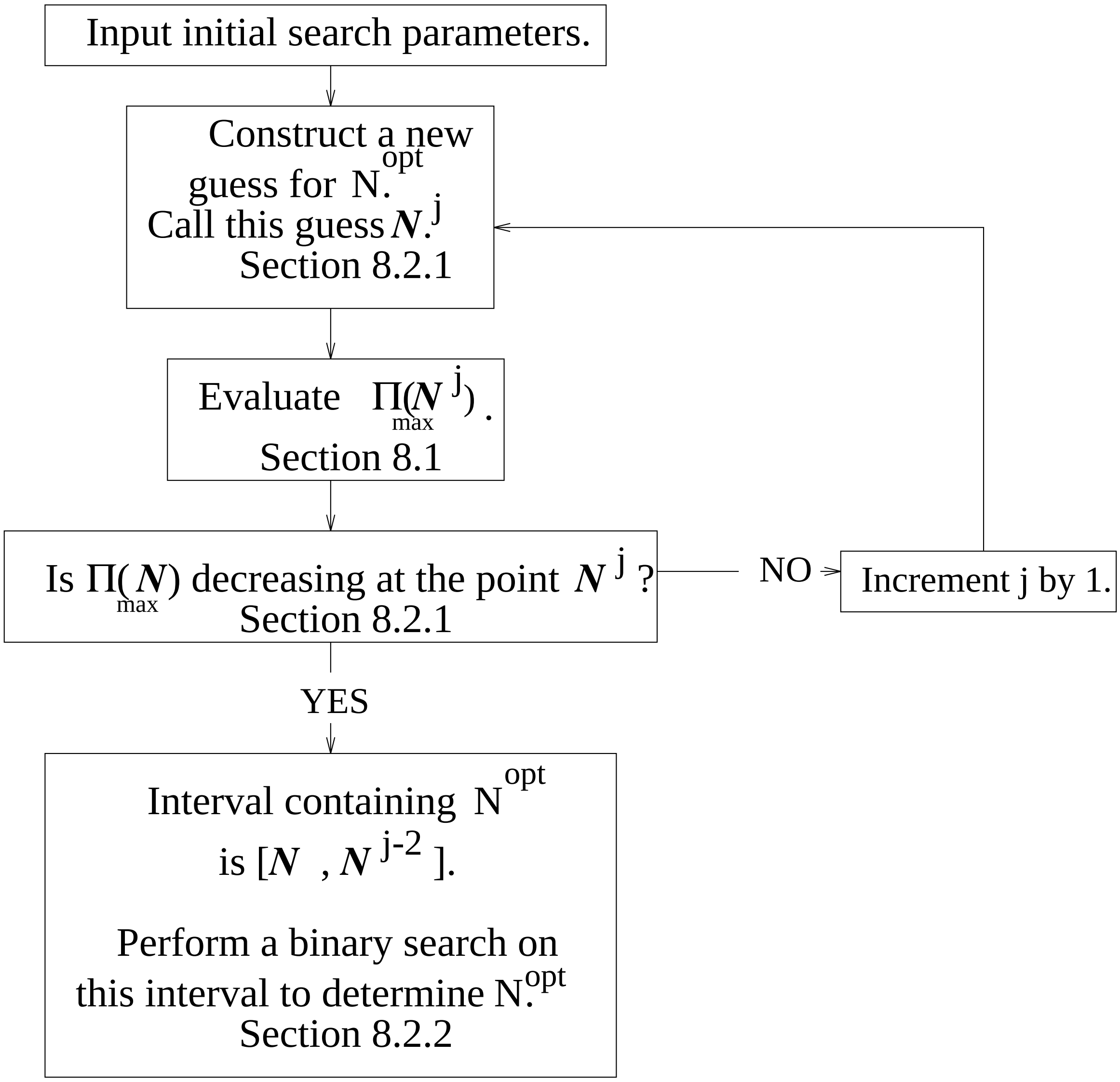

Figure 8-1: Block diagram of the PMP algorithm

8.1 CPMP algorithm

The CPMP and dual algorithms are similar. The only difference between these two algorithms is the way in which **g** is defined and calculated. When solving the CPMP problem we define and calculate **g** according to

$$g_i = \frac{\Pi(N_1, \ldots, N_i + \delta N, \ldots, N_{k-1}) - \Pi(N_1, \ldots N_i, \ldots, N_{k-1})}{\delta N} \tag{8.1}$$

Unlike in the dual algorithm, the gradient is always calculated the same way. This is because there is no reference to easily determine in which direction the next step is taken.

8.2 One dimensional profit maximization problem (ODPMP)

The solution to the ODPMP may be found by performing a one-dimensional linear search. The linear search method we develop is based on the property that $\Pi_{max}(N^{total})$ has a single maximum. In Section 7.4 we argue that $\Pi_{max}(N^{total})$ has this property. The linear search method also requires that we are able to evaluate $\Pi_{max}(N^{total})$. The CPMP algorithm evaluates $\Pi_{max}(N^{total})$ and determines the buffer distribution that achieves $\Pi_{max}(N^{total})$.

The goal is to find the value of N^{total} that maximizes $\Pi_{max}(N^{total})$. This value of N^{total}, which is called N^{opt}, has a cost of $\Pi_{max}(N^{opt}) = \Pi^{opt}$. The following paragraph describes the method used to determine N^{opt}.

Unlike the primal algorithm, no initial guess is required. The linear search routine first determines an interval which contains N^{opt}, then performs a binary search to find N^{opt}. The binary search routine we employ in the PMP algorithm, which is described in Section 8.2.2, is slightly different than the binary search routine of the primal algorithm (Section 5.1.3).

The linear search method of the primal problem would not work in the PMP algorithm for the following reason. In the primal problem we search for the minimum value of N^{total} such that $P_{max}(N^{total}) \geq P^*$. Therefore we are able to use P^*, and other information, to estimate N^*. In the PMP formulation we do not know the value of Π^{opt} so we are unable to estimate N^{opt} using the method described in Section 5.1.2.

8.2.1 Determining an interval

The major issue in constructing the procedure to search $\Pi_{max}(N^{total})$ is how to determine a value of N^{total} such that $N^{total} > N^{opt}$. This value of N^{total} is required to determine the interval which is searched. The procedure is constructed to take advantage of the knowledge that N^{opt} is generally increasing in k.

The method to estimate the end points of the interval which contains N^{opt} proceeds by guessing successively larger values for N^{total}. Let N^j denote the j^{th} guess of N^{total}. Successive guesses are calculated until (8.2) is satisfied.

$$\Pi_{max}(N^j) \leq \Pi_{max}(N^{j-1}) \tag{8.2}$$

The points N^{j-2} and N^j are the end points of the interval that contains N^{opt}. The procedure for determining successive guesses and the end points of this interval is as follows:

1. Let $N^0 = 0$, $N^1 = (factor)(k-1)$, and $j = 1$.
2. If (8.2) is satisfied go to 3. Otherwise increment j by 1 and set $N^j = (factor)N^{j-1}$.
3. If $j > 1$ the interval is $[N^{j-2}, N^j]$. Otherwise the interval is $[0, 4(k-1)]$.

In our implementation $factor$ is an input provided by the user. The default for $factor$ is 4. If N^{total} is expected to be large $factor$ should be reduced to avoid the region where $P(N_1, \ldots, N_{k-1})$ is very flat.

8.2.2 Binary Search

The binary search we use in the PMP algorithm differs from the binary search we use in the primal algorithm (Section 5.1.3) in one important respect. To explain the difference it is convenient to characterize the search method as dividing the line segment into two segments and discarding one of the segments. In the primal algorithm we determine which segment to discard by comparing $P_{max}(N^j)$ to P^*. If $P_{max}(N^j) < P^*$ we make N^j the minimum point of the new interval. Otherwise N^j is the maximum point. To determine which segment to discard for the PMP problem we determine the sign of the first derivative of $\Pi_{max}(N^{total})$ at the point N^j. To determine this we calculate the value of $\Pi_{max}(N^j - \delta N)$. In our implementation of the PMP algorithm $\delta N = 1$.

The binary search procedure uses two variables N^{over} and N^{under}. N^{over} is the best recorded guess such that $N^{over} > N^{opt}$ and N^{under} is the best recorded guess such that $N^{under} < N^{opt}$. N^l is the most recent guess of N^{opt}. The procedure that determines this interval is described in Section 8.2.1. The binary search is conducted as follows:[1]

1. Let $[N^{l-1}, N^l]$ be the interval that includes N^{opt} and let $N^{under} = N^{l-1}$ and $N^{over} = N^l$. Increment l by one.

2. The next guess is $N^l = (N^{over} + N^{under})/2$

3. Let $N^{l+1} = N^l - 1$. If $\Pi_{max}(N^{l+1}) < \Pi_{max}(N^l)$ then $N^{under} = N^l$. Otherwise $N^{over} = N^{l+1}$.

4. Increment l by 2. If $N^{over} - N^{under} \leq \delta N$ go to 5. Otherwise go to step 2.

5. If $\Pi_{max}(N^{under}) \geq \Pi_{max}(N^{over})$ then $N^{opt} = N^{under}$. Otherwise $N^{opt} = N^{over}$.

[1] The first time the binary search is invoked the end points are N^{j-2} and N^j. For this case we let the next guess $N^l = N^{j-1}$ because we have already evaluated $\Pi_{max}(N^{j-1})$.

8.2.3 Initializing the CPMP

When either the CPMP or dual algorithm is invoked a value must be specified for each N_i. The method we use in the primal algorithm (Section 5.2) to calculate this value of N_i is based on the property that $\mathbf{g} \geq 0$. When (8.1) is used to calculate g some, and possibly all, components of g may be negative. Consequently, (5.6) may no longer be used to calculate $\mathbf{N^{initial}}$. Let $\mathbf{N^{best}}$ be the most profitable guess that has been recorded so far and let N^j be the next guess for N^{opt}. The formula we use to calculate the initial guess, which is provided to the CPMP algorithm, is

$$
\begin{aligned}
\phi &= \frac{1}{N^{best}(N^j - N^{best})} \\
N^{initial} &= N^j \\
\mathbf{N^{initial}} &= \mathbf{N^{best}}(1+\phi)
\end{aligned} \tag{8.3}
$$

Before the interval including N^{opt} is determined $\mathbf{N^{best}} = \mathbf{N^{j-1}}$. After this interval has been determined the best recorded guess is not necessarily the last guess.

This method for calculating the next guess is based on the assumption that N_i^j/N^j will be close to N_i^{j-1}/N^{j-1}. We are requiring the vector $\mathbf{N^{initial}}$ to point in the same direction as $\mathbf{N^{j-1}}$.

8.3 Pseudo code for the PMP algorithm

In this section the pseudo code for the PMP algorithm is described.

1. Retrieve transfer line parameters, $factor$, and s.

2. Calculate the interval $[N^{j-2}, N^j]$ as described in Section 8.2.1.

3. Search $[N^{j-2}, N^j]$ for N^{opt} as described in Section 8.2.2.

4. Round $\mathbf{N^{opt}}$ according to the procedure outlined in Section 3.7 and call this vector $\mathbf{N^{final}}$. The answer to the PMP problem is $\mathbf{N^{final}}$ and the profitability of

the proposed line is $\Pi(\mathbf{N^{final}})$.

8.4 Implementation issues

ADDX and DDX algorithms The comments in Section 3.9 apply here as well.

Choosing δN In our implementation we choose to increment the total buffer space by 1 when calculating the gradient of $\Pi_{max}(N^{total})$. In the binary search routine we also choose the interval of uncertainty to have a length of 1. This seems the natural choice for integer buffer sizes but other values may be chosen and could be better. If δN is too small the noise from the evaluation algorithm will be too large for the PMP algorithm to work correctly.

Chapter 9

Behavior of the PMP algorithm

In this chapter we present examples to demonstrate that the PMP algorithm is a useful line design tool. As in previous chapters, the execution time of the algorithm, total buffer space required, and computed buffer distributions are reported. In this chapter all of the examples are done with the continuous material flow line model.

We begin by discussing the accuracy of the solutions in Section 9.1. Next, in Section 9.2.1, we examine the behavior of the execution time as the cost coefficient for throughput, Φ, increases. Then, in Section 9.2.3, we describe how the execution time of the algorithm varies when the length of a transfer line increases.

We know of no results in the literature that are appropriate for comparison with the PMP algorithm. Seong, Chang and Hong (1994b) solve the constrained profit maximization problem (CPMP) for a short, balanced line. We compare the solutions proposed by Seong *et al.* with the solutions found by the CPMP algorithm in Section 9.3.

9.1 Accuracy

We establish the accuracy of PMP solutions by comparing the output of the PMP algorithm with the results of an exhaustive search. The search is performed for three different five-machine lines. The reliability parameters, $r_i = 0.1$ and $p_i = 0.01$, are identical for every machine in all three lines. Table 9.1 describes the processing rates of the machines and the cost coefficients.

The transfer lines are chosen to parallel the three different buffer behaviors discussed in Section 7.4. Buffer 1, in line 1, will tend to fill, and buffer 4 in line 1 will tend to empty. Consequently, $\bar{n}_4$ should approach a limit as $N_4 \to \infty$. Line 2 is designed with faster downstream machines. The machines in line 3 are identical to the machines in line 2 but the cost coefficients differ. Table 9.2 describes the solutions to the PMP for all three lines. The solution to the ODPMP is N^{opt}. The results found for line 3 using an exhaustive search are listed as 3^{ES}. The solutions the PMP algorithm determines are identical to the optimal solutions found by the exhaustive search for lines 1 and 2. Line 3 is the most difficult case we can construct because there is no cost for buffer sizes and all average inventory levels should approach a limit as the buffer sizes increase because every machine is faster than the adjacent upstream machine. We explain why this is a difficult case in Section 7.4. For this case $a_i = 0 \; \forall \; i$. For line 3, the PMP algorithm rounds to a non-optimal solution, but the profit of the unrounded solution is larger than the profit of optimal integer solution[1].

	μ_i							
Line	1	2	3	4	5	$a_i \; \forall \; i$	$b_i \; \forall \; i$	Φ
1	1.03	1.01	1.02	1	1.04	1	1	1,000
2	1	1.01	1.02	1.03	1.04	1	1	1,000
3	1	1.01	1.02	1.03	1.04	0	1	1,000

Table 9.1: Description of three different five-machine lines with $r = 0.1$ and $p = 0.01$

To describe the solutions to the PMP, we introduce the notation, M_i^j, which represents machine i in line j. Lines 1 and 2 are identical except that M_1^1 and M_4^1 are switched. Line 1 has the faster machine at the beginning of the line. The values for Π in Table 9.2 show that line 1 is less profitable than line 2.

Notice the differences in the final buffer distributions for lines 1 and 2. To compensate for M_1^1 being faster than M_4^1, buffer 1 is smaller in line 1 than in line 2.

[1]The difference in the cost of the optimal integer solution and the rounded PMP solution for line 3 is approximately 0.0005%. If the profit of the line 3 is \$1,000,000 the difference in the cost calculated by the two methods would be \$5.40.

Line	Buffer i 1	2	3	4	N^{opt}	Π^{opt}	Two-machine evals.
1	4	13	15	10	43	737.16	10,209
2	6	14	15	8	43	740.78	13,191
3	16	39	47	47	149	813.24	23,724
3^{ES}	17	39	47	47	150	813.24	

Table 9.2: PMP solutions for different five-machine lines

Conversely, to compensate for M_4^1 being slower than M_1^1, N_4 is larger in line 1.

The computational effort required to determine a solution appears to be proportional to N^{opt}. This is not surprising because the segment of the curve $\Pi_{max}(N^{total})$ that is searched will generally be longer when N^{opt} is larger.

9.2 Behavior of the algorithm

This section describes examples that illustrate how the PMP algorithm behaves. The first examples show how computation time varies with the cost coefficients. Next, we consider how the computation time and the cost varies with the length of the line.

9.2.1 Increasing Φ

Figure 9-1 is a graph of computational effort vs. Φ for the unbalanced eight-machine line described in Figure 9.3. For this example $factor = 4$ (Section 8.2.1) and $s = 1$ (Section 3.4). Increasing Φ may be contrasted with increasing P^* in the primal problem because the values N^{opt} and N^* increase when Φ and P^* increase. The computational effort required to achieve an optimal solution is not always increasing in Φ. This is because the length of the segment of the curve $\Pi_{max}(N^{total})$, which is searched for N^{opt}, essentially determines the search time. The number of evaluations required is mainly influenced by the length of the segment of the curve on which we perform the binary search.

	Machines							
	1	2	3	4	5	6	7	8
r_i	0.08	0.1	0.1	0.12	0.08	0.11	0.11	0.1
p_i	0.01	0.011	0.011	0.01	0.01	0.011	0.011	0.011
μ_i	1	1.05	1.05	1.07	1.07	1.05	1.05	1.10
ρ_i	0.89	0.946	0.946	0.988	0.951	0.955	0.955	0.991
a_i	2	2	2	2	2	2	2	
b_i	1	1.1	1.2	1.3	1.4	1.5	1.5	

Table 9.3: Description of an eight-machine line

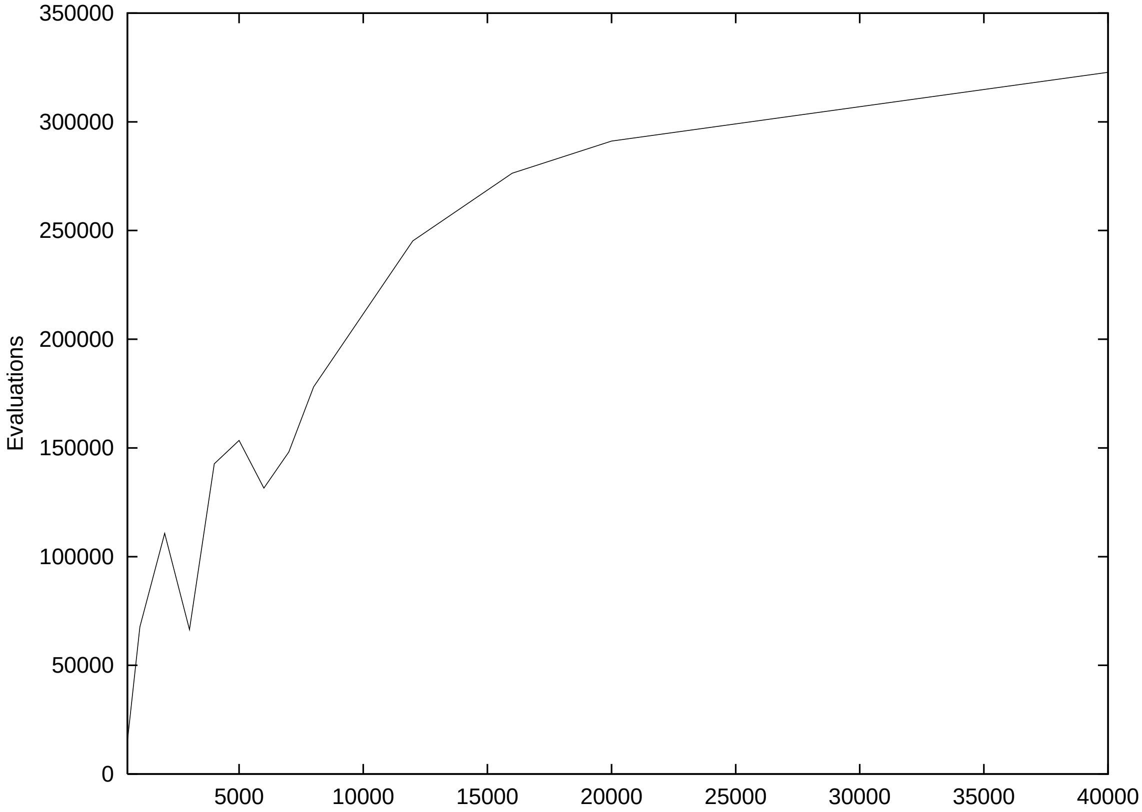

Figure 9-1: Two-machine evaluations vs. Φ

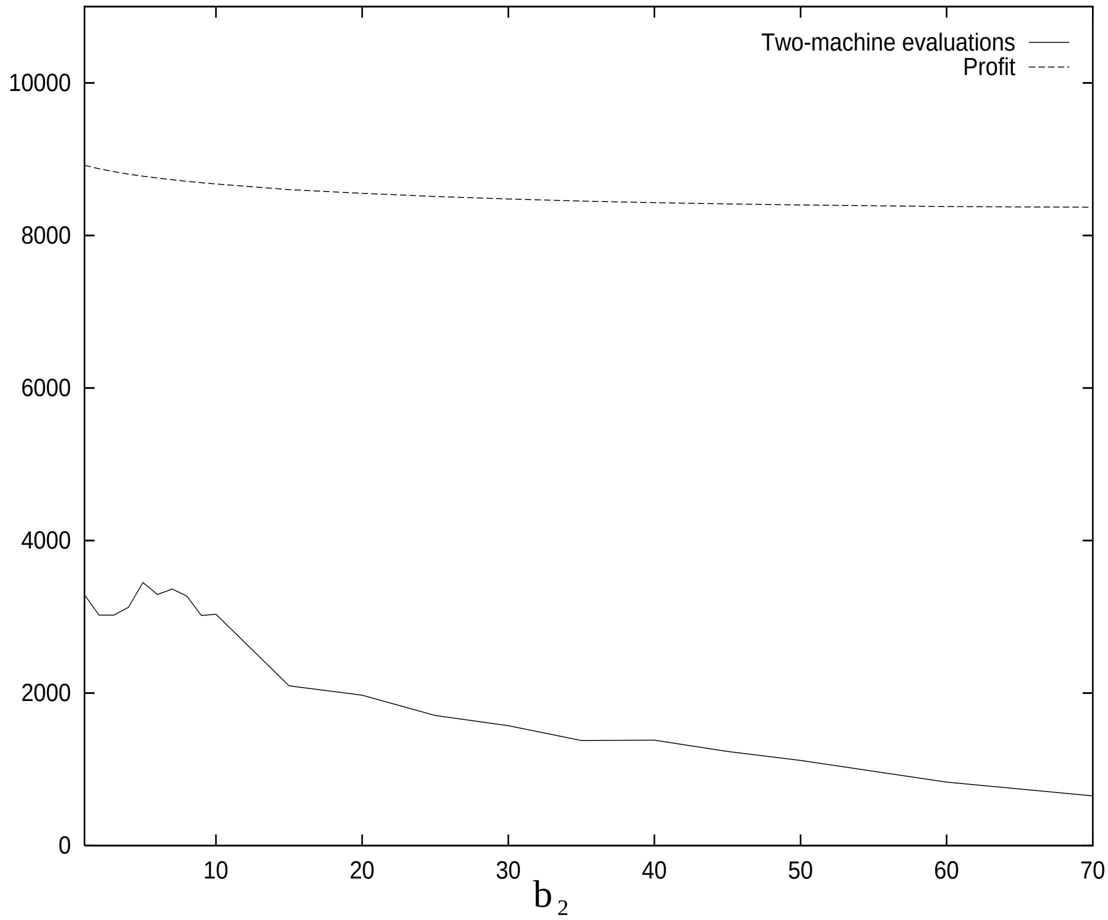

Figure 9-2: Two-machine evaluations and profit vs. b_2

9.2.2 Increasing b_i

Figure 9-2 is a graph of two-machine evaluations and profit vs. b_2 for a line consisting of the first three machines of line 3 in Table 9.2. For this example $\Phi = 3,000$ so that the variation in b_2 is reflected in both the computational effort and the solution. Both the computation time and profit decrease as b_2 increases. This decrease in profit is consistent with the argument presented in Section 7.4 concerning the tradeoff between the marginal increase in production rate vs. the marginal increase in inventory cost. The decrease in computation time is probably a consequence of the the decrease in N^{opt} when b_2 is increased.

Figure 9-3 is a graph of N^{opt}, N_2, $\bar{n}_2$, and $\bar{n}_1$ vs. b_2. As we would expect, as b_2 increases, both N^{total} and N_2 decrease. For this case, $N_1 = N^{total} - N_2$. N_1 also

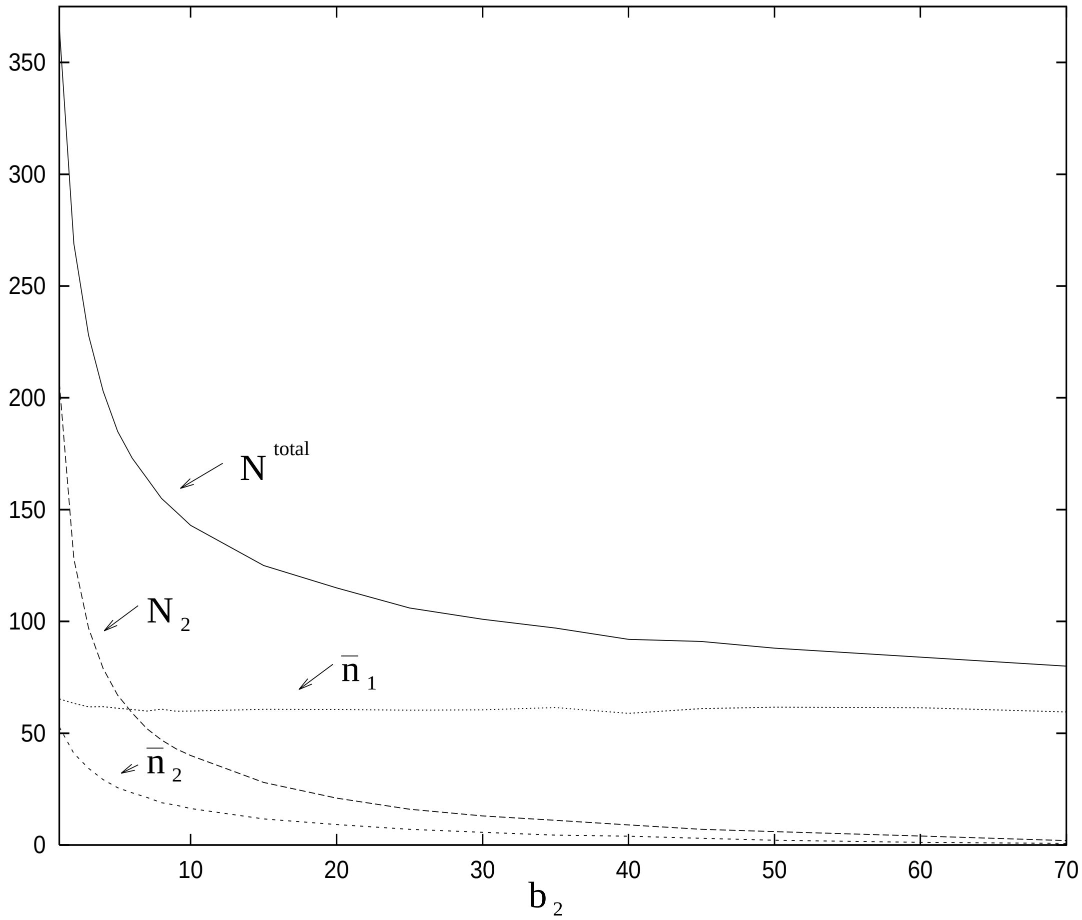

Figure 9-3: Buffer sizes and average inventory levels vs. b_2

decreases, but more slowly than N_2 and $N_1 \to 76$ when b_2 gets large. The behavior of $\bar{n}_1$ is surprising, because it seems to be independent b_2.

9.2.3 Long lines

Table 9.5 illustrates the impact increasing the length of a transfer line has on the computational effort. Each long line is composed of five-machine blocks. Each block is the line described in Table 9.4. $\Phi = 10,000$, $s = 1$ and $factor = 2$ for all of the cases. The significant increase in computational effort occurs for two reasons. The first is the increased number of two-machine evaluations the ADDX algorithm requires for convergence. The second is that each gradient calculation requires an additional long line evaluation for each added machine.

	Machines				
	1	2	3	4	5
r_i	0.08	0.1	0.1	0.12	0.08
p_i	0.01	0.011	0.011	0.01	0.01
μ_i	1	1.05	1.05	1.07	1.07
ρ_i	0.89	0.946	0.946	0.988	0.951
a_i	1	1	1	1	
b_i	.5	.55	.6	.65	

Table 9.4: Description of a five-machine line block

Number of machines	Two-machine evaluations	$\Pi(N_1, \ldots, N_{k-1})$	N^{total}
5	38871	8579	207
10	463224	8172	382
15	2044224	7878	447
20	5235678	7647	607

Table 9.5: Long line computation times

9.3 Comparison with the literature

In this section we compare the solutions the CPMP algorithm proposes with the solutions Seong, Chang and Hong (1994b) propose for a balanced five-machine line. We are unable to compare the efficiencies of the algorithms because Seong *et al.* do not report the computational effort required to achieve a solution.

For these cases buffer space has no cost, so $a_i = 0 \; \forall \; i$. In addition, the inventory cost of a part does not change when additional processing is done. Therefore $b_i = b \; \forall \; i$ and the cost of inventory is $b \sum_{i=1}^{k-1} \bar{n}_i$, where b is specified. Seong *et al.* also state that they find the optimal solution using simulation and report this result. We contrast the performance measures Seong *et al.* calculate with the performance measures calculated by the ADDX algorithm.

The CPMP problem is solved for a five-machine line, consisting of identical machines. The machine parameters are $r_i = 0.1$, $p_i = 0.1$, and $\mu_i = 1 \; \forall \; i$. (Notice that the repair and failure probabilities are equal. Therefore $\rho_i = e_i = 0.5 \; \forall \; i$ and $P < 0.5$. These are much less efficient machines than the machines in other examples.) The results are reported in Table 9.6. The table describes the buffer allocations propose by the three different optimization procedures. The procedures are the Seong *et al.* algorithm, simulation results which Seong *et al.* claim are optimal, and the CPMP algorithm. We do not know how Seong *et al.* verify the optimality of their simulated solutions. Table 9.6 reports the specified value of Φ and b for each case, and duplicates the performance measures of $\bar{n}_i$, $P(N_1, \ldots, N_{k-1})$ and $\Pi(N_1, \ldots, N_{k-1})$ calculated by Seong *et al.* The case designations correspond to the cases reported by Seong *et al.*. Table 9.7 describes the value of these performance measures when the proposed buffer allocations are evaluated using the ADDX algorithm. All proposed buffer distributions are evaluated using the ADDX algorithm so that comparisons are a fair as possible.

	Solution		Reported results		
Case	Procedure	$\mathbf{N}$	$\bar{\mathbf{n}}$	P	Π
6-a	Simulation	$(21, 29, 29, 21)$	$(13, 15.1, 13.9, 8.0)$	0.3942	0.3942
$\Phi = 1$	Seong *et al.*	$(21, 29, 29, 21)$	$(13, 15.1, 13.9, 8.0)$	0.3942	0.3942
$b = 0$	CPMP	$(21, 29, 29, 21)$	$(13, 15.1, 13.3, 8.2)$	0.3942	0.3942
6-b	Simulation	$(1, 11, 18, 70)$	$(0.56, 3.9, 5.15)$	0.3119	2.4657
$\Phi = 10$	Seong *et al.*	$(0, 20, 34, 46)$	$(0, 4.6, 5.3, 5.7)$	0.3215	2.4415
$b = 0.05$	CPMP	$(1, 11, 18, 70)$	$(0.57, 3.5, 3.9, .15)$	0.3119	2.4655
6-c	Simulation	$(14, 26, 31, 39)$	$(8.1, 11.1, 11.6, 9.1)$	0.3881	13.5323
$\Phi = 40$	Seong *et al.*	$(14, 25, 30, 31)$	$(8.1, 10.7, 10.9, 9.3)$	0.3871	13.5298
$b = 0.05$	CPMP	$(14, 26, 31, 29)$	$(8.1, 11.1, 11.2, 9.3)$	0.3878	13.5325
6-d	Simulation	$(0, 3, 8, 89)$	$(0, 1.1, 1.7, 2.9)$	0.2611	2.0414
$\Phi = 10$	Seong *et al.*	$(0, 1, 28, 71)$	$(0, 0.3, 2.7, 2.8)$	0.2589	2.0166
$b = 0.1$	CPMP	$(0, 3, 8, 89)$	$(0, 1.1, 1.7, 2.9)$	0.2611	2.0413
6-e	Simulation	$(7, 23, 32, 38)$	$(3.9, 7.8, 8.6, 8.5)$	0.3678	11.8333
$\Phi = 40$	Seong *et al.*	$(7, 23, 32, 38)$	$(3.9, 7.8, 8.6, 8.5)$	0.3678	11.8333
$b = 0.1$	CPMP	$(7, 23, 32, 38)$	$(3.9, 7.8, 8.6, 8.5)$	0.3677	11.8333
6-f	Simulation	$(7, 23, 32, 38)$	$(3.9, 7.8, 8.6, 8.5)$	0.3678	11.8333
$\Phi = 40$	Seong *et al.*	$(7, 23, 32, 38)$	$(3.9, 7.8, 8.6, 8.5)$	0.3678	11.8333
$b = 0.1$	CPMP	$(7, 23, 32, 38)$	$(3.9, 7.8, 8.6, 8.5)$	0.3677	11.8333

Table 9.6: Comparison of algorithms

In case 6-a, $b = 0$ and $\Phi = 1$. This case is the dual formulation we describe in Section 2.2. Notice that the production rate and profit are equal. For this case all three methods determine the same solution. The average inventory levels in buffers 3 and 4 reported by Seong *et al.* correspond to their simulation results but their $\bar{n}_3$ is

	Solution		ADDX results		
Case	Procedure	$\mathbf{N}$	$\bar{\mathbf{n}}$	P	Π
6-a	Simulation	$(21, 29, 29, 21$	$(13, 15.1, 13.3, 8.2)$	0.3942	0.3942
$\Phi = 1$,	Seong *et al.*	$(21, 29, 29, 21)$	$(13, 15.1, 13.3, 8.2)$	0.3942	0.3942
$b = 0$	CPMP	$(21, 29, 29, 21)$	$(13, 15.1, 13.3, 8.2)$	0.3942	0.3942
6-b	Simulation	$(1, 11, 18, 70)$	$(0.57, 3.5, 3.9, .15)$	0.3119	2.4655
$\Phi = 10$	Seong *et al.*	$(0, 20, 34, 46)$	$(0, 4.6, 5.3, 5.6)$	0.3215	2.4415
$b = 0.05$	CPMP	$(1, 11, 18, 70)$	$(0.57, 3.5, 3.9, .15)$	0.3119	2.4655
6-c	Simulation	$(14, 26, 31, 29)$	$(8.1, 11.1, 11.2, 9.3)$	0.3878	13.5325
$\Phi = 40$	Seong *et al.*	$(14, 25, 30, 31)$	$(8.1, 10.7, 10.9, 9.3)$	0.3871	13.5298
$b = 0.05$	CPMP	$(14, 26, 31, 29)$	$(8.1, 11.1, 11.2, 9.3)$	0.3878	13.5325
6-d	Simulation	$(0, 3, 8, 89)$	$(0, 1.1, 1.7, 2.9$	0.2611	2.0414
$\Phi = 10$	Seong *et al.*	$(0, 1, 28, 71)$	$(0, 0.3, 2.7, 2.8)$	0.2589	2.012
$b = 0.1$	CPMP	$(0, 3, 8, 89)$	$(0, 1.1, 1.7, 2.9)$	0.2611	2.0413
6-d	Simulation	$(7, 23, 32, 38)$	$(0, 1.1, 1.7, 2.9$	0.2611	2.0414
$\Phi = 10$	Seong *et al.*	$(0, 1, 28, 71)$	$(0, 0.3, 2.7, 2.8)$	0.2589	2.012
$b = 0.1$	CPMP	$(0, 3, 8, 89)$	$(0, 1.1, 1.7, 2.9)$	0.2611	2.0413
6-e	Simulation	$(7, 23, 32, 38)$	$(3.9, 7.8, 8.6, 8.5)$	0.3678	11.8333
$\Phi = 40$	Seong *et al.*	$(7, 23, 32, 38)$	$(3.9, 7.8, 8.6, 8.5)$	0.3678	11.8333
$b = 0.1$	CPMP	$(7, 23, 32, 38)$	$(3.9, 7.8, 8.6, 8.5)$	0.3677	11.8333

Table 9.7: Evaluation of Seong *et al.* results using the ADDX

7% larger and $\bar{n}_4$ is 2.5% smaller than the inventory levels calculated by the ADDX algorithm.

In case 6-b, the CPMP algorithm and simulation method determine the same solution. The Seong *et al.* algorithm proposes a different buffer allocation, which has a higher production rate but is less profitable. The performance measures calculated by the ADDX algorithm, for $\mathbf{N} = (0, 20, 34, 46)$, were identical to those reported by Seong *et al.*. The value of $\bar{n}_1$ calculated by the simulation differs from the value calculated by the ADDX algorithm by 1%.

The first two cases show that there is some disparity in the average buffer levels calculated using the ADDX algorithm and the levels reported by Seong *et al.*

In case 6-c, the total buffer space Seong *et al.* report for the simulation is 110. This is probably a typographical error[2], so we assume $N_4 = 29$. The CPMP algorithm and the simulation determine the same buffer allocation. The Seong *et al.* algorithm

[2]The total buffer space in all other eleven cases is 100.

determines a slightly different allocation but the difference in profitability is less than 0.02%.

In case 6-d, the cost of inventory has been increased and Φ has been decreased. The simulation and CPMP find the same solution. The difference in profit between the buffer allocation that the Seong *et al.* algorithm proposes and the allocation the two other methods propose is either 1.5% or 3%, depending upon which evaluation method is used.

In case 6-e, Φ is increased. The final allocations are identical for all three methods.

In case 6-f, b is decreased This case has the largest value of Φ and smallest value of b for cases 6-b thru 6-f. Consequently, the solution is closest to the solution for the dual, case 6-a. As in all the other cases, the CPMP algorithm and simulation determine the identical buffer allocations. The profitability of the solution that Seong *et al.* calculate differs by less than 0.02% from the solution calculated by the other two methods.

If the claims that Seong *et al.* make concerning the optimality of the simulated solution are true, and we think they are, the results in this section are further evidence that the CPMP algorithm is accurate. The results reported in this section demonstrate that the accuracy of the CPMP compares favorably with the accuracy of the algorithm Seong, Chang, and Hong (1994b) describe.

Chapter 10

Behavior of lines

In this chapter we use the primal and PMP algorithms to confirm and develop our intuition concerning the behavior of transfer lines. We begin by exploring how the relative processing rates and placement of machines, in a three-machine line, impact profitability. Then we hold ρ_i constant, and vary μ_i and r_i for the bottleneck machine in a three machine line. We also examine the sensitivity of both the primal and PMP algorithms to reliability parameters, processing rates, and cost coefficients. All experiments in this chapter are based on the continuous material flow model and use the ADDX algorithm to evaluate performance measures.

10.1 Machine placement along the line

In this section, we explore the effect of changing the order of the machines in a three-machine, continuous material line. Though this option is usually not available to the line designer after the machines have been built, the intuition developed in this exercise is valuable when determining the specification for a production line that has yet to be built. The PMP is solved for all possible configurations of the three machines described in Table 10.1. This is done for two different sets of cost coefficients.

The results are given in Table 10.2. The results show that the machine with the smallest value of ρ should be put at the start of the line. It is less clear whether it is more profitable to put the machine with the largest ρ at the the end of the line or in

Machine	r_i	p_i	μ_i	ρ_i
M_A	0.1	0.01	1.0	0.91
M_B	0.1	0.01	1.2	1.09
M_C	0.1	0.01	1.4	1.27

Table 10.1: Machine parameters

the middle.

Assume that, for processing reasons, either M_C or M_B must be the first machine. The cost coefficients for inventory may determine the placement of the machines. This can be seen by comparing configurations 3 with 4 and 5 with 6 for cases 1 and 2. For cases where inventory is inexpensive we should put the machine with the largest ρ in the middle of the line because this is the place where disruptions are greatest. This is similar to a work force allocation problem[1]. For cases where inventory is expensive, placing the machine with the larger ρ at the end of the line will be more profitable. Most of the total buffer space, for configurations 3 and 5, is allocated to buffer 2. Since M_3 is faster than M_2, $\bar{n}_2$ is small.

	Case 1 $\Phi = 5000$ $a_1 = a_2 = 1$ $b_1 = b_2 = 1$		Case 2 $\Phi = 5000$ $a_1 = a_2 = 0.1$ $b_1 = 2$ $b_2 = 2.5$	
Configuration	$\Pi_{max}(N^{total})$	N^{total}	$\Pi_{max}(N^{total})$	N^{total}
1. $M_A M_B M_C$	4460	59	4517	99
2. $M_A M_C M_B$	4473	52	4522	84
3. $M_B M_A M_C$	4408	67	4444	93
4. $M_B M_C M_A$	4437	42	4422	39
5. $M_C M_A M_B$	4408	68	4445	99
6. $M_C M_B M_A$	4423	47	4410	44

Table 10.2: Profitability of various three-machine configurations

[1]See Hillier *et al.* (1966) or Buzacott and Shantikumar (1993).

10.2 Varying r, and μ for a constant ρ and p

In this section we present a result that demonstrates that increased variability adversely impacts profitability. This observation is well known[2] for systems in general and may be shown in a variety of ways including a queueing analysis. The purpose for presenting the result in this context it to show that the PMP algorithm may be used to confirm our intuition about the behavior of the transfer line models we study in this thesis and to quantify the impact of changing parameters.

Figure 10-1 is a graph of profit vs. r_2 for a three-machine line. The first machine in the line is M_C and the third machine is M_B, both machines are described in Table 10.1. As r_2 is increased μ_2 is decreased so that both $\rho_2 = \frac{\mu_2 r_2}{r_2+p_2} = 0.91$ and $p_2 = 0.01$ are constant. The efficiency of machine 2 is $e_2 = \frac{r_2}{r_2+p_2}$.

Consequently, when $r_2 \gg p_2$, increasing r_2 does not substantially increase the efficiency, which is one indication of the variability. Since ρ_2 is constant, increasing r_2 does not affect μ_2 either. Therefore profit increases very little when $r_2 \gg p_2$. Profit is increased only in cases where increasing r_2 increases e_2 appreciably, i.e. where r_2 is comparable to p_2.

10.3 Sensitivity to errors in estimating r and p

In this section we describe the results of two experiments. The experiments are performed to develop our intuition concerning the impact of errors in estimates of the reliability parameters. The influence that these errors have on the optimal value of the objective function is explored. Understanding the impact of errors in estimating parameters is important when assessing the actual vs. predicted performance of a transfer line.

[2]See Gershwin (1994) for a discussion of the effect of machine variability on efficiency for a two-machine deterministic processing time line.

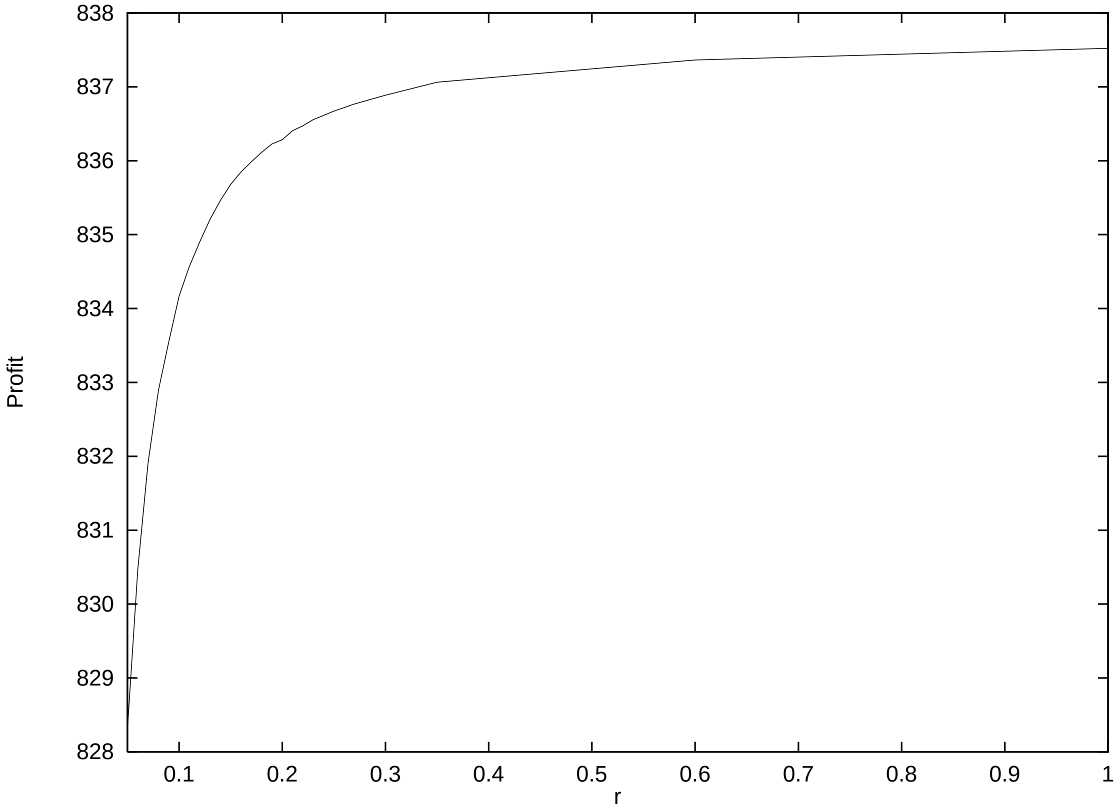

Figure 10-1: Profit vs. r_2 for constant p_2 and ρ_2

10.3.1 Impact of estimation errors with increasing P^*

In this section we examine how the cost of the optimal solution of the primal problem is influenced by changes in repair rates for different values of P^*. Figure 10-2 is a graph of the error in N^* vs. P^*, caused by overestimating r_5 or underestimating p_5, for the nine-machine, continuous material line described in Table 10.3. The estimation errors due to errors in r_5, which are defined in (10.1), are calculated according to the following procedure:

1. Calculate N^* for $r_5 = 0.1$ and $p_5 = 0.01$ ($\rho_5 = 0.91$). Call this value N^{base}.
2. Calculate N^* for $r_5 = 0.09$ and $p_5 = 0.01$ ($\rho_5 = 0.90$). Call this value N^{error}.
3. The definition of the error in N^* is

$$E_N^r = \frac{N^{error} - N^{base}}{N^{base}} \tag{10.1}$$

The estimation errors for p_5 are calculated in an analogous manner. We have chosen to vary the reliability parameters for machine 5 because it is the slowest machine in the line. Therefore, changing r_5 and p_5 will have the largest impact. The change in total buffer space, when r_5 is decreased by 10%, is approximately 5% for values of $P^* \ll P(\infty)$, but the errors increase significantly as $P^* \to P(\infty)$. The same behavior is true for errors in p_5, but the error in N^{total} is smaller. The errors are smaller when we vary p_5 because the change in ρ_5 is less. When r_5 is changed, ρ_5 decreases from 0.9091 to 0.9000. When p_5 is increased by 10% ρ_5 decreases from 0.9091 to 0.9009.

	Machines								
	1	2	3	4	5	6	7	8	9
r_i	0.1	0.1	0.1	0.1	r_5	0.1	0.1	0.1	0.1
p_i	0.01	0.01	0.01	0.01	p_5	0.01	0.01	0.01	0.01
μ_i	1.01	1.02	1.03	1.04	1.00	1.01	1.02	1.03	1.04
ρ_i	0.918	0.927	0.936	0.945	$\frac{r_5}{r_5+p_5}$	0.918	0.927	0.936	0.945

Table 10.3: Description of a nine-machine line

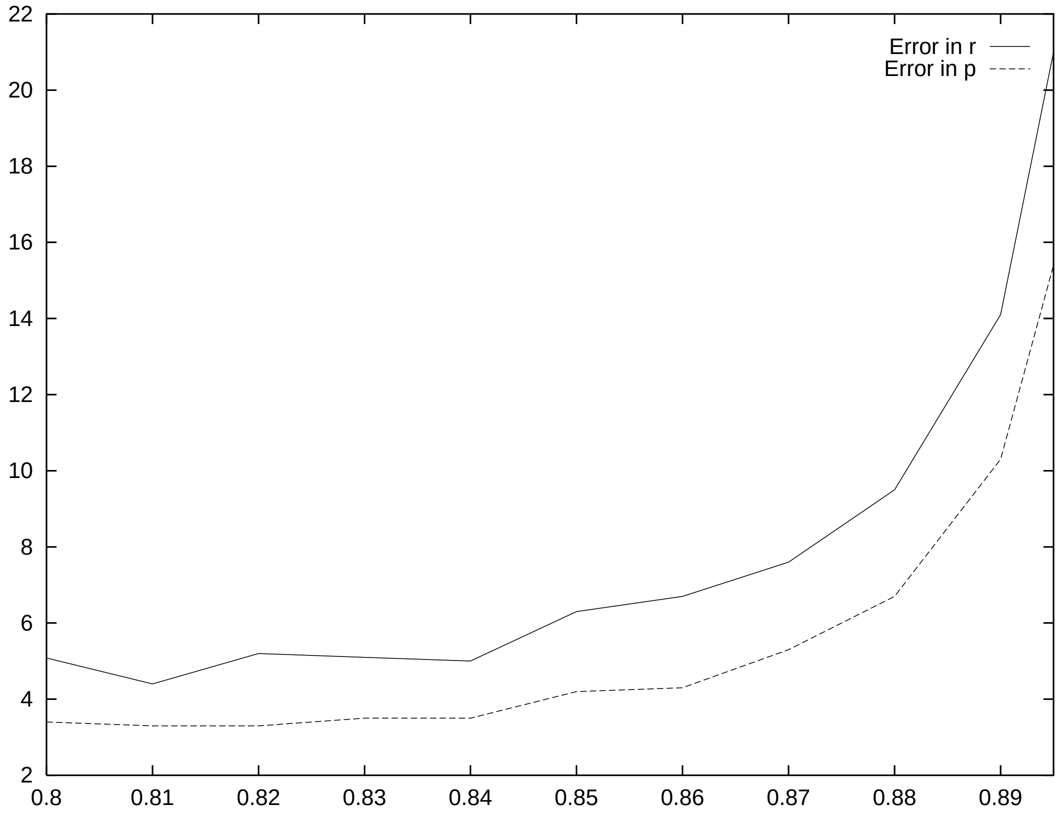

Figure 10-2: Errors in N^* vs. P^*

An alternative method for describing the impact of errors in estimating reliability parameters would be to examine how the buffer sizes in the vicinity of machine 5 change. Table 10.4 shows how the buffer distribution changes for two different values of P^*. In all cases, the size of buffer i is non-decreasing in decreasing r_5. The buffers at the beginning and end of the line do not change appreciably when r_5 is decreased. The buffers close to machine 5 account for most of the change in N^*. The changes in N_4 and N_5, for cases 1 and 2, are approximately 8% and 25% respectively. The changes in N_1, N_2, N_8 and N_9 are very small.

			Buffer i								
Case	P^*	r_5	1	2	3	4	5	6	7	8	N^*
1	0.85	0.1	20	29	31	37	38	35	29	16	235
	0.85	0.09	20	30	34	40	42	37	30	17	250
2	0.89	0.1	57	67	68	86	101	86	66	41	572
	0.89	0.09	58	69	75	110	136	96	68	41	653

Table 10.4: Buffer allocation for different values of P^* and r_5

10.3.2 Impact of estimation errors in r on Π^{opt}

In this section we describe how changing r impacts Π^{opt}, the maximum profitability of the transfer line. We perform the experiment on the nine-machine line described in Table 10.3. For this experiment, $\Phi = 2,000$ and $a_i = b_i = 1 \; \forall \; i$. Table 10.5 describes the results.

Case 1 is the reference case. In case 2 we reduce r_5 by 10%. N^{opt} does not change, but the profitability of the transfer line decreases. In case 4 we reduce r_9. The profitability of the line decreases approximately 50% as much as when the bottleneck machine is made less reliable. In cases 3 and 5, we increase the the reliability of machines 5 and 9 respectively. Π^{opt} increases in both cases, but the increase is approximately 50% larger in the case where the performance of machine 5 is improved. This is to be expected since machine 5 is the bottleneck. When we reduce r_9 ρ_9 is reduced to 0.927. Machine 9 is not the bottleneck but reducing the repair rate does decrease profit. For this case, the behavior of the line is impacted by perturbing r_i for both the machine with the largest and smallest values of ρ.

Case	Buffer i 1	2	3	4	5	6	7	8	N^{total}	Π^{opt}
1. $r_5 = 0.1$	4	14	17	19	20	18	16	8	116	1427
2. $r_5 = 0.09$	4	14	17	19	20	19	15	8	116	1419
3. $r_5 = 0.11$	5	14	16	19	19	18	16	9	116	1433
4. $r_5 = 0.1$ and $r_9 = 0.09$	4	14	17	19	20	19	17	10	120	1422
5. $r_5 = 0.1$ and $r_9 = 0.11$	5	14	17	19	20	19	15	7	116	1430

Table 10.5: Buffer allocation and Π^{opt} for different values of r

Chapter 11

Conclusions

11.1 Summary

This thesis formulates four transfer line design problems. The goal of the first problem, designated the *primal*, is to choose the smallest total buffer space that meets or exceeds a production rate target. The second problem, denoted the *dual*, seeks to choose individual buffers sizes, for a given total buffer space, so that production rate is maximized. The objective of the third problem, called the *profit maximization problem (PMP)*, is to choose the individual buffer sizes that maximize the profitability of the transfer line. The fourth problem is the *constrained profit maximization problem (CPMP)*. The goal in this problem is to maximize profitability for a specified total buffer space. In all four problems, the machines are specified and the goal is to determine optimal buffer sizes. Four efficient, accurate, optimization algorithms, denoted the primal, dual, PMP, and CPMP algorithms are constructed to solve these problems. All four are based on gradient methods.

These four problems are solved using two different transfer line models. The models are the deterministic processing time model, for cases where all machines have identical cycle times, and the continuous material flow model, for cases where the cycle times of the machines are different. We employ the DDX (Dallery, David, and Xie 1988) algorithm to evaluate the performance measures for the deterministic processing time model. The ADDX (Burman 1995) algorithm is used to evaluate the performance measures for the continuous material flow model. These algorithms use analytical methods and decomposition (Gershwin 1987,1994) to determine the performance measures of the transfer line. These algorithms are efficient and able to evaluate transfer lines consisting of many machines quickly. We know of no other practical method for evaluating long transfer lines.

After formalizing the problems and describing the models and problems we describe the four algorithms. The formulation of both the primal (2.1) and dual (2.2) problems is comparable to other formulations that have appeared in the literature. We only know of one paper in the literature which addresses the dual problem and two papers which address the primal problem which are reasonable candidates for comparison with the present work. We contrast our results with the results presented in these papers in Sections 4.3.and 6.3. The comparison indicates that both the primal and dual algorithms outperform the other algorithms in all of the cases we study.

We do not know of any papers that explicitly address the unconstrained profit maximization problem as formulated in this thesis. Therefore, we are unable to compare its performance to alternative algorithms. One paper does address the CPMP and we compare the results reported in this paper with our solutions in Chapter 9. All of the algorithms help develop intuition about transfer lines and general manufacturing systems. In Chapter 10 we demonstrate how these algorithms may be used to build intuition.

11.2 Further research

11.2.1 Alternative algorithms

In Chapter 6 we discuss methods for improving the primal algorithm. One idea is to only do one gradient calculation and one line search for each instance of the dual. This algorithm was implemented. Preliminary results indicate that this modification reduces the computational effort by approximately 30%. In the examples with transfer lines consisting of less than 15 machines this method calculates a total buffer space which was identical to the total buffer space the primal algorithm calculates. When the 20 machine line of Section 6.3.2 is optimized this method computes a solution with 2% more buffer space than the primal finds. These results indicate that for approximate solutions this method may be acceptable. Alternatively, the primal algorithm can be modified only in the early stages of the algorithm, each dual instance performs one gradient calculation and one line search. This modification was done but the method to determine when the primal algorithm is in the early stages needs to be improved.

Though not discussed in this thesis, alternative methods for approximating $P_{max}(N^{total})$ were also tried. $P_{max}(N^{total})$ was approximated as an exponential function. When a guess for N^* satisfied the production rate constraint, a binary search is performed. In certain cases this method outperformed the primal algorithm. The reason we choose to use a linear approximation is that we want to avoid the part of $P_{max}(N^{total})$ where gradients are difficult to calculate.

11.2.2 Alternative problem formulations

Gershwin and Goldis (1995) propose many different problem formulations and present a general formulation which encompasses all the problems we study in this thesis. The specific modifications to our problem formulations that we are interested in are

- Including a production rate constraint in the profit maximization problem.
- Including the option for choosing the reliability parameters of the individual machines or choosing the machines from a pool of candidate machines for all three formulations.
- Including a fixed cost for buffers greater than size zero in all three formulations.

11.2.3 Model extensions

An important extension is to construct an assembly/disassembly model, for machines with different cycle times, and incorporate this model in the optimization algorithms we describe in this thesis.

Bibliography

[1] I. Adan and J. Van Der Wal (1989), "Monotonicity of the Throughput in Single Server Production and Assembly Networks with Respect to the Buffer Sizes," *Queueing Networks with Blocking,* H. G. Perros and T. Altiok, Editors, Elsevier Science Publishers, pp. 345-356, 1989.

[2] T. Altiok and S. Stidham Jr. (1983), "The Allocation of Interstage Buffer Capacities in Production Lines," *IEE Transactions,* **15** pp. 292-299.

[3] V. Anantharam and P. Tsoucas (1990), "Stochastic Concavity of Throughput in Series of Queues with Finite Buffers," *Advances in Applied Probability*, Vol. 22, pp. 761-763, 1990.

[4] D. R. Anderson and C. L. Moodie (1969), "Optimal Buffer Storage Capacity in Production Line Systems," *International Journal of Production Research*, Vol. 7 No. 3, pp. 233-240, 1969.

[5] K. Barten (1962), "A Queueing Simulator for Determining Optimum Inventory Levels in a Sequential Process," *Journal of Industrial Engineering*, Volume 13, No. 4, July-August, 1962, pp. 245-252.

[6] M. Burman (1995), "New Results in Flow Line Analysis", PhD Dissertation, M.I.T, Cambridge Ma.

[7] J. A. Buzacott (1967), "Automatic Transfer Lines with Buffer Stocks," *International Journal of Production Research*, Vol. 5, No. 3, pp. 183-200.

[8] J. A. Buzacott and J. G. Shanthikumar (1993), *Stochastic Models of Manufacturing Systems*, Prentice Hall, 1993.

[9] S. P. Bradley, A. C. Hax, T. L. Magnanti (1977), *Applied Mathematical Programming*, Addison-Wesley, 1977.

[10] M. Caramanis (1987), "Production System Design: A Discrete Event Dynamic System and Generalized Benders' Decomposition Approach," *International Journal of Production Research*, Vol. 25, No. 8, pp. 1223-1234, 1987.

[11] We-Min Chow (1987), "Buffer Capacity Analysis for Sequential Production Lines with Variable Processing Times," *International Journal of Production Research*, Vol. 25, No. 8, pp. 1183-1196, 1987.

[12] Y. Dallery and S. B. Gershwin (1992), "Manufacturing Flow Line Systems: A Review of Models and Analytical Results," *Queueing Systems Theory and Applications, Special Issue on Queueing Models of Manufacturing Systems*, Volume 12, No. 1-2, December, 1992, pp. 3-94.

[13] Y. Dallery, R. David, and X.-L. Xie (1987), "Approximate Analysis of Transfer Lines with Unreliable Machines and Finite Buffers," Technical Report LAG Number 87-64, June, 1987, Grenoble.

[14] Y. Dallery, R. David, and X.-L. Xie (1988), "An Efficient Algorithm for Analysis of Transfer Lines with Unreliable Machines and Finite Buffers," *IIE Transactions,* Vol. 20, No. 3, pp. 280-283, September, 1988.

[15] Y. Dallery, R. David, and X.-L. Xie (1989), "Approximate Analysis of Transfer Lines with Unreliable Machines and Finite Buffers," *IEEE Transactions on Automatic Control*, Vol. 34, pp. 943-953.

[16] Y. Dallery, Z. Liu and D. Towsley (1994), "Equivalence, Reversibility, Symmetry and Concavity Properties in Fork/Join Queueing Networks with Blocking," to appear in *J. ACM.*

[17] E. Dogan and T. Atiok (1995), "Gradient Estimation of the Output Rate in Transfer Lines," *preprint*, Rutgers University, Dept. of Industrial Engineering

[18] S. B. Gershwin and I. C. Schick (1983), "Modeling and Analysis of Three-Stage Transfer Lines with Unreliable Machines and Finite Buffers," *Operations Research,* Vol. 31, No. 2, March-April, 1983, pp. 354-380.

[19] S. B. Gershwin (1987), "An Efficient Decomposition Method for the Approximate Evaluation of Tandem Queues with Finite Storage Space and Blocking," *Operations Research*, pp. 291-305, March-April, 1987.

[20] S. B. Gershwin (1994), *Manufacturing Systems Engineering,* Prentice-Hall, 1994.

[21] S. B. Gershwin and Y. Goldis (1995), "Efficient Algorithms for Transfer Line Design", Laboratory for Manufacturing and Productivity Report LMP-95-005, 1995.

[22] P. Glasserman and D. D. Yao (1992), "Structured Buffer-Allocation Problems," *Journal of Discrete Event Dynamic Systems,* to appear.

[23] J. M. Hatcher (1969), "The Effect of Internal Storage on the Production Rate of a Series of Stages Having Exponential Service Times," *AIIE Transactions*, Volume 1, Number 2, June, 1969, pp. 150-156.

[24] F. S. Hillier, and R. W. Boling (1966), "The Effect of Some Design Factors on the Efficiency of Production Lines with Variable Operation Times," *Journal of Industrial Engineering*, Volume 17, pp. 651-658.

[25] F. S. Hillier and K. C. So (1991), "The Effect of Machine Breakdowns and Interstage Storage on the Performance of Production Line Systems," *International Journal of Production Research*, Vol. 29, No. 10, pp. 2043-2055, 1991.

[26] F. S. Hillier, K. C. So, and R. W. Boling (1993), "Toward Characterizing the Optimal Allocation of Storage Space in Production Line Systems with Variable

Processing Times," *Management Science*, Volume 39, Number 1, January, 1993, pp. 126-133.

[27] Y. C. Ho, M. A. Eyler, and T. T. Chien (1979), "A Gradient Technique for General Buffer Storage Design in a Production Line," *International Journal of Production Research*, Vol. 17, No. 6, pp. 557-580.

[28] Y. C. Ho, M. A. Eyler, and T. T. Chien (1983), "A New Approach to Determine Parameter Sensitivities of Transfer Lines," *Management Science*, Volume 29, Number 6, June 1983, pp. 700-714.

[29] D. Jacobs and S. M. Meerkov (1993), "A System-Theoretic Property of Serial Production Lines: Improvability," The University of Michigan College of Engineering Control Group Reports, Report No. CGR-93-1, January, 1993.

[30] D. Jacobs and S. M. Meerkov (1994), "System-Theoretic Properties of the Process of Continuous Improvement in Production Systems," *Proceedings of the American Control Conference*, Baltimore, Maryland, June, 1994, pp. 3323-3327.

[31] M. A. Jafari, J. G. Shanthikumar (1989), "Determination of Optimal Buffer Storage Capacities and Optimal Allocation in Multistage Automatic Transfer Lines," *IIE Transactions*, Vol. 21, No. 2, pp. 130-134, June, 1989.

[32] C.-T. Kuo, J.-T. Lim and S. M. Meerkov (1994), "Improvability Analysis of a Machining Transfer Line: An Application," The University of Michigan College of Engineering Control Group Reports, Report No. CGR-94-08, April, 1994.

[33] C.-M. Liu and C.-L. Lin (1994), "Performance Evaluation of Unbalanced Serial Production Lines," *International Journal of Production Research*, Vol. 32, No. 12, pp. 2897-2914.

[34] D. G. Luenberger (1984), *Linear and Nonlinear Programming,* Addison Wesley, 1984.

[35] L. E. Meester and J. G. Shanthikumar (1990), "Concavity of the Throughput of Tandem Queueing Systems with Finite Buffer Storage Space," *Advances in Applied Probability*, Vol. 22, pp. 764- 767, 1990.

[36] Okamura, K. and Yamashina, H. Analysis of the effect of buffer storage capacity transfer line systems. AIEE Transactions, **11** (2), 206-224, 1977.

[37] T. Park (1993), "A Two-Phase Heuristic Algorithm for Determining Buffer Sizes of Production Lines," *International Journal of Production Research*, Vol. 31, No. 3, pp. 613-631, 1993.

[38] E. L. Plambeck, B.-R. Fu, S. M. Robinson, and R. Suri (1993), "Throughput Optimization in Tandem Production Lines Via Nonsmooth Programming," *Proceedings of the 1993 Summer Computer Simulation Conference*, Boston, 1993.

[39] D. Seong, S. Y. Chang, and Y. Hong (1994a), "Heuristic Algorithms for Buffer Allocation in a Production Line with Unreliable Machines," Department of Industrial Engineering, POSTECH, Korea; accepted for publication, *International Journal of Production Research.*

[40] D. Seong, S. Y. Chang, and Y. Hong (1994b), "An Algorithm for Buffer Allocation with Linear Resource Constraints in a Continuous Flow Production Line," Department of Industrial Engineering, POSTECH, Korea, Technical Report IE-TR-94-05.

[41] J. G. Shanthikumar and D. D. Yao (1989), "Monotonicity and Concavity Properties in Cyclic Queueing Networks with Finite Buffers," *Queueing Networks with Blocking,* H.G. Perros and T. Altiok, Editors, Elsevier Science Publishers, pp. 325-344, 1989.

[42] R. Suri and B. Fu (1994), "On using Continuous Flow Lines to Model Discrete Production Lines" *Discrete Event Dynamic Systems*, Vol. 4, No. 2, pp. 129-169