Fluid Approximation and other Methods for Hard Combinatorial Optimization Problems

Tal Raviv

Fluid Approximation and other Methods for Hard Combinatorial Optimization Problems

Submitted in Partial Fulfillment of the

Requirements for the
Degree of Doctor of Philosophy

Tal Raviv

Submitted to the Senate of

the Technion - Israel Institute of Technology

Kislev, 5764 HAIFA November 2003

The Research Thesis Was Done Under The Supervision of Michal Penn and Shmuel Onn in the Faculty of Industrial Engineering and Management.

My deepest gratitude to my wife Ravit and my parents Ayala and Michael for supporting me along the way of completing this dissertation and for urging me to start it in the first place.

Special thanks to Professor Michal Penn for being an extremely efficient, devoted and pleasant advisor. From the joint work with Michal I learned how to conduct scientific studies and how to publish them. The opportunities Michal created for me are my spring board to academic career. I hope I will have the chance to continue to cooperate with her in the years to come.

Thanks for Professor Shmuel Onn who adopted me right from my first semester at the Technion. I learned a lot from my work with him and enjoyed it very much.

Thanks for professor Gideon Weiss from the University of Haifa for his valuable comments regarding this dissertation.

Finally, thanks for Professors Boaz Golany, Moshe Kress, Avishay Mandelbaum and Reuven Rubinstein and from Technion and to Professor Gideon Weiss from The University of Haifa. The scientific collaboration with them during my years at the Technion was an enriching experience.

THE GENEROUS FINANCIAL HELP OF THE TECHNION IS GRATEFULLY

ACKNOWLEDGED.

Contents

List of Figures

List of Tables

Abstract

This dissertation consists of two parts. The first part deals with two problems related to manufacturing. Namely, optimal configuration of quality control stations in a serial production line and long run profit maximization of a job shop system. The second part deals with the *separation problem*.

The first part includes three chapters. In the first chapter we study unreliable serial production lines with known failure probabilities for each operation. Each such production line consists of a series of stations; existing machines and optional quality control stations (QCS). Our aim is to decide where and if to install the QCSs on the line, so as to maximize the expected net profit per time unit from the system.

Following some recent studies, where a fluid approach was used to solve combinatorial optimization problems (mainly scheduling problems), we extend the set of such problems and use the fluid view to analyze and optimize the performance of a QCS system.

We use two methods: mixed integer programming and dynamic programming to solve the fluid counterpart of the QCS problem. Numerical experiments were conducted to demonstrate and compare the practical efficiency of these methods. As observed, the dynamic programming approach is more efficient than the MIP approach. The computational complexity of both methods is unknown. We then present a fully polynomial-time approximation scheme for the problem that uses a simpler dynamic program as a subroutine.

Finally we show how to use the solutions of the fluid counterpart problem to obtain optimal solutions for the discrete QCS problem where inventory costs are not taken into account. Furthermore, we show how the fluid approach can be used to construct approximation methods for the more general case where inventory costs are incorporated in the model.

In the second chapter we extend the QCS model to include work in process holding cost, arbitrary revenue function and capital cost of QCSs. We present a practical method to approximate the solution of very large instances of the problem using branch and bound strategy with the developed dynamic programming procedure as a subroutine.

The last chapter in the first part uses the fluid approximation approach to solve

the problem of constructing an optimal product mix and cyclic dispatching rule simultaneously in order to maximize the long run average profit per time unit obtained from a given job shop system. A fluid counterpart model is presented and solved as a simple linear program model. A dispatching rule based on the fluid solution is presented and proved to be optimal once sufficient safety stocks are provided. Methods to calculate and reduce the minimum required safety stock are then presented.

The second part of the dissertation includes a single chapter which deals with the following problem: Given a pair of disjoint subsets S and T of some universe U, construct a collection of rules that for any given point p in U determine whether or not p is close to S and far from T in some sense. One can study this problem in two different levels: providing these rules for some collections of sets; and designing a mechanism that constructs such a collection automatically for any given set.

Applications of this problem arise in the design of decision support systems for medical diagnostic, risk assessment of credit and others. Various methodologies for dealing with this problem have been considered in the literature, including statistics, neural networks, Logical Data Analysis and others.

In this chapter we present a different approach, based on the construction of a system of linear equalities and inequalities to separate the sets. In many cases this method allows a description of complex sets using a small number of equalities and inequalities.

Each of the four chapters is a self contained and is to be submitted for publication separately.

Notations and Abbreviations

$\lfloor x \rfloor$	The floor function of x
$\lceil x \rceil$	The ceiling function of x
$\otimes$	The outer product of vectors of the same dimension if $A = u \otimes v$ then $A_{ij} = u_i v_j$
$\langle \cdot, \cdot \rangle$	The inner product of two matrices of the same dimensions $\langle U, V \rangle \equiv \sum_{i,j} U_{i,j} \cdot V_{i,j}$
A^d	The set of all vectors of d elements over A
$\text{aff}(S)$	The affine hull of a set S
$\text{conv}(S)$	The convex hull of a set S
DP	Dynamic Program
$\mathbf{e}_i$	The i^{th} standard unit vector in an Euclidean real space of a known dimension
G/G/1	A queueing system with a single server, arbitrary arrival process and arbitrary processing time distribution
LP	Linear Program
MIP	Mixed Integer Program
M/M/1	A queueing system with a single server, Poisson arrival process and exponential processing time distribution
$P(A)$	The power set of a set A i.e., the collection of all subsets of A
QCS	Quality Control Station(s)
QC_i	The QCS located immediately after the i^{th} machine in the line.
$\mathbb{R}$	The set of real numbers
$\mathbb{R}_+$ ($\mathbb{R}_-$)	The set of non-negative (non-positive)
WIP	Work In Process
$\mathbb{Z}$	The set of integers
$\mathbb{Z}_+$ ($\mathbb{Z}_-$)	The set of non-negative (non-positive)

Chapter 1

Introduction

This dissertation consists of two parts. The first part deals with two problems related to manufacturing. Namely, optimal configuration of quality control station in a serial production line and long run profit maximization of a job shop system. The second part deals with the *separation problem.* Here we present some background and literature survey for the methods we used and for the problems studied.

1.1 Fluid Approximation

Many systems in the fields of manufacturing, transportation and data communication can be considered as discrete flow control systems, e.g., systems consisting of a network and some distinct objects that should be transferred along the network according to some rules. Often, optimization problems related to these systems are NP-Hard. For example, many scheduling problems are such problems and are well known for their intractability.

Given a discrete flow control system, its *fluid counterpart system* can be defined by relaxing the discrete nature of its objects and adjusting some of the rules that control the movements of the objects within the system. The fluid counterpart system, if properly designed, may imitate quite accurately the discrete one. Since the continuous nature of the fluid systems makes them easier to analyze and optimize, it might be useful to construct such a system whenever one wishes to devise a method to evaluate or optimize a discrete flow system. This construction may lead to efficient exact approximation or heuristics procedures and sometimes even to an efficient optimization algorithms.

The fluid approximation view is extensively used for analyzing queueing systems, see for example Chen and Mandelbaum [9]. However, only recently it has been used for combinatorial optimization problems. In particular this approach was previously used to tackle the High Multiplicity Job Shop Problem by Bertsimas and Gammarnik [5], by Boudoukh, Penn and Weiss [7]; by Dai and Weiss [13]; Bertsimas and Sethuraman [6] and by Penn and Raviv [32]. Bertsimas and Gammarnik also studied a Packet Routing Problem. Penn and Raviv [33] considered a Vehicle Routing Problem.

Here we extend the set of combinatorial optimization problems solved by the fluid approach by studying the Quality Control Station Configuration Problem (in Chapter 2) and to the Maximum Profit Job Shop Problem (in Chapter 4).

1.2 The Quality Control Configuration Problem

It is well observed in the literature that incorporating inspection stations into multi-stage production systems exert influence on the final cost and the quality of the final product. Models and optimization algorithms for the problem of installing inspection stations is dated back to 1965, see Lindsay and Bishop [25]. A survey on the problem of optimal allocation of quality control stations (sometimes referred as inspection station) in multi-stage systems, appears in Raz [34]. For more recent studies focused on systems with imperfect inspection facilities, see for example [35], for study on systems allowing rework and repair see [42] and for finite planning horizon look at [23].

To the best of our knowledge, all previous studies considered the effect of the inspection procedure on the average cost per product and overlooked its important effect on the line throughput. In fact, most studies explicitly or implicitly, assume a unit processing time in each of the machines and inspections facilities along the production line. Hence, under the unit processing time assumption, the first machine is always a bottleneck station and so installing additional quality control stations (QCS) can not increase the throughput of the system.

Furthermore, it turns that the quality control station configuration substantially affects the quantity of work in process within the system and thus the actual costs.

This phenomenon was first pointed out in a descriptive manner by Drezner, Gurnani and Akella [19] but was never incorporated into an optimization algorithm.

The first two chapters of this dissertation extend the QCS models studied in the literature and propose methods to solve them. In Chapter 2, we present a version of the problem where the aim is to obtain a maximum profit per time unit. This objective captures the effect of the QCS configuration on the line throughput.

This model is extended in the second chapter. In addition this model allows arbitrary revenue function.

In Chapter 3 we define and solve a QCS Configuration model of a serial production line where holding cost of work in process as well as capital costs are considered. We present a method to analyze and optimize the performance of such a system. Two optimization problems are considered: Minimization of the expected operational cost under a given production rate and maximization of the expected profit where a QCS configuration and an arrival rate are to be decided simultaneously. As far as we know, this work presents the first attempt to optimize the QCS configuration where throughput and WIP (work in process) are taken into account. Also, we allow an arbitrary revenue function for the yield of this system.

1.3 The Maximum Profit Job Shop Problem

Here we consider a production model where many instances of a small amount of product types are to be produced according to a Job Shop setting. The planner has to determine simultaneously the production mix and the schedule in order to maximize the expected steady state profit.

We open the discussion with a simplified version of the problem in which the long run average gross profit per time unit is maximized. In this version we allow the system to use arbitrarily large (finite) safety stocks. Consequently, a long initialization phase, to build this safety stocks, may be required. In addition, the size of the system buffers and the average level of work in process may be large.

Ideally, one would like to maximize the net profit, where holding cost, space cost (of the buffers) and delay cost (of the initialization phase) are considered. We could not meet this ultimate goal. Instead, we suggest a three-step optimization process. First, the gross profit is maximized, using a fluid approach. Then, the solution is modified in order to construct a cyclic schedule with very short cycles that still yields approximately optimal gross profit. It is shown numerically, on standard bench mark problems, that a compromise of 1% on the optimality of the gross profit enables the creation of very short and simple cyclic schedule. Once a short cycle is constructed, it is possible to change its sequence. The sequence's change should not affects the cycle length but it may reduce the level of work in processes, buffers spaces, required safety stocks or any weighted combination of the three. This last phase can be carried out by standard scheduling and combinatorial optimization techniques and is out of the scope of this paper.

1.4 Separation Index of Sets

A fundamental problem in automated data analysis is the following: given an universe U and a pair of disjoint sets $S, T \subset U$, construct a collection of rules that for any given point $p \in U$ determine whether or not p is close to S and far from T in some sense. One can study this problem in two different levels: providing these rules for some families of sets; and designing a mechanism that constructs such rules automatically for any given set.

Application of such separation problems arise in the design of decision support systems for medical diagnostic, risk assessment of credit and others. Consider for example a data base containing the results of some set of tests conducted on 1000 patients, for systolic and diastolic blood pressure, heart rate, body temperature and existence of proteins in the urine. Clearly, the results of the tests conducted on each patient can be represented approximately by an integer vector with five elements. Assume that these patients can be divided into set S and T of those who finally developed a certain disease and those who did not, respectively. It is possible to design a diagnostic expert system that can use this data set in order to devise a set of rules that returns a positive answer when an input of a vector which is similar to the members of S in some sense and negative otherwise.

Various methodologies were used for these problems including statistic and neural networks. For instance, the so called Logical Data Analysis (LDA) methodology developed by Hammer [21] deals with logical methods for constructing boolean function $f : \{0,1\}^d \rightarrow \{0,1\}$ providing the separation. More recent work by Ekin, Hammer and Kogan [16] tests the applicability of this method for decision support systems.

In this paper we consider the use of a system of linear equalities and inequalities rather than a logical function to separate the sets. In many cases this method allows a description of complex sets using a small number of equalities and inequalities. More importantly, we believe that in many real life applications this kind of representation captures very well the essential nature of the problems since the set of "positive points" tends to be in some sense "convex".

For the medical diagnostic problem described above our method can provide a polyhedra that contains all the vectors representing the positive (ill) patient and none of those of the negative (healthy) patients.

The paper focuses on the problem of constructing a minimal linear system that performs the separation (in a sense to be rigorously defined in the next section). Clearly, using a small separation it is easy to check whether a given point belongs to the set.

An additional benefit of having a finite set described by a compact system of equalities and inequality is that one can use it to optimize any convex function over the set. This can be carried out by various integer programming techniques such as branch & bound, cutting schemes or Gröbner bases methods [37, 4]. Lovaš and Schrijver [26] presented a cutting schemes in which a relaxation of the convex hull of S, given by a set of n inequalities, is converted into a tighter description of $\text{conv}(S)$ with $O(n^2)$ number of inequalities. This operation if applied repeatedly returns the inequalities description of $\text{conv}(S)$. While some separation can be always obtained merely from a membership oracle (see [31]), it is likely that an explicit system with as few inequalities as possible will provide a better starting point for such a cutting scheme.

Part I

Manufacturing System Problems

Chapter 2

Fluid Approximation Model for the Quality Control Station Configuration Problem

Many systems in the fields of manufacturing, transportation and data communication can be considered as discrete flow control systems, e.g., systems consisting of a network and some distinct objects that should be transferred along the network according to some rules. Often, optimization problems related to these systems are NP-Hard. For example, many scheduling problems are such problems and are well known for their intractability.

Given a discrete flow control system, its *fluid counterpart system* can be defined by relaxing the discrete nature of the objects and adjusting some of the rules that control the movements of the objects within the system. The fluid counterpart system, if properly designed, may imitate quite accurately the discrete one. Since the continuous nature of the fluid systems makes them easier to analyze and optimize, it might be useful to construct such a system whenever one wishes to devise a method to evaluate or optimize a discrete flow system. This construction may lead to efficient approximation or heuristics procedures and sometimes even to an efficient optimization algorithms.

The fluid approximation view is extensively used for analyzing queueing systems, see for example Chen and Mandelbaum [9]. However, only recently it has been used for combinatorial optimization problems. In particular this approach was previously used to tackle the High Multiplicity Job Shop Problem by Bertsimas and Gammarnik [5], by Boudoukh, Penn and Weiss [7]; by Dai and Weiss [13]; Bertsimas and Sethuraman [6] and by Penn and Raviv [32]. Bertsimas and Gammarnik also studied a Packet Routing Problem. Penn and Raviv [33] considered a Vehicle Routing Problem.

Here we extend the set of combinatorial optimization problems solved by the fluid approach by studying the Quality Control Station Configuration Problem. We model a serial production line where Quality Control Stations (QCS) are to be installed along the line with the aim of maximizing the expected profit of the system. We believe that such a model captures many real-life manufacturing situations. In this paper we define and solve a fluid counterpart model of the QCS Configuration problem. We present a method to analyze and optimize the performance of such a system.

It is well observed in the literature that incorporating inspection stations into multi-stage production systems exert influence on the final cost and the quality of the final product. Models and optimization algorithms for the problem of installing inspection stations is dated back to 1965, see Lindsay and Bishop [25]. A survey on the problem of optimal allocation of inspection stations in multi-stage systems, appears in Raz [34]. For more recent studies focused on systems with imperfect inspection facilities, see for example [35], for study on systems allowing rework and repair see [42] and for finite planning horizon look at [23].

However, to the best of our knowledge, all previous studies considered the effect of the inspection procedure on the average cost per product and overlooked the important effect of the inspection procedure on the line throughput. In fact most studies explicitly or implicitly, assume a unit processing time to all of the machines and inspections facilities along the production line. Under this assumption the first machine is always a bottleneck in the system and thus no improvement in throughput can be achieved by installing additional Quality Control Stations (QCS).

Performing quality inspections throughout a production or a service process may save labor, materials and energy, helps to reduce the level of work in process and increases the output rate of the system. On the other hand, quality control operations incur their own costs. Thus, it is desirable to design a quality control system that maximizes the net profit of the system.

The procedures presented in this paper are based on a fluid view to analyze and optimize the QCS configurations. In Section 2.1 we define the model and present a fluid system that imitates the discrete one. In Section 2.2 we first study a simple version of the fluid problem where the aim is to determine an optimal flow for a given QCS configuration. For that problem we use Linear Programming to obtain the optimal profit per time unit. We then turn to the more general case where both QCS configuration and flow rates are to be determined with the aim of profit maximization. We formulate this problem as a compact mixed integer program (MIP) and as a dynamic program (DP). Our MIP formulation uses a linear number of constraints and variables with respect to the production line length. In Section 2.3 we present a fully polynomial-time approximation scheme (FPTAS) for the fluid problem based on a simplified version of the dynamic program algorithm and parametric optimization approach. In Section 2.4 we show how the optimal solution of the fluid model can be used to construct an optimal solution for the discrete model. In section 2.5 we show a method to approximate the optimal solution of the more general problem where a limitation on the amount of work in process is introduced. In Section 2.6 we present the results of the numerical experiments we conducted to demonstrate and compare the practical efficiency of both the MIP and DP algorithm presented in Section 2.2. The DP method turns to be very fast and able to solve problems with 200 machines in a fraction of a second and it out performs the MIP method. We conclude (Section 2.7) with a discussion.

2.1 Problem Definition and Fluid Relaxation

Consider a serial production line with N machines, infinite number of identical products to be produced, and with no limitation on the intermediate inventory levels. That is, unlimited number of products that completed operation on a machine may wait in a buffer in front of the next machine until being processed. Processing of a product consists of a series of operations, each processed on a single machine. For each product we assume operation i has a deterministic processing time x_i and a cost per product c_i. These values are the same for all products. Operation i performed on a **non-defective** product succeeds with a known probability p_i and fails with probability $(1-p_i)$. A product is considered defective if one or more of its operations failed and non-defective otherwise. We use q_{ij} to denote the probability that a product that left machine j as non-defective remains non-defective when it leaves machine i. Clearly,

$$q_{ij} = \prod_{k=i+1}^{j} p_k. \tag{2.1}$$

Note that this is true even if the failure events are dependent across machines since p_i is the conditional probability of a successful operation given all previous operations were successful. A quality control station (QCS) can be installed after each machine and is capable of detecting any flaw caused by any of the previous operations. Once a flaw is detected, the product is discarded from the line. The cost per operation of a QCS, if installed at position i, (right after machine i) is c'_i. The machines and the QCSs are jointly referred in the sequel as stations. Each non-defective (defective) finished product has its own revenue (loss) denoted by r_G (r_B). Our aim is to maximize the expected net profit of the system, per time unit, by determining the locations of the quality control stations and constructing a dispatching rule that governs the operation of the system. A *Dispatching rule* is a set of rules by which the timing of releasing jobs to stations are determined. A configuration is denoted by the set Y of machines that are followed by a QCS; When it convenient and clear from the context we refer to Y as a characteristic vector of this set. That is, $Y_i = 1$ if a QCS is installed after machine i and $Y_i = 0$ otherwise.

One intuitive such a dispatching rule is the greedy one, i.e., let each machine starts processing the next job whenever it is ready and a job is available to be processed in its buffer. Clearly in a serial production line with cost associated with each operation and where the aim is to maximize the expected profit per time unit the greedy rule is not optimal in general. For a serial production line, with no QCSs, one needs to adjust the processing rate of all machines to the rate of the slowest machine. QCSs allow machines located before the bottleneck machine to work faster.

A fluid relaxation of this problem is to allow each product to be divided into infinitesimally small fractions (molecules). Different fractions of the same product can be processed by different machines. The processing cost and the expected processing times of each product-fraction is proportional to its size. The failure and the actual processing time of each product-fraction is independent random variables for all other fractions of the same product. That is, by the law of large number, the total processing times of all the fractions of a single product on a station equals the expected processing time on this station and a constant portion of the fraction is discarded at each QCS. The net revenue from all the fractions representing a single product is also a constant. That is, although each fraction behave in a stochastic manner, the total cost at each station and the total revenue from the fluid flows out of the system is determine deterministically by a dispatching rule / control policy.

The fluid counterpart problem can be defined as follows: a fluid material is to be pumped into the fluid system instead of discrete objects. Machine i (QC_i) of the fluid system is capable of pumping one unit of volume, say liter, in x_i (x'_i) units of time, and thus the maximum potential pumping rate of machine i is $\frac{1}{x_i}$ ($\frac{1}{x'_i}$) units of volume per time unit. Each machine is a flow preserver. That is, the flow rate into the machine must equal the flow rate out of it. In the QCSs, some of the fluid that represents the defective product-fraction, leaks out. That is, if the inflow rate into QC_j is F_j, and the last QCS is installed after machine j then the outflow rate from QC_j is $q_{ij} \cdot F_j$ and the remaining fluid leaks out of the system via QC_j.

A solution for the fluid system is given by the pair (F, Y), the flow rate through each machine and the QCS configuration respectively. $L_i(Y)$ denotes the location of the last QCS before machine i according to configuration Y. Our goal is to find a feasible solution (F, Y) that maximizes the expected net profit of the system. Note that the fluid system can operate optimally with no intermediate buffers at all. Also, the performance of the discrete system is bounded above by the performance of its fluid counterpart.

2.2 Analyzing and Optimizing the Fluid system

In this section we consider a fluid system. We first present a method to determine the maximum feasible flow rate for a given system and QCS configuration and then turn to the problem of determining optimal QCS configuration and optimal flow rate simultaneously.

Given a QCS configuration Y, the following Linear Program finds a static flow control that maximizes the expected profit from the system.

Linear Program 2.2.1 (optimal solution for the fluid system)

$$\max F_N \left[q_{L_N(Y),N} r_G - (1 - Y_N)(1 - q_{L_N(Y),N}) r_B\right] - \sum_{i=1}^{N} F_i(c_i + Y_i c_i') \tag{2.2}$$

Subject to

$$F_i \leq \frac{1}{x_i} \qquad \forall i = 1, ..., N \tag{2.3}$$

$$F_i \leq \frac{1}{x_i'} \qquad \forall i = 1, ..., N : Y_i = 1 \tag{2.4}$$

$$F_{i+1} = F_i \qquad \forall i = 1, ..., N - 1 : Y_i = 0 \tag{2.5}$$

$$F_{i+1} = q_{i,L_i(Y)} F_i \qquad \forall i = 1, ..., N - 1 : Y_i = 1 \tag{2.6}$$

$$F_i \geq 0.$$

Observe that the variables $F_1, \ldots, F_N$ are linearly dependent and so,

$$F_i = F_1 q_{L_i(Y),0} \qquad \forall i = 2, \ldots, N \tag{2.7}$$

Now the above Linear Program can be reformulates with a single variable F_1.

Linear Program 2.2.2 (solution for the fluid system using one variable)

$$\max F_1 \left[r_G q_{N,0} - r_B(1 - Y_N)(1 - q_{N,L_N(Y)}) - \sum_{i=1}^{N} q_{L_i(Y),0}(c_i + Y_i c_i') \right] \tag{2.8}$$

Subject to

$$F_1 q_{L_i(Y),0} \leq \frac{1}{x_i} \qquad i = 1, ..., N \tag{2.9}$$

$$F_1 q_{L_i(Y),0} \leq \frac{1}{x_i'} \qquad i = 1, ..., N : Y_i = 1 \tag{2.10}$$

$$F_1 \geq 0$$

If $r_G q_{N,0} - r_B(1 - Y_N)(1 - q_{N,L_N(Y)}) - \sum_{i=1}^{N} q_{L_i(Y),0}(c_i + Y_i c_i') \leq 0$, then the optimal solution is to set $F_1 = 0$. In such a situation, the expected profit from a single product is negative and thus the best is not to produce at all. In the following, we ignore this case and assume positive profits. Therefore, our aim is to find the largest value of F_1 that agrees with constraint (2.9) and (2.10) and hence solves LP 2.2.2. It is not difficult to see that the optimal solution is

$$F_1^* = \min_i \left\{ \frac{1}{q_{L_i(Y),0} \cdot \max(x_i, Y_i \cdot x_i')} \right\} \tag{2.11}$$

all other decision variables of LP 2.2.1, $F_2, ..., F_N$, can be easily calculated using (2.7).

From (2.11) and (2.7) we conclude that for a given QCS configuration the computational complexity of optimizing the expected net profit from the fluid system is $O(N)$. We turn now to discuss the more complicated problem of obtaining simultaneously an optimal QCS configuration and an inflow rate. We start by giving some definitions.

Here Y is a binary vector of decision variables with the same meaning as before. The decision variables FG_i (FB_i) denote the flow rate of the fluid representing non-defective (defective) jobs into machine i. The problem of determining the QCS configuration and flow rate in each station in order to maximize the net profit per time unit obtained by the fluid system can be solved by the following Mixed Integer Program:

Mixed Integer Program 2.2.3

$$\max \quad r_G \cdot FG_{N+1} - r_B \cdot FB_{N+1} - \sum_{i=1}^{N} [R_i + c_i \cdot (FB_i + FG_i)] \tag{2.12}$$

subject to

$$FG_i + FB_i \leq \frac{1}{x_i} \qquad \forall i = 1, \ldots, N : x_i \geq x_i' \tag{2.13}$$

$$FG_i + FB_i \leq \frac{1}{x_i} + Y_i \cdot \left(\frac{1}{x_i'} - \frac{1}{x_i} \right) \qquad \forall i = 1, \ldots, N : x_i < x_i' \tag{2.14}$$

$$FG_{i+1} = FG_i \cdot p_i \qquad \forall i = 1, \ldots, N \tag{2.15}$$

$$FB_{i+1} \geq (1 - p_i) \cdot FG_i + FB_i - \frac{1}{x_i} \cdot Y_i \qquad \forall i = 1, \ldots, N \tag{2.16}$$

$$R_i \geq c_i' \cdot (FG_i + FB_i) - \frac{c_i'}{x_i} \cdot (1 - Y_i) \qquad \forall i = 1, \ldots, N \tag{2.17}$$

$$Y_i \in \{0, 1\} \qquad \forall i = 1, \ldots, N$$

$$FB_i, FG_i \geq 0 \qquad \forall i = 1, \ldots, N + 1$$

$$R_i \geq 0 \qquad \forall i = 1, \ldots, N$$

To simplify the notations we add a zero cost dummy machine $N+1$. The flow rate of defective and non-defective fluid into this machine represents the outflow of these types of fluid from the system. The auxiliary variable R_i represents the cost per time unit incurred by QC_i if such a QCS is installed. Constraints (2.13) and (2.14) assure that the actual flow through each station does not exceed its capacity. Constraint (2.15) is used to coordinate the flow rate of non-defective fluid through each pair of consecutive machines. Constraint (2.16) assures that FB receives a proper value in the following way. If machine i is **not** immediately preceded by a QCS, then the flow of defective fluid via this machine equals to the flow of defective fluid in the preceding machine $(i-1)$ plus the fluid that got damaged in machine i. If machine i is located right after a QCS, then the minimization will draw FB to zero. Constraint (2.17) forces R_i to obtain the cost per time unit incurred by the QCS, and again the optimization makes it zero if no QCS is installed at position i.

This is a compact Mixed Integer Program with only N binary variables, $2N+2$ continuous variables and $5N$ constraints. Due to our computational experiments such a program can be solved for fairly large systems in few minutes (see Section 2.6).

Note that by constraint (2.15) all variables $FG_1, ..., FG_N$ are linearly dependent and so could be replaced by a single variable. However, running numerical examples on CPLEX 8.0 shows that this modified formulation is much more sensitive to rounding errors; these errors worse both the accuracy of the solution obtained and the running time.

We turn now to show a dynamic programming procedure that solves the fluid problem. This method turns to perform significantly faster as observed by our computational experiments. In addition, it is used as a basis for various approximation methods as described in the next sections.

The dynamic programming procedure consists of N steps, starting from the last step and proceeds backward. For each i, in step i we determine the optimal value of Y_i as a function of two state variables. Our state variables are:

L_i - The position of the last QCS installed before machine i. $L_i \in \{0, ..., i-1\}$ with $L_i = 0$ indicating that no QCS was installed before machine i.

F_i - The flow rate into machine i.

The function $g_i(F_i, L_i; Y_i)$ defined below, represents the profit from the subsystem that consists of machines $\{i, ..., N\}$, with F_i as the flow rate into machine i, and L_i as the location of the previous QCS before i. Note that L_i controls the percentage of the defective material into machine i in this flow. Recall that if $Y_i = 1$ then a QCS is installed right after machine i and otherwise, if $Y_i = 0$, then no QCS is installed.

Algorithm 2.2.4 ***The QCS Dynamic Programming Algorithm***
Input: *A QCS problem given by* $(\mathbf{p}, \mathbf{x}, \mathbf{x}', \mathbf{c}, \mathbf{c}', \mathbf{r_G}, \mathbf{r_B})$
The function g_i is constructed by the following recursive relation:

$$g_i(F_i, L_i; Y_i) = \begin{cases} -F_i \cdot (c_i + c'_i \cdot Y_i) + g^*_{i+1}\left(F_{i+1}(F_i, L_i, Y_i), L_{i+1}(L_i, Y_i)\right) & F_i \in [0, a_i(Y_i)] \\ -\infty & \textit{otherwise} \end{cases} \tag{2.18}$$

with

$$a_i(Y_i) = \begin{cases} \frac{1}{x_i} & Y_i = 0 \\ \min\left(\frac{1}{x_i}, \frac{1}{x'_i}\right) & Y_i = 1 \end{cases}$$

and with the following transitions functions:

$$F_{i+1}(F_i, L_i, Y_i) = \begin{cases} F_i & Y_i = 0 \\ F_i \cdot q_{i,L_i} & Y_i = 1 \end{cases} \tag{2.19}$$

and

$$L_{i+1}(L_i, Y_i) = \begin{cases} L_i & Y_i = 0 \\ i & Y_i = 1. \end{cases} \tag{2.20}$$

Observe that the flow rate into machine $i+1$ equals the flow into machine i if no QCS is installed at position i. Otherwise, if a QCS is installed, the flow rate into machine $i+1$ equals the flow rate into machine i times q_{i,L_i}. That is, q_{i,L_i} is the proportion of fluid that flow through QC_i without being discarded.

The function g^*_i is constructed by

$$g^*_i(F_i, L_i) = \max_{Y_i} g_i(F_i, L_i; Y_i). \tag{2.21}$$

The optimal decision at each step i (whether to install a QCS at position i or not) is determined by

$$Y_i^*(F_i, L_i) = argmax_{Y_i} g_i(F_i, L_i; Y_i). \tag{2.22}$$

The rate of the raw material into the first machine is then determined by

$$\max_{F_1} g_1^*(F_1, 0). \tag{2.23}$$

The special structure of g^* (see below) makes this last optimization problem trivial. The initial conditions for step N are:

$$g_N(F_N, L_N; Y_N) = \begin{cases} T(F_N, L_N, Y_N) & F_N \in [0, \frac{1}{x_N}] \\ -\infty & otherwise \end{cases} \tag{2.24}$$

with

$$T(F, L, Y) = F \cdot [r_G \cdot q_{N,L} + (1 - Y) \cdot r_B \cdot (1 - q_{N,L}) - (c_N + c'_N \cdot Y)]. \quad \square$$

In the Appendix we demonstrate the performance of this DP algorithm on a simple four machines problem.

The structure of the function g^*, as in (2.21) is discussed below. The observations made here are used for the development of our approximation method in the next section.

Definition 2.2.1 *For two sets of k rational numbers $a_1, a_2, \ldots, a_k$ with $a_i \neq a_{i+1}$ for all $i \in 1, ..., k-1$ and $0 < u_1 < u_2 < \cdots < u_k$, the following function*

$$f(x) = \begin{cases} a_1 \cdot x & x \in [0, u_1] \\ a_2 \cdot x & x \in (u_1, u_2] \\ \vdots & \\ a_k \cdot x & x \in (u_{k-1}, u_k] \\ -\infty & otherwise \end{cases} \tag{2.25}$$

is called a ***k-piecewise linear homogeneous function****, abbreviated* k-PWLH.

Figure 2.1 demonstrates Definition 2.2.1. The following simple lemmas on PWLH functions are needed in the sequel, their proofs are omitted.

Lemma 2.2.2 *If $f_1(x)$ and $f_2(x)$ are two k_1 and k_2-PWLH functions, then the function $g(x) = \max\{f_1(x), f_2(x)\}$ is a k-PWLH function with $k \leq k_1 + k_2$.*

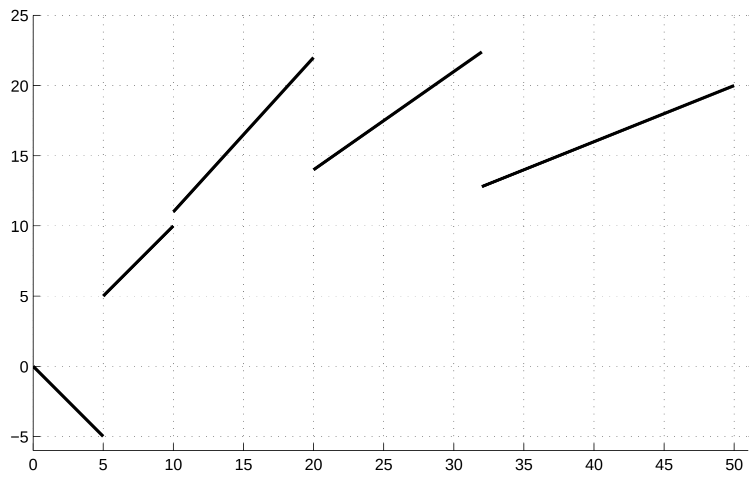
Figure 2.1: An example of a 5-piecewise linear homogenous function with $u_1 = 5, u_2 = 10, u_3 = 20, u_4 = 32, u_5 = 50, a_1 = -1, a_2 = 1, a_3 = 1.1, a_4 = 0.7, a_5 = 0.4$.

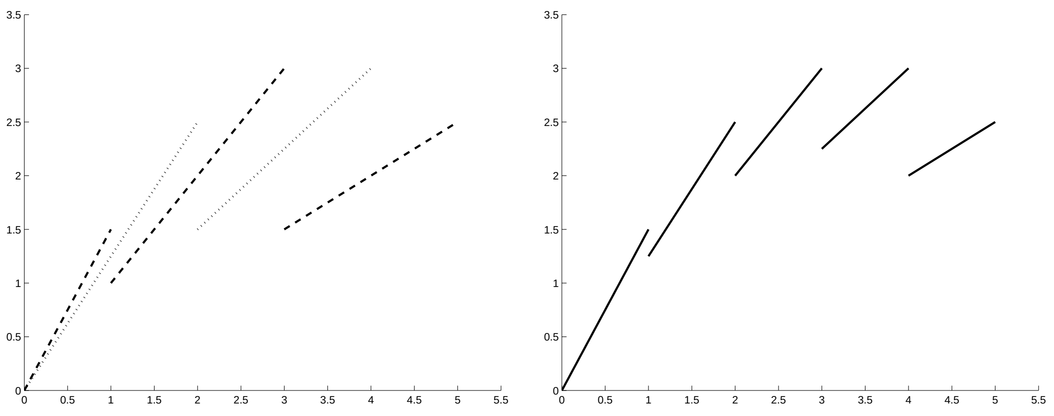
Figure 2.2: A 5-PWLH function obtained from the maximum of 2-PWLH and 3-PWLH functions. The left chart shows two functions. f_1 is a 2-PWLH function represented by the dotted lines and f_2 is a 3-PWLH function represented by the dashed lines. The right chart illustrates the function $f(x) = max\{f_1(x), f_2(x)\}$ which is a 5-PWLH function.

Figure 2.2 demonstrates Lemma 2.2.2.

Lemma 2.2.3 *If $f_1(x)$ and $f_2(x)$ are two k-PWLH functions defined on the same set of intervals, then the functions $g(x) = max\{f_1(x), f_2(x)\}$ and $h(x) = f_1(x) + f_2(x)$ are m-PWLH functions with $m \leq k$.*

Lemma 2.2.4 *Consider a k-PWLH function $f_1(x)$ and a 1-PWLH function $f_2(x)$, then the function $f_1(x) + f_2(x)$ is an m-PWLH function with $m \leq k$.*

Recall that the function obtains a value of minus infinity "outside" its domain and thus the actual domain of the summed function $f_1(x)+f_2(x)$ is the intersection of the actual domain of both f_1 and f_2.

Lemma 2.2.5 *For any L_i, the function $g_i^*(F_i, L_i)$ is a k-PWLH function of F_i with $k \leq 2^{N-i}$.*

Proof. We prove this lemma by induction. Consider the function $g_N^*(F_N, L_N)$. For any given value of L_N this function is obtained as the maximum of the two 1-PWLH functions, $g_N(F_N, L_N; 0)$ and $g_N(F_N, L_N; 1)$ defined on the same interval $[0, \frac{1}{x_N}]$. So by Lemma 2.2.3 the function g_N^* is a 1-PWLH function and thus the basis step is proved. We continue with the induction step. Recall that,

$$g_i^*(F_i, L_i) = \max_{Y_i \in \{0,1\}} g_i(F_i, L_i; Y_i). \tag{2.26}$$

Note that the function (2.26) is obtained as a maximization of the functions

$$g_i(F_i, L_i; 0) = \begin{cases} -F_i \cdot c_i + g_{i+1}^*(F_i, L_i) & F_i \in [0, \frac{1}{x_i}] \\ -\infty & \textit{otherwise}, \end{cases} \tag{2.27}$$

and

$$g_i(F_i, L_i; 1) = \begin{cases} -F_i \cdot (c_i + c_i') + g_{i+1}^*(F_i \cdot q_{i,L_i}, i) & F_i \in [0, \min(\frac{1}{x_i}, \frac{1}{x_i'})] \\ -\infty & \textit{otherwise}. \end{cases} \tag{2.28}$$

Observe that, by our induction hypothesis, both functions, (2.27) and (2.28), obtained as a sum of a 1-PWLH function $[F_i \cdot c_i$ or $F_i \cdot (c_i + c_i')]$ and a k-PWLH function with $k \leq 2^{N-(i+1)}$ (the function g_{i+1}^*) and so by Lemma 2.2.4 both are k-PWLH with $k \leq 2^{N-(i+1)}$. Now since the function $g_i^*(F_i, L_i)$ (2.26) obtained as a maximization of these two function, Lemma 2.2.2 implies that g^* is a k-PWLH functions with $k \leq 2 \cdot 2^{N-(i+1)} = 2^{N-i}$. ■

Propsition 2.2.6 *The complexity of algorithm 2.2.4 is at most $O(2^N)$.*

Proof. Clearly the time and space complexity of calculating the functions $g^*(F_i, L_i)$ and storing them in the dynamic programming tables depend linearly on the number of pieces. Thus the overall complexity of our algorithm depends linearly on the total number of pieces in all of these functions.

Now, by Lemma 2.2.5 the total number of pieces is bounded above by

$$\sum_{i=1}^{N} i \cdot 2^{N-i} = O(2^N). \tag{2.29}$$

■

However, the complexity of Algorithm 2.2.4 is not clear yet. We believe that our upper bound is either not tight or that it is attained only in some "pathological" cases. This because the functions $g^*(F_i, L_i)$ admit additional structure. The number of pieces of the g_i^* functions in our numerical study (see Section 2.6) was fewer than 2^N. In fact, it never exceeded $4 \cdot N$ in single function.

The DP approach has additional virtue over the MIP approach. Note that once the DP tables in the backward iteration are constructed one can use them to perform sensitivity analysis on the relations between the output rate of our system and the profit. Observe that for a given output rate F_n, the corresponding profit is just $g_1^*\left(\frac{F_N}{q_{N,0}}, 0\right)$. This kind of analysis can be useful if the market price depends on the production rate (e.g., our firm admits a monopolistic power). Lemma 2.2.7 to follow implies that the expected profit per item is a non-increasing function of the production rate. Note that this property is stronger than decreasing marginal profit. For our case the marginal profit is strictly negative at each break point u_i and constant everywhere else.

Lemma 2.2.7 *For any L_i the function $g_i^*(F_i, L_i)$ is a k-PWLH function with a list of decreasing slopes $(a_1, \dots, a_k)$, that is $a_1 > a_2 > \cdots > a_k$.*

Proof. We prove the lemma by a backward induction on the index i (actually, the induction is on m, with $i = N - m$). Clearly the lemma holds for N since this function is a 1-PWLH. Now, assume the lemma holds for $i + 1$, then for given L_i and Y_i the function $g_i(F_i, L_i; Y_i)$ is obtained by a summation of a 1-PWLH function with $F_i \in [0, v]$ and a k-PWLH function for some k. Thus,

$$g_i(F_i, L_i; Y_i) = \begin{cases} (a + a_1) \cdot F_i & F_i \in [0, u_1] \\ (a + a_2) \cdot F_i & F_i \in (u_1, u_2] \\ \vdots & \\ (a + a_r) \cdot F_i & F_i \in (u_{r-1}, min(u_r, v)] \\ -\infty & otherwise. \end{cases} \tag{2.30}$$

Note that the pieces of g_i admit the same boundaries as the first r pieces of g^*_{i+1} with r being the index of the first piece for which $u_r \geq v$ or $r = k$ if no such piece exists. Clearly, the function g_i maintains the property of decreasing slopes. Now, the function $g^*_i(F_i, L_i)$ is the maximum of the two functions $g_i(F_i, L_i; 0)$ and $g_i(F_i, L_i; 1)$ both admit the property of decreasing slopes. Let $a_1, ..., a_k$ and $u_1, ..., u_k$ be the corresponding slopes and boundaries of g^*.

Now, any piece in g^* is associated with a piece in $g_i(F_i, L_i; 0)$ or in $g_i(F_i, L_i; 1)$. We denote the slopes of $g_i(F_i, L_i; y)$ by $a^y_1, ..., a^y_k$. Consider a pair of consecutive pieces in g^* denoted by j and $j+1$. Either both pieces are associated with a pair of consecutive pieces in the same function or each piece is associated with a different function.

In the first case, assume the pieces j and $j+1$ are associated with pieces l and $l+1$ of the function $g_i(F_i, L_i; y)$ then by the first part of our proof $a_j = a^y_l > a^y_{l+1} = a_{j+1}$.

In the second case, assume without loss of generality that piece j is associated with piece l of the function $g_i(F_i, L_i; 0)$ and piece $j+1$ is associated with piece m of the function $g_i(F_i, L_i; 1)$. By the first part of this proof, $a^1_{m-1} > a^1_m$, but since piece j is associated with piece l of $g_i(F_i, L_i; 0)$, $a^0_l > a^0_{m-1}$ and so $a_j = a^1_l > a^1_m = a_{j+1}$ and we are done. ■

Observed that $g^*(x, L_i) \leq 0$ if and only if the function slope at x is non-positive. This is since any piece of g is homogeneous and the function is defined for non-negative values of x. Also, Lemma 2.2.7 implies that if $g^*_i(x, L_i)$ is non-positive for some x, then for any $x' > x$ the value of $g^*_i(x', L_i)$ remains non-positive. The above two observations imply the following. Let (u_i, u_{i+1}) be the first segment with non-positive slope. Then piece $i+1$ and all its following pieces, can be omitted from g^* (implying the values returned by this function being $-\infty$) without affecting the optimal solution achieved by Algorithm 2.2.4. Furthermore, due to the monotonic structure of $g^*_i(F_i, L_i)$, as characterized in Lemma 2.2.7, it can be replaced by a simpler function $g^{**}_i(F_i, L_i)$ that contains the pieces up to the global maximum of $g^*_i(F_i, L_i)$ and obtains the value of $-\infty$ elsewhere. Clearly when maximizing $g^*(F_1, 0)$ over F_1, at the first step [see (2.23)], we need to check the values of the function only at the break points $u_1, u_2, ..., u_k$. We select one with the highest value $a_i \cdot u_i$ or decide to produce nothing if all values are negative. It is not difficult to see that g^*_1

can be replaced by g_1^{**}. The following proposition extends the above remark to any g_i^*.

Propsition 2.2.8 *Replacing the functions $g_i^*(\cdot)$ by $g_i^{**}(\cdot)$ in Algorithm 2.2.4 keeps the set of optimal solutions unchanged.*

Proof. Consider the decision taken at any forward $i > 1$ step of the algorithm. The decision is done based on F_i, the flow rate obtained from the previous step, and L_i, the location of the previous QCS. Note that once the QCSs were set for positions between 1 and $i-1$ then for all $j < i$, F_j is an increasing function of F_i. We denote this function by $F_j(\cdot)$. Now, if u_i^* is the global maximum of $g_i^*(x, L_i)$ for a given L_i, then in order to prove the proposition we need to show that $F_i \leq u_i^*$. Assume by contradiction that $F_i > u_i^*$. Clearly $F_j(F_i) > F_j(u_i^*)$ and thus,

$$g_i^*(F_i) - \sum_{j=1}^{i-1} F_j(F_i) \cdot (c_j + c_j' \cdot Y_j) < g_i^*(u_i^*) - \sum_{j=1}^{i-1} F_j(u_i^*) \cdot (c_j + c_j' \cdot Y_j)$$

where the left hand side is the value of the solution assuming a flow rate of F_i into machine i and the right hand side is the same with a flow rate of u_i^*. Thus, the optimal value of F_i obtained by our procedure can not be greater than u_i^*. We conclude that the flow rate determined at each forward step is at most the one that maximizes g^* and so all pieces right to the maximum can be eliminated.■

Using proposition 2.2.8 one can reduce the size of the dynamic programming tables by omitting all the pieces right to the global maximum and thus reducing computation times. The value of the function for all points greater than this point is set to $-\infty$. Our numerical experiments indicate a significant reduction of computational times and memory usage achieved by this insight.

2.3 Approximation Scheme for the Fluid Problem

Recall that the complexity status of the fluid problem presented in Section 2.2 is still unknown. However, our numerical studies presented in Section 2.6 shows that randomly generated large instances of the problem can be solved in very short time by our dynamic programming algorithm presented in the previous section. In this section a fully polynomial-time approximation scheme (FPTAS) for the fluid QCS problem is presented. For definition and discussion of FPTAS see for example, [3]. We note that the QCS fluid problem is in "a good company" of more generic flow problems for which the complexity status is yet unclear but exact algorithms that perform well on test problems as well as FPTAS were introduced. For example, the optimal control of fluid networks. See Weiss [40], and Fleischer and Sethuraman [17] for an exact algorithm and the FPTAS respectively.

We start with few notations. Let f be a flow into the system. f is said to be feasible for a given QCS configuration Y if it is a solution for Linear Program 2.2.2 with Y. If f is not feasible for any configuration then f is said to be an *infeasible flow*. Clearly, the domain of a feasible flow is bounded above by $\frac{1}{x_1}$ as the fluid rate into the system can not exceed the processing rate of the first machine. Note that, for a given QCS configuration the expected profit from a unit of fluid is constant and independent of the inflow rate. Let us denote this value by $V(Y)$. QCS configuration Y is said to be optimal for f if amongst all feasible configurations for f, Y maximizes $V(\cdot)$. Such an optimal configuration is denoted by $Y^*(f)$. In case of several optimal configurations, one is taken arbitrarily. By convention, for infeasible flow f, $Y^*(f)$ is the $Null$ configuration and $V(Null) = -\infty$.

The maximal profit from a given inflow rate is denoted by $P^*(f) \equiv f \cdot V\left(Y^*(f)\right)$. The domain of $P^*(\cdot)$ is the set of all feasible flows in a system with a QCS installed after each machine. We denote the problem of calculating $P^*(f)$ by $QCS(f)$. Solving $QCS(f)$ is essentially finding the optimal configuration $Y^*(f)$. While there is some interest in the problem $QCS(f)$ by itself, we show that it can be used as a basis for an efficient approximation algorithm for the general fluid QCS problem discussed in the previous section. This problem can be rephrased in terms of $QCS(f)$ as

$$\max_f P^*(f) \tag{2.31}$$

and so $P^*(f) \equiv g_1^*(f, 0)$ with g^* as defined in (2.21).

We turn now to present a dynamic programming algorithm that solves $QCS(f)$ in $O(N^2)$.

Algorithm 2.3.1 ***The QCS(f) Dynamic Programming Algorithm***
Input*: A QCS model* $(\mathbf{p}, \mathbf{x}, \mathbf{x}', \mathbf{c}, \mathbf{c}', r_G, r_B)$ *and an inflow rate* f*.*
Here we use a single state variable L_i *that represents the location of the last QCS*

before machine i; and a single decision binary variable Y_i which is whether to install a QCS after machine i or not. Note that the inflow rate into each station can be calculated from f and L_i. The function $h_i(L_i;Y_i)$ is the optimal value of the rest of the line (revenue minus all expenses in machines $\{i+1,\dots,N\}$ assuming (L_i,Y_i). It is recursively constructed as follow:

$$h_i(L_i;Y_i) = \begin{cases} -f\cdot q_{L_i,0}\cdot(c_i + c_i'\cdot Y_i) + h^*_{i+1}\left(L_{i+1}(L_i,Y_i)\right) & f\cdot q_{L_i,0}\in[0,\frac{1}{x_i}] \\ -\infty & \textit{otherwise} \end{cases} \tag{2.32}$$

with the following transitions function:

$$L_{i+1}(L_i,Y_i) = \begin{cases} L_i & Y_i = 0 \\ i & Y_i = 1. \end{cases} \tag{2.33}$$

The initial condition for h_N is

$$h_N(L_N;Y_N) = \begin{cases} T(L_N,Y_N) & f\cdot q_{L_N,0}\in[0,\frac{1}{x_N}] \\ -\infty & \textit{otherwise} \end{cases} \tag{2.34}$$

with

$$T(L,Y) = q_{L,0}\cdot f\cdot[r_G\cdot q_{N,L} - (1-Y)\cdot r_B\cdot(1-q_{N,L}) - (c_N + c_N'\cdot Y)].$$

The function h_i^* is constructed by

$$h_i^*(L_i) = \max_{Y_i} h_i(L_i;Y_i). \tag{2.35}$$

If at any stage $h_i^*(L_i) = -\infty$ for $L_i = 0,\dots,i-1$ then the algorithm can be stopped and declare that no "profitable configuration exists for inflow rate f". The optimal decision at each step i (whether to install a QCS at position i or not) is determined by

$$Y_i^*(L_i) = \textit{argmax}_{Y_i} h_i(L_i;Y_i). \tag{2.36}$$

■

Note that once the flow rate into the first machine in the line is known, the flow of the non defective products through each machine is known. Thus, only the location of the previous QCS is needed as a state variable in our dynamic programming formulation.

In the sequel we made the standard assumption that real numbers can be stored in a constant memory space and arithmetic and comparison operations on reals take constant time.

Propsition 2.3.1 *The time and memory complexity of Algorithm 2.3.1 algorithm is $O(N^2)$.*

Proof. At any stage i the function $h_i(L_i; Y_i)$ is calculated in a constant number of operations for two possible values of Y_i ($\{0,1\}$) and for i possible values of L_i ($\{0,\ldots,i-1\}$). Thus, there are $2N(N+1)$ such calculations in total. The forward iterations, to determine the optimal configuration, clearly takes $O(N)$. Hence, the overall time complexity of the procedure is $O(N^2)$. The results of each stage i are stored in i reals $[h_i^*(L_i)]$ and i boolean variables $[Y_i^*(L_i)]$ and thus the total memory complexity is $O(N^2)$ as well. Also note that before starting to run the dynamic programm proper, the values of q_{ij} should be calculated. This can be done in $O(N^2)$ using the recursion formula $q_{i,j} = q_{i-1,j}p_i$. The space required to store the table is clearly $O(N^2)$. ■

In the sequel we explore some properties of the function $V(Y^*(f))$ that leads to an approximation scheme for the problem of determining optimal QCS configuration and optimal flow simultaneously. Recall that this problem is denoted the *QCS problem.*

Lemma 2.3.2 *For any $0 \leq f_1 \leq f_2$ the inequality $V(Y^*(f_1)) \geq V(Y^*(f_2))$ holds.*

Proof. Note that $P^*(f) \equiv g_1^*(f,0) = f \cdot V(Y^*(f))$ and so our result follows directly from Lemma 2.2.7 ■

The economic interpretation of Lemma 2.3.2 is that our system admits non-increasing marginal profit which is a common phenomenon when a production takes place using scarce resources.

Lemma 2.3.3 *Consider an inflow rate f_1 and its optimal QCS configuration $Y^*(f_1)$. Assume $f_2 > f_1$ and f_2 is a feasible inflow rate for the configuration $Y^*(f_1)$ then $Y^*(f_1)$ is also optimal for f_2.*

Proof. By Lemma 2.3.2, $V(Y^*(f_1)) \geq V(Y^*(f_2))$. By the fact that $Y^*(f_1)$ is feasible for f_2 it follows that $V(Y^*(f_1)) \leq V(Y^*(f_2))$. Thus $V(Y^*(f_1)) = V(Y^*(f_2))$ and the optimality of $Y^*(f_1)$ for f_2 follows. ■

Lemma 2.3.4 *For any QCS problem,* $V\left(Y^*(\frac{1}{max_i x_i})\right) = V\left(Y^*(f)\right)$ *for all* $f \in [0, \frac{1}{max_i x_i}]$.

Proof. First recall that $P^*(f)$ is a PWLH function and $V(Y^*(f))$ is piecewise constant. Thus, there exists an $\epsilon > 0$ such that $V(Y^*(\epsilon)) = V(Y^*(f)$ for all $f \in [0, \epsilon]$. Now, for any flow rate $f \leq \frac{1}{max_i x_i}$ any QCS configuration is feasible, and in particular $Y^*(\epsilon)$ and so by Lemma 2.3.3, $Y^*(\epsilon)$ is also optimal for f. ■

A simple consequence of Lemma 2.3.4 and Lemma 2.3.2 is that the optimal flow rate is either zero or at least the rate of the slowest machine $(\frac{1}{max_i x_i})$. This actually proves the following,

Propsition 2.3.5 *If the first machine in the line is the slowest one then Algorithm 2.3.1 with flow rate* $\frac{1}{x_1}$ *solves the QCS problem (of determining QCS configuration and flow rate simultaneously).*

That is, this special case can be solved in $O(N^2)$ time. Later on we shall see that this case is of a particular interest.

Lemma 2.3.6 *Let* $f_1 > 0$ *be a feasible inflow rate into the first machine, then for any* $f_2 > f_1$, $P^*(f_2) \leq \frac{f_2}{f_1} \cdot P^*(f_1)$

Proof. First note that $P^*(f) \equiv g_1^*(f, 0) = f \cdot V(Y^*(f))$. Now by Lemma 2.2.7 $V(Y^*(f_2)) \leq V(Y^*(f_1))$ and our result follows by substituting $V(Y^*(f)) = \frac{P^*(f)}{f}$ in both sides. ■

Clearly, the inflow rate into the system is bounded above by $\frac{1}{x_1}$. Thus the segment $[0, \frac{1}{x_1}]$ contains all feasible flow rates. Let us define the following finite subset of this segment $F = \{\frac{1}{max_i x_i} \cdot (\frac{1}{1-\delta})^i | i = 0, \ldots, \lfloor log_{\frac{1}{1-\delta}}(\frac{max_i x_i}{x_1})\rfloor\} \cup \{0, \frac{1}{x_1}\}$.

In the following we denote the value of an optimal solution of the QCS problem at hand by OPT.

Propsition 2.3.7 *Let $f \in F$ be an inflow rate with $P^*(f) \geq P^*(f')$ for all $f' \in F$. Then, $P^*(f) \geq (1-\delta) \cdot OPT$.*

Proof. If the optimal flow rate is zero then $OPT = 0$ and the proposition trivially holds. For the non trivial case. let z^* be an optimal flow rate. Note that $z^* \geq \frac{1}{max_i x_i}$ this since by Lemma 2.3.4, $V\left(Y^*(\frac{1}{max_i x_i})\right) = V\left(Y^*(f)\right)$ for all $f \in [0, \frac{1}{max_i x_i}]$. Now, let f' be the largest point in F such that $f' \leq z^*$. Note that there must be such a point in F since $\frac{1}{max_i x_i} \in F$. Let f'' the smallest point in F such that $f'' \geq z^*$. If $f' = f''$ then $z \in F$ and we are done. Otherwise, by the construction of F, $f'' \leq \frac{f'}{(1-\delta)}$. Now by Lemma 2.3.6, $OPT = P^*(z^*) \leq \frac{f''}{f'} P^*(f') \leq \frac{1}{1-\delta} P^*(f')$ and so $P^*(f') \geq (1-\delta) \cdot OPT$. ■

Algorithm 2.3.2 ***Approximation Algorithm for the QCS Fluid Problem***

```
Input: A QCS problem given by $(\mathbf{p}, \mathbf{x}, \mathbf{x'}, \mathbf{c}, \mathbf{c'}, \mathbf{r_G}, \mathbf{r_B})$ and a required relative error $\delta$.
Let $MaxProfit = 0$;
Set $f = \frac{1}{max_i x_i}$;
While $f < \frac{1}{x_1}$ do
    Calculate $Y^*(f)$ and $V(Y^*(f))$ using Algorithm 2.3.1;
    If $f \cdot V(Y^*(f)) > MaxProfit$ then
        Let $MaxProfit = f \cdot V(Y^*(f))$;
        Let $BestConfiguration = Y^*(f)$;
    Let $f = f \cdot \frac{1}{1-\delta}$;
End Do;
If $MaxProfit > 0$ then
    Return $BestConfiguration$;
Else
    Return "No profitable configuration exists";
```

The approximation guarantee $ALG \geq (1-\delta) \cdot OPT$ is obtained directly from Proposition 2.3.7. Algorithm 2.3.2 consists of about $log_{\frac{1}{1-\delta}}(\frac{max_i x_i}{x_1}) = log_{1-\delta}(\frac{x_1}{max_i x_i})$ calls to the $O(N^2)$ Algorithm 2.3.1 and thus its complexity is

$$O\left(N^2 \cdot log_{1-\delta}\left(\frac{x_1}{\max_i x_i}\right)\right).$$

Now clearly this complexity is polynomially bounded in the input size (when $1/\delta$ considered part of the input) and thus Algorithm 2.3.2 is a FPTAS. Clearly, the memory complexity of 2.3.2 is $O(N^2)$ since the memory used by each call to Algorithm 2.3.1 can be reused.

Below we suggest an enhanced version Algorithm 2.3.2 that takes advantage of the special structure of the piecewise homogeneses function $P^*(f)$ to improve the running time of the algorithm for most instances although it does not affect the worst case computational complexity.

Algorithm 2.3.3 ***Enhanced Approximation Algorithm for the QCS Fluid Problem***
Input: *A QCS problem given by* $(\mathbf{p}, \mathbf{x}, \mathbf{x}', \mathbf{c}, \mathbf{c}', \mathbf{r_G}, \mathbf{r_B})$ *and required relative error* δ.

```
Let $MaxProfit = 0$;
Set $f = \frac{1}{max_i x_i}$;
While $f < \frac{1}{x_1}$ do
    Calculate $Y^*(f)$ and $V(Y^*(f))$ using Algorithm 2.3.1;
    If $V(Y^*(f)) \leq 0$ Then Exit Loop;
    Let $\mathcal{Y} = Y^*(f)$;
    Let $f$ be the maximum feasible flow for configuration $\mathcal{Y}$ [Obtained by (2.11)];
    If $f \cdot V(Y^*(f)) > MaxProfit$ then
        Let $MaxProfit = f \cdot V(Y^*(f))$;
        Let $BestConfiguration = Y^*(f)$;
    Let $f = \max\left(f, \frac{MaxProfit}{V(Y^*(f))}\right) \cdot \frac{1}{1-\delta}$;
End Do;
If $MaxProfit > 0$ then
    Return $BestConfiguration$;
Else
    Return "No profitable configuration exists";
```

Note that the distance between each pair of consecutive inflow rates for which we calculate the optimal QCS configuration is generally larger in the enhanced version of the algorithm and thus in most cases less calls to Algorithm 2.3.1 are required. This is achieved by,

1. Once the optimal configuration for each tested flow is calculated, (2.11) is used to obtain the maximum feasible flow for this configuration. This flow is then used as a start point for the next iteration. Note that by doing so we also enhanced the accuracy of the algorithm.

2. We use the fact that the slope of the objective function is non-increasing to obtain an upper bounds on the optimal solution for some interval that follows the segment under consideration. The segment length is "stretched" to cover all points for which the upper bound can be applied.

We now propose two modifications of Algorithm 2.3.1 which may shorten its running time and memory requirements, although does not affect our worst case analysis of $O(N^2)$. Both take advantage of the fact that the algorithm is called numerous times by Algorithm 2.3.2 (and Algorithm 2.3.3) with the same system but with different values of f. First note that the q_{ij} matrix remains constant during all the iterations of Algorithm 2.3.2 and thus can be calculated and stored once for the whole process. Second, for each step i, it is possible to a-priori eliminate some of the possible values of the state variable L_i based on the model data. That is, it is possible to say that the last QCS must be install in the positions $\{\underline{L}_i, \ldots, i-1\}$ for some $0 \leq \underline{L}_i \leq i$ (with $\underline{L}_i = 0$ indicates that there are no restrictions on the place of the last QCS and it might be the case that there is no QCS at all prior to machine i). One method to construct $\underline{L}_i$ is to use the following local consideration: if $(1-q_{ij}) \cdot c_i > c'_{i-1}$ then $\underline{L}_i \geq j+1$. This since if no QCS is installed after machine j, then it worth to install a QCS at machine $i-1$ only to save the expenses incurred by defective jobs in machine i. Also it is clear that $\underline{L}_i \geq \underline{L}_{i-1}$. Now we can use the following recursion formula,

$$\underline{L}_i = \max\left\{\underline{L}_{i-1}, \max_j \left(\{j : (1-q_{ij}) \cdot c_i > c'_{i-1}\} \cup \{0\}\right)\right\}$$

starting with $\underline{L}_1 = 0$. It is easy to see that the time complexity of this recursive process carried out for each $i = 1, \ldots, N$ is $O(N^2)$ and that additional $O(N^2)$ integers should be stored. Now, once L_i is known, the value of $h_i^*(L_i)$ can be calculated only for $L_i \in \{\underline{L}_i, \ldots, i-1\}$ instead of $L_i \in \{0, \ldots, i-1\}$. Note that this modification may be also applied for Algorithm 2.2.4 described in the previous section.

2.4 Adapting the fluid Solution to the Discrete System

In the previous sections, we showed how an optimal QCS configuration can be found for the fluid QCS system. In this section we show how the fluid view and the results of the previous section can be used to solve the discrete problem. Throughout we assume that the intermediate buffer spaces are unlimited and there are no holding costs for the intermediate inventory. This assumption is somewhat relaxed in the next section. Recall that we are interesting in a QCS configuration that maximizes the expected profit per time unit in **steady state**.

For the discrete system we define the *production rate* of a station under some dispatching rule to be the expected number of products that processed in a unit of time. We continue to use F_i and Y_i as in the discussion of the fluid system. Let F'_i denote the flow rate through QC_i.

Definition 2.4.1 *A dispatching rule for the discrete system is said to be* ***stable*** *if for all* i*,*

1. $F_i = F'_i$ *whenever* QC_i *is installed.*

2. $F_i = F_{i-1} - Y_i(1 - q_{i-1,L_{i-1}(Y)})F_{i-1}$

An optimal dispatching rule is one that maximizes the expected profit from the system at a steady state.

Lemma 2.4.2 *Any optimal dispatching rule is a stable one.*

Proof. Consider a non-stable dispatching rule. Then at least one of the two conditions of Definition 2.4.1 is violated. In such a case, it is possible to reduce the rate of the first station in the pair (by inserting idle time) in order to coordinate it with the next station. Such a modification will not affect the steady state departure rate from the second station and thus admits no effect on the processing rate in all other machines in the system and on the departure rate of the products from the system. The only change to the total profit is that the average cost per time unit incurred by the first station of the pair is reduced and so the total profit is increased. Hence a non-stable rule can not be optimal. ■

Propsition 2.4.3 *The optimal profit at steady state of a discrete QCS system is bounded above by the optimal profit of its fluid counterpart.*

Proof. Consider a stable dispatching rule with the following production rates $F_1, \ldots, F_N$ of all machines and QCSs respectively. Clearly, $F_1, \ldots, F_N$ must be a feasible solution of Linear Program 2.2.1 and the objective function value of this program is exactly the gross profit from the system (without fixed costs, which are embedded for a given configuration and identical for both the fluid and discrete systems). That is, the optimal solution of 2.2.1 is an upper bound for the optimal solution of the discrete system for the given configuration. Now, since this is the case for any given configuration, then the optimal profit of the fluid system is an upper bound on the optimal profit of the discrete one. ■

In the following we present an optimal dispatching rule for the discrete QCS system. Our rule is based on the optimal solution of the fluid counterpart system.

Definition 2.4.4 (Fluid Based Dispatching Rule - FBDR) *For a given QCS system with configuration Y, let (F, Y) be a feasible solution of its fluid counterpart. The FBDR is as follows: A token arrives at an infinite bucket every $\frac{1}{F_1}$ units of time. Whenever the first machine is ready and there is at least one token in the bucket, a token is taken out and the machine starts processing a new job. In all other stations (machines and QCSs) operation starts whenever the station is ready and a job is waiting in its buffer.*

The token may be viewed as sufficient raw materials for the processing of a single job. The FBDR is related to the fluid solution in the sense that a job enters the first machine at time t only if at this time, the volume of fluid stuff that already processed by the fluid counterpart system is at least as the number of jobs processed by the first machine in the discrete system.

At first look, this on-line dispatching rule may seem strange since we force the first machine to remain idle part of the time. If our objective function was to minimize the makespan or the total completion time, this rule may produce a schedule which is inferior to schedules with no inserted idle times. However, we seek a dispatching rule that maximizes the steady state expected net profit. If we enter jobs to the system whenever the first machine is ready, that is on average every x_1 units of time where $x_1 \leq \frac{1}{F_1}$, then we may deliver jobs in a rate that can not be handled by the stations down the line. Since the output rate of the fluid system is an upper bound on that of the discrete one, our system will be congested and the inventory level in at least one of the buffers will grow indefinitely.

Theorem 2.4.5 *Let Y be a QCS configuration of a serial production line and let F be a feasible flow in its fluid counterpart with respect to Y. Then using FBDR yields the same profit as using F in the fluid system. In particular if (F^*, Y^*) is an optimal solution of the fluid counterpart system then using Y^* as the QCS configuration and the FBDR, based on F as the dispatching rule, maximizes the expected profit per time unit from the system.*

Proof. To prove the theorem we show that the expected number of defective and non-defective jobs carried by each station per time unit in the discrete system, is the same as the flow rate of the corresponding flows in the fluid counterpart system. Recall, that the optimal profit of the fluid system is an upper bound on the profit of the discrete one. We prove the theorem by induction.

We cite a simple known fact from queuing theory stating that the departure rate from a $G/G/1$ system is the minimum between the arrival and the service rates. That is, in a tandem of servers where the arrival rate for all servers is lower than their service rate then the actual processing rate of each server is equal to its arrival rate.

Recall that x_i stands for the expected processing time on machine i. Therefore, since $F_1 \leq \frac{1}{x_1}$, the expected total number of completed jobs in a line with a single station is equal to the inflow rate F_1. The expected number of non-defective [defective] jobs is p_1F_1 $[(1-p_1)F_1]$ as in the fluid system. This is our induction base.

Now, assume the theorem holds for a system with n stations (machines or QCSs) then we show its correctness for an $(n+1)$-stations system.

Under this induction hypothesis the departure rates of non-defective (defective) products from station n is p_nFG_n $(FB_n + (1-p_n)FG_n)$ if station n is a machine and p_nFB_n (0) if it is a QCS. The arrival rate to machine $n+1$ is the sum of these rates $(FG_n + FB_n)$. In both of these cases this is at most the service rate $\frac{1}{x_{n+1}}$ of the next station in the line since $(F_1, \dots, F_N)$ is a feasible solution for fluid problem. ■

Note that our definition of stability is slightly weaker than the one generally used in queueing theory. In particular we allow the arrival rate at any station to be the same as its maximum potential processing rate. Consequently, the queues in front of the bottleneck station will explode. This is still consistent with our problem definition since we imposed no-restrictions on the amount of work-in-process in the system and assume no holding cost for WIP (Work in Process). However, if the FBDR is to be used in practice, it should be based on a flow rate which is slightly less than the optimal solution of the fluid system. It is a challenge to calculate analytically the average WIP for any such arrival rate. One may use simulation in order to determine the maximum flow rate that will produce an acceptable level of WIP in the system. Note that in general, the WIP level in a queueing system operated under heavy traffic is very sensitive to the arrival rate and thus small reduction of the arrival rate may be suffice for many practical purposes.

2.5 Limited Work In Process

In this section we study a special case of the QCS model for which the system can produce optimal profit with very low level of work in process. Based on this case we proposed an approximation procedure for the more general case. For that mater we restrict our problem by requiring that the processing times of the machines and the QCSs are **deterministic**.

For a given Configuration we divide the system into segments. Each segment consists of a QCS and all the machines that precede it up to the previous QCS (not inclusive). If the last station in the line is not a QCS then the last segment consists of all the machines after the last QCS in the line. A segment is called *ready* if all its stations are ready, and *busy* otherwise. We use $\tilde{x}_i$ to denote the processing time of the slowest station of the i^{th} segment and $\tilde{F}_i$ the denote the flow rate through the segment in a a given solution. The following Synchronized Fluid Based Dispatching Rule can be used for a system that admits the above conditions.

Definition 2.5.1 (Synchronized Fluid Based Dispatching Rule (SFBDR)) *For a given QCS system with a configuration Y, let (F, Y) be a feasible solution of its fluid counterpart. The SFBDR is as follows: A Token arrives at an infinite bucket every $\frac{1}{F_1}$ units of time. If there is a token waiting in the bucket and the first segment* *is ready; or if there is a job waiting to be processed in one of the intermediate buffers before segment i and this segment is ready, then the following procedure is applied:*

1. *If there is a non-defective product on the last station of the segment (which is a QCS) it moves to the buffer before the next segment.*

2. *All jobs on all other machines along the segment progress one station downstream.*

3. *A job is loaded on the first machine of the segment and operations start concurrently on all the stations along the segment. For the first segment, in addition, a token is taken out of the bucket.*

Note that once a job finishes its operation it waits on the machine until the next cycle of the segment begins.

Clearly this policy requires buffers only between segments. Note however that each station serves as a buffer for its following station when it is idle. During the initialization phase, the system is gradually filled with products, one additional station in each cycle. Cycles are triggered by the arrival of jobs (or token) to the segment's feeding buffer (bucket). If the segment is still busy with the previous cycle, the jobs (tokens) are queued. We first show that this policy is optimal if there are no limitations on the intermediate inventory.

Propsition 2.5.2 *Let Y be a QCS configuration of a serial line with deterministic processing times and let F be a feasible flow in its fluid counterpart with respect to Y. Then using SFBDR yields the same profit as using F in the fluid system.*

The proof of Proposition 2.5.2 is very similar to that of Theorem 2.4.5. Both are proved by induction where here the induction is on the number of segments and not on the number of machines. Note that each segment plays the same role as a station in the previous proof and the segment rate equals the rate of its slowest station. Indeed, according to SFBDR, if a product enters the first machine of the segment, then a finished product is available in the last machine within $\frac{1}{F_i}$ units of time (although a different job then the one entered) and the segment is ready to accept a new job from its buffer. Thus, one can consider a segment as machine and hence the similarity of the proofs.

SFBDR takes advantage of the deterministic processing times to coordinate the operations of the stations along the segment and to reduce the number of niches where intermediate inventory is stored. Next proposition indicates where further reduction is applicable. Let $B_i(t)$ be the inventory level, at time t, of the buffer that precedes segment i. We first present the following simple lemma that is needed for the proof of Proposition 2.5.4.

Lemma 2.5.3 *Let $t_k(j)$ be the arrival time of the k^{th} job at buffer B_j (the one that feeds segment j). If $t_{k+1}(j) - t_k(j) > \tilde{x}_j$ for all k, then $B_j(t) = 0$ for all t.*

Proof. It should be shown that the lemma holds for all k and times $t_k(j)$. Our proof is by induction on k. Clearly at $t_1(j)$, when the first job arrives at the buffer, segment j is ready and thus the job can be immediately taken out of the buffer. Assume the same holds for time $t_k(j)$ (induction hypothesis). We now show that this is also true for $t_{k+1}(j)$. First note that $t_{k+1}(j) \geq t_k(j) + \tilde{x}_{j-1}$ and that cycle k starts at segment j no later than $t_{k+1}(j) - \tilde{x}_{j-1}$ (by our induction hypothesis, all cycles up to k were started immediately as the job was available in their feeding buffer). Thus, cycle k was finished by time $t_{k+1}(j)$ and so segment j was ready to accept the $(k+1)^{th}$ job immediately as it arrives to its feeding buffer at time $t_{k+1}(j)$. ■

Note that $t_k(j)$ is defined only for indices of products k that actually reach segment j and were not discarded from the line earlier. Clearly if job k was discarded it does not add anything to the intermediate inventory down the stream.

Propsition 2.5.4 *Let S be a system with deterministic processing times that operates under the SFBDR. If, in S, there is a pair of a segments $j < i$ for which $\tilde{x}_j \geq \tilde{x}_i$ then $B_i(t) = 0$ at any time t.*

Proof. Let h be the largest index of a segment such that $j \leq h < i$ and $\tilde{x}_h \geq \tilde{x}_i$. We first argue that the time between any two consecutive arrivals of jobs to buffer m such that $h < m \leq i$, is at least $\tilde{x}_h$ and that $B_m(t) = 0$ for all t. By Lemma 2.5.3 this is the case for segment $h + 1$, our induction base. Let us assume that our argument holds for segment m and show that it holds for $m + 1$ as well. Note that by the way we selected h, $\tilde{x}_m < \tilde{x}_h$ and so by the second part of our induction hypothesis each cycle on segment m starts exactly when a job arrives at buffer B_m

and exactly at these times jobs are transferred from segment m to buffer B_{m+1} if not discarded by a QCS. That is, the arrival time of the $(k+l)^{th}$ job to buffer B_m is at most the arrival time of the k^{th} job to buffer B_{m+1} when l is the number of stations in segment m, assuming no jobs were discarded by the QCS. Put it formally,

$$t_k(m+1) \geq t_{k+l}(m), \tag{2.37}$$

and so it turns that

$$t_{k+1}(m+1) - t_k(m+1) \geq t_{k+l+1}(m) - t_{k+l}(m) \geq \tilde{x}_h \geq \tilde{x}_{m+1} \tag{2.38}$$

for all k. The second inequality is due to the first part of our induction hypothesis. We conclude that the inter arrival intervals of segment $m+1$ are at least as long as $\tilde{x}_{m+1}$ and so whenever a job arrives at buffer B_{m+1} it is taken immediately to be processed by segment $m+1$. Continue by induction until $m = i$ and we are done. ■

An important implication of Proposition 2.5.4 is that under SFBDR, if the first segment is one of the slowest ones in a line with deterministic processing times, then the level of the intermediate inventory is kept zero **along** the line. Thus, an optimal solution for our original problem (with no limitation on the intermediate inventory) can be achieved with no use of intermediate inventory at all.

Corollary 2.5.5 *If the first machine is a slowest one, then an optimal solution of the fluid QCS problem is also optimal for the discrete QCS problem with limited sizes buffers or with holding costs of intermediate inventory taken into account.*

Proof. If the first machine is a slowest one, then the first segment is always a slowest segment under any QCS configuration, and thus the proof follows directly from Proposition 2.5.4.■

Note however that holding costs of jobs that are on stations and not in buffers are not taken into account by the result of this corollary. Partial remedy for this inaccuracy can be achieved by adding the holding costs to the operational costs of the stations. This will not solve to problem completely, since we allow jobs to be stored on idle machines.

Corollary 2.5.5 and the following lemma lead to a simple efficient approximation algorithm for the bounded buffer version of our problem (Theorem 2.5.7).

Lemma 2.5.6 *Increasing x_i, the processing time of machine i, by a factor of $\delta > 1$, decreases the steady state expected profit of the system by a factor of $\frac{1}{\delta}$ at most.*

Proof. By Equation (2.11), the inflow rate into the last segment $\tilde{F}_k$ can be reduced by at most a factor of δ when reducing the rate of one of the machines by the same factor. Now since the net profit of a given QCS configuration as expressed in (2.8) is a linear function of $\tilde{F}_K$, the effect of this modification can be at most δ. ■

Theorem 2.5.7 *For any given QCS problem, if $\delta = \frac{x_1}{\max_i x_i}$, then the problem can be δ-approximate by an $O(N^2)$ algorithm.*

Proof. The above approximation obtained by artificially enlarging the processing time of the first machine to make it the slowest one, and then use Algorithm 2.2.4 to solve it. Note that by Proposition 2.3.5 its time complexity is $O(N^2)$. ■

2.6 Numerical Results

In this section we describe the numerical experiments that were carried out in order to test the applicability of the exact dynamic programming Algorithm 2.2.4 and Mixed Integer Programming formulation 2.2.3. We created 16 groups of instances; each such group contains 50 data sets (800 instances in total). The groups differ from each other by three criteria:

1. **The system size.** Production lines with $n = 50, 100, 200, 400$ machines were tested.

2. **The tendency of the processing rates along the line.** For instances denoted by R the expected processing times were sampled from a common distribution (i.i.d) and for those denoted by I the expected processing were generated in a way that insures strictly increasing tendency.

3. **Success probabilities.** Groups denoted by L posses relatively low failure rates while H denote the high ones.

We used a Pentium 4, 2Ghz with 512Mb RAM runs under Windows 2000 as a testing machines.

The Dynamic Programming algorithm was implemented in *Microsoft Visual C++ 6.0* with Algorithmic Solution library LEDA (see [28]). The Mixed Integer Programs were solved by state of the art MIP solver, CPLEX 8.0, on the same computer.

The operation costs c_i and c_i' were randomly sampled from the set $\{1, ..., 10000\}$. The high success probabilities (group L) were randomly sampled from the interval $[\frac{n-0.5}{n}, 1]$, and the low success probabilities (group H) were chosen from the interval $[\frac{n-4}{n}, 1]$. All probabilities were rounded to 4 digits after the decimal point for the 50 and 100 machines problems and for 5 digits for the larger problems. For type R the processing times x_i of the machine operations were randomly chosen from the set $\{1, ..., 1000\}$. For type I we chose x_1 from the set $\{1, ..., 1000\}$ and for all other machines $x_i = \lceil x_{i-1} \cdot (1 + D_i) \rceil$ and $D_i \sim U[0, \frac{1}{n}]$, with the D_i-s independently generated for each instance. For all types, the revenue per non-defective product, r_G, was chosen from the interval $[0.75C, 3.75C]$ where $C = \frac{\sum_{i=1}^n c_i}{\prod_{i=1}^n p_i}$ is the expected profit per item if no QCSs are used. The revenue per defective product was selected from [-0.5C,0]. Each of the 800 test problems was solved by the two versions of our dynamic programming algorithm: The standard one, denoted by DP, and the one based on the observation of Proposition 2.2.8 denoted by DP**.

For both DP and DP**: Table 2.1 presents:

1. The average total number of pieces summed over all the constructed functions $g^*(\cdot)$ and $g^{**}(\cdot)$ in a solution of a single instance (column "# pieces"). Note that the number of pieces fairly measures the memory usage of the algorithm.

2. The average CPU time (in seconds).

The last three columns contain the performances of CPLEX on our MIP formulation. We set the time limit to 900 seconds for all instances. The first column, in the MIP section, shows the average running times until a solution within 0.1% from optimum is obtained. We consider such a gap as an exact solution for all practical purposes.

This average takes into account partial running times of the problems that exceeded the time limit and thus aborted. The second column shows the percentage of the problems (out of 50 in each class) that were solved in 900 sec. The last column presents the average time needed for CPLEX to find a feasible solution within 2% from a proven optimum.

Our test data sets are available for downloading from "http://ie.technion.ac.il/∼talraviv/Publications".

Class			DP		DP**		MIP		
Type	Risk	N	Number of pieces	Time exact	Number of pieces	Time exact	Time approx. 0.1%	% solved	Time approx. 2%
R	H	50	5,984	0.02	3,099	0.02	0.32	100	0.05
R	L	50	3,147	0.02	2,232	0.01	5.87	100	0.05
I	H	50	14,341	0.03	6,395	0.02	0.36	100	0.09
I	L	50	12,563	0.03	7,678	0.02	7.26	100	0.24
R	H	100	29,279	0.11	15,356	0.09	14.44	100	0.18
R	L	100	14,597	0.07	10,104	0.07	>702	40	0.28
I	H	100	85,641	0.15	30,873	0.09	14.61	100	0.21
I	L	100	94,895	0.23	51,024	0.15	>857	10	1.27
R	H	200	147,355	0.63	75,431	0.38	>663	42	0.50
R	L	200	70,302	0.51	47,046	0.38	>900	0	0.37
I	H	200	623,179	1.21	203,462	0.53	>658	46	0.38
I	L	200	738,225	2.00	352,313	1.10	>900	0	0.53
R	H	400	726,590	6.13	355,843	4.72	>683	36	0.67
R	L	400	318,780	5.37	207,142	4.62	>900	0	1.41
I	H	400	4,550,716	8.49	1,454,590	3.31	>432	52	0.92
I	L	400	5,790,973	23.11	2,394,068	7.51	>900	0	1.30

Table 2.1: Performances of the Dynamic Programming and MIP Algorithms

Table 2.1 shows that the Dynamic Programming Algorithm is significantly superior to the Mixed Integer Programming approach if the problems are to be solved to optimality. Nevertheless, if one is to solve the problem for a fairly large production lines, with up to 100 machines, MIP is a reasonable approach, especially since it is much easier to implement. Also, if an optimality tolerance of 2% is acceptable, then the MIP approach is extremely fast, even for larger problems, and the average running time grows very slowly in the size of the problems. The table also demonstrates the advantage of using the truncated version of the g^* function in the Dynamic Programming Algorithm. This advantage becomes more prominent for harder and larger data sets.

An important observation obtained from our experiment is that the number of pieces in each of the g^* functions, created by the Dynamic Programming algorithm, grows slowly with the number of machines. There is not a single problem, out of the 800 problems we tested, for which the number of these pieces in a single function exceeds 3 times the number of machines. Recall that the theoretic bound we have so far on this number is 2^n. This phenomenon raises the conjecture that the number of pieces is $O(n)$ and hence the worst case running time of our Dynamic Programming of $O(n^3)$.

For the Dynamic Programming Algorithm, the problems with increasing processing times (group I), turn to be much harder in terms of computational time and memory usage. This is due to the structure of the recursion function. If machine i is slower than some of the machine following it, some of the pieces in the functions g_i are dropped because they relate to a larger flow rate than the maximum possible rate of machine i. This is not the case if each machine is faster than its subsequent ones. Our Numerical study confirms this observation. The running time of the MIP is not significantly affected by this property.

On the other hand, the success probabilities seem to affect the MIP approach more significantly. The problems with lower success probabilities, group L, turn to be much harder to be solved by CPLEX than those of group H. We assume this is due to numerical difficulties raised by the differences in the values of the variables FG_i that become larger as the p_is become smaller.

2.7 Discussion

The paper demonstrates the usefulness of fluid approach in analyzing and optimizing serial production lines in large systems with similar items. Our research joins a relatively new trend of applying fluid approximation methods for combinatorial optimization problems and continues the legacy of its application in queueing theory.

Based on the solution for the fluid counterpart system and the token based dispatching rules it is proven that if the level of the intermediate inventory is unlimited, then the steady state throughput of the discrete system and that of its fluid counterpart are the same. In these cases, discrete systems can be optimized relatively easy. This observation is similar to our results on vehicle routing (see [33]) and job shop problems See Chapter 4 of this dissertation.

For the limited intermediate inventory with deterministic processing times we showed that, based on the fluid counterpart system and the fluid based dispatching rule, if the first machine is the slowest one then the optimal configurations for the discrete and the fluid systems coincide. This last observation is the basis for our δ-approximation algorithm for the general case, where δ is the ratio between the processing time of the first machine and that of the slowest one. We believe it is a challenging task to develop other approximation methods that do not depend on processing times.

We proposed two methods for solving the fluid system, one that is based on Mixed Integer Formulation and the other on Dynamic Programming. Our Dynamic Programming Algorithm is a *Mixed Dynamic Programming Algorithm* as continues and discrete state variables are used. Although we did not prove a polynomial running time of the Dynamic Programming Algorithm, our numerical experiments on large systems led us to conjecture that this complexity is $O(n^3)$. The MIP formulation is a practical choice for small systems or if compromise in optimality is acceptable. Since in real life problems the data is probably noisy, such a compromise is reasonable.

Although we study here the steady state performances of some discrete systems, we believe that the method is also well suited for high multiplicity problems where a large finite number of products are to be produced, and the goal is to minimize the completion time.

A simple extension to our model is to include fixed costs of the QCSs, such as capital and maintenance associated with each installed QCS. Let f_i stands for such a cost for QCS_i. Then, in the MIP formulation we simply subtract the term $\sum_{i=1}^{n} f_i Y_i$ from the objective function. The Dynamic Programming Algorithm can be adapted for this extension by slightly modifying the recursion formula as follows (2.18),

$$g_i(F_i, L_i; Y_i) = \begin{cases} -F_i \cdot (c_i + c_i' \cdot Y_i) - Y_i f_i + g_{i+1}^{*}\left(F_{i+1}(F_i, L_i, Y_i), L_{i+1}(L_i, Y_i)\right) & F_i \in [0, \frac{1}{x_i}] \\ -\infty & otherwise \end{cases}$$

Similar modification can be applied to the recursion formula (2.32) of the $QCS(f)$ Dynamic Programming Algorithm in order to introduce capital cost for the solution of this model. Note however that our approximation scheme in Algorithm very easily as well. Clearly, the relation between the optimal solution of the fluid system and discrete one remains unchanged for both of these extensions.

There is an extensive literature on the Buffer Allocation Problem (BAP) see for example [39, 18, 2]. In this problem one has to determine an optimal allocation of a given number of buffer spaces to niches in order to maximize throughput. A natural extension of the BAP is to perform buffer allocation concurrently with QCS configuration in order to maximize the total net profit. Standard heuristic methods may be used to search for a "good" combination. If one considers the case of deterministic processing times, then by Proposition 2.5.4 the search space may be reduced by eliminating all the solutions with buffers which are not right after a QCS or at the end of the segment with a slower segment upstream.

The model presented in this paper can be extended to capture a variety of possible additional configurations in manufacturing environments such as allowing sample inspection of jobs, re-work, breakdowns of machines and for other manufacturing environments such as job shop, assembly lines, multi-stage shop, etc. In addition, the idea presented here can be adapted for other areas as determining optimal nodes for performing integrity checks along a communication network.

Appendix - Demonstration of Algorithm 2.2.4

Consider a serial production line with four machines with the following parameters:

	M_1	M_2	M_3	M_4
x_i	2	3	4	5
x'_i	1	1	1	1
p_i	0.9	0.8	0.9	0.95
c_i	1	2	3	2
c'_i	2	3	3	5

In addition, the revenue (loss) obtained from delivering a non-defective (defective) product is 50 (30). First we use p_i and (2.1) to calculate the matrix q_{ij}.

	$j = 0$	$j = 1$	$j = 2$	$j = 3$	$j = 4$
$i = 0$	1	-	-	-	-
$i = 1$	0.9	1	-	-	-
$i = 2$	0.72	0.8	1	-	-
$i = 3$	0.648	0.72	0.9	1	-
$i = 4$	0.6156	0.684	0.855	0.95	1

First we construct the function $g_4^*(F_4, L_4)$.

L_4	$g_4(F_4, L_4; 0)$	$g_4(F_4, L_4; 1)$	$g_4^*(F_4, L_4)$	$Y_4^*(F_4, L_4)$
0	$F_4\,[50 \cdot 0.6156 - 30 \cdot (1 - 0.6156) - 2]$ $= 17.248F_4$	$F_4\,[50 \cdot 0.6156 - 7]$ $= 23.78F_4$	$23.78F_4$ $F_4 \in [0, 0.2]$	1
1	$F_4\,[50 \cdot 0.684 - 30 \cdot (1 - 0.684) - 2]$ $= 22.72F_4$	$F_4\,[50 \cdot 0.684 - 7]$ $= 27.2F_4$	$27.2F_4$ $F_4 \in [0, 0.2]$	1
2	$F_4\,[50 \cdot 0.855 - 30 \cdot (1 - 0.855) - 2]$ $= 36.4F_4$	$F_4\,[50 \cdot 0.855 - 7]$ $= 35.75F_4$	$36.4F_4$ $F_4 \in [0, 0.2]$	0
3	$F_4\,[50 \cdot 0.95 - 30 \cdot (1 - 0.95) - 2]$ $= 44F_4$	$F_4\,[50 \cdot 0.95 - 7]$ $= 40.5F_4$	$44F_4$ $F_4 \in [0, 0.2]$	0

Equipped with g_4^* and Y_4^* we are ready to calculate g_3^* and Y_3^* in the following step.

L_3	$g_3(F_3, L_3; 0)$	$g_3(F_3, L_3; 1)$	$g_3^*(F_3, L_3)$	$Y_3^*(F_3, L_3)$
0	$-3F_3 + 23.78F_3 = 20.78F_3$ $F_3 \in [0, 0.2]$	$-6F_3 + 0.648 \cdot 44F_3 = 22.512F_3$ $F_3 \in [0, 0.25]$	$22.512F_3$ for $F_3 \in [0, 0.25]$	1
1	$-3F_3 + 27.2F_3 = 24.2F_3$ $F_3 \in [0, 0.2]$	$-6F_3 + 0.72 \cdot 44F_3 = 25.68F_3$ $F_3 \in [0, 0.25]$	$25.68F_3$ for $F_3 \in [0, 0.25]$	1
2	$-3F_3 + 36.4F_3 = 33.4F_3$ $F_3 \in [0, 0.2]$	$-6F_3 + 0.9 \cdot 44F_3 = 33.6F_3$ $F_3 \in [0, 0.2222]$	$33.6F_3$ for $F_3 \in [0, 0.2222]$	1

We turn to calculate g_2^* and Y_2^*.

L_2	$g_2(F_2, L_2; 0)$	$g_2(F_2, L_2; 1)$	$g_2^*(F_2, L_2)$	$Y_2^*(F_2, L_2)$
0	$20.512F_2$ for $F_2 \in [0, 0.25]$	$-5F_2 + 0.72 \cdot 33.6F_2 = 19.192F_2$ for $F_2 \in [0, 0.3086]$	$20.512F_2$ for $F_2 \in [0, 0.25]$ $19.192F_2$ for $F_2 \in (0.25, 0.3086]$	0 1
1	$23.68F_2$ for $F_2 \in [0, 0.25]$	$-5F_2 + 0.8 \cdot 33.6F_2 = 21.88F_2$ for $F_2 \in [0, 0.2778]$	$23.68F_2$ for $F_2 \in [0, 0.25]$ $21.88F_2$ for $F_2 \in (0.25, 0.2778]$	0 1

Finally g_1^* and Y_1^* are calculated.

L_1	$g_1(F_1, L_1; 0)$	$g_1(F_1, L_1; 1)$	$g_1^*(F_1, L_1)$	$Y_1^*(F_1, L_1)$
0	$19.512F_1$ for $F_1 \in [0, 0.25]$ $18.192F_1$ for $F_1 \in (0.25, 0.3086]$	$-3F_1 + 0.9 \cdot 23.68F_1 = 18.312F_1$ for $F_1 \in [0, 0.2778]$ $16.69F_1$ for $F_1 \in (0.2778, 0.3086]$	$19.512F_1$ for $F_1 \in [0, 0.25]$ $18.312F_1$ for $F_1 \in (0.25, 0.2778]$ $18.192F_1$ for $F_1 \in (0.2778, 0.3086]$	0 1 0

Now, the optimal values of Y_1 and F_1 are founded from the last table by checking the value of $g_1^*(F_1, 0)$ at the end of the pieces. In this case g_1^* obtains its maximum at the end of the third piece at the point $F_1 = 0.3086$. $g_1^*(0.3086, 0) = 18.192 \cdot 0.3086 = 5.614$ and so $Y_1^* = 0$. Substitute the values of Y_1^* and F_1 recursively in the functions g_2^*, g_3^*, g_4^* we obtain $Y_2^* = 1$, $Y_3^* = 1$, $Y_4^* = 0$. So the optimal expected profit per time unit from the system is 5.614 and it is obtained if QCSs are installed at positions 2 and 3. Note that if we wish to operate the system without intermediate inventory at all, the processing time of the first machine should be artificially increased to 5, that is, a feeding rate of $F_1 = 0.2$. For this case we shall have a profit of $0.2 \cdot 19.512 = 3.9024$. The optimal configuration for this case is to have a single QCS between the third and fourth machines. Note that while the approximation ratio guaranteed for this problem by our method is $2/5 = 0.4$, in this case we can bound the ratio further to $3.9024/5.614 \approx 0.7$.

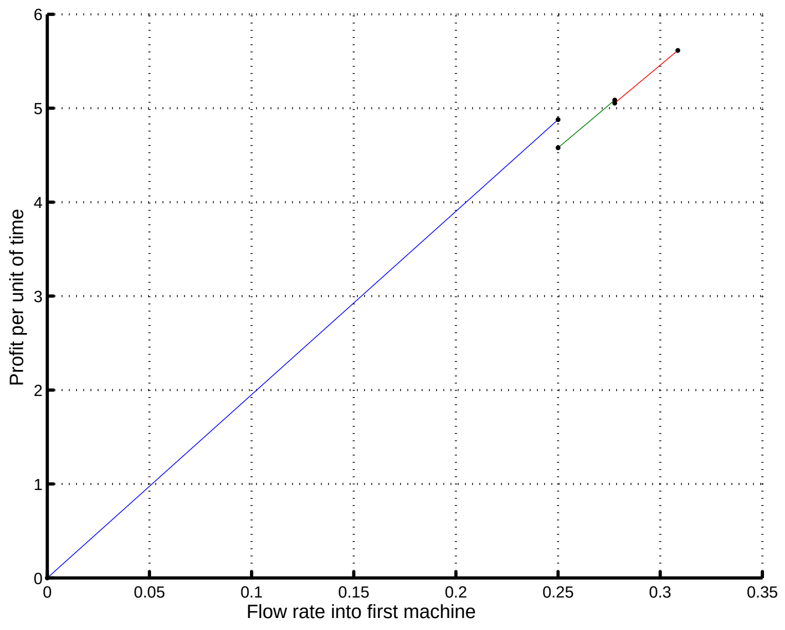

Figure 2.3: The function $g_1^*(x, 0)$. The solid (dotted) lines represent the pieces in which the optimal decision for the first step is (not) to install a QCS after the first station. It is easily observed that the optimal decision for the first step is not to install a QCS after the first machine.

Chapter 3

Solving the Quality Control Station Configuration with Holding Costs

It is well observed in the literature that incorporating inspection stations into multi-stage production systems exert influence on the final cost and the quality of the final product. Models and optimization algorithms for the problem of installing inspection stations is dated back to 1965, see Lindsay and Bishop [25]. A survey on the problem of optimal allocation of quality control stations (sometimes referred as inspection station) in multi-stage systems, appears in Raz [34]. For more recent studies focused on systems with imperfect inspection facilities, see for example [35], for study on systems allowing rework and repair see [42] and for finite planning horizon look at [23].

However, to the best of our knowledge, all previous studies considered the effect of the inspection procedure on the average cost per product and overlooked its important effect on the line throughput. In fact, most studies explicitly or implicitly, assume a unit processing time in each of the machines and inspections facilities along the production line. Hence, under the unit processing time assumption, the first machine is always a bottleneck station and so installing additional quality control stations (QCS) can not increase the throughput of the system.

Furthermore, it turns that the quality control station configuration substantially affects the quantity of work in process within the system and thus the actual costs.

This phenomenon was first pointed out in a descriptive manner by Drezner, Gurnani and Akella [19] but was never incorporated into an optimization algorithm.

In this paper we define and solve a QCS Configuration model of a serial production line where QCSs are to be installed along the line, and present a method to analyze and optimize the performance of such a system. Two optimization problems are considered: Minimization of the expected operational cost under a given production rate and maximization of the expected profit where a QCS configuration and an arrival rate are to be decided simultaneously. As far as we know, this work presents the first attempt to optimize the QCS configuration where throughput and WIP are taken into account. We believe that such a model captures many real-life manufacturing situations.

The cost minimization problem is solved using a polynomial time dynamic programming algorithm assuming exponentially distributed processing times and Poisson arrival process of jobs into the system. The profit maximization problem is approximated under the same assumptions using a branch and bound strategy that employs the dynamic programming algorithm as a subroutine. The main contributions of this paper are the following. Formulating a cost minimization problem under a certain throughput requirement and incorporating holding costs in the objective function. In addition, formulating a profit maximization problem for under the same Markovian assumption coupled with a general revenue function. Also, And presenting efficient algorithms for solving the above problems.

In Section 3.1 the cost minimization problem is defined. Section 3.2 is devoted to the method of calculating the operational cost of a system with a *given* QCS configuration. Section 3.3 presents a polynomial time dynamic programming algorithm to obtain a minimal cost QCS configuration for a given pre-specified production rate under the assumption of Poisson arrival process of jobs into the system and exponential distribution of the processing times. In Section 3.4 we define the profit maximization problem and present a branch and bound approximation strategy to solve it. In Section 3.5 we present our numerical experiments that show the practical efficiency of our algorithms. In addition, these experiments demonstrate that the above method is a good heuristic for non-Poisson arrival processes and in particular for the deterministic arrival process. which is a more applicable dispatching policy for serial production lines.

3.1 Problem Definition and Notations

Consider a serial production line that consists of N machines. A stream of identical jobs arrives at the first machine, according to some stationary stochastic process with an average arrival rate of a. The jobs are then processed serially along the line. Furthermore, come out of the system as finished products. Each machine starts processing its next operation as soon as it turns ready and a job is available in its feeding buffer. For each machine, the processing times of the jobs on that machine are i.i.d random variables with a mean of x_i. For machine i, for each i, there is a known success probability p_i, that a non-defective job that enters machine i remains non-defective and a failure probability of $(1 - p_i)$ that it turns defective. Any job that enters a machine as a defective one remains defective. The success and the failure events on each machine are independent. Let q_{ij}, for $i > j$, denote the probability that a non defective job leaving machine j remains non-defective after leaving machine i. Here q_{i0} is the probability that a job remains non-defective by the time it leaves machine i and $q_{ii} = 1$ by convention. Clearly for all $i > j$,

$$q_{ij} = \prod_{k=j+1}^{i} p_k. \tag{3.1}$$

A job is considered defective if one or more of its operations failed and non-defective otherwise. A quality control station (QCS) can be installed after any machine and is capable of detecting any flaw caused by any of the previous operations. A QCS is installed after machine i is denoted by QC_i; all the jobs that are being processed by machine i must pass through it.

Two kinds of costs are taken into consideration, variable and capital (fixed) costs. The variable cost of an operation carried on by machine i (QC_i) is denoted by c_i (c'_i). The capital (fixed) costs of the machines are considered as sunk costs and thus are not incorporated in the cost function. The fixed costs of installed QCSs are denoted by f'_i if QC_i exists. f'_i stands for the capital cost per unit of time associated with the installation of QC_i (regardless of the number of jobs that are actually examined by that control station). Each defective completed product that comes out of the system causes a loss of r_B. In addition, a holding cost of h_i (h'_i) is payed for every time unit that a job is being processed or waits to be process on machine i (QC_i). We consider a minimization problem where the aim is to minimize the expected cost per time unit at **steady state** by determining the number and the locations of the installed quality control stations. In the sequel we also consider the a profit maximization problem to be defined in Section 3.4.

Let Y denote a given QCS configuration. That is, Y represents the set of the QCS's locations $Y \subseteq \{1, \ldots, N\}$ or the characteristic (0-1) vector of this set with $Y_i = 1$ ($Y_i = 0$) means that the configuration includes (does not include) QC_i. For a given configuration Y, let $L_i(Y)$ denote the location of the last QCS before machine i, that is, $L_i(Y) = max\{j < i : Y_j = 1\}$ with the convention that $L_i(Y) = 0$ is to say that no QCS is installed before position i.

The tuple $(\mathbf{p}, \mathbf{x}, \mathbf{x}', \mathbf{c}, \mathbf{c}', \mathbf{f}', \mathbf{h}, \mathbf{h}', r_B)$ with an arrival rate of value a is refereed to as a *QCS(a) problem.* A system compound of a QCS(a) and a given configuration Y is denoted by (QCS(a),Y). Given a (QCS(a),Y) system, $A_i(Y, a)$ is defined to be the arrival rate of the jobs into machine i.

3.2 Analyzing a Given QCS System

In this section we study a given QCS system and find out the conditions, in terms of the arrival rates, under which the QCS system is stable. In addition, under the Poisson arrival process, we determine the operational cost of the system.

Definition 3.2.1 *A (QCS(a),Y) system is said to be stable if the expected amount of work in process in each of the system's buffers converges to some constant as the length of time the system operates goes to infinity.*

Propsition 3.2.2 *Let (QCS(a),Y) be a given system and assume the following inequalities hold*

$$a \cdot q_{L_i(Y),0} < \frac{1}{x_i} \qquad \forall i = 1, ..., N \tag{3.2}$$

and

$$Y_i(a \cdot q_{L_i(Y),0}) < \frac{1}{x'_i} \qquad \forall i = 1, ..., N, \tag{3.3}$$

then the system is stable and for each i, $A_i(Y, a) = a \cdot q_{L_i(Y),0}$.

Proof. It is clear that if and only if the arrival rate to each of the stations along the line is less than its service rate, then the system is stable. Thus, showing that under the proposition's conditions, for each machine i (QC_i), the arrival rate into machine i (QC_i) is $a \cdot q_{L_i(Y),0}$, is suffice. The proof is by induction on the first i machines along the line. Clearly, $A_1(Y,a) = a$ and since $q_{L_1(Y),0} = q_{0,0} = 1$ it is apparent that $A_i(Y,a) = a \cdot q_{L_1(Y),0}$. Assume the proposition holds for the first i machines. Note that in a stable $G/G/1$ system the steady state departure rate equals the arrival rate. Thus, if a QC_i is not installed, then $A_{i+1}(Y,a) = A_i(Y,a)$ and $L_{i+1}(Y) = L_i(Y)$, implying that $q_{L_{i+1}(Y),0} = q_{L_i(Y),0}$ and we are done. If QC_i exists, then it discards a portion of $q_{i,L_i(Y)}$ of the jobs that enter machine i and so

$$A_{i+1}(Y,a) = q_{i,L_i(Y)} \cdot q_{L_i(Y),0} \cdot a = \prod_{j=L_i(Y)}^{i} p_j \cdot \prod_{j=0}^{L_i(Y)} p_j \cdot a = q_{i0} \cdot a = q_{L_{i+1}(Y),0} \cdot a.$$

Note that $A_i(Y,a)$ is also the arrival rate into QC_i if QC_i exists. ■

Corollary 3.2.3 *Given a $(QCS(a),Y)$ then the system is stable if and only if*

$$a < \min\left\{\min_i \frac{1}{x_i \cdot q_{L_i(Y),0}}, \min_{i:Y_i=1} \frac{1}{x_i' \cdot q_{L_i(Y),0}}\right\}.$$

Note that Proposition 3.2.2 and its corollary need only the stationarity of the arrival process. In what follows, in order to get stronger results, we further assume that the arrival process is a Poisson process.

Lemma 3.2.4 *Consider a given stable $(QCS(a),Y)$ system with an exponentially distributed processing and QCS service times, as well as a Poisson arrival process to the first machine. Then, the arrival process into each of the machines and the QCSs is also a Poisson arrival process.*

Proof. A well known fact from queuing theory is that in a stable $M/M/1$ system the departure process is also a Poisson process with the same arrival and departure rates. Also, given a Poisson process with a rate of λ, and a random partitioning of the Poisson process by selecting some of the basic events with probability p, and not the others with probability $1-p$, results in a new Poisson process with a rate of $p\lambda$. Applying these two simple facts iteratively on the machines and the QCSs proves our claim. ■

Using Lemma 3.2.4 for a given stable $(QCS(a),Y)$ system, the expected total cost for a job being processed by machine i and its following QC_i can be easily calculated as shown below. We denote these costs by $C_i(Y,a)$ and $\mathcal{C}'_i(Y,a)$, respectively. This calculation is based on the known fact that the expected waiting time in a stable $M/M/1$ system with a service rate μ and an arrival rate λ, is $1/(\mu-\lambda)$. See for example [10].

$$\mathcal{C}_i(Y,a) = c_i + \frac{h_i}{\frac{1}{x_i} - a \cdot q_{L_i(Y),0}} \tag{3.4}$$

$$\mathcal{C}'_i(Y,a) = Y_i \left(c'_i + \frac{h'_i}{\frac{1}{x'_i} - a \cdot q_{L_i(Y),0}} \right). \tag{3.5}$$

Hence, the expected operating cost of the system per time unit is given by

$$\begin{aligned} C(Y,a) \equiv \quad & a \cdot \textstyle\sum_{i=1}^{N} q_{L_i(Y),0} \cdot [\mathcal{C}_i(Y,a) + \mathcal{C}'_i(Y,a)] + \sum_{i=1}^{N} Y_i f'_i \\ & +(1-Y_N) \cdot a \cdot q_{L_Y(N),0} \cdot (1 - q_{N,L_Y(N)}) \cdot r_B. \end{aligned} \tag{3.6}$$

Note that $\mathcal{C}_i(Y,a)$ is defined for arrival rates $a < (x_i \cdot q_{L_i(Y),0})^{-1}$, thus the queue in front of machine i is finite. Similarly, the domain of $\mathcal{C}'_i(Y,a)$ whenever $Y_i = 1$ is $a < (x'_i \cdot q_{L_i(Y),0})^{-1}$, otherwise it is $\mathbb{R}_+$. Clearly, the domain of $C(Y,a)$ is the intersection of these domains. Note, however, that in some places, if no confusion arises, we assume that all the three functions are defined over $\mathbb{R}_+$ and obtain the value of ∞ outside their actual domains.

3.3 An Optimal QCS Configuration for a Given Arrival Rate

In this section we turn to solve the combinatorial optimization problem of determining an optimal QCS configuration that minimizes the cost function over all 2^N possible configurations. In particular, we present a dynamic programming algorithm that solves the problem in a time complexity of $O(N^2)$. The optimal value of this optimization problem is denoted by

$$C^*(a) = \min_{Y \in \{0,1\}^N} C(Y,a)$$

and an optimal QCS configuration that materializes this cost is denoted by

$$Y^*(a) = \text{argmin}_{Y \in \{0,1\}^N} C(Y,a).$$

In fact, there might be more than one QCS configuration that minimizes the expected cost, in such a case $Y^*(a)$ represents any arbitrarily chosen optimal configuration. A brute force procedure for solving the problem is simply to enumerate all possible configurations and calculate $C(Y,a)$ for each one of them. Clearly, this approach is not efficient and is applicable only for systems with a very few tens of machines. Note, however, that in various industries, such as semiconductors and automobile productions lines, lines that consist of hundreds of operations are frequently used. We turn now to present the dynamic programming procedure.

Algorithm 3.3.1 ***The QCS(a) Dynamic Programming Algorithm***

Input: A QCS(a) problem defined by $(\mathbf{p}, \mathbf{x}, \mathbf{x}', \mathbf{c}, \mathbf{c}', \mathbf{f}', \mathbf{h}, \mathbf{h}', r_B)$ and an arrival rate a.
The recursion function $g_i(L_i; Y_i)$ denotes the total cost that incurred by the tail of the system that begins at machine i, assuming QC_{L_i} is the last installed control station before machine i, and given the existence $(Y_i = 1)$ or absence $(Y_i = 0)$ of QC_i. Here, L_i is a state variable and Y_i is a decision variable. For all $i = 1, \ldots, N-1$, the function g_i is constructed by the following recursive relation:

$$g_i(L_i;Y_i) = a \cdot q_{L_i,0} \cdot \left[c_i + \frac{h_i}{\frac{1}{x_i} - a \cdot q_{L_i,0}} + \left(c_i' + \frac{h_i'}{\frac{1}{x_N'} - a \cdot q_{L_i,0}}\right) \cdot Y_i\right] + f_i' Y_i + g_{i+1}^*\left(L_{i+1}(L_i, Y_i)\right) \tag{3.7}$$

for $a \cdot q_{L_i,0} \in [0, \min\{\frac{1}{x_i}, \frac{1}{x_i'} + (1-Y_i)\cdot\infty\})$, and $g_i(L_i;Y_i) = \infty$ otherwise. We use the following transitions function:

$$L_{i+1}(L_i, Y_i) = \begin{cases} L_i & Y_i = 0 \\ i & Y_i = 1. \end{cases} \tag{3.8}$$

The initial condition for g_N is

$$\begin{aligned} g_N(L_N;Y_N) = \ & a \cdot q_{L_N,0} \cdot \left[c_i + \frac{h_N}{\frac{1}{x_N} - a \cdot q_{L_N,0}} + \left(c_N' + \frac{h_N'}{\frac{1}{x_N'} - a \cdot q_{L_N,0}}\right) \cdot Y_N\right] + \\ & f_N' Y_N + (1 - Y_N) \cdot a \cdot q_{L_N,0} \cdot (1 - q_{N,L_N}) \cdot r_B \end{aligned} \tag{3.9}$$

for $a \cdot q_{L_N,0} \in [0, \min\{\frac{1}{x_N}, \frac{1}{x_N'} + (1-Y_N)\cdot\infty\})$ and $g_N(L_N;Y_N) = \infty$ otherwise.

The function g_i^* is constructed by

$$g_i^*(L_i) = \min_{Y_i} g_i(L_i;Y_i). \tag{3.10}$$

If, at any stage, $g_i^*(L_i) = \infty$ for all $L_i = 0, \ldots, i-1$, then the arrival rate a is not feasible for the problem and the algorithm terminates. The optimal decision at each step i (whether to install a QCS at position i or not) is determined by

$$Y_i^*(L_i) = \text{argmin}_{Y_i} g_i(L_i; Y_i). \tag{3.11}$$

■

Example 3.3.1 below demonstrates the use of Algorithm 3.3.1.

Example 3.3.1 Consider a serial production line with four machines, independent success events and the following parameters:

	M_1	M_2	M_3	M_4
x_i	2	2	4	5
x_i'	1	3	2	1
p_i	0.9	0.8	0.9	0.95
c_i	1	2	3	2
c_i'	2	3	3	5
h_i	0	0.08	0.07	0.10
h_i'	0.06	0.08	0.08	0.10
f_i'	1	1	2	1

In addition, r_B, the penalty for a defective product is 30 and the arrival rate a is 0.22. We start with the preprocessing procedure of calculating the matrix q_{ij} by using (3.1).

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$
$i=0$	1	-	-	-	-
$i=1$	0.9	1	-	-	-
$i=2$	0.72	0.8	1	-	-
$i=3$	0.648	0.72	0.9	1	-
$i=4$	0.6156	0.684	0.855	0.95	1

First we construct the function $g_4^*(L_4)$.

L_4	$g_4(L_4;0)$	$g_4(L_4;1)$	$g_4^*(L_4)$	$Y_4^*(L_4)$
0	∞	∞	∞	-
1	$0.22 \cdot 0.9 \left[2 + \frac{0.1}{0.2-0.22\cdot 0.9} + (1-0.684)\cdot 30\right]$ $= 12.173$	$0.22 \cdot 0.9 \left[2 + \frac{0.1}{0.2-0.22\cdot 0.9} + 5 + \frac{0.1}{1-0.22\cdot 0.9}\right] + 1$ $= 12.311$	12.173	0
2	$0.22 \cdot 0.72 \left[2 + \frac{0.1}{0.2-0.22\cdot 0.72} + (1-0.855)\cdot 30\right]$ $= 1.387$	$0.22 \cdot 0.72 \left[2 + \frac{0.1}{0.2-0.22\cdot 0.72} + 5 + \frac{0.1}{1-0.22\cdot 0.72}\right] + 1$ $= 2.508$	1.387	0
3	$0.22 \cdot 0.648 \left[2 + \frac{0.1}{0.2-0.22\cdot 0.648} + (1-0.95)\cdot 30\right]$ $= 0.747$	$0.22 \cdot 0.648 \left[2 + \frac{0.1}{0.2-0.22\cdot 0.648} + 5 + \frac{0.1}{1-0.22\cdot 0.648}\right] + 1$ $= 2.263$	0.747	0

Equipped with g_4^* and Y_4^* we are ready to calculate g_3^* and Y_3^* in the following step.

L_3	$g_3(L_3;0)$	$g_3(L_3;1)$	$g_3^*(L_3)$	$Y_3^*(L_3)$
0	$.22 \cdot 1 \left[3 + \frac{0.08}{0.25-0.22\cdot 1}\right] + \infty$ $= \infty$	$0.22 \cdot \left[3 + \frac{0.08}{0.25-0.22} + 3 + \frac{0.07}{0.5-0.22}\right] + 2 + 0.747$ $= 4.709$	4.709	1
1	$0.22 \cdot 0.9 \left[3 + \frac{0.08}{0.25-0.22\cdot 0.9}\right] + 12.173$ $= 13.072$	$0.22 \cdot 0.9 \left[3 + \frac{0.08}{0.25-0.22\cdot 0.9} + 3 + \frac{0.07}{0.5-0.22\cdot 0.9}\right] + 2 + 0.747$ $= 4.286$	4.286	1
2	$0.22 \cdot 0.72 \left[3 + \frac{0.08}{0.25-0.22\cdot 0.72}\right] + 1.387$ $= 2.000$	$0.22 \cdot 0.72 \left[3 + \frac{0.08}{0.25-0.22\cdot 0.72} + 3 + \frac{0.07}{0.5-0.22\cdot 0.72}\right] + 2 + 0.747$ $= 3.868$	2.000	0

We turn to calculate g_2^* and Y_2^*.

L_2	$g_2(L_2;0)$	$g_2(L_2;1)$	$g_2^*(L_2)$	$Y_2^*(L_2)$
0	$0.22 \left[2 + \frac{0.08}{0.5-0.22}\right] + 4.709$ $= 5.212$	$0.22 \cdot \left[2 + \frac{0.08}{0.5-0.22} + 3 + \frac{0.08}{0.333-0.22}\right] + 1 + 2.000$ $= 4.318$	4.318	1
1	$0.22 \cdot 0.9 \left[2 + \frac{0.08}{0.5-0.22\cdot 0.9}\right] + 4.286$ $= 4.734$	$0.22 \cdot 0.9 \left[2 + \frac{0.08}{0.5-0.22\cdot 0.9} + 3 + \frac{0.08}{0.333-0.22\cdot 0.9}\right] + 1 + 2.001$ $= 4.160$	4.160	1

Finally g_1^* and Y_1^* are calculated.

L_1	$g_1(L_1;0)$	$g_1(L_1;1)$	$g_1^*(L_1)$	$Y_1^*(L_1)$
0	$0.22 \cdot \left[1 + \frac{0}{0.5-0.22}\right] + 4.318$ $= 4.538$	$0.22 \cdot \left[1 + \frac{0}{0.5-0.22} + 2 + \frac{0.06}{1-0.22}\right] + 1 + 4.16$ $= 5.837$	4.538	0

Now, the optimal value obtained from the last table (describing $g_1^*(a)$) is 4.538 with $Y = (0,1,0,0)$. In this case, the arrival rate into the first two machines and into QC_2 is 0.22, while the arrival rate into the last two machines is $0.22 \cdot 0.72 = 0.1584$.

Figure 3.1 illustrates the functions $C(Y,a)$ for all possible QCS configurations Y, and the function $C^*(a)$ (in solid line) which is obtained as the minimization of these functions. Observe that the domain of $C^*(a)$ can be divided into several segments, where each segment is associated with a certain configuration that is optimal through. That is, in each such segment, the function $C^*(a)$ coincides with the function $C(Y,a)$ for some constant Y. We further study the structure of these functions in the next section.

In the sequel we assume that real numbers can be stored in a constant memory space and arithmetic and comparison operations on reals take constant time. This assumption, although not accurate, is widely accepted since real numbers are generally handled with limited accuracy by the computer.

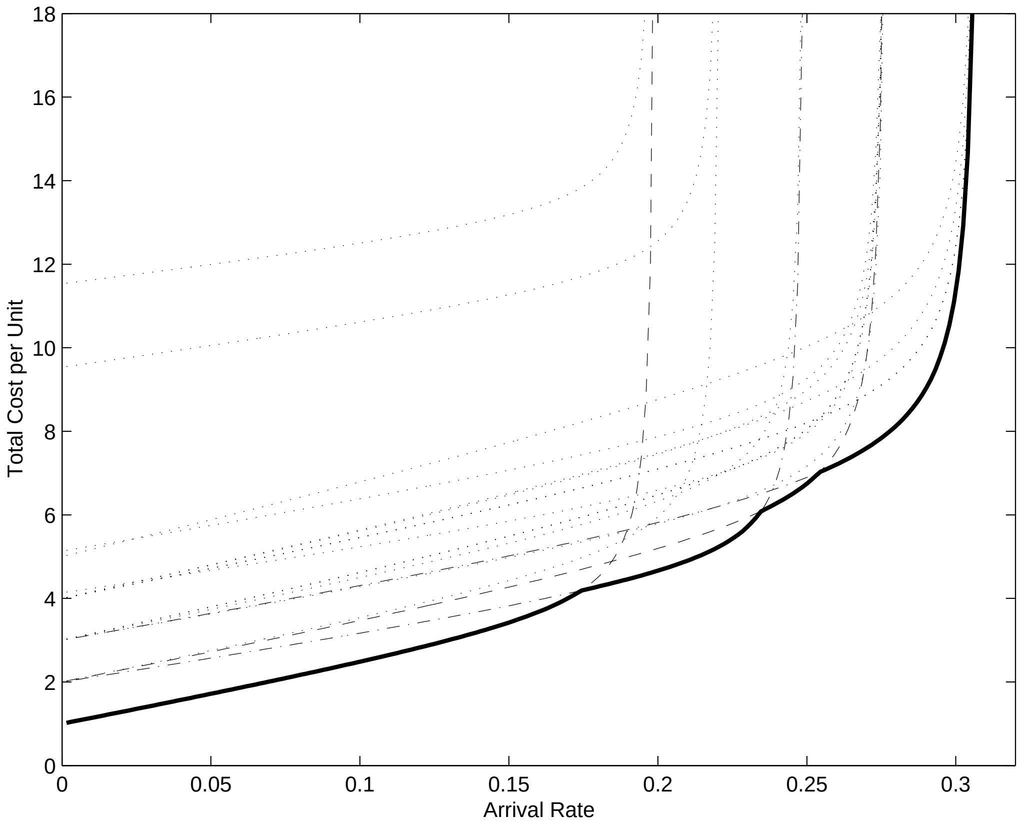

Figure 3.1: An illustration of the functions $C(Y, a)$ (in dotted and dashed lines) for all 16 possible QCS configurations of the problem described in Example 3.3.1. The function $C^*(a)$ (in solid line) is obtained by the minimization of these functions.

Propsition 3.3.2 *The time and the memory complexity of Algorithm 3.3.1 is $O(N^2)$.*

Proof. The preprocessing of calculating the values of q_{ij} takes $O(N^2)$ real arithmetic operations. This can be done using the recursion formula $q_{ij} = q_{i-1,j}p_i$. At any stage i, the function $g_i(L_i; Y_i)$ is calculated in a constant number of operations for two possible values of Y_i ($\{0, 1\}$) and for i possible values of L_i ($\{0, \ldots, i-1\}$), thus there are $N \cdot (N+1)$ such calculations in total. The stage of forward iterations, to determine the optimal configuration, clearly takes $O(N)$ and thus the overall time complexity of the procedure is $O(N^2)$. The results of each stage i are stored in i reals $[g_i^*(L_i)]$ and i boolean variables $[Y_i^*(L_i)]$ and thus the total memory complexity is also $O(N^2)$. ■

3.4 Parametric QCS Problem

In this section we extend the problem to capture the case when the arrival rate is a decision variable rather than part of the input. Thus, our aim is to optimize the QCS configuration and the production rate simultaneously. We consider a problem defined by $(\mathbf{p}, \mathbf{x}, \mathbf{x}', \mathbf{c}, \mathbf{c}', \mathbf{f}', \mathbf{h}, \mathbf{h}', r_B)$ coupled with a revenue function $r(x)$. The function $r(x)$ describes the expected revenue per time unit as a function of the departure rate of the non-defective products from the last machine. If the firm plays in a competitive market, then this function is linear and homogenous as the production rate of the firm admits no influence on the market price. For the discussion below we use the weak assumption that the revenue function is K-Lipschitz continuous over the relevant domain. That is, it is continues and derivable almost every where with its derivative bounded above by some finite constant K. This assumption is not very restrictive since in general the total revenue is hardly fluctuated by small changes in the supply.

Our extended parametric problem denoted by QCS is to determine the arrival rate and the QCS configuration simultaneously in order to maximize the expected profit per time unit from the system in steady state. For a given QCS configuration Y and an arrival rate a, the total profit is,

$$\begin{aligned} P(Y,a) = \; & r(q_{N,0} \cdot a) - a \cdot \textstyle\sum_{i=1}^{N} q_{L_i(Y),0} \cdot [\mathcal{C}_i(Y,a) + \mathcal{C}_i'(Y,a)] - \sum_{i=1}^{N} Y_i f_i' \\ & -(1 - Y_N) \cdot a \cdot q_{L_Y(N),0} \cdot (1 - q_{N,L_Y(N)}) \cdot r_B. \end{aligned} \tag{3.12}$$

Now, the optimal profit for a given arrival rate a is just

$$P^*(a) = r(q_{N,0} \cdot a) - C^*(a)$$

and hence can be easily calculated using Algorithm 3.3.1. Therefore, our extended parametric problem can be formulated as

$$\max_a P^*(a) \tag{3.13}$$

and in this form, it is reduced to an optimization problem in a single continues variable. The “bad news” however are that generally $P^*(a)$ is not concave or unimodal and hence standard line search techniques will not solve (3.13). In the sequel we explore some useful properties of $C^*(a)$ and $P^*(a)$ that form the basis for our approximation method.

Lemma 3.4.1 *The function $C^*(a)$ is continuous, piecewise convex and piecewise derivable.*

Proof. Recall that the function $C^*(a)$ is obtained as a minimization over all possible configurations of $C(Y, a)$. Thus, it is suffice to show that $C(Y, a)$ is convex, continues and derivable with respect to a. Let us write this function explicitly

$$\begin{aligned} C(Y,a) \equiv \quad & a \cdot \textstyle\sum_{i=1}^{N} q_{L_i(Y),0} \cdot [\mathcal{C}_i(Y,a) + \mathcal{C}'_i(Y,a)] + \sum_{i=1}^{N} Y_i f'_i \\ & +(1-Y_N) \cdot a \cdot q_{L_Y(N),0} \cdot (1 - q_{N,L_Y(N)}) \cdot r_B. \end{aligned}$$

Thus, $C(Y, a)$ is a sum of linear functions and the functions

$$\mathcal{H}_i(Y,a) = \frac{h_i \cdot q_{L_i(Y),0} \cdot a}{q_{L_i(Y),0} \cdot a - \frac{1}{x_i}} \qquad \textit{and} \qquad \mathcal{H}'_i(Y,a) = \frac{h'_i \cdot q_{L_i(Y),0} \cdot a}{q_{L_i(Y),0} \cdot a - \frac{1}{x'_i}}.$$

Deriving $\mathcal{H}_i(Y, a)$ twice we obtain,

$$\frac{\partial^2 \mathcal{H}_i(Y,a)}{\partial a} = \frac{2\, q_{L_i(Y),0}\, h_i}{\left(a\, q_{L_i(Y),0} - \frac{1}{x_i}\right)^2} - \frac{2\, a\, q_{L_i(Y),0}\, h_i}{\left(a\, q_{L_i(Y),0} - \frac{1}{x_i}\right)^3} = -\frac{2\, q_{L_i(Y),0}\, h_i\, x_i^2}{\left(a\, x_i\, q_{L_i(Y),0} - 1\right)^3}.$$

Since we are interested in stable systems, it follows from Corollary 3.2.3 that the relevant arrival rates are those for which $a \cdot q_{L_i(Y),0} < \frac{1}{x_i}$ and $a \cdot q_{L_i(Y),0} < \frac{1}{x'_i}$. Thus, it is easy to see that the second derivative is positive for all a in the relevant domain and the convexity of $\mathcal{H}$ is established. Now, since $C(Y, a)$ is obtained as a sum of derivable and convex functions it follows that it is convex and derivable. The continuity of $C^*(a)$ follows from the continuity of $C(Y, a)$ for each possible configuration.

Lemma 3.4.2 *For any pair of points $a_1 \geq a_0$ in the domain of $C^*(a)$, if $C^*(a)$ is derivable at a_1 then its derivative is bounded below by*

$$\frac{\partial C^*(a)}{\partial a}(a_1) \geq \sum_{i=1}^{N} \left\{ q_{i-1,0} \left(c_i + \frac{h_i}{\frac{1}{x_i} - a_0 \cdot q_{i-1,0}} \right) + \frac{a_0 \cdot h_i \cdot q_{i-1,0}^2}{\left(\frac{1}{x_i} - a_0 \cdot q_{i-1,0}\right)^2} \right\}.$$

Proof. Let us write $C^*(a_1)$ explicitly in terms of the optimal configuration $Y^*(a_1)$ at a_1,

$$\begin{aligned} C^*(a_1) = {} & a_1 \cdot q_{L_N(Y^*(a_1)),0} \cdot (1 - q_{N,L_N(Y^*(a_1))}) \cdot r_B + \\ & a_1 \cdot \textstyle\sum_{i=1}^{N} q_{L_i(Y^*(a_1)),0} \cdot [\mathcal{C}_i(Y^*(a_1), a_1) + \mathcal{C}'_i(Y^*(a_1), a_1)] + \\ & \textstyle\sum_{i=1}^{N} f'_i Y^*(a_1)_i. \end{aligned}$$

Let us denote $\mathcal{Y} \equiv Y^*(a_1)$. Note that if $C^*(a)$ is derivable at a_1, then there is a neighborhood of a_1 for which $\mathcal{Y}$ remains an optimal configuration. Now,

$$\begin{aligned} \tfrac{\partial C^*(a)}{\partial a}(a_1) = {} & q_{L_N(\mathcal{Y}),0} \cdot (1 - q_{N,L_N(\mathcal{Y})}) \cdot r_B + \\ & \textstyle\sum_{i=1}^{N} q_{L_i(\mathcal{Y}),0} \left\{ \mathcal{C}_i(\mathcal{Y}, a_1) + a_1 \tfrac{\partial \mathcal{C}_i(\mathcal{Y},a)}{\partial a}(a_1) \right\} + \\ & \textstyle\sum_{i=1}^{N} q_{L_i(\mathcal{Y}),0} \left\{ \mathcal{C}'{}_i(\mathcal{Y}, a_1) + a_1 \tfrac{\partial \mathcal{C}'_i(\mathcal{Y},a)}{\partial a}(a_1) \right\} \\ \geq {} & \textstyle\sum_{i=1}^{N} q_{L_i(\mathcal{Y}),0} \left\{ \mathcal{C}_i(\mathcal{Y}, a_1) + a_1 \tfrac{\partial \mathcal{C}_i(\mathcal{Y},a)}{\partial a}(a_1) \right\} \\ = {} & \textstyle\sum_{i=1}^{N} \left\{ q_{L_i(\mathcal{Y}),0} \left(c_i + \frac{h_i}{\frac{1}{x_i} - a_1 \cdot q_{L_i(\mathcal{Y}),0}} \right) + \frac{a_1 \cdot h_i \cdot q^2_{L_i(\mathcal{Y}),0}}{\left(\frac{1}{x_i} - a_1 \cdot q_{L_i(\mathcal{Y}),0}\right)^2} \right\} \\ \geq {} & \textstyle\sum_{i=1}^{N} \left\{ q_{i-1,0} \left(c_i + \frac{h_i}{\frac{1}{x_i} - a_1 \cdot q_{i-1,0}} \right) + \frac{a_1 \cdot h_i \cdot q^2_{i-1,0}}{\left(\frac{1}{x_i} - a_0 \cdot q_{i-1,0}\right)^2} \right\} \\ \geq {} & \textstyle\sum_{i=1}^{N} \left\{ q_{i-1,0} \left(c_i + \frac{h_i}{\frac{1}{x_i} - a_0 \cdot q_{i-1,0}} \right) + \frac{a_0 \cdot h_i \cdot q^2_{i-1,0}}{\left(\frac{1}{x_i} - a_0 \cdot q_{i-1,0}\right)^2} \right\} \end{aligned} \tag{3.14}$$

The first inequality is due to the facts that

$$q_{L_N(\mathcal{Y}),0} \cdot (1 - q_{N,L_N(\mathcal{Y})}) \cdot r_B \geq 0$$

and

$$\left\{ \mathcal{C}'_i(\mathcal{Y}, a_1) + a_1 \frac{\partial \mathcal{C}'_i(\mathcal{Y}, a)}{\partial a}(a_1) \right\} \geq 0$$

for all i. This is because $\mathcal{C}'_i(\mathcal{Y}, a)$ is a non-negative and increasing function of a. The second inequality in (3.14) is due to the fact that $q_{L_i(Y),0} \geq q_{i-1,0}$, since $L_i(Y) \leq i-1$ for any configuration Y. Now it is apparent that the expression

$$q \left(c_i + \frac{h_i}{\frac{1}{x_i} - a_1 \cdot q} \right) \tag{3.15}$$

is non-decreasing in q within the relevant domain. To see why the expression

$$\frac{a_1 \cdot h_i \cdot q^2_{L_{i-1},0}}{\left(\frac{1}{x_i} - a_1 \cdot q_{L_{i-1},0}\right)^2} \tag{3.16}$$

is also non-decreasing, we derive it with respect to q and obtain

$$\frac{2\,a_1^2\,h_i\,q^2}{\left(\frac{1}{x_i}-a_1\,q\right)^3}+\frac{2\,a_1\,h_i\,q}{\left(\frac{1}{x_i}-a_1\,q\right)^2}$$

which is also non-negative in the relevant domain of a and for all positive x and non-negative h and q. The last inequality of (3.14) follows from the fact that expressions (3.15) and (3.16) are also non-decreasing in a in the relevant domain; and $a_0 < a_1$. ■

Clearly, the slope of the function $P^*(a_1)$ for any point $a_1 > a_0$ is bounded above by the Lipschitz constant K minus the lower bound obtained by Lemma 3.4.2 for a_0. We denote this upper bound by γ_{a_0}.

Corollary 3.4.3 *For any pair of feasible arrival rates $a_1 < a_2$,*

$$P^*(a_2) < P^*(a_1) + (a_2 - a_1)\cdot\gamma_{a_1}.$$

Based on Corollary 3.4.3 and on Algorithm 3.3.1 we present below Algorithm 3.4.1 which is a branch and bound approximation procedure for solving the parametric QCS problem. Let $\mathcal{A} > 0$ and $\mathcal{R} \geq 0$ be the desired absolute and relative optimality errors respectively. That is, if the value of the optimal solution is OPT then our algorithm terminates with a solution which is at least $\min\{OPT\cdot(1-\mathcal{R}), OPT-\mathcal{A}\}$.

Algorithm 3.4.1 uses the following data structure to represent any segment of possible arrival rates. Each such segment is characterized by the following properties: *StartPoint*, *EndPoint*, $Profit$, *UpperBound* and *Recalc*. The boundaries of the segment are stored in *StartPoint* and *EndPoint*. The profit from using an optimal configuration with respect to the arrival rate that equals *StartPoint*, is stored in $Profit$. The auxiliary flag, *Recalc*, marks whether the value of $Profit$ was already calculated by calling Algorithm 3.3.1.

Starting with a data structure called *list* that contains at the beginning a single segment of all feasible arrival rates, the algorithm repeatedly removes a segment from the list and calculates the optimal profit assuming the arrival rate equals the start point of the removed segment. Then, based on Corollary 3.4.3, it calculates upper bounds on the profit for the first and the second half of the segment. These two segments are added to the list only if their upper bounds are greater than the best known solution plus the allowed error.

Algorithm 3.4.1 ***The QCS branch and Bound Procedure***

```
Input: a QCS problem ($\mathbf{p}, \mathbf{x}, \mathbf{x}', \mathbf{c}, \mathbf{c}', \mathbf{f}', \mathbf{h}, \mathbf{h}', r_B, r(x)$), optimality errors $\mathcal{A}$ and $\mathcal{R}$
Segment.StartPoint $\leftarrow 0$;     Segment.EndPoint $\leftarrow \frac{1}{x_1}$;
Segment.Profit $\leftarrow 0$;
Segment.Recalc $\leftarrow$ True
Segment.UpperBound $\leftarrow \frac{\gamma_0}{x_1}$;
Set List empty;
Add Segment to List;
Let BestKnownProfit $= 0$;
While List not empty Do
     Remove from List an item with the largest UpperBound and store it in Segment;
    If Segment.Recalc Then
        Calculate profit and Optimal Configuration at StartPoint using Algorithm 3.3.1
            and store in Segment.Profit and Y, respectively;
        Segment.Recalc $\leftarrow$ False;
        If Segment.Profit $>$ BestKnownProfit Then
            BestKnownProfit $\leftarrow$ Segment.Profit;   BestKnownConfig $\leftarrow Y$;
            Remove from List all items with UpperBound $\leq$
            $\min(BestKnownProfit \cdot (1+\mathcal{R}), BestKnownProfit + \mathcal{A})$;
        Segment.UpperBound $\leftarrow Profit + \gamma_{StartPoint} \cdot (EndPoint - StartPoint)$;
    If Segment.UpperBound $>$
     $\min(BestKnownProfit \cdot (1+\mathcal{R}), BestKnownProfit + \mathcal{A})$ Then
        Segment1.StartPoint $\leftarrow$ (Segment.EndPoint + Segment.StartPoint)/2;
        Segment1.EndPoint $\leftarrow$ Segment.EndPoint;
        Segment1.UpperBound $\leftarrow$ Segment.UpperBound;
        Segment1.Recalc $\leftarrow$ True;
        Add Segment1 to List;
        Segment.EndPoint $\leftarrow$ (Segment.EndPoint + Segment.StartPoint)/2;
        Segment.UpperBoound $\leftarrow$ Segment.Profit + $(EndPoint - StartPoint) \cdot \gamma_{StartPoint}$;
        If Segment.UpperBound $>$
        $\min(BestKnownProfit \cdot (1+\mathcal{R}), BestKnownProfit + \mathcal{A})$ Then
            Add Segment to List;
End Do;
```

Algorithm 3.4.1 maintains a list of *active segments* which are continues subsets of the set of feasible rates. For each segment in the list, the start point, the end point, an upper bound on the expected profit from the system with arrival rates within the segment, an upper bound on the slope of $P^*(a)$, and the optimal profit at the start point are kept. The list of active segments is sorted according to the segments upper bounds (implemented as a *skip list* so insertion takes $O(\log n)$ and all other relevant operations take $O(1)$, see [28]). We start with a list that consists of a single active segment that contains all feasible rates, say $[0, \frac{1}{x_1}]$, and an upper bound of ∞. At each iteration, our algorithm removes from the list a segment with the largest upper bound and then computes the expected profit at the start point of the removed segment using Algorithm 3.3.1. Based on this value, a tighter upper bound relative to this segment is calculated using Corollary 3.4.3. If the obtained upper bound on the profit is smaller than the minimum value between the best known solution plus $\mathcal{A}$ and the best known solution times $(1+\mathcal{R})$, then the segment is discarded from further consideration. If the solution at the start point is greater than the best known solution then it is stored as the new best known solution. The segment is then divided into two sub-segments of identical length. These two segments are returned to the list of active segments. The first segment of the two is stored with a new and reduced upper bound, using the fact that it is shorter. The second segment is stored with the upper bound calculated for the segment that was removed from the list. Whenever the best known solution is updated, the list is cleaned from segments with upper bounds which are within the allowed error from the current best known solution. The process terminates when the list is empty. ■

Theorem 3.4.4 *Algorithm 3.4.1 terminates in a finite number of iterations and achieves an approximate solution of value* $\min\{OPT - \mathcal{A}, OPT \cdot (1-\mathcal{R})\}$.

Proof. To see the termination of the algorithm observe that at any step of the algorithm, a segment is removed from the list and two, one or none new segments of half length of the removed one, are added to the list. So clearly, the lengths of the segments added to the list gradually decrease by at least half each iteration.

We show that no segment of length shorter than $\frac{\mathcal{A}}{2\gamma_0}$ can be added to the list, and thus, at some point, the list becomes empty and the algorithm stops. Consider a step in which the algorithm processes a segment of length $l \leq \frac{\mathcal{A}}{\gamma_0}$, say the segment $[a_0, a_0+l]$. At this point, the value of $P^*(a_0)$, was already evaluated and thus $P^*(a_0)$ is not greater than the value of the current "best known solution". The slope of the segment is not greater than γ_0 and so the value at any point within the segment is not greater than $P^*(a_0) + l \cdot \gamma_0 \leq P^*(a_0) + \mathcal{A}$ and thus no halves of this segment are added to the list. Hence, at some step, all segments in the list must be shorter than $\frac{\mathcal{A}}{\gamma_0}$ and the algorithm terminates. Clearly, from the above discussion, the algorithm terminates with the required approximate solution. ■

Note that at the worst case our branch and bound procedure calls Algorithm 3.3.1, $\frac{\gamma_0}{x_1\mathcal{A}}$ times. Our numerical tests presented in Subsection 3.5.1 shows that instances up to thousand machines can be approximated with a relative error of 1% in very few seconds.

Remark 3.4.5 If the absolute error is set to $\mathcal{A} = 0$, then the theoretic convergence of Algorithm 3.4.1 can not be guaranteed regardless of the magnitude of $\mathcal{R}$. This is because if the optimal profit is of value zero and the bound on the slope of the function obtained by Corollary 3.4.3 is strictly positive at some neighborhood of the optimal arrival rate, then the stopping condition of the algorithm never holds for $\mathcal{A} = 0$. However, this has no practical implication since there is always some absolute error allowed by the floating point accuracy of the computer.

Remark 3.4.6 Note that although throughout this paper we assumed the Poison arrival process, the developed methods produce good approximate solution (in the heuristic sense) to other arrival processes. This phenomenon is indicated in the literature and was also observed by our numerical experiments as described in the next section. It is widely believed that in a tandem of N stations, if the arrival process is stationary and ergodic with a rate of α and the system is stable, then the departure process from the n^{th} station converges to a Poisson process with a rate of α as $n \to \infty$. This conjecture is known as Reiman and Simon conjecture and was partially proved by Mountford and Prabhakar [29] for the case of identical stations. Further more, a simulation study conducted by Suresh and Whitt [38] indicates that the convergence rate, in terms of the number of machines in the tandem, is fairly high if the arrival process admits low variability and in particular when the arrival process is deterministic (e.g., the inter-arrival times between any successive arrivals are constant).

3.5 Numerical Results

Our study consists of three sets of experiments described in the subsections below. In 3.5.1 we test the applicability of Algorithms 3.3.1 and 3.4.1 for very large instances of the problem. We solved instances with 1000 machines in a very short time. In 3.5.2 and 3.5.3 we check the assumption that large instances of the problem are not that sensitive to the type of the arrival process but rather to its rate. In particular, we examine this assumption for the deterministic arrival process. In 3.5.2 we show that for given configuration and arrival rate, the average amount of work in process is similar for the Poisson and the deterministic arrival processes. Recall that we denote an arrival process deterministic if the times between any successive arrivals are constant. Note that, for a fixed arrival rate, the arrival process affects the WIP and thus if the WIP is similar for both cases, then the total operational costs per time unit are also similar. In 3.5.3 we show that the optimal QCS configuration for a Poisson process remains nearly optimal when the Poisson arrival process is replaced by deterministic one of the same rate. We compare the results obtained from Algorithm 3.3.1 with the simulation results of all 2^N possible QCS configurations. For obvious reasons, this experiment is restricted to short production lines. We have tested it on eight machines lines. The created test problem instances differed by the following three criteria:

1. **Success probabilities**: Groups denoted by L possess relatively low success probabilities while H denotes those possess high ones. The success probabilities of the 'H' instances were generated such that $q_{N,0} = \prod_{i=1}^{N} p_i \approx 0.8$ and for the 'L' instances it is $q_{N,0} \approx 0.4$.

2. **Tendency of the processing rates along the line**: For instances denoted by R the expected processing times were sampled from a common distribution (i.i.d) and for those denoted by I, the expected processing times were generated

in a way that insures strictly increasing processing times in i, the index of the station.

3. **Tendency of the holding costs along the line**: In problems denoted by R, the holding costs $\mathbf{h}$ and $\mathbf{h}'$ were taken from a common distribution (i.i.d) for all stations and for those denoted by I (D) the holding costs were generated to be monotonously increasing (decreasing) in i, the machine index.

There are 12 combinations of these criteria. A problem instance is denoted by three letters and the number of machines. For example, a problem denoted by $HRD100$ is one with High success probabilities, arbitrary Random processing times, Decreasing holding costs and 100 machines. We believe that these 12 combinations represent variety of systems that can be found in reality. Note that we did not include in our data set problems with decreasing processing times. Such systems are in general easier to analyze since their bottleneck machine is always the first one and thus the QCS configuration admits no effect on the line throughput.

3.5.1 The Efficiency of the Algorithms

In order to check the applicability of our Algorithm 3.4.1 we randomly generated 600 instances with 1000 machines each. The instances are divided into 12 groups as described above (50 systems in each group). For each system we constructed two revenue functions. One is a linear homogenous function $r(x) = \alpha \cdot x$ and one is a non-linear of the shape $r(x) = \beta\sqrt{x}$. The constants α and β were randomly selected in a manner that assures the existence of a profitable solution; this, in order to avoid trivial instances of the problem. Our algorithm was applied for these 1200 problems; the running times in seconds and the number of iterations (calls to Algorithm 3.3.1) were collected. Statistics results of this experiment are presented in Table 3.1. The relative optimality error was set to 0.001 (0.1%) and the absolute optimality gap was set to 0 (which practically means that the absolute error is set to the numerical accuracy of the computer).

The algorithm was implemented in Microsoft Visual C++ with LEDA (see [28]) on an Intel Pentium 4, 2Ghz CPU with 512Mb RAM. The source code and data set are available from our site http://ie.technion.ac.il/∼talraviv/Publications.

Model	Linear $r(x) = C \cdot x$			Concave $r(x) = C \cdot \sqrt{(}x)$		
	Average time	Worst time	Average # iterations	Average time	Worst time	Average # iterations
LII1000	0.997	1.673	8.54	3.667	4.567	30.46
LID1000	1.076	1.783	8.70	3.896	4.596	31.14
LIR1000	1.024	1.562	8.44	3.767	4.507	30.52
LRI1000	2.004	2.604	12.00	4.542	6.069	32.60
LRD1000	1.937	2.774	11.44	4.562	5.498	33.06
LRR1000	1.986	2.603	11.82	4.542	5.758	32.52
HII1000	1.716	2.413	11.94	4.640	5.438	35.54
HID1000	1.711	2.494	12.26	4.717	5.978	36.64
HIR1000	1.751	2.473	12.10	4.740	5.899	36.10
HRI1000	1.973	3.265	12.46	4.561	5.508	33.86
HRD1000	1.913	2.864	12.50	4.581	5.739	34.36
HRR1000	1.928	2.834	12.42	4.489	5.398	33.62

Table 3.1: The average and worst case running times of Algorithm 3.4.1 in seconds and the average number of calls to Algorithm 3.3.1 are presented for the two different revenue functions.

From Table 3.1 it is apparent that Algorithms 3.3.1 and 3.4.1 can be employed to solve efficiently the problems presented in the paper under diverse sets of conditions and for any reasonable size. In particular, we believe that 1000 machines is a reasonable upper bound on the size of serial production lines encountered in real life and the relative optimality guarantee of 0.1% is in most cases more accurate than the problem parameters. Note that Algorithm 3.3.1 is a subroutine called numerous times in the solution process of Algorithm 3.4.1. Thus, the problem of determining an optimal QCS configuration for a given arrival rate in a thousand machines line is solved within a fraction of a second.

3.5.2 Insensitivity of the Total WIP to the Arrival Process

Recall that in Remark 3.4.6 we argued that in long production lines, the optimal configuration for a system with a Poisson arrival process is likely to be optimal, or near optimal, for other arrival processes of the same rate. Also, by flow conservation, it is clear that at steady state the arrival rate of jobs into each station is not affected by the type of the arrival process but only by its rate. Thus, the only component of the cost affected by the arrival process is the holding cost of the work in process. Here we use simulation to demonstrate the fact that in long production lines, replacing the Poisson arrival process by a deterministic one, admits only a minor effect on the expected level of work in process and thus on the total cost per time unit.

We constructed 12 instances of 100 machines each, one for each of the categories described at the beginning of this section. The data set is available from our site. For each of these instances, we calculated a tight upper bound λ_{max} on the arrival rate using Corollary 3.2.3 and assuming all QCSs are installed. We used the dynamic programming Algorithm 3.3.1 to find the optimal QCS configurations, assuming Poisson arrival process for three different arrival rates, $0.5\lambda_{max}$ (Light), $0.8\lambda_{max}$ (Medium) and $0.95\lambda_{max}$ (High). We then used simulation to evaluate the average amount of work in process at steady state, assuming deterministic arrival process in the same rates.

For our 36 test problems, the average level of the WIP in the deterministic arrival case, was about 83-92% of the expected level of the Poisson arrival case, as shown in the right most column of Table 3.2. As expected, it is apparent from the table that the levels of WIP are lower for the deterministic arrival process as compare to Poisson process. This is due to the larger variability of the Poisson Process as compare to the deterministic one. Note that for a given system, the ratio between the two WIP levels for Poisson and deterministic arrival process is not very sensitive to the arrival rate and to the specific QCS configuration. This important observation makes possible the use of an optimal solution of the Poisson arrival case as an approximate solution for the deterministic one in the parametric QCS problem.

Figure 3.2 demonstrates the similarity in WIP levels of the systems with Poisson and the deterministic arrival processes. We consider the problem HRR100 shown in Table 3.2, with a medium arrival rate and an optimal QCS configuration for this rate (under the Poisson arrival process). We simulated the same system with a

Problem	Arrival Rate	Poisson Expected WIP	QCSs locations (optimal for Poisson arrivals)	Deterministic Average WIP	Ratio
LII100	Light	83.500	{9, 19, 28, 40, 48, 62, 75, 88}	73.1563±0.234881	0.876
LII100	Medium	251.349	{4, 9, 13, 19, 27, 34, 40, 48, 58, 66, 75, 88}	215.948±0.933386	0.859
LII100	Heavy	546.464	{4, 7, 9, 13, 19, 26, 31, 36, 44, 52, 58, 62, 68, 75, 80, 88, 93}	454.764±2.89178	0.832
LID100	Light	84.725	{8, 19, 29, 40, 53, 63, 74, 88}	73.054±0.227335	0.862
LID100	Medium	251.494	{7, 16, 19, 26, 36, 45, 53, 63, 74, 83, 92}	213.251±1.09989	0.848
LID100	Heavy	545.095	{4, 7, 10, 16, 20, 26, 31, 36, 40, 45, 53, 59, 66, 74, 79, 88, 92}	451.915±2.86822	0.829
LIR100	Light	84.711	{16, 26, 41, 48, 61, 70, 88}	72.3496±0.224759	0.854
LIR100	Medium	252.344	{7, 13, 20, 26, 31, 41, 49, 54, 61, 70, 79, 88}	213.644±1.09059	0.847
LIR100	Heavy	544.479	{4, 7, 11, 16, 20, 26, 31, 36, 44, 49, 54, 59, 65, 73, 79, 88, 93}	451.495±2.99134	0.829
LRI100	Light	26.810	{9, 21, 27, 44, 54, 66, 78, 89}	23.2582±0.05171	0.868
LRI100	Medium	57.929	{9, 14, 21, 27, 35, 44, 54, 59, 66, 73, 81, 89}	50.2465±0.16899	0.867
LRI100	Heavy	92.249	{1, 6, 11, 14, 21, 27, 35, 44, 54, 59, 66, 73, 81, 89}	78.6892±0.41424	0.853
LRD100	Light	26.955	{13, 27, 44, 54, 61, 69, 81, 89}	23.3166±0.04993	0.865
LRD100	Medium	58.155	{9, 14, 21, 27, 44, 54, 61, 69, 81, 89}	49.8765±0.15807	0.858
LRD100	Heavy	92.630	{1, 5, 11, 14, 21, 27, 44, 54, 61, 69, 81, 89}	78.5607±0.42037	0.848
LRR100	Light	26.540	{12, 21, 26, 44, 59, 69, 81}	23.0335±0.04873	0.869
LRR100	Medium	57.833	{9, 14, 21, 26, 44, 59, 69, 81, 84}	49.5396±0.16372	0.857
LRR100	Heavy	92.493	{1, 6, 11, 14, 21, 27, 44, 54, 59, 69, 81, 92}	78.0101±0.3817	0.843
HII100	Light	72.318	{22, 50, 69}	65.6968±0.12966	0.908
HII100	Medium	202.430	{22, 36, 54, 69, 81}	183.648±0.6844	0.907
HII100	Heavy	397.066	{8, 22, 36, 51, 60, 69, 81, 91}	357.881±2.1994	0.901
HID100	Light	72.074	{32, 50, 69}	65.6725±0.12118	0.911
HID100	Medium	202.339	{16, 34, 50, 69, 82}	183.575±0.601957	0.907
HID100	Heavy	396.768	{16, 34, 46, 60, 69, 81, 91}	354.541±2.04448	0.894
HIR100	Light	72.141	{15, 37, 54, 81}	66.1531±0.12889	0.917
HIR100	Medium	201.893	{16, 37, 54, 66, 81}	183.02±0.672422	0.9071
HIR100	Heavy	396.545	{15, 27, 37, 49, 62, 72, 81, 91}	358.81±2.05424	0.905
HRI100	Light	39.911	{6, 18, 34, 50, 65, 85}	36.3588±0.05524	0.911
HRI100	Medium	100.139	{10, 18, 34, 50, 65, 75, 85}	90.8305±0.2501	0.907
HRI100	Heavy	192.310	{2, 6, 14, 18, 24, 35, 48, 57, 65, 75, 85, 89}	172.365±1.18	0.896
HRD100	Light	39.866	{10, 18, 34, 48, 75, 89}	36.2967±0.05183	0.910
HRD100	Medium	100.007	{2, 10, 18, 34, 44, 50, 65, 75, 85, 89}	91.0597±0.27339	0.911
HRD100	Heavy	192.270	{2, 8, 14, 18, 24, 34, 44, 48, 57, 65, 75, 85, 89}	171.986±1.1113	0.895
HRR100	Light	39.544	{10, 24, 34, 48, 63, 75, 82}	36.4027±0.05455	0.921
HRR100	Medium	99.586	{10, 24, 34, 48, 57, 63, 75, 82, 90}	91.2471±0.28752	0.916
HRR100	Heavy	191.904	{2, 10, 14, 24, 35, 48, 57, 65, 75, 85, 90}	172.534±1.2376	0.899

Table 3.2: A comparison between the expected total level of WIP for the Poisson and the deterministic arrival processes. For the deterministic case, the average total WIP level was estimated using simulation and is presented within a 95% confidence interval. The ratio between the two levels is presented in the right most column.

deterministic arrival process to estimate the expected WIP in each of the system's stations. We then compare these values with the expected WIP level in the system with a Poisson arrival process (calculated analytically). In Figure 3.2 the ratio between these two values is plotted against the station index (either machine or QCS). It is apparent from the graph that for stations located at the beginning of the line the ratio fluctuates widely while for further on stations it tends to stabilize close to unity.

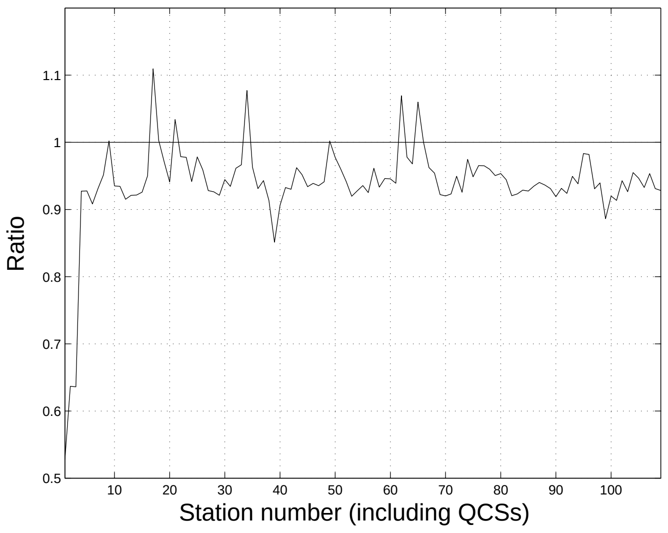

Figure 3.2: The ratio between the WIP levels assuming Poisson and deterministic arrival processes of the same rate.

3.5.3 Comparison of the Exact and Approximate Solutions

As mentioned above, we believe that optimal QCS configurations for the Poisson arrival process frequently remain optimal or near optimal for non-Poison arrival processes and in particular for the deterministic arrival process as defined above. In this section we supply further numerical support for this assumption.

Twelve systems of eight machines each based on our 12 categories described at the beginning of this section were constructed. Each problem was solved, using Algorithm 3.3.1 for three different arrival rates. The arrival rates were selected in order to cover diverse sets of conditions, according to the following method. Corollary 3.2.3 was used to obtain λ_{max}, an upper bound on the feasible arrival rates assuming all eight QCSs are installed. The following arrival rates $\lambda_{low} = 0.5\lambda_{max}$, $\lambda_{med} = 0.8\lambda_{max}$ and $\lambda_{high} = 0.95\lambda_{max}$ were considered.

The procedure recently proposed by Nelson et al. [30] was used to obtain a near optimal configuration for the problem with deterministic arrival process. This procedure finds, with a pre-specified probability $(1 - \alpha)$, a solution which is optimal or within a pre-specified *Indifference Zone* from the optimum. We applied the above procedure with an indifference zone of 2% and $\alpha = 0.05$. It should be noted that Nelson et al. procedure is practical only for very small instances of our problem since the procedure repeatedly runs numerous simulation sessions for each of the exponentially many possible configurations. In our case, each of the eight machines problem with the above confidence level and indifferent zone, took several minutes to solve.

For the optimal solutions, those obtained by Algorithm 3.3.1 and by Nelson's procedure, further simulations under the deterministic arrival process were conducted, and an estimator of their expected cost with a relative confidence interval of 0.1% was obtained.

Table 3.3 compares the solutions obtained by Algorithm 3.3.1 with those obtained by Nelson et al. method where both used for the problem with deterministic arrival process. The values in the "ratio" columns were calculated as follow,

$$100 \times \left(\frac{\text{Optimal profit of a solution obtained by Nelson procedure}}{\text{Optimal profit of a solution obtained by Algorithm 3.3.1}} - 1 \right).$$

Prob-	Low Rate ($0.5\lambda_{max}$)			Medium Rate ($0.8\lambda_{max}$)			High Rate ($0.95\lambda_{max}$)		
lem	Ratio	Optimal Configuration		Ratio	Optimal Configuration		Ratio	Optimal Configuration	
		DP 3.3.1	Nelson		DP 3.3.1	Nelson		DP 3.3.1	Nelson
LII8	-	00010001		-	01010101		1.7%	11111101	11110101
LID8	-	00100001		0.4%	01010001	00110001	-	11111001	
LIR8	-1.1%	00100001	00100011	-	00100101		-	10101011	
LRI8	-0.0%	01000001	00100001	-0.4%	01000001	00100001	-0.0%	00100001	00010001
LRD8	-1.2%	00100001	01000001	-1.3%	00100001	01000001	-0.1%	00010001	01010001
LRR8	-	01000001		-	01000001		-	01000001	
HII8	-	00000001		-0.2%	00010001	01000011	-	01010010	
HID8	-	00000000		-1.2%	00100000	0000000	-	00100100	
HIR8	-	00000010		-	00000010		-	00100010	
HRI8	-1.1%	00000001	00000010	-	00000010		-	00000010	
HRD8	-	00000010		-	00000010		-	00000010	
HRR8	-1.6%	00000010	00000001	-2.4%	00000010	00000001	-0.3%	00000010	00000011

Table 3.3: A comparison between the profit and the QCS configurations for the eight machines system obtained by Algorithm 3.3.1 and by Nelson et al. procedure.

From Table 3.3 it is apparent that for our 36 test problems, Algorithm 3.3.1 returns solutions which are either optimal or very close to optimal. The differences between the solutions obtained by both methods can be partly explained by the estimation error.

Recall that the total costs of a given system under Poisson arrival process and under deterministic one differs only in the holding costs. Thus, for low holding costs it is not surprising that the optimal solutions are similar for both cases. In order to show that the above phenomenon holds also for relatively high holding costs we examined the proportion of holding costs relative to the total expenses. We observed that for the instances presented in Table 3.3 holding costs were on the average 26.1%

of the total expense with a range of 10.2% to 48.2%. We conclude that our method works well even if holding costs are a substantial part of the total costs.

3.6 Discussion

In this paper we presented a dynamic programming algorithm and a branch and bound strategy to solve the problem of determining an optimal QCS configuration along a serial production line. Two versions of this problem were considered. Namely, minimization of the cost per time unit under a given production rate and maximization of the profit where the QCS configuration and the production rate are to be selected simultaneously.

Note that in the minimization problem the production rate is defined by the number of jobs arrive at the first machine per time unit. However, this is completely equivalent to define the rate in terms of non-defective products completed products delivered for the system. This is because the ratio between the departure rate of non-defective products out of the system and the arrival rate of jobs into the system (given by $q_{N,0}$) is uniquely determined by the success probabilies and is not affected by the QCS configuration.

We point out that the model discussed in this paper, as oppose to previous studies in the literature, captures the effect of the inspection process on the line throughput and on the level of work in process. We suggest the following managerial insights,

1. Installing additional QCS may increase the line throughput by reducing the load caused by defective products on bottleneck machines.

2. The effect of installing additional QCSs on the WIP level, assuming fixed production rate, is determined by two conflicting factors. The reduction of the load on the machines and QCSs toward the end of the line obtained by installing additional QCSs, reduces the queues in front of these stations and thus reduces the expected WIP there. On the other hand, an additional QCS hads its own queue and processing time which increases the total expected WIP level in the system. It is not possible to make a general statement whether the former or the later factor governs. It is a challenge to theoretically analyze the connection between these two conflicting factors for some special cases.

3. In many cases, QCSs installed before loaded machines increase the throughput and reduce the WIP while a QCS installed after the slowest machine in the line has no effect on the throughput and only minor effect on the WIP level.

Throughout the analytical part of the paper we assumed that the arrival process of jobs into the system is Poisson. This assumption is not suitable for most real life production environments where the jobs are dispatched into the system by a decision of the system operator. We used numerical experiments to demonstrate that the Poisson arrival assumption leads to near optimal solutions also for deterministic arrival processes. This is true at least for small instances of the problem, for which we were able to estimate the optimal solutions by enumerative method. However, we expect the method to be even more accurate for larger instances since the arrival process into each machine i stochastically approach Poisson process as $i \to \infty$. See the discussion on Reiman and Simon conjecture in Remark 3.4.6 above.

In addition to the Poisson arrival assumption the following assumptions were made:

1. **Idleness policy:** Each machine starts an operation whenever it becomes ready and a job is available in its buffer. Note that this rule is not necessarily an optimal policy, especially if holding costs vary along the line.

2. **Random variables:** The operations' failures on each machine are independent across jobs. This assumption may not hold in various cases when the failures depend on the state of the machine or other external factors such as dusty environment.

3. **Inspection Policy:** Once a QC_i is installed, it inspects all the jobs that depart from machine i. Note that this policy may be sub-optimal even under the assumption that the failures events on each machine are independent. In particular, for slow QCSs it might be better to inspect subsets of the jobs, so part of the benefit from inspecting is gained without creating a new bottleneck in the system. In general, the decision whether to check a job or not should be made on-line, based on the buffers state along the line.

4. **Perfect Inspections:** We assume a perfect inspection process. In practice

two types of errors may occur: A defective job may not be identified as such and a non-defective job may be rejected as defective one. In addition, in some cases, the inspection operation itself may damage the product.

Relaxing the above assumptions is an important direction for further research. However, despite of the assumptions mentioned above we believe that our model is useful for designing Quality Control Systems. In particular, we feel that the understanding of the various factors influencing the system performance, and the insights gained by our analysis, are of much importance for further research on this applicable problem.

Note that although our aim in this study was to optimize the steady state performances of the systems, we believe that the method is also well suited for high multiplicity problems where a large but finite number of identical or similar products are to be produced, and the goal is to minimize the total cost or maximize the total profit. The model presented in this paper can be extended to capture a variety of manufacturing environments such as allowing repairs, reworks, machine breakdowns and for other manufacturing environments such as job shop, assembly lines, multi-stage shop, etc. In addition, the ideas presented here can be adapted for other areas such as determining optimal nodes for performing integrity checks along a communication network or during a long service process.

Chapter 4

Long Run Maximum Profit Job Shop Problem

Many systems in the fields of manufacturing, transportation and data communication can be considered as discrete flow control systems, e.g., systems consist of a network and some distinct objects that should be transferred along its arcs according to some rules. Often, optimization problems related to these systems are NP-Hard. For example, many scheduling problems are such problems and are well known for their intractability. Surprisingly, it turns out that in a way, large instances of this kind of problems are not so difficult to approximate using fluid approximation techniques.

Given a discrete flow control system, its *fluid counterpart system* can be defined by relaxing the discrete nature of the objects, that is allowing each discrete object to be divided into infinitesimally small fractions. Since the continuous nature of the fluid systems make them easier to analyze and optimize, it is often useful to construct such a system whenever one wishes to devise a method for evaluation or optimization of the discrete flow system. This construction may lead to efficient approximation or heuristics procedures and in some cases even to an efficient optimization algorithms.

The fluid view is extensively used for analyzing queueing systems, see for example Chen and Mandelbaum [9], and Dai and Weiss [12]. However, only recently it has been used for solving combinatorial optimization problems. In particular this approach was previously used to tackle the High Multiplicity Job Shop Problem by Bertsimas and Gammarnik [5], by Boudoukh, Penn and Weiss [7]; by Dai and

Weiss [13]; Bertsimas and Sethuraman [6]. Bertsimas and Gammarnik also studied a Packet Routing Problem. Penn and Raviv [33] considered a Vehicle Routing Problem.

Here we consider a production model where many instances of a small amount of product types are to be produced according to a Job Shop setting. The planner has to determine simultaneously the production mix and the schedules in order to maximize the expected steady state profit.

We open the discussion with a simplified version of the problem in which the long run average gross profit per time unit is maximized. In this version we allow the system to use arbitrarily large (finite) safety stocks. Consequently, a long initialization phase, to build this safety stocks, may be required. In addition, the size of the system buffers and the average level of work in process may be large.

Ideally, one would like to maximize the net profit, where holding cost, space cost (of the buffers) and delay cost (of the initialization phase) are considered. We could not meet this ultimate goal. Instead, we suggest a three phase optimization process. At the first step, the gross profit is minimized, using a fluid approach. At the second phase the solution is modified in order to construct a cyclic schedule with very short cycles that still yields approximately optimal gross profit. We show numerically, on standard bench mark problems, that a compromise of 1% on the optimality of the gross profit enables the creation of very short and simple cyclic schedule. Once a short cycle is constructed, it is possible to sequence it in a manner that minimizes the measures of holding costs, buffers spaces, required safety stocks or any weighted combination of the three. This last phase can be carried out by standard scheduling and combinatorial optimization techniques and is out of the scope of this paper.

The rest of this paper is organized as follow: In Section 4.1 we define the problem and formulate a fluid relaxation of it. In Section 4.2 we construct a dispatching rule based on the fluid solution and prove its optimality. In section 4.3 we provide an upper bound on the difference between to the total number of products produced in the discrete system and the amount of fluid flows out of the fluid one. In Section 4.4 we explore the cyclic nature of the obtained schedule and show how to construct an approximately optimal schedules with a simple cycle structure. In section 4.5 we discuss a method to initialize the system and to adjust it to modification in optimal production mix. Some of the concepts presented in the paper are demonstrated using a numerical example in Section 4.6 and their practicality is supported by an extensive numerical experiments reported in Section 4.7. We conclude by conducting a short discussion and drawing few directions for further research in Section 4.8.

4.1 Problem Definition and Fluid Relaxation

Our aim is to operate a job shop system in a manner that maximizes its operational profit. We first define two types of such profit.

Definition 4.1.1 (Gross operational profit) . *The gross operational profit per product is the total revenue for the product minus direct costs associated with its production such as materials, labor and energy. Throughout we assume that this profit per product is a constant regardless of the quantities of products produced. That is, the gross operational profit from the system is proportional to the number and types of products.*

Definition 4.1.2 (Net operational profit) *The net operational profit from the system is its gross operational profit minus the following additional costs:*

1. *Holding costs - alternative cost of work in process.*
2. *Space cost - alternative cost of the space allocated for the system buffers*
3. *Delay cost - alternative cost of the time required for the initialization phase.*

The terminology used above is precisely defined below. These three types of costs are interrelated but not necessarily proportional to each other.

In the sequel we omit the word "operational" when referring to gross and net operational profit since we only consider these kinds of profit.

In this paper we take the following three-step approach. First, the gross profit is minimized, using a fluid approach. Then, the solution is modified in order to construct a cyclic schedule with very short cycles that still yields approximately optimal gross profit. It is shown on numerically, on standard bench mark problems, that a compromise of 1% on the optimality of the gross profit enables the creation of very short and simple cyclic schedule. Once a short cycle is constructed, it is possible to change its sequence. The sequence's change should not affects the cycle length but it may reduce the level of work in processes, buffers spaces, required safety stocks or any weighted combination of the three. This last phase can be carried out by standard scheduling and combinatorial optimization techniques and is out of the scope of this paper.

Consider the following Job Shop like problem: Let M be a set of different machines and R a set of product types. The production process of a product of type r consists of a series of K_r operations that should be carried out on the machines according to a given order. We refer to this order as the product route. We allow reentrant route, that is, a route may visit the same machine several times. The o^{th} operation of route r is denoted by (r, o) and the time needed to carry out operation (r, o) is $T_{r,o}$. The machine that carry out operation (r, o) is denoted by $M(r, o)$ Let σ_i denote the set of operations carried out by machine i. $\sigma(r, o) \equiv \sigma_{M(r,o)}$ and $\sigma(r, o)^- \equiv \sigma(r, o) \setminus \{(r, o)\}$. As in the classic Job Shop problem, each machine may work on a single job (product instance) at a time and a job may be processed by a single machine at a time. We consider non-preemptive policy, that is, an operation can not be aborted before it is finished. Each completed product of type r yields a gross profit of P_r. Without loss of generality we assume $P_r > 0$ since otherwise products of type r will not be produced at all. The aim is to select a subset of products and to construct a schedule that maximizes the total average gross profit per time unit achieved if the system is to be operated **indefinitely**.

The *fluid counterpart* of our system is defined as follows: Machine i is represented by pump i and route r by a pipe that traverses the pumps according to route r. Each unit of fluid in a pipe represents a single job. The maximum rate in which pump i can draw fluid from a pipe entrance is $\frac{1}{T_{r,o}}$ assuming it devotes all its effort to this buffer. However a pump may divide its effort among several entering pipes. For example, if a pump devotes α of its effort to (r, o) then the inflow from this entrance is limited to $\frac{\alpha}{T_{r,o}}$. We associate a gross profit of P_r with each unit of volume of fluid coming out of pipe r. The aim is to maximize the average gross profit of the system for a unit of time. Note that this description is equivalent to allowing each product to be divided into infinitesimal fractions and allow fractions of the same product to be concurrently processed by different machines.

The net profit of the fluid system equals its gross profit. The fluid system use no WIP and thus of no holding cost occurs; It uses no buffers and thus have no space costs; and it needs no initialization phase and thus admit no delay cost.

The solution to the fluid problem is given by the following simple Linear Program:

Linear Program 4.1.1

$$\max \sum_{r=1}^{R} P_r x_r$$

$$\sum_{(r,o)\in\sigma_i} T_{ro} x_r \leq 1 \qquad \forall i = 1, ..., M \tag{4.1}$$

$$x_r \geq 0 \qquad \forall r = 1, ..., R.$$

Here x_r denotes the flow rate via pipe r. We shall see that valuable information can be derived from an optimal solution of Linear Program 4.1.1. This information can be helpful when seeking for an optimal solution of the more interesting discrete system.

4.2 Adapting the Fluid Solution to the Discrete System

In this section we present a method to build an infinite schedule for the discrete system based on the optimal solution of its fluid counterpart. Our schedule consists of two phases: The *initialization phase* and the *regular phase*. The initialization phase is finite and its schedule can be described as a solution of a standard job-shop problem. Once the initialization phase is over, all the machines become ready. The regular phase can be described by a *dispatching rule*, that is, a method to decide for each machine what to do whenever it turns ready.

In the discrete system each operation (r, o) for $o = 2, ..K_r$ is associated with a buffer. Each such buffer contains all the jobs that have already finished operation $(r, 0-1)$ but not yet started operation (r, o). Let $B_{(r,o)}(t)$ denote the number of jobs in buffer (r, o) at time t.

The purpose of the initialization phase, in our proposed method, is to fill the buffers with large enough quantity of work in process in order to allow smooth operation of the system during the regular phase with respect to our dispatching rule. That is, to enable the operation of the dispatching rule in such a manner that it never tries to dispatch a job that is not available in its buffer. The level of work in process accumulated in the buffers at the end of the initialization phase is called *safety stock*. The two related problems of how to calculate the minimum level of safety stock required for our dispatching rule and how to construct the schedule of the initialization phase accordingly, are discussed later on. For now we use the term *sufficient safety stock*, as if these problems already resolved. Below we discuss the dispatching rule for the regular phase. We use the argument t to denote the time passed since the beginning of this phase. Thus, the level of the safety stock used is $B_{ro}(0)$. Let $D_{r,o}(t)$ denote the total number of operations of type (r,o) completed by time t.

Recall that in the fluid counterpart system each operation (r,o) is being processed in a constant rate x_r and thus during t units of time, $t \cdot x_r$ units of volume goes through the pipe associated with route r.

Definition 4.2.1 *The* ***lateness*** *of operation (r,o) at time t is $\mathcal{L}_{ro}(t) = t \cdot x_r - D_{ro}(t)$. It represents the difference between the volume of the fluid that moved along route r by time t in the fluid system and the number of jobs that completed operation (r,o) by time t in the discrete system.*

We are now ready to present the dispatching rule that governs the operation of the system during the regular phase.

Definition 4.2.2 (FBDR - Fluid based dispatching rule) *At any given time t, if machine i is idle and $\max_{(r,o)\in\sigma_i} \mathcal{L}_{ro}(t) \geq 0$, then process a job from buffer (r,o) with maximum value of $\mathcal{L}_{ro}(t)$. Otherwise, machine i stays idle until the situation is changed. Ties are broken in favor of operations with larger value x_r and if few operations have equal value of x_r by lexicographic order of (r,o).*

Propsition 4.2.3 *Consider a discrete system working indefinitely according to FBDR. At any time t when operation of type (r,o) is* ***not*** *being processed there is a finite constant γ_{ro}, independent of t, such that $\mathcal{L}_{ro}(t) \leq \gamma_{ro}$.*

In the next section we establish an upper bound on γ_{ro}, which indirectly proves Proposition 4.2.3. Theorems 4.2.6, 4.2.5 and 4.2.9, in the sequel, shows the virtues of FBDR.

Lemma 4.2.4 *Consider a discrete system working indefinitely according to FBDR. At any time t we have $\mathcal{L}_{ro}(t) < \gamma_{ro} + x_r T_{ro}$. That is, the lateness of operation (r, o) is at most $\gamma_{ro} + x_r T_{ro}$.*

Proof. By Proposition 4.2.3 this is a true whenever the operation is not being processed. Consider a time t when operation (r, o) is being processed and assume the processing was started at time t_0. Then

$$\mathcal{L}_{ro}(t) = tx_r - D_{ro}(t).$$

Now, clearly $D_{ro}(t) = D_{ro}(t_0)$ and so

$$\mathcal{L}_{ro}(t) = t_o x_r - D_{ro}(0) + (t - t_0)x_r < \mathcal{L}_{ro}(t) + x_r T_{ro}.$$

The last inequality is due the fact that $t - t_0 < T_{ro}$. ■

Theorem 4.2.5 *If a system is scheduled according to FBDR, then a safety stock of $\lfloor \gamma_{r,o-1} + 1 \rfloor$ jobs in buffer (r, o), for each (r, o), suffices.*

Proof. We prove the theorem by showing that when using FBDR and allowing negative levels of stock in the system's buffers, if $\mathcal{B}_{ro}(0) = \lceil \gamma_{r,o-1} \rceil$ then $\mathcal{B}_{ro}(t) \geq 0$ for all t. The number of jobs in buffer (r, o) at time t is

$$\mathcal{B}_{ro}(t) \geq \mathcal{B}_{ro}(0) + D_{(r,o-1)}(t) - D_{r,o}(t).$$

Note that this is an inequality rather than equality since while operation (r, o) is being processed the number of jobs that were taken from the buffer exceeds $D_{ro}(t)$ by one. By the definition of the lateness,

$$\mathcal{B}_{ro}(t) \geq \mathcal{B}_{ro}(0) + D_{(r,o-1)}(t) - tx_r + tx_r - D_{r,o}(t) = \mathcal{B}_{ro}(0) - \mathcal{L}_{r,o-1}(t) + \mathcal{L}_{r,o}(t).$$

Now consider R, the set of points of time when the machine $\sigma(r, o)$ is ready and an operation (r, o) is about to be dispatched. For these times we have $\mathcal{L}_{r,o}(t) \geq 0$ and so,

$$\mathcal{B}_{ro}(t) \geq \mathcal{B}_{ro}(0) - \mathcal{L}_{r,o-1}(t).$$

In order to have a job available for this operation it must be the case that $\mathcal{B}_{ro}(t) \geq 1$. Now, by proposition 4.2.3, $\mathcal{L}_{r,o-1}(t) \leq \gamma_{r,o-1}$ and so if $\mathcal{B}_{ro}(0) = \lfloor \gamma_{r,o-1} + 1 \rfloor$ then

$$\mathcal{B}_{ro}(t) \geq \lfloor \gamma_{r,o-1} + 1 \rfloor - \gamma_{r,o-1} > 0.$$

Now since $\mathcal{B}_{ro}(t)$ is integer we have

$$\mathcal{B}_{ro}(t) \geq 1.$$

■

Theorem 4.2.5 established the proper definition of the term *sufficient safety stock* by showing that there is always a finite level of initial stock that allows the system to operate indefinitely under FBDR.

Theorem 4.2.6 *Using FBDR relative to* $\mathbf{x}$*, a solution of Liner Program 4.1.1, with sufficient safety stocks and for* t *units of time, the average gross profit per time unit of the discrete system converges to the gross profit per time unit of the fluid one as* $t \to \infty$*.*

Proof. It is enough to show that by time t, the number of products of each type completed by the discrete system is within a constant of the amount of fluid that flowed out of the corresponding pipe in the fluid system. The difference between these two values is $\mathcal{L}_{r,K_r}(t) = tx_r - D_{r,K_r}(t)$, which is bounded above by a constant as showed in Proposition 4.2.3 and Lemma 4.2.4, plus some finite time needed to carry out the initialization phase.■

Lemma 4.2.7 *Assume a system operates according to FBDR with sufficient safety stocks. Then, at any given time* t *and for any operation* (r,o)*, the following inequality holds*

$$\mathcal{L}_{r,o}(t) \geq -1 + x_r T_{ro}.$$

Proof. First note that $\mathcal{L}_{r,o}(t)$ decreases by one each time when an operation (r,o) is completed and increases everywhere else. That is, $\mathcal{L}_{r,o}(t)$ achieves its global minimum either when an operation of type (r,o) finishes its processing or at $t = 0$ ($\mathcal{L}_{r,o}(0) = 0$). Hence, it is enough to prove the inequality at these termination points when an operation is completed. Assume an instance of operation (r,o) that starts at time t_0, then its lateness is $\mathcal{L}_{r,o}(t_0) \geq 0$. The operation is finished at time $t_0 + T_{r,o}$ where $D_{r,o}(t_0 + T_{ro}) = D_{r,o}(t_0) + 1$ and so

$$\mathcal{L}_{ro}(t_0 + T_{ro}) = (t_0 + T_{ro})x_r - (D_{r,o}(t_0) + 1) = \mathcal{L}_{ro}(t_0) - 1 + x_r T_{ro} \geq -1 + x_r T_{ro}.$$

The last inequality is due to the fact that under FBDR, jobs are dispatched only if $\mathcal{L}_{r,o}(t_0) \geq 0$. Also note that by constraint (4.1) it is always true that $x_r T_{ro} - 1 \leq 0$. ■

In particular since $x_r > 0$ for any product type r that is produced we have that $\mathcal{L}_{ro}(t) > -1$.

Propsition 4.2.8 *Using FBDR with sufficient safety stock implies that the amount of work in process of jobs that already completed operation $(r, o-1)$ but yet not finished operation (r,o) is bounded above by*

$$B_{r,o}(t) \leq B_{r,o}(0) + 1 + \lfloor \gamma_{r,o} - x_r T_{r,o-1} \rfloor$$

where $B_{ro}(o)$ is the amount of safety stock used.

Proof. Consider the number of jobs in buffer (r,o) at time t,

$$B_{r,o}(t) \leq B_{r,o}(0) + D_{r,o-1}(t) - D_{r,o}(t).$$

(sharp inequality occurs only when operation (r,o) is being processed). Now, by the definition of the lateness $D_{ro}(t) = t \cdot x_r - \mathcal{L}_{ro}(t)$ and so,

$$B_{r,o}(t) \leq B_{r,o}(0) + [t \cdot x_r - \mathcal{L}_{r,o-1}(t)] - [t \cdot x_r - \mathcal{L}_{r,o}(t)]$$

$$B_{r,o}(t) \leq B_{r,o}(0) - \mathcal{L}_{r,o-1}(t) + \mathcal{L}_{r,o}(t).$$

By Lemma 4.2.7, $\mathcal{L}_{r,o-1}(t) \geq -1 + x_r T_{r,o-1}$ and by Proposition 4.2.3, $\mathcal{L}_{r,o}(t) \leq \gamma_{r,o}$. Hence,

$$B_{r,o}(t) \leq B_{r,o}(0) + 1 + \gamma_{r,o} - x_r T_{r,o-1}.$$

Now, since both $B_{r,o}(t)$ and $B_{r,o}(0)$ must be integers

$$B_{r,o}(t) \leq B_{r,o}(0) + 1 + \lfloor \gamma_{r,o} - x_r T_{r,o-1} \rfloor.$$

■

The following proposition shows that in long routes the total WIP level of products from each type, at any given time, can be bounded above approximately by the amount of required safety stock for that route.

Propsition 4.2.9 *The maximum total levels of WIP, along all buffers and machines of a given route r while using FBDR, is bounded above by*

$$1 + \lfloor \gamma_{r,K_r} \rfloor + \sum_{o=2}^{K_r} \mathcal{B}_{ro}(0). \tag{4.2}$$

Proof. First observe that at any given time, the number of products along each route r equals the number of products in the system at the beginning of the regular phase (safety stock) plus the number of products that started their first operation since the beginning of the phase, minus the number of products that completed their last operation. That is

$$\sum_{o=2}^{K_r} \mathcal{B}_{ro}(t) \leq \sum_{o=2}^{K_r} \mathcal{B}_{ro}(0) + D_{r,1}(t) + 1 - D_{r,K_r}(t).$$

and so

$$\sum_{o=2}^{K_r} \mathcal{B}_{ro}(t) \leq \sum_{o=2}^{K_r} \mathcal{B}_{ro}(0) - \mathcal{L}_{r,1}(t) + 1 + \mathcal{L}_{r,K_r}(t).$$

Now, since the total WIP level of a route increases only when a job starts being processed on the first machine and at such times $\mathcal{L}_{r,1}(t) \geq 0$ and since $\mathcal{L}_{r,K_r}(t) \leq \gamma_{r,K_r}$ we have

$$\sum_{o=2}^{K_r} \mathcal{B}_{ro}(t) \leq +1 + \gamma_{r,K_r} + \sum_{o=2}^{K_r} \mathcal{B}_{ro}(0).$$

Now, (4.2) follows from the fact that the total number of products is in integer. ■

It is worth to stress here that the actual required safety stock and buffers' size for each operation when using FBDR is generally significantly lower than the bounds presented here. Thus, the best way to calculate the safety stock requirements is by numerical means described and demonstrated in Sections 4.4, 4.6 and 4.7.

4.3 Upper Bounds on The Lateness

The bounds, presented in the previous section, on the required safety stock and the maximum level of work in process are given in terms of an upper bound on the lateness denoted by $\gamma_{r,o}$. In proposition 4.2.3 we stated that γ_{ro} is finite. Here we pay the debet of proving this statement by presenting some upper bounds on this constant. In Proposition 4.3.1 we provide an upper bound on the lateness. We then present a tighter, and more complex, upper bound in Proposition 4.3.6.

We start with few definitions. Let $\mathbf{x} = (x_1, \ldots, x_R)$ be the solution of the fluid problem obtained from Linear Program 4.1.1. Consider a given operation (r, o) carried out by machine $M(r, o)$. Let us denote by $\sigma^{fast(r,o)}$ the set of all operations carried out by this machine with a production rate of at least x_r and by $\sigma^{slow(r,o)}$ the set of the rest of the operations. That is, $\sigma^{fast(r,o)} \equiv \{(l, u) \in \sigma(r, o) : x_l \geq x_r\}$ and $\sigma^{slow(r,o)} \equiv \sigma(r, o) \setminus \sigma^{fast(r,o)}$. We use the notation $\sigma^{fast(r,o)^-} \equiv \sigma^{fast(r,o)} \setminus \{(r, o)\}$. Now the main result of this section is

Propsition 4.3.1 *Assume a system scheduled according to FBDR. Then, at any given time t when $M(r, o)$ is ready, the lateness is bounded above by*

$$\mathcal{L}_{ro}(t) \leq \frac{\sum_{(l,u)\in\sigma^{fast(r,o)^-}} \left[T_{lu} - (x_l - x_r)T^2_{l,u}\right] + \sum_{(l,u)\in\sigma^{slow(r,o)}} (T_{l,u} - x_l T^2_{l,u})}{\sum_{(l,u)\in\sigma^{fast(r,o)}} T_{lu}} \equiv \gamma^*_{ro}.$$

The proof of this Proposition follows from the following two lemmas.

Lemma 4.3.2 *Consider a job shop scheduled according to FBDR. Assume a pair of operations types (l, u) and (r, o), both processed on the same machine and $x_l \geq x_r$. Then, at time t, $\mathcal{L}_{l,u}(t) \geq \mathcal{L}_{r,o}(t) - 1 + (x_l - x_r)T_{l,u}$.*

Proof. let us define the function

$$\Delta^{\mathcal{L}}(t) \equiv \mathcal{L}_{l,u}(t) - \mathcal{L}_{r,o}(t)$$

$$\Delta^{\mathcal{L}}(t) = t(x_l - x_r) - D_{l,u}(t) + D_{r,o}(t)$$

Now since $x_l - x_r \geq 0$ then the only time when $D_{l,u}(t)$ increases is when operation (l, u) is completed and thus, these are the only points of time when $\Delta^{\mathcal{L}}(t)$ decreases. Hence, local minima points can be found only at these times. Assume t_0 is such a time and so at time $t_0 - T_{l,u}$ the processing of the operation started. Since the operations are scheduled according to FBDR, $\Delta^{\mathcal{L}}(t_0 - T_{l,u}) \geq 0$. Now

$$\Delta^{\mathcal{L}}(t_0) = \Delta^{\mathcal{L}}(t_0 - T_{l,u}) - 1 + (x_l - x_r)T_{l,u} \geq -1 + (x_l - x_r)T_{l,u}$$

and we are done. ■

Lemma 4.3.3 *Consider a discrete system with sufficient safety stock. Then,at any given time t, when machine i is ready, we have $\sum_{(r,o)\in\sigma_i}\mathcal{L}_{r,o}(t)T_{ro}\leq 0$.*

Proof. We first show that the lemma holds whenever the machine is idle for some time. Assume by contradiction that at time t machine i is idle and $\sum_{(r,o)\in\sigma_i}\mathcal{L}_{r,o}(t)T_{ro} > 0$. Then, there is at least one operation $(r,o)\in\sigma_i$ such that $\mathcal{L}_{ro}(t)>0$. Now since the system operates under FBDR, the machine should have been started to perform operation (r,o). Note also that while operation (r,o) is not being processed $\mathcal{L}_{ro}(t)$ is a continuous increasing function of t. Thus, at any time when a machine switch from idleness to busyness, it begins to process an operation with $\mathcal{L}_{ro}(t)=0$.

Assume a time t_2 when the machine is ready but not idle. That is an operation just finished on the machine and the machine is about to process its next operation. Let t_1 denote the last time when the machine switched from idleness to busyness. If no such point in time exists, let t_0 be the starting time of the regular phase. Now,

$$\sum_{(r,o)\in\sigma_i} D_{r,o}(t_2)\cdot T_{r,o} - \sum_{(r,o)\in\sigma_i} D_{r,o}(t_1)\cdot T_{r,o} = t_2 - t_1. \tag{4.3}$$

since the machine was busy throughout the time interval $[t_1,t_2]$. On the other hand by constraint (4.1) we have

$$t_2\cdot\sum_{(r,o)\in\sigma_i} x_r\cdot T_{r,o} - t_1\cdot\sum_{(r,o)\in\sigma_i} x_r\cdot T_{r,o} \leq t_2 - t_1. \tag{4.4}$$

Subtracting (4.3) from (4.4) we obtain

$$\sum_{(r,o)\in\sigma_i}\mathcal{L}_{ro}(t_2)T_{ro} - \sum_{(r,o)\in\sigma_i}\mathcal{L}_{ro}(t_1)T_{ro}\leq 0$$

Now, by the first part of this proof $\sum_{(r,o)\in\sigma_i}\mathcal{L}_{ro}(t_1)=0$ and so

$$\sum_{(r,o)\in\sigma_i}\mathcal{L}_{ro}(t_2)T_{ro}\leq 0.$$

■

Proof of Proposition 4.3.1: by Lemma 4.3.3 for any given time t,

$$\sum_{(r,o)\in\sigma_i}\mathcal{L}_{r,o}(t)T_{ro}\leq 0.$$

Thus, for operation (r, o),

$$\sum_{(l,u)\in\sigma(r,o)^-} \mathcal{L}_{lu}(t)T_{lu} \leq -\mathcal{L}_{r,o}(t)T_{ro}. \tag{4.5}$$

Now (4.5) can be written as

$$\sum_{(l,u)\in\sigma^{fast(r,o)^-}} \mathcal{L}_{lu}(t)T_{lu} + \sum_{(l,u)\in\sigma^{slow(r,o)}} \mathcal{L}_{lu}(t)T_{lu} \leq -\mathcal{L}_{r,o}(t)T_{ro}.$$

Now, by Lemma 4.2.7, $\mathcal{L}_{lu}(t) \geq x_l T_{lu} - 1$ and by Lemma 4.3.2 all the operations $(l, u) \in \sigma^{fast(r,o)}$ admit $\mathcal{L}_{l,u}(t) \geq \mathcal{L}_{r,o}(t) - 1 + (x_l - x_r)T_{l,u}$ Hence,

$$\sum_{(l,u)\in\sigma^{fast(r,o)^-}} [\mathcal{L}_{r,o}(t) - 1 + (x_l - x_r)T_{l,u}]T_{lu} + \sum_{(l,u)\in\sigma^{slow(r,o)}} [x_l T_{lu} - 1]T_{lu} \leq -\mathcal{L}_{r,o}(t)T_{ro}$$

$$\sum_{(l,u)\in\sigma^{fast(r,o)}} \mathcal{L}_{r,o}(t)\cdot T_{lu} \leq \sum_{(l,u)\in\sigma^{fast(r,o)^-}} \left[T_{lu} - (x_l - x_r)T_{l,u}^2\right] + \sum_{(l,u)\in\sigma^{slow(r,o)}} (T_{l,u} - x_l T_{l,u}^2).$$

Thus,

$$\mathcal{L}_{ro}(t) \leq \frac{\sum_{(lu)\in\sigma^{fast(r,o)^-}} [T_{lu} - (x_l - x_r)T_{lu}^2] + \sum_{(lu)\in\sigma^{slow(r,o)}} (T_{lu} - x_l T_{lu}^2)}{\sum_{(lu)\in\sigma^{fast(r,o)}} T_{lu}} \equiv \gamma_{ro}^*. \tag{4.6}$$

■

Remark 4.3.4 Note that γ_{ro}^* does not depend on t. Moreover, a simpler but less tight version of the bound given in (4.6) is

$$\mathcal{L}_{r,o}(t) \leq \frac{\sum_{(lu)\in\sigma(r,o)^-} T_{lu}}{\sum_{(lu)\in\sigma^{fast(r,o)}} T_{lu}} \leq \frac{\sum_{(lu)\in\sigma(r,o)^-} T_{lu}}{T_{ro}}. \tag{4.7}$$

That is, the lateness can be bounded above by a constant which is independent of the optimal solution of Linear Program 4.1.1.

In the sequel we further study the function $\mathcal{L}_{ro}(t)$ to obtain a tighter upper bound on it.

Lemma 4.3.5 *Consider a subset $S \subseteq \sigma_i$ of operation types carried out by machine i, $\{O_1, O_2, ..., O_{|S|}\}$. Let $r(O_j)$ denote the route to which O_j belongs. Assume, without loss of generality, that the operations are ordered such that $x_{r(O_1)} \geq x_{r(O_2)} \geq \cdots \geq x_{r(O_{|S|})}$. Then, at any given time t_0 when machine i is ready, the following inequality holds*

$$\sum_{j=1}^{|S|} \mathcal{L}_{O_j}(t_0) \cdot T_{O_j} \geq \sum_{j=1}^{|S|} T_{O_j} \left(-1 + x_{r(O_j)} \sum_{l=1}^{j} T_{O_l} \right).$$

Proof. Consider a time t_0 when the machine is ready. Assume first that by time t_0 at least one instance of each operation in S was already processed. Let us denote by Q_1 the last operation type from S processed before time t_0, its distinct predecessor from S by Q_2 and so on up to $Q_{|S|}$. Note that between the end time of operation Q_{j+1} and the start time of operation Q_j we allow the processing of operations not in S, idle times or processing of operations Q_l with $l < j$. Recall that since all the jobs are dispatched by FBDR, then $\mathcal{L}_{Q_j}(t) \geq 0$ at time t when operation Q_j starts. Thus, at time t_0 we have,

$$\begin{aligned}
\mathcal{L}_{Q_1}(t_0) &\geq -1 + x_{r(Q_1)} T_{Q_1} \\
\mathcal{L}_{Q_2}(t_0) &\geq -1 + x_{r(Q_2)} (T_{Q_1} + T_{Q_2}) \\
&\vdots \\
\mathcal{L}_{Q_{|S|}}(t_0) &\geq -1 + x_{r(Q_{|S|})} (T_{Q_1} + T_{Q_2} + \cdots + T_{Q_{|S|}})
\end{aligned}$$

This is due to the fact that since the last time operation Q_i started, its lateness was decreased by one (upon its completion) and increased in a rate of $x_{r(Q_i)}$ for at least $T_{Q_1} + \cdots + T_{Q_i}$ units of time. Now by multiplying each inequality i by T_{Q_i} we obtain,

$$\begin{aligned}
\mathcal{L}_{Q_1}(t_0) \cdot T_{Q_1} &\geq -T_{Q_1} + x_{r(Q_1)} T_{Q_1} T_{Q_1} \\
\mathcal{L}_{Q_2}(t_0) \cdot T_{Q_2} &\geq -T_{Q_2} + x_{r(Q_2)} T_{Q_2} (T_{Q_1} + T_{Q_2}) \\
&\vdots \\
\mathcal{L}_{Q_{|S|}}(t_0) \cdot T_{Q_{|S|}} &\geq -T_{Q_{|S|}} + x_{r(Q_{|S|})} T_{Q_{|S|}} (T_{Q_1} + T_{Q_2} + \cdots + T_{Q_{|S|}}).
\end{aligned}$$

To complete the proof we show that the sum of the expressions on the right hand side

$$\sum_{j=1}^{|S|} -T_{Q_j} + x_{r(Q_j)} T_{Q_j} (T_{Q_1} + T_{Q_2} + \cdots + T_{Q_j}) \tag{4.8}$$

is minimized if $x_{r(Q_1)} \geq x_{r(Q_2)} \geq \cdots \geq x_{r(Q_{|S|})}$. Assume by contradiction that this is not the case. That is, there is at least one pair Q_j and Q_{j+1} such that $x_{r(Q_j)} < x_{r(Q_{j+1})}$ and let us see how (4.8) is affected by swapping the order of these two operations. The difference caused by the replacement is

$$-x_{r(Q_j)} T_{Q_j} (T_{Q_1} + T_{Q_2} + \cdots + T_{Q_j}) + x_{r(Q_{j+1})} T_{Q_{j+1}} (T_{Q_1} + T_{Q_2} + \cdots + T_{Q_{j-1}} + T_{Q_{j+1}})$$
$$-x_{r(Q_{j+1})} T_{Q_{j+1}} (T_{Q_1} + T_{Q_2} + \cdots + T_{Q_{j+1}}) + x_{r(Q_j)} T_{Q_j} (T_{Q_1} + T_{Q_2} + \cdots + T_{Q_{j+1}})$$
$$= T_{Q_j} T_{Q_{j+1}} (x_{r(Q_j)} - x_{r(Q_{j+1})}).$$

Now, by our assumption $x_{r(Q_j)} < x_{r(Q_{j+1})}$ and hence swapping O_j and O_{j+1} decreases the sum and the presumed minimality of the order is contradicted. Also note that if $x_{r(Q_j)} = x_{r(Q_{j+1})}$, replacing them does not affect the sum.

Finally, we note that this lower bound holds also for the case when the first instance of some operation types was not yet processed. In this case their lateness is greater than zero. This is not less than their contribution to the lower bound if the operations are ordered in decreasing order of their rates. ■

Using Lemma 4.3.5 we can provide a tighter upper bound than the one presented in Proposition 4.3.1.

Propsition 4.3.6 *Consider an operation (r, o) and let $O_1, ..., O_{|\sigma^{slow(r,o)}|}$ be a list of operations performed by $M(r, o)$ with $x_{r(O_j)} < x_r$. Assume the list $O_1, ..., O_{|\sigma^{slow(r,o)}|}$ is ordered by the non-increasing values of $x_{r(O)}$s. Then,*

$$\mathcal{L}_{ro}(t) \leq \frac{\sum_{(lu)\in\sigma^{fast(r,o)-}} [T_{lu} - (x_l - x_r)T_{lu}^2] + \sum_{j=1}^{|\sigma^{slow(r,o)}|} T_{O_j} \left(1 - x_{r(O_j)} \sum_{l=1}^{j} T_{O_l}\right)}{\sum_{(lu)\in\sigma^{fast(r,o)}} T_{lu}} \equiv \gamma_{ro}^{**} \tag{4.9}$$

Proof. As in the proof of 4.3.1 we have,

$$\mathcal{L}_{r,o}(t)T_{ro} + \sum_{(l,u)\in\sigma^{fast(r,o)^-}} \mathcal{L}_{lu}(t)T_{lu} + \sum_{(l,u)\in\sigma^{slow(r,o)}} \mathcal{L}_{lu}(t)T_{lu} \le 0. \tag{4.10}$$

Now by Lemma 4.3.2,

$$\sum_{(l,u)\in\sigma^{fast(r,o)^-}} \mathcal{L}_{l,u}(t)T_{lu} \ge \sum_{(l,u)\in\sigma^{fast(r,o)^-}} \left[\mathcal{L}_{r,o}(t)T_{lu} - T_{lu} + (x_l - x_r)T_{l,u}^2\right]$$

and by Lemma 4.3.5

$$\sum_{(l,u)\in\sigma^{slow(r,o)}} \mathcal{L}_{lu}(t)\cdot T_{lu} \ge \sum_{j=1}^{|\sigma^{slow(r,o)}|} T_{O_j}\left(-1 + x_{r(O_j)}\sum_{l=1}^{j} T_{O_l}\right).$$

Now substituting the left hand side of the last two inequalities in (4.10) we get

$$\mathcal{L}_{r,o}(t)\sum_{(l,u)\in\sigma^{fast(r,o)}} T_{lu} + \sum_{(l,u)\in\sigma^{fast(r,o)^-}} \left[-T_{lu} + (x_l - x_r)T_{l,u}^2\right] +$$

$$\sum_{j=1}^{|\sigma^{slow(r,o)}|} T_{O_j}\left(-1 + x_{r(O_j)}\sum_{l=1}^{j} T_{O_l}\right)\mathcal{L}_{lu}(t)T_{lu} \le 0.$$

Finally, (4.9) is obtained by a simple algebraic manipulation. ■

Using the same notations as in Proposition 4.3.6, it is easy to see that,

$$\sum_{j=1}^{|\sigma^{slow(r,o)}|} T_{O_j}\left(1 - x_{r(O_j)}\sum_{l=1}^{j} T_{O_l}\right) \le \sum_{(l,u)\in\sigma^{slow(r,o)}} (T_{l,u} - x_l T_{l,u}^2)$$

Thus, γ_{ro}^{**} is a tighter upper bound then γ_{ro}^{*}.

4.4 FBDR and Cyclic Schedules

In this section we show that during the regular phase, a system ran by FBDR, repeats the sequence of its operations every finite time. Then, we use this observation to build a near optimal schedule with a simple structure. Such a schedule can be further refine to improve the secondary objectives of reducing the required safety stocks and WIP.

Definition 4.4.1 ***Cyclic Schedule:*** *A schedule is called cyclic if there are constants t_0 and d such that at any time $t \geq t_0$ the system status is the same as in $t+d$ and the number of operations finished during the time interval $(t_0, t_0+d]$ equals the number of operations finished during the time interval $(t_0+n \cdot d, t_0+(n+1) \cdot d]$ for any positive integer n. All times are specified with respect to the beginning of the regular phase. By a system status we refer to the operations being processed by the machines, the time passed since the beginning of these operations and the buffers content. The constant d is the cycle length.*

Propsition 4.4.2 *The schedule obtained by applying FBDR with sufficient safety stock is a cyclic one.*

Proof. Let t_0 be the starting time of the regular phase and let d be a common denominator of $x_1, ..., x_R$, where $\mathbf{x} = (x_1, ..., x_R)$ is a solution of Linear Program 4.1.1. We first show that at time $n \cdot d$, with any nonnegative integer n, all machines are ready to accept their next operation. That is, the machine status is the same in t_0. Note that for all r, the $(n \cdot d \cdot x_r + 1)^{th}$ operation cannot starts time $n \cdot d$. This is since if $D_{ro}(t) = (n \cdot d \cdot x_r)$ for $t < nd$ then $\mathcal{L}_{ro}(t) < 0$ and in such a case the next instance would not be dispatched by FBDR. That is, for all operations (r, o), $\mathcal{L}_{ro}(nd) \geq 0$. On the other hand, by Lemma 4.3.3 we have that $\sum_{(r,o)\in\sigma_i} \mathcal{L}_{r,o}(t)T_{ro} \leq 0$ and so it must be the case that $\mathcal{L}_{ro}(nd) = 0$ for all operations. We see that by time nd the $(ndx_r)^{th}$ operation already completed while the $(ndx_r + 1)^{th}$ not yet started. This is true for all operations and thus at this time the machine is ready.

Second, since $\mathcal{L}_{ro}(n \cdot d) = \mathcal{L}_{ro}((n+1) \cdot d) = 0$ for all integers n (established by the first part of this proof), then the number of operations of type (r, o) processed by the system during the time interval $[n \cdot d, (n+1) \cdot d]$ must be $x_r \cdot d$ and since this is true for all the operations along route r, then the number of products entered buffer (r, o) from the machine that carry out operation $(r, o-1)$ is the same as the number of products taken out of the buffer by the machine that carry out operation (r, o). Hence, $B_{ro}(n \cdot d) = B_{ro}((n+1) \cdot d)$ for all positive integers n. ■

Corollary 4.4.3 *The cycle length of a system controlled by FBDR is the minimal common denominator of the flow rates $x_1, ..., x_R$.*

Observation 4.4.4 *If the system is controlled by FBDR and d is the cycle length, then the number of operations of type* (r, o) *performed in a cycle is* $d \cdot x_r$. *We use the notation* $J_r = d \cdot x_r$ *for the number of operations per cycle.*

Once identifying the cyclic nature of the schedules created by FBDR, we can simulate our system for a single cycle in order to calculate the exact minimal amount of safety stocks and buffers sizes required. We start the simulation with zero level safety stock but allowing the buffer level to be negative throughout the cycle. The safety stock required for each buffer is the absolute value of the smallest level reached in this buffer throughout the cycle. If such a safety stock level is used then the maximum WIP level throughout the cycle can be obtained by subtracting the minimum (negative) WIP level from the maximum WIP level in the simulation. This maximum level of WIP determines the space needed to be allocated for the system buffers. Note that the term "simulation" is somewhat misleading in this context; Recall that we are simulating the operation of a deterministic system - so this is **not** a monte-carlo simulation.

Note also that our schedule remains optimal under any modification of the sequence and the timing of the operations within the cycle, as long as the number of operations of each type and the cycle length are not modified. Thus one can use the rich body of literature on cyclic scheduling, see for example [22], [24], [7] and [20] in order to "improve" the schedule in terms of required safety stocks and average amount of WIP.

However, a barrier to apply such an approach in practice is that a cycle may consist of arbitrarily large number of operations. Thus, it is generally too costly to conduct a simulation of a cycle. Also, Optimization of the cycle sequence is prohibitively hard for such long cycles. In the sequel we present a method to build near optimal cycles with few operations.

Consider an optimal solution x_r of Linear Program 4.1.1. A feasible cycle of length at most $\mathcal{T}$ can be constructed according to such a solution by modifying J_r to be

$$J'_r = \lfloor \mathcal{T} \cdot x_r \rfloor. \tag{4.11}$$

The newly constructed cycle is of length

$$d' = \max_i \sum_{(r,o)\in\sigma_i} T_{ro} J'_r \leq \mathcal{T}$$

and the flow rate in the corresponding fluid system

$$x'_r = \frac{J'_r}{d'}. \tag{4.12}$$

Observe that x'_r is a feasible solution to Linear Program 4.1.1 and hence we can use it to control the discrete one using FBDR.

Propsition 4.4.5 *Let $\delta(\mathcal{T})$ denote the ratio between the solution obtained by 4.12 and the optimal fluid solution, then*

$$\delta(\mathcal{T}) \geq 1 - \frac{\sum_{r=1}^{R} P_r \cdot \mathbb{I}_{x_r>0}}{\mathcal{T} \cdot \sum_{r=1}^{R} P_r x_r}.$$

Proof. The average gross profit for a time unit obtained by applying the modified solution obtained by (4.12) is

$$\sum_{r=1}^{R} \frac{P_r \cdot \lfloor \mathcal{T} \cdot x_r \rfloor}{d'} \geq \sum_{r=1}^{R} \frac{P_r \cdot \lfloor \mathcal{T} \cdot x_r \rfloor}{\mathcal{T}}$$

and the average gross profit per time unit obtained from the original optimal fluid solution is

$$\sum_{r=1}^{R} P_r \cdot x_r.$$

Hence,

$$\delta(\mathcal{T}) \geq \frac{\sum_{r=1}^{R} P_r \cdot \lfloor \mathcal{T} \cdot x_r \rfloor}{\mathcal{T} \cdot \sum_{r=1}^{R} P_r \cdot x_r} \geq \frac{\sum_{r=1}^{R} P_r \cdot (\mathcal{T} \cdot x_r - \mathbb{I}_{\{x_r>0\}})}{\mathcal{T} \cdot \sum_{r=1}^{R} P_r \cdot x_r} = 1 - \frac{\sum_{r=1}^{R} P_r \cdot \mathbb{I}_{\{x_r>0\}}}{\mathcal{T} \cdot \sum_{r=1}^{R} P_r x_r} \tag{4.13}$$

■

Corollary 4.4.6 *An approximation factor of δ is obtained by constructing a cycle of length*

$$\mathcal{T} = \frac{\sum_{r=1}^{R} P_r \cdot \mathbb{I}_{\{x_r>0\}}}{(1-\delta) \sum_{r=1}^{R} P_r x_r}.$$

We described above a rounding procedure of the optimal solution of the fluid system. Using this procedure one can build shorter cycles with gross profit that yields a profit within any prespecified gap from optimum. One can get even shorter cycles for the same compromise on optimality of the profit using the following Mixed Integer Program.

Mixed Integer Program 4.4.1

$$\min \sum_{r\in R} K_r j_r$$

$$\sum_{(r,o)\in\sigma_i} T_{ro} j_r \le d \qquad \forall i \in M$$

$$\sum_{r\in R} P_r j_r \ge \left(\sum_{r\in R} P_r x_r\right) \delta \cdot d$$

$$d \ge \epsilon$$

$$\mathbf{j} \in \mathbb{Z}_+^R, d \in \mathbb{R}$$

Here we use x_r, the optimal solution of Linear Program 4.1.1, as data. The first constraint assures that the length of the cycle chosen is enough to perform all the operations on each of the machines. The second constraint assures that the ratio between the average gross profit of the constructed cycle ($\sum_{r\in R} P_r j_r/d$) and the profit of its fluid counterpart system ($\sum_{r\in R} P_r x_r$) is at least δ. The third constraint is aim to eliminate the solution $d = 0$ and $j_r = 0$ for all r. This solution implies an undefined profit's ratio of 0/0. The constant ϵ can be any small enough positive number. To improve numerical stability it is better to use as large as possible ϵ and so we set $\epsilon = \min_r \max_i \sum_{(r,o)\in\sigma_i} T_{ro}$. Clearly, the length of any nonempty cycle is larger than the time it takes to process the maximum length operation in at least one of the routes and in particular the route that minimize this value. Note that if $x_r = 0$ for all r then it is unworthy to produce anything. In this case the right hand side of the second constraint of the mixed integer program vanished and so the solution is $J_r = 0$ for all r and $d = \epsilon$.

Alternatively the objective function could have been to minimize the cycle time d. However, since we want to obtain cycles that are as easy as possible to describe and analyze numerically, it seems more beneficial to have the "shortest" cycle in terms of operations and not in terms of units of time.

Using the solution of Mixed Integer Program 4.4.1 a new $\delta-$approximate solution of the fluid problem with $x'_r = j_r/d$ is constructed to be used as basis for FBDR. Note that 4.4.1 is a very lean mixed integer program with only $|R|$ integer decision variables and $|M|+1$ constraints. Our numerical experiments show that this program can be solved for problems with few tens of routes using commercial solver in very short time. Clearly the cycle created by Corollary 4.4.6 is a feasible solution for Mixed Integer Program 4.4.1 (with $d = \mathcal{T}$ and $j_r = J_r$) and hence generally it contains more operations.

Finally we note that Mixed Integer Program 4.4.1 admits a feasible solution for any value of $\delta \leq 1$ and in particular letting d be the common denominator of $x_1, ..., x_R$ and $j_r = x_r \cdot d$ as in the proof of Proposition 4.4.2 is always a feasible solution. Nevertheless, using values of δ very closed to unity, say $1 - 10^{-5}$, may result in computational instability caused by the actual floating point representation used by the solver.

4.5 Transient Phases

In real life systems the optimal product mix is changed from time to time due to changes in the products prices, raw material and other production costs, technological changes and other factors. Adjusting the system to new product mix and in particular performing initialization phase is discussed in this section. Clearly, it is desirable to perform the transition phase in the shortest possible time and then return to regular phase in which the system attains its upper bound performances as its fluid counterpart. A special case of the transition phase is the initialization phase discussed above.

The transient phase problem is equivalent to the classical job shop problem and thus intractable. For small enough systems with small safety stocks it might be possible to use exact optimization or approximation methods described in the literature of the classical job-shop problem. For the sake of completeness of this paper, we present a fluid heuristic toward this problem.

We use $\mathcal{S}_{ro}$ to denote the required safety stock for the new optimal or approximate product mix as obtained by the methods described in the previous section. Let $B'_{r,o}$ denote the number of jobs pending in buffer (r, o) or on the machine that performs operation $(r, o-1)$ at the time when the transient phase begins. The number of operations of type (r, o) needed to be preformed in the transient phase is

$$M_{ro} = \max\left\{ \max_{k=2,\dots,o} \left(\sum_{l=k}^{o} (B'_{r,l} - \mathcal{S}_{r,l}) \right), \max_{k=o+1,\dots,K_r} \left(\sum_{l=o+1}^{k} (\mathcal{S}_{r,l} - B'_{r,l}) \right), 0 \right\}.$$

Where $D_{r,o}(t)$ denote the number of operations (r, o) started since the beginning of the **phase**.

Consider now the following greedy heuristic for the transient phase problem: at any time t when a machine becomes ready and a job of type (r, o) with $\frac{D_{ro}(t)}{M_{ro}} < 1$ is available in one or more of its buffers, process the one with the smallest value of $\frac{D_{ro}(t)}{M_{ro}}$. If no such a job is available, the machine stays idle until the situation is changed. Ties are broken in favor of jobs with smaller o and smaller r (in this order). The phase is ended when $\frac{D_{ro}(t)}{M_{ro}} = 1$ for all the operations on all the machines.

4.6 Numerical Example

In this section we demonstrate some of the concepts presented in this paper on a small 4-4-4 job shop problem with the parameters listed in Table 4.1.

product number	product path	processing times				product price
		M_1	M_2	M_3	M_4	
1	$2 \to 3 \to 1 \to 4$	4	7	12	15	27
2	$1 \to 4 \to 3 \to 2$	11	14	15	10	70
3	$4 \to 3 \to 2 \to 1$	7	15	12	8	44
4	$3 \to 2 \to 4 \to 1$	13	10	12	15	51

Table 4.1: Sample data of small maximum profit job shop problem with four product types each consists of 4 operations on 4 different machines.

Solving Linear Program 4.1.1 for this problem returns

$$x = (0.00000000001890, 0.00671140988385, 0.03355704684015, 0.04026845574536)$$

with the objective function value of about 4.02685. This result can be taken as is in order to operate the system under FBDR. Observe that very small number of jobs of types 1 are going to be produced while about 8% of the produced products are going to be of type 2, 42% of type 3 and 50% are of type 4. In order to operate the system with this solution one can obtain the required safety stocks by employing the results of Proposition 4.3.1 and Proposition 4.3.6. Consider for example the maximum lateness of the third operation of product type 4 carried out by machine 4. This value determines the required safety stocks for the next operation in the route, carried out by machine 1. We first demonstrate the calculation of the bound obtained by Proposition 4.3.6 employing the formula

$$\gamma^{**}_{ro} = \frac{\sum_{(lu)\in\sigma^{fast(r,o)-}} [T_{lu} - (x_l - x_r)T^2_{lu}] + \sum_{j=1}^{|\sigma^{slow(r,o)}|} T_{O_j}\left(1 - x_{r(O_j)}\sum_{l=1}^{j} T_{O_l}\right)}{\sum_{(lu)\in\sigma^{fast(r,o)}} T_{lu}}$$

Here $O_1 = (1,4)$, $O_2 = (2,2)$ and $O_3 = (3,1)$ we get

$$\gamma^{**}_{4,3} = \frac{T_{1,4} - x_1T^2_{1,4} + T_{2,2} - x_2T_{1,4}(T_{1,4} + T_{2,2}) + T_{3,1} - x_3T_{3,1}(T_{1,4} + T_{2,2} + T_{3,1})}{T_{4,3}} \approx 1.46.$$

We conclude that the safety stock needed for operation $(4,4)$ is at most 2. In Table 4.2 we present the upper bound γ^{**}_{ro} for all operations,

product	Operation Number			
number	o=1	o=2	o=3	o=4
r=1	0.57	0.54	0.62	0.50
r=2	0.55	0.68	0.67	0.58
r=3	1.96	1.59	1.16	1.30
r=4	2.74	2.57	2.43	1.46

Table 4.2: Upper bounds on the maximum lateness for all the operations

The Safety stock needed for each buffer (r,o) in order to operate the system smoothly is $\lfloor \gamma_{r,o-1} + 1 \rfloor$.

The common denominator of the values x_r presented above is clearly huge. As discussed in Section 4.4 it might be better to use a slightly different solution of Linear Program 4.1.1 with x_r's that admit much smaller common denominator and yield schedules with shorter cycle times. One approach is to use the result of Corollary 4.4.6. Assume we agree to obtain a schedule that guarantee an average gross profit per time unit which is at least $\delta = 0.99$ of the optimal one. We first calculate an upper bound on the cycle length, in time units, $\mathcal{T}$.

$$\mathcal{T} = \frac{\sum_{r=1}^{R} P_r \cdot \mathbb{I}_{\{x_r>0\}}}{(1-\delta)\sum_{r=1}^{R} P_r x_r} = \frac{27 + 69 + 44 + 51}{0.01 \cdot (27 \cdot x_1 + 69 \cdot x_2 + 44 \cdot x_3 + 51 \cdot x_3)} \approx 4768.$$

Then we create the cycle structure:

$$J_1 = \lfloor x_1 \cdot \mathcal{T} \rfloor = 0, \quad J_2 = \lfloor x_2 \cdot \mathcal{T} \rfloor = 32, \quad J_3 = \lfloor x_3 \cdot \mathcal{T} \rfloor = 159, \quad J_4 = \lfloor x_4 \cdot \mathcal{T} \rfloor = 191.$$

Now the actual cycle length is $d = max_i \sum_{(lu)\in\sigma_i} T_{lu}J_r = 4745$ with machine 4 as the bottleneck machine (that is $maxarg_i \sum_{(lu)\in\sigma_i} T_{lu}J_r = 4$). Finally in order to apply FBDR we calculate the flow rate of each machine $x_1 = 0, x_2 = 32/4745, x_3 = 159/4745, x_4 = 191/4745$. The expected net profit per time unit from this solution is $\sum_{r=1}^{4} J_r \cdot P_r \approx 4.0261$ which is very close to the expected gross profit from the optimal production mix of about 4.0268.

Alternative approach is to run Mixed Integer Program 4.4.1 to find the shortest possible cycle that guarantee approximation ratio of say $\delta = 0.99$. For our demonstration problem the solution for this program is $J_1 = 0, J_2 = 1, J_3 = 2, J_4 = 2$ the cycle time for this case is $d = 65$ and the expected gross profit per time unit is about 4.0154 which is approximately 0.997 of the optimal solution. The advantage of the schedule produced by this method is that it consists of very short cycles. The safety stocks required for this schedule are easily calculated by simulating the system with empty initial buffer and allowing the stock level to be negative. Table 4.3 presents the actual required safety stocks for the problem presented in Table 4.1.

product	Operation number			Products
number	o=2	o=3	o=4	per Cycle
r=1	-	-	-	0
r=2	0	1	1	1
r=3	0	1	1	2
r=4	1	1	1	2

Table 4.3: Actual required safety stocks for the problem described in Table 4.1 with a cycle obtained by MIP 4.4.1 with $\delta = 0.99$.

.

Note that the average required safety stocks here are about 0.77 as compare to an average of 1.75 obtained by the upper bound on γ presented in Table 4.2. In larger examples this difference tends to be even larger as shown in Table 4.5 at the next section.

4.7 Numerical Study

In this section we demonstrate the practicality of the methods described in the paper using some famous benchmark job-shop problems extended with a list of prices of the products. We constructed two 10x10 and four 20x20 test problems based on [1] (denoted by 'abz5' and 'abz6') and [41] (denoted by 'yn1',...,'yn4'). The test data sets were downloaded from OR-Lib at "http://mscmga.ms.ic.ac.uk/jeb/orlib/-jobshopinfo.html". In addition, since all OR-Lib problems are of non re-entrant lines while our approach treats re-entrant lines as well, we generated new three problem instances. These problem instances are denoted by 'pr1','pr2' and 'pr3'. They are all re-entrant and consist of 10 machines, 20 product types and each product route consists of 40 operations each. The processing times of all the operations of these three instances were sampled uniformly from the sets $\{10, \ldots, 49\}$, $\{1, \ldots, 99\}$ and $\{1, \ldots, 999\}$ respectively.

Finally we "cooked" an extreme example, denoted by 'ext' in order to reveal one of the weaknesses of our method. Half of the jobs in 'ext' are "light" ones and half are "heavy". The processing times of all the operations of the light jobs are uniformly sampled from the set $\{1, \ldots, 10\}$ while the processing times of all operations of the heavy jobs are uniformly sampled from the set $\{990, \ldots, 999\}$. As we expected this kind of problems does not fit the fluid approach very well. This is since during the processing of a heavy job many light jobs are accumulated in the buffer in front of the machine processing this job. Our numerical tests demonstrate that.

For each of these ten job shop problems we randomly generated 20 profit vectors. A profit vector assigns a gross profit per product for each product type. The profits were taken from Normal distributed with mean $\mu_r = \sum_{o=1}^{K_r} T_{ro}$ and σ^2 equals the variance of the processing times over all the operations in the route, rounded to the nearest integer. By using prices proportional to the total processing times of the jobs we created instances with many product types involved in the optimal product mix. We did it in order to avoid trivial cases where most of the products are not produced.

For each of the 200 test problems we constructed four approximate fluid based solutions. Two solutions are based on Corollary 4.4.6 with an approximation ratio of $\delta = 0.99$ and $\delta = 0.999$ and two were obtained by a solution of MIP 4.4.1 with the same approximation ratio. The MIPs were solved using CPLEX 8.0 on a mobile Pentium III, 1GHz with 384Mb RAM. The solution times of the programs solved ranged from fraction of second to about 200 seconds on the worst case out of the 400 problems that were solved. The solution times of the programs with $\delta = 0.999$ were significantly longer than these of the programs with $\delta = 0.99$ although the number of variables and constraints is the same. Our test data is available from our site at "http://iew3.technion.ac.il/∼talraviv/Publications".

Table 4.4 presents the average number of products per cycle for each of the four types of the approximate solution. Column 2 indicates the dimensions of the problem in triplet *(#machines - #products types - #operations in each route)*. In column 3 the range of the processing times for all operation is presented. Columns 4-5 present the average number of products in a cycle constructed according to Corollary 4.4.6. The total number of products is then obtained by the summation of J_r. In columns 6-7 we present the number of products per cycle obtained using MIP 4.4.1. Note that we present the cycle length in terms of the number of products, and not the number of operations, in order to make a fair compression between problems with short and long routes.

The cycles obtained by MIP 4.4.1 are about 100 times shorter than those created according to Corollary 4.4.6. Also, note that in both solutions approach some of the product types was not produced in the optimal solution (that is, $J_r > 0$).

We simulated the cyclic schedules obtained by Corollary 4.4.6 and by Mixed Integer Program 4.4.1 with $\delta = 0.99$. The safety stocks, buffer sizes and the average levels of WIP for each operation required by both methods are compared.

Each line in Table 4.5 summarizes the 20 experiments we conducted for each job shop problem (with 20 different profit vectors). In column 2 the average upper bound on the required safety stocks is presented. In columns 3-5 we present the average required safety stocks per operation, the average buffer size required and the average WIP per operation for the cyclic schedules constructed using Corollary 4.4.6. In columns 6-8 the same is presented for schedule constructed using MIP 4.4.1. All values are averages over all operations in the 20 instances.

It is clear from Table 4.5 that in most cases the approximated fluid schedules constructed by both methods require small safety stock and work in process. This is true for all of our examples expect for the 'ext'. Indeed, we expect the fluid

Problem name	Problem size	T_{ro} Range	Corollary 4.4.6		MIP 4.4.1	
			$\delta = 0.99$	$\delta = 0.999$	$\delta = 0.99$	$\delta = 0.999$
1	2	3	4	5	6	7
abz5	10-10-10	50-99	918	9206	9	40
abz6	10-10-10	20-98	926	9290	20	135
yn1	20-20-20	10-49	1980	19872	32	230
yn2	20-20-20	10-49	1881	18864	24	163
yn3	20-20-20	10-49	2002	20081	27	198
yn4	20-20-20	10-49	1927	19329	27	188
pr1	10-20-40	11-49	1953	19572	25	172
pr2	10-20-40	2-99	1987	19908	24	181
pr3	10-20-40	2-997	1926	19298	26	169
ext	10-10-10	1-999	50492	504947	979	2320

Table 4.4: Average number of products per cycle using the cycles constructed by Corollary 4.4.6 and by Mixed Integer Program 4.4.1 with various approximation ratios.

approach to be useful whenever a large number of **similar** items are to be treated. In 'ext' problem the light jobs are significantly different from the heavy ones and thus many light jobs are queued in the buffers whenever a heavy job is being processed. The theoretical bounds on the maximum lateness and, as result, on the minimum required safety stocks, buffer sizes and average level of WIP are significantly larger than their actual values when using the approximate solution obtained by both methods.

The schedule constructed using MIP 4.4.1 dominates those created based on Corollary 4.4.6 by all three criteria. We assume that this is due to the fact that significantly shorter cycles are constructed by this method and thus there is a smaller chance for synchronization problem to happen (e.g very large value of $\mathcal{L}_{ro}$ at some times throughout the cycle).

4.8 Discussion

In this paper we show how to construct a dispatching rule that asymptotically maximizes the average gross profit per time unit. The rule is based on the optimal

Problem name	Average $\lceil \gamma_{ro} \rceil$	Corollary 4.4.6			MIP 4.4.1		
		SST	Buffers	WIP	SST	Buffers	WIP
1	2	3	4	5	6	7	8
abz5	2.49	1.18	1.87	0.98	0.85	1.13	0.68
abz6	2.72	1.01	1.81	0.89	0.83	1.23	0.72
yn1	3.57	0.93	1.75	0.87	0.77	1.20	0.72
yn2	3.34	0.95	1.72	0.86	0.80	1.16	0.72
yn3	3.68	0.93	1.73	0.86	0.81	1.21	0.74
yn4	3.39	0.96	1.73	0.88	0.74	1.14	0.69
pr1	3.48	0.99	1.95	0.97	0.75	1.30	0.73
pr2	3.69	0.99	1.97	0.97	0.71	1.29	0.70
pr3	3.75	0.97	1.88	0.94	0.80	1.44	0.78
ext	121.76	16.81	34.14	16.59	10.72	21.16	10.52

Table 4.5: Upper bound for Safety Stock vs. actual requirements of safety stocks, buffers size and average WIP for schedules constructed by Corollary 4.4.6 and MIP 4.4.1 with gross profit approximation guarantee of $\delta = 0.99$.

solution of fluid relaxation of the problem. The created schedule consists of a finite initialization phase, where safety stocks are built, and an infinite regular operation phase. Throughout the regular operation the system delivers an optimal average gross profit per time unit. By the term gross profit we refer to the fact that initialization costs and work in process holding costs are not charged.

The disadvantage of this asymptotically optimal solution is that it requires arbitrarily high level of safety stocks. Thus, the initialization phase may take longtime, the expected WIP and the required buffer spaces may be large. We indirectly treat these points in the second phase of our method where we build nearly optimal schedule (in terms of gross profit) which consists of short cycles. Such cycles require less safety stocks. Also, one can adjust the internal sequence of the cycles to further reduce the safety stocks.

We are aware that this three-step approach may lead to suboptimal solutions but we note that our method is a practical way to attack this intractable problem. We show the applicability of this method on 200 test problems. We used standard OR-Lib benchmark problems and composed in addition harder test problems with reentrant lines. One can use these problems instances as a test bed for further research.

We noted that in most of our test problems the cycle created by MIP 4.4.1 consists of few operations. Therefore, it is possible to improve the sequence of the cycle, to reduce safety stocks and WIP, using exact combinatorial optimization methods such as integer programming.

One possible direction for further research is to devise methods to improve the sequence of the cycles obtained by Mixed Integer Program 4.4.1. Such methods can be based on exact combinatorial optimization techniques (for example, integer programming) or on heuristic search method.

We note that our method can be easily extended to flex-shop, a generalization of job shop in which each product can be produced using some different routes. For this case we simply represent each route as a distinct product, solve Linear Program 4.1.1 and proceed as usual. It is also easy to extend our method to different production regimes such as assembly lines, multistage shops and others.

Other possible extension is to use FBDR as an online rule for stochastic systems with random processing times as well as machine failures, as proposed in [7]. For such systems the fluid counterpart system can be built based on the expected processing times and expected operational times of the machines. The basic difficulty in this case is that the schedule is not cyclic and no sufficient amount of safety stocks that guarantee a smooth operation of the system can be calculated.

Part II

The Separation Index Problem

Chapter 5

Separation Index of Sets - Bounds, Algorithms and Complexity

A fundamental problem in automated data analysis is the following: given an universe U and a pair of disjoint sets $S, T \subset U$, construct a collection of rules that for any given point $p \in U$ determine whether or not p is close to S and far from T in some sense. One can study this problem in two different levels: providing these rules for some families of sets; and designing a mechanism that constructs such rules automatically for any given set.

Application of such separation problems arise in the design of decision support systems for medical diagnostic, risk assessment of credit and others. Consider for example a data base containing the results of some set of tests conducted on 1000 patients, for systolic and diastolic blood pressure, heart rate, body temperature and existence of proteins in the urine. Clearly, the results of the tests conducted on each patient can be represented approximately by an integer vector with five elements. Assume that these patients can be divided into set S and T of those who finally developed a certain disease and those who did not, respectively. It is possible to design a diagnostic expert system that can use this data set in order to devise a set of rules that returns a positive answer when an input of a vector which is similar to the members of S in some sense and negative otherwise.

Various methodologies were used for these problems including statistic and neural networks. For instance, the so called Logical Data Analysis (LDA) methodology developed by Hammer [21] deals with logical methods for constructing boolean function $f : \{0,1\}^d \rightarrow \{0,1\}$ providing the separation. More recent work by Ekin, Hammer and Kogan [16] tests the applicability of this method for decision support systems.

In this paper we consider the use of a system of linear equalities and inequalities rather than a logical function to separate the sets. In many cases this method allows a description of complex sets using a small number of equalities and inequalities. More importantly, we believe that in many real life applications this kind of representation captures very well the essential nature of the problems since the set of "positive points" tends to be in some sense "convex".

For the medical diagnostic problem described above our method can provide a polyhedra that contains all the vectors representing the positive (ill) patient and none of those of the negative (healthy) patients.

The paper focuses on the problem of constructing a minimal linear system that performs the separation (in a sense to be rigorously defined in the next section). In particular we are interested in the special case of separating some set $S \in \{0,1\}^d$ from its complement. Clearly, if a small representation is obtained it is easy to check whether a given point belongs to the set.

An additional benefit of having a finite set described by a compact system of equalities and inequalities is that one can use it to optimize any convex function over the set. This can be carried out by various integer programming techniques such as branch & bound, cutting schemes or Gröbner bases methods [37, 4]. Lovaš and Schrijver [26] presented a cutting schemes in which a relaxation of the convex hull of S, given by a set of n inequalities, is converted into a tighter description of conv(S) with $O(n^2)$ number of inequalities. This operation if applied repeatedly returns the inequalities description of conv(S). While some separation can be always obtained merely from a membership oracle (see [31]), it is likely that an explicit system with as few inequalities as possible will provide a better starting point for such a cutting scheme.

In section 5.1 the basic concepts of *Separation* and *Separation Index* are presented and some essential notations are introduced. In section 5.2 we provide some upper bounds for the Separation index of binary sets. A lower bound on the separation index of a set based on some combinatorial properties is presented in Section 5.3. The computability of the problem is demonstrated through an Integer Programming formulation in Section 5.4. In Section 5.5 proofs for the hardness of various versions of the separation index problem are given. In section 5.6 examples of the minimal representation of some sets are given.

5.1 Basic concepts and notations

Throughout we assume that all sets are finite subsets of some universe $U \subseteq \mathbb{Z}^d$. The universe will be clear for the context and often we take $U = \{0,1\}^d$. Let $[n] \equiv \{1,\ldots,n\}$. For any pair of vectors, $\mathbf{x} > \mathbf{y}$ means that $\mathbf{x}$ is point-wise greater than $\mathbf{y}$ i.e., $\forall i : x_i > y_i$. Vectors and matrices appear in **boldface** font while scalar appears in *italic* font.

For a given hyperplane $X = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{ax} = b\}$ with $\mathbf{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}_+$ we denote the open half-space $X = \{x \in \mathbb{R}^n : \mathbf{ax} > b\}$ by $X^>$. We use similar notation for $X^<$ and for closed half-spaces $X^\leq$ and $X^\geq$.

For a function, not necessarily bijective, $f : X \to Y$, and $y \in Y$, we set $f^{-1}(y) \equiv \{x : f(x) = y\}$.

$\mathbf{e}_i$ stands for the i^{th} standard unit vector in Euclidean real space. The outer product of an ordered pair u, v of vectors is the matrix $u \otimes v$ whose $(i,j)^{th}$ entry is $u_i v_j$. The inner product of two matrices U, V of the same dimensions is $\langle U, V\rangle \equiv \sum_{i,j} U_{i,j} \cdot V_{i,j}$.

We use $conv(S)$ to denote the convex hull of S and aff(S) stand for its affine hull.

Definition 5.1.1 *A* ***Separation*** *of an ordered pair of sets* (S,T) *is a system of linear weak inequalities* $\mathbf{Ax} \leq \mathbf{b}$ *and equalities* $\mathbf{A^{eq}x} = \mathbf{b^{eq}}$ *such that* $S \subset \{\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}, \mathbf{A}^{eq}\mathbf{x} = \mathbf{b}^{eq}\}$ *and* $T \cap \{\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}, \mathbf{A}^{eq}\mathbf{x} = \mathbf{b}^{eq}\} = \emptyset$.

Propsition 5.1.2 *An* (S,T)*-separation exists if and only if* $conv(S) \cap T = \emptyset$.

Proof. Clearly if conv$(S) \cap T = \emptyset$ then hyperplane representation of $conv(S)$ is one possible separation. Conversely, recall that $conv(S)$ is the minimal convex-hull containing all points of S. Hence, the solution set of any system of equalities and inequalities valid for all points of S contains $conv(S)$ and so if a separation exists no point $\mathbf{t} \in T$ belongs to $conv(S)$. ■

Definition 5.1.3 *The* ***Separation Index*** *of an ordered pair of sets (S, T), denoted by sep(S, T), is the number of inequalities in a minimal separation. If no separation exists then by convention sep$(S, T) = \infty$. A pair of sets (S, T) is said to be separable sep$(S, T) < \infty$.*

Later, in Theorem 5.1.9, we show that any separation can be replaced by an equivalent separation with a single equality and the same set of inequalities. Therefore, the more interesting question is how many inequalities are required.

Propsition 5.1.4 *For any pair of sets (S, T) we have sep$(S, T) = 0$ if and only if*

$$\textit{aff}(S) \cap T = \emptyset.$$

Proof. The affine hull of S can be defined by a system equalities. If it is disjoint from T then no inequalities are needed to carry out the separation.■

Propsition 5.1.5 *For any pair of sets (S, T) we have sep$(S, T) = 1$ if and only if*

$$\textit{conv}(S) \cap \textit{conv}(\textit{aff}(S) \cap T) = \emptyset \qquad \textit{and} \qquad \textit{aff}(S) \cap T \neq \emptyset.$$

Proof. All point of T outside aff(S) can be eliminated by equalities so inequalities are only needed to separate S from the set aff$(S) \cap T$. The fact that a single hyperplane separates two nonempty sets if and only if their convex hulls are disjoint follows directly from the basic separation Theorem (See for example [36]). ■

Propsition 5.1.6 *For a pair of sets (S, T) we have sep$(S, T) \leq 1$ if and only if in the optimal solution of the following Linear Program $\epsilon > 0$.*

Linear Program 5.1.1

$$\max \epsilon$$

$$\begin{array}{rcll} \mathbf{as} & \leq & b & \forall s \in S \\ \mathbf{at} & \geq & b + \epsilon & \forall \mathbf{t} \in T \cap \textit{aff}(S) \\ \mathbf{a} & \in & \mathbb{R}^d & b, \epsilon \in \mathbb{R} \end{array}$$

Proof. For an optimal solution $(\mathbf{a}^*, b^*, \epsilon^*)$ if $\epsilon^* > 0$ then the separation can be carried out by $\mathbf{a}^* x \leq b^*$ and a system of equalities that defines aff(S). Conversely, if the $\mathbf{a}^*\mathbf{x} \leq b^*$ is such a separating inequality then $(\mathbf{a}^*, b^*, \min\{\mathbf{a}^*\mathbf{t} - b^* : \mathbf{t} \in T\})$ is a feasible solution for Linear Program 5.1.1 with $\epsilon > 0$. ■

Now, since checking whether all points $\mathbf{t} \in T$ belong to aff(S) can be carried out in polynomial time of $|T|$ it follows that deciding $sep(S, T) = 1$ can be done in polynomial time (of $|S| + |T|$). We shall later see that for any $k \geq 2$ deciding $sep(S, T) \leq k$ is $\mathcal{NP}$-Complete.

One immediate candidate for (S, T) separation is the hyperplanes description (H-Representation) of conv(S). For any separable sets this is indeed a valid separation. However, in general there are separations with fewer inequalities than the H-Representation. Figure 5.1 demonstrates this statement. The black points in the figure represent the points of S and the white ones represent the points of $\bar{S}$. The octagon is the convex hull of S while the rhombus represents a possible separation with half the number of inequalities.

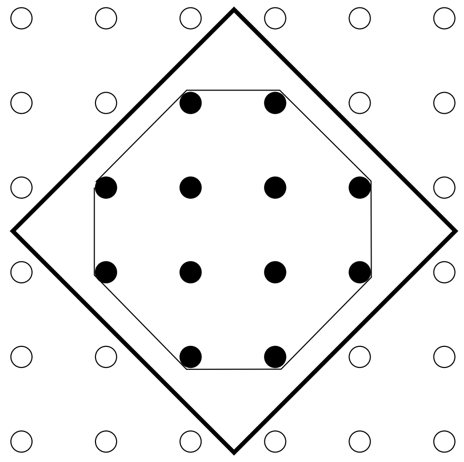

Figure 5.1: The black points represent a set $S \subset \{0, \ldots, 5\}^2$ and the white ones are T. The Octagon is the H-Representation of the set while the rhombus represents another IP-separation with only four inequalities.

Another example is provided by the set of characteristic vectors of all cuts (in edges) of the complete graph K_n. Denote this set by $\mathcal{C}_n \subset \{0,1\}^{\binom{n}{2}}$ and consider its separation from the set $\bar{\mathcal{C}_n} \equiv \{0,1\}^{\binom{n}{2}} \setminus \mathcal{C}_n$. Deza and Deza [14] showed that for

K_7 the number of facets of conv($\mathcal{C}_7$) is 116,764. In Theorem 5.2.1.b we show that the Separation index of any binary set is bounded above by $|S|$. For this particular case, the number of cuts in K_7 graph is only $2^{7-1} = 64$. Moreover, in example 5.6.8 we show that $sep(\mathcal{C}_n, \bar{\mathcal{C}}_n) \leq n(n-1)$.

For a given set S let us denote its complement in U by $\bar{S} = U \setminus S$. We now define a useful special cases of separation and separation index:

Definition 5.1.7 *An $(S, \bar{S})$-separation is called an* ***IP-separation*** *of S .*

We use the term IP-separation due the fact that such a system of equalities and inequalities can be used in an Integer Programming formulation.

Definition 5.1.8 *The separation index of S from $\bar{S}$ is called the* ***IP-Separation Index*** *of S. It is denoted by such that $ip(S) = sep(S, \bar{S})$.*

We conclude this section by giving a motivation for measuring the separation only by the number of inequalities (and not counting the equalities). It is clear that the number of equalities needed is at most d. We now show that it always can be shrunk even to one. The next theorem extends a result of Bradley [8].

Theorem 5.1.9 *Let S and T be bounded sets in $\mathbb{Z}^d$ If aff$(S) \cap T = \emptyset$ then a single equality that holds for every point in S and violated for any point in T exists.*

Proof. Clearly aff(S) can be presented by a system of linear equalities

$$\text{aff}(S) \equiv \{\mathbf{x} : \mathbf{A}\mathbf{x} = \mathbf{b}\} \tag{5.1}$$

with $\mathbf{A} \in \mathbb{Z}^{m \times d}$, $b \in \mathbb{Z}^m$ and $m \leq d$. For each row A_i let $a_i \equiv \min\{A_i x : x \in S \cup T\}$ and $q = 1 + \max\{A_i x - a_i : i = 1, ..., m, x \in S \cup T\}$. Consider the following set

$$X \equiv \{x : \sum_{i=1}^{m} (A_i x - a_i) q^{i-1} = \sum_{i=1}^{m} (b_i - a_i) q^{i-1}\}. \tag{5.2}$$

The claim that $S \subseteq X$ follows from the fact that (5.2) can be obtain by subtracting a_i and multiplying by q^{i-1} both sides of each row i in the system of (5.1) and then summing all its equations.

It is left to show that $X \cap T = \emptyset$. Consider a point $\mathbf{t} \in T$. Let us construct the vector $\mathbf{y}, \mathbf{z} \in \mathbb{Z}^m$ such that $y_i = \mathbf{A}_i \mathbf{t} - a_i$ and $z_i = b_i - a_i$ for all i. Now since $\mathbf{t}$

violates at least one inequality in (5.2) then $\mathbf{y} \neq \mathbf{z}$. Note that by the constriction all elements of $y_i, z_i \in \{0, \ldots, q-1\}$ and the two vectors can be seen as a representation of the non-negative integer numbers $\sum_{i=1}^{d} y_i q^{i-1}$ and $\sum_{i=1}^{d} z_i q^{i-1}$ in basis q. Now, since a non-negative integer admit a unique representation in any basis then the fact that $\mathbf{y} \neq \mathbf{z}$ implies that the equation of (5.1) does not holds and so $\mathbf{t} \notin X$. ■

A simple consequence of Theorem 5.1.9 is that any S, T separation, in particular one with minimum number of inequalities, can be transformed into a system with a single equality and the same inequalities. From this reason we ignore the number of equalities when measuring the "size" of a separation. The task of constructing an the affine hull is as easy as solving a system of linear equalities. Hence, in the sequel, in order to simplify notation, we assume $T \subset \text{aff}(S)$.

5.2 Upper Bounds on the Separation Index

In this section we focus on binary sets $S \subseteq \{0,1\}^d$ (so the universe is now throughout $U = \{0,1\}^d$).

Theorem 5.2.1 *For any disjoint $S, T \subset \{0,1\}^d$ the separability index is bounded above by :*

(a) sep$(S,T) \leq |T|$

(b) sep$(S,T) \leq |S|$

The proof of the above theorem is constructive, and produces a separation with up to $\min(|T|, |S|)$ inequalities.

Proof of part (a) of Theorem 5.2.1. For $\mathbf{v} \in \{0,1\}^d$, let $\text{supp}(\mathbf{v}) \equiv \{i \in [n] : v_i \neq 0\}$ be its *support*. It is not hard to verify that an (S,T) separation is provided by,

$$\left\{ \mathbf{x} \in \{0,1\}^d : \sum_{i \notin \text{supp}(\mathbf{v})} x_i + \sum_{i \in \text{supp}(\mathbf{v})} (1 - x_i) \geq 1, \quad \mathbf{v} \in T \right\} \tag{5.3}$$

■

Clearly, there is no symmetry between the two claims of Theorem 5.2.1 and in general $sep(S,T) \neq sep(T,S)$. The proof for (b) evolves from a series of lemmas and an algorithm.

Lemma 5.2.2 *For a given set $A \subseteq \{0,1\}^d$ and $k \in \{0,1\}$ let us denote*

$$A_{(k)} = \{\mathbf{x} \in \{0,1\}^{d-1} : \exists \mathbf{y} \in A : (x_i = y_i \quad \forall i = 1..d-1, y_d = k)\}$$

Then, for any non-empty sets (S,T) the inequality $sep(S,T) \leq sep(S_{(0)}, T_{(0)}) + sep(S_{(1)}, T_{(1)})$ holds.

Proof. Let us define $\tilde{A}_{(k)} = \{x \in A : x_d = k\}$ for $k \in \{0,1\}$ and use a similar notation for $\tilde{T}_k$. If the following system of inequalities

$$\begin{array}{l} a^0_{11}x_1 + \cdots + a^0_{1,d-1}x_{d-1} \geq b^0_1 \\ \vdots \\ a^0_{m1}x_1 + \cdots + a^0_{m,d-1}x_{d-1} \geq b^0_m \end{array} \tag{5.4}$$

is an $(S_{(0)}, T_{(0)})$ separation then it is also $(\tilde{S}_{(0)}, \tilde{T}_{(0)})$ separation. The following system is also an $(\tilde{S}_{(0)}, \tilde{T}_{(0)})$ separation

$$\begin{array}{l} a^0_{11}x_1 + \cdots + a^0_{1,d-1}x_{d-1} + \left(b^0_1 - \sum_{i:a^0_{1,i}<0} a^0_{1,i}\right) x_d \geq b^0_1 \\ \vdots \\ a^0_{m1}x_1 + \cdots + a^0_{m,d-1}x_{d-1} + \left(b^0_m - \sum_{i:a^0_{m,i}<0} a^0_{m,i}\right) x_d \geq b^0_m \end{array} \tag{5.5}$$

but note that all the points of $\tilde{S}_{(1)} \cup \tilde{T}_{(1)}$ are valid for this system. Note that in case $T_{(0)} = \emptyset$ then (5.4) and (5.5) are empty systems. Similarly, the separation of $S_{(1)}$ from $T_{(1)}$ can be transformed to to the following separation of $\tilde{S}_{(1)}$ from $\tilde{T}_{(1)}$.

$$\begin{array}{l} a^1_{11}x_1 + \cdots + a^1_{1,d-1}x_{d-1} + \left(b^1_1 - \sum_{i:a^1_{1,i}<0} a^1_{1,i}\right) (1 - x_d) \geq b^1_1 \\ \vdots \\ a^1_{m1}x_1 + \cdots + a^1_{m,d-1}x_{d-1} + \left(b^1_m - \sum_{i:a^1_{m,i}<0} a^1_{m,i}\right) (1 - x_d) \geq b^1_m. \end{array} \tag{5.6}$$

This system is valid for all points $\tilde{S}_{(1)}$, $\tilde{T}_{(0)}$ and $\tilde{S}_{(0)}$ but violated by all points of $\tilde{T}_{(1)}$. Now, the joint system (5.5) and (5.6) is an (S,T) separation. ■

The following lemma uses the same notations as in Lemma 5.2.2,

Lemma 5.2.3 *for any non-empty set S, with $S_{(i)} \neq \emptyset$ and $S_{(j)} = \emptyset$ the inequality sep$(S, T) \leq$ sep$(S_{(i)}, T_{(i)})$ holds for all sets T.*

Proof. Assume, w.l.g., that $i = 0$. Consider a pair of sets (S, T), assume that $S_{(1)} = \emptyset$ and that (5.4) is an $(S_{(0)}, T_{(0)})$ separation then

$$\begin{array}{l} a^1_{11}x_1 + \cdots + a^1_{1,d-1}x_{d-1} - \left(b^1_1 - \sum_{i:a^1_{1,i}>0} a^1_{1,i}\right) x_d \geq b^1_1 \\ \vdots \\ a^1_{m1}x_1 + \cdots + a^1_{m,d-1}x_{d-1} + \left(b^1_m - \sum_{i:a^1_{m,i}>0} a^1_{m,i}\right) x_d \geq b^1_m \end{array}$$

is an (S, T) separation. ■

The following algorithm produces a separation of the set S with at most $|S|$ inequalities and leads to the proof of Theorem 5.2.1.b .

Algorithm 5.2.1 ***Create a separation***

***Input**: A pair of sets $S, T \in \{0, 1\}^d$.*
***Output**: An (S, T)-separation with at most $|S|$ inequalities.*

step 1 : Check if it is possible to separate S from T by a single inequality (using Linear Program 5.1.1 for example). If the answer is "yes" return the inequality and quit. For empty T return empty system of inequalities.

step 2 : Divide S and T into $S_{(0)}, S_{(1)}, T_{(0)}, T_{(1)} \in \{0, 1\}^{d-1}$. Recursively apply the Algorithm for (S_0, T_0) and (S_1, T_1) and merge the output of both problems as described in the proof of Lemma 5.2.2 and Lemma 5.2.3.

Propsition 5.2.4 *Algorithm 5.2.1 stops with a valid (S, T)-separation.*

Proof. Observe that for any pair of disjoint sets $S, T \in \{0, 1\}^1$, *sep*$(S, P) \leq 1$. Thus, in our algorithm after at most d recursive calls the stopping condition in the first step holds. ■

Proof of Theorem 5.2.1 part (b) . Each time Algorithm 5.2.1 stops at step 1 and return an inequality a non empty subset of S is eliminated from further consideration. In step 2 no additional inequalities are produced. Thus, the total number of inequalities can not exceed $|S|$. ■

Corollary 5.2.5 *For any set $S \subseteq \{0,1\}^d$ we have the bound ip$(S) \leq 2^{d-1}$*

Proof. Immediate by $|S| + |\overline{S}| = 2^d$, ip$(S)$ $=$ $sep(S, \bar{S}) \leq |\overline{S}|$ and ip(S) $=$ $sep(S, \bar{S}) \leq |S|$ ■

We shall see in Example 5.3.9 that this is an attainable upper bound.

Should algorithm 5.2.1 be used as a heuristic method to obtaining a "small as possible" separation, some refinements may be done to improve performances. First, note that Linear Program 5.1.1 always terminates with a feasible solution. It is possible to use this solution to construct a primal feasible solutions for the Linear Problems of (S_0, T_0) and (S_1, T_1). The former is obtained simply by deleting a_d from the solution and the latter is obtained by replacing b by $b - a_d$ and deleting a_d. Hence, it is possible to save time by re-optimizing 5.1.1 at each iteration instead of solving it from scratch. Second, recall that we do not actually look for an optimal solution of 5.1.1, just one with positive ϵ. Hence, the optimization process can be stopped once such value is obtained. Finally, there are methods to find a separating inequality other than linear programming. For example the method based on threshold graphs presented in [11] produce a valid inequality with $0-1$ coefficients. Its time complexity in terms of our problem input is only $O(d^2 \cdot min(|S|, |T|))$. Note that because of the $0-1$ restriction this method can not guarantee to find a separating inequality whenever it exists. However, it will always find a separation in iterations where $|S_{(k)}| = 1$ which is enough for the termination of our algorithm with at most $|S|$ inequalities.

It is not easy to extend the constructive proof of 5.2.1.a from binary to general sets in $\mathbb{Z}^d$. However, we can do it in a non constructive way.

Propsition 5.2.6 *Consider a pair of finite and separable sets $S, T \in \mathbb{Z}^d$. Then $sep(S,T) \leq |T|$.*

Proof. Assume by contradiction that A is a minimum set of inequalities that separates S from T and $|A| > |T|$. Let us label each point $\mathbf{t} \in T$ by the lowest index of inequalities in A that it violates. So we get a function $f : T \to A$. Now since $|A| > |T|$ there must be at least one inequality $a \in A$ such that $f^{-1}(a) = \emptyset$. Clearly A remain a valid separation after dropping this inequality and hence the contradiction. ■

5.3 Lower bounds for the IP-Separation index

In this section we show how to decompose the problem of determining the Separation index of binary sets into several sub problems and a lower bound on $ip(S)$ is then derived.

We first present some definitions. Let us define the Hamming distance between the points q and s as $d_1(q, s) \equiv \sum_{i=1}^{d} |q_i - s_i|$. Note that for $q, s \in \{0, 1\}^d$ this is exactly the number of coordinates in which s and q differ. A pair of points $q, s \in \{0, 1\}^d$ said to be adjacent if $d_1(q, s) = 1$.

Definition 5.3.1 *The Hamming graph $H(S)$ of a set $S \subseteq \{0, 1\}^d$ is defined by the vertex set $V \equiv S$ and edge set $E \equiv \{\{q, s\} \subseteq S : d_1(q, s) = 1\}$.*

Definition 5.3.2 *[Total a-Order] Consider a given vector $\mathbf{a} \in \mathbb{R}^d$ and a pair of vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. We say that $\mathbf{x}$ is an $\boldsymbol{a}$-predecessor of $\mathbf{y}$ (denoted $\mathbf{x} \prec_a \mathbf{y}$) if $\mathbf{ax} < \mathbf{ay}$ or $\mathbf{ax} = \mathbf{ay}$ and $\mathbf{x}$ is a lexicographic predecessor of $\mathbf{y}$ (that is for the minimum coordinate index i in which $\mathbf{x}$ and $\mathbf{y}$ differs $x_i < y_i$). If for all $\mathbf{y} \in S \setminus \{x\}$ the vector $\mathbf{x}$ admits $\mathbf{y} \prec_a \mathbf{x}$ then $\mathbf{x}$ is the $\boldsymbol{a}$-maximum of S*

Lemma 5.3.3 *For a point $\mathbf{s} \in \{0, 1\}^d$ either $\mathbf{s}$ is an $\boldsymbol{a}$-maximum of $\{0, 1\}^d$ or a point $\mathbf{q}$ exist such that $d_1(\mathbf{s}, \mathbf{q}) = 1$ and $\mathbf{s} \prec_a \mathbf{q}$.*

Proof. Assume by contradiction that for some point $\mathbf{x} \in S$ and for any $\mathbf{s} \in \{q \in S : d_1(x, q) = 1\}$, $\mathbf{s} \prec_a \mathbf{x}$ and that $\mathbf{x}^* \neq \mathbf{x}$ is the $\mathbf{a}$-maximum point. Define the set of coordinates $K = \{i : x_i \neq x_i^*\}$. If $\exists i \in K$ such that $a_i \neq 0$ then one of the following must holds for at least one $i \in K$.

- $x_i = 0$ and $a_i > 0$
- $x_i = 1$ and $a_i < 0$

Now, it is easy to see that by toggling x_i (that is, let $x_i = 1 - x_i$) we obtain a new point y with $d_1(x, y) = 1$ and $\mathbf{x} \prec_a \mathbf{y}$; A contradiction.

If no such a coordinate exist, i.e $a_i = 0$ for all $i \in K$ then it is clear that $\mathbf{ax} = \mathbf{ax}^*$ and $\exists i \in K$ such $x_i = 0$ because $\mathbf{x}^* \succ_{lex} \mathbf{x}$, and so by toggling this coordinate we get a new point $\mathbf{y}$ such that $d_1(x, y) = 1$ and $\mathbf{x} \prec_a \mathbf{y}$. This is again a contradiction. ■

Lemma 5.3.4 *For any $\mathbf{a} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ the Hamming graph $H(Q)$ of the set $Q = \{x \in \{0,1\}^d : \mathbf{ax} > b\}$ is connected.*

Proof. We argue that for any pair of points $\mathbf{x}, \mathbf{y} \in Q$ there is an $x - y$ path in H(Q). To see this it is enough to show that for any $\mathbf{x} \in Q$ if $\mathbf{x}^*$ is an $\mathbf{a}$-maximum point then there is a $\mathbf{x} - \mathbf{x}^*$ path in $H(Q)$.

It is clear that $\mathbf{x}^* \in Q$ (providing that Q is non-empty). Now if $\mathbf{x} \neq \mathbf{x}^*$ then by Lemma 5.3.3 there must be a point $\mathbf{y}$ in its neighborhood $\{q \in \{0,1\} : d(x, q) = 1\}$ such that $\mathbf{x} \prec_a \mathbf{y}$ more over $\mathbf{y} \in Q$ since $\mathbf{ay} \geq \mathbf{ax} > b$ then $\mathbf{y} \in Q$. Now add the edge $(\mathbf{x}, \mathbf{y})$ to our $\mathbf{x} - \mathbf{x}^*$ path. Applying this procedure iteratively until we end in $\mathbf{x}^*$ (recall that Q is finite). ■

Lemma 5.3.5 *For any collection of n sets $S_i \subset \{0,1\}^d$ the following inequality holds*

$$ip\left(\bigcap_{i=1}^{n} S_i\right) \leq \sum_{i=1}^{n} ip(S_i)$$

Proof. Combining all the separation of $S_1, \ldots, S_n$ from their complements is a separation of $\bigcap_{i=1}^{n} S_i$ from its complement. ■

Now, we are ready to prove the main statement of this section that provides us with a lower bound for ip(S).

Theorem 5.3.6 *Consider a set S and a partition of $\bar{S}$ into k distinct subsets $(Q_1, \ldots, Q_k)$ such that any Q_i induces a maximal connected component of $H(\bar{S})$. Then IP-Separation index is given by $ip(S) = \sum_{i=1}^{k} ip(\bar{Q}_i)$.*

Proof. First note that $\text{ip}(S) \leq \sum_{i=1}^{k} \text{ip}(\bar{Q}_i)$. This is since $\{Q_i\}$ is a partition of $\bar{S}$ and the claim follows by Lemma 5.3.5.

Now to prove that $\text{ip}(S) \geq \sum_{i=1}^{k} \text{ip}(\bar{Q}_i)$ it suffices to prove that any inequality that is valid for for S and violated by at least one point $\mathbf{x} \in Q_i$ is valid for any point $\mathbf{y} \in Q_j$ for any $i \neq j$. Assume by contradiction that $\mathbf{a} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ exist such that $S \subseteq \{\mathbf{x} : \mathbf{ax} \leq b\}$ and $\mathbf{x}, \mathbf{y} \in \{\mathbf{x} : \mathbf{ax} \ > b\}$ such that $\mathbf{x} \in Q_i$ and $\mathbf{y} \in Q_j$. Now by lemma 5.3.4 both $\mathbf{x}$ and $\mathbf{y}$ belongs to the same connected component, and the contradiction follows. ■

For a given set S let us denote the number of the connected components of $H(\bar{S})$ by $\#C(S)$. We point out that the relevant number of connected components in the Hamming graph of $\bar{S}$ even if $\bar{S}$ contains points not in the affine hull of S.

Theorem 5.3.7 *The IP-Separation index of a given set $S \subseteq \{0,1\}^d$ is bounded below by $\#C(S)$.*

Proof. For a given IP-Separation let us divide the set of points in $\bar{S}$ into maximal connected component in $H(\bar{S})$. Now for each such a components Q, we have $ip(Q) \geq 1$ and so by Theorem 5.3.6 $ip(S) \geq \#C(S)$ ■

A special case of a connected component is an *Isolated point*, i.e, a component that contains a single point.

Corollary 5.3.8 *If every point of $\bar{S}$ is an isolated point in $H(\bar{S})$ then $ip(S) = |\bar{S} \cap aff(S)|$*

Proof. By Theorem 5.2.1.a $\text{ip}(S) \leq |\bar{S} \cap \text{aff}(S)|$ for any set and by Theorem 5.3.6 $ip(S) \geq \#C(\bar{S}) = |\bar{S} \cap \text{aff}(S)|$ ■

Example 5.3.9 *[Parity set] Consider the set of points $S \subset \{0,1\}^d$ such that for any $x \in S$, $|supp(x)|$ is an even number. Clearly each point in $\bar{S}$ is an isolated one and $aff(\bar{S}) = \{0,1\}^d$. Hence, by Corollary 5.3.8, $ip(S) = |\bar{S}| = 2^{d-1}$. This demonstrates the fact that the upper bound of Corollary 5.2.5 is attainable.*

5.4 Computability

In this short section we demonstrate the fact that the Separation Index of any pair of separable sets is computable. We do it by formulating a mixed integer program that admits a solution with positive value if and only if $sep(S,T) \leq m$. The separation index can be obtained by solving the problem for $m = 1, 2, \ldots, \min(|S|, |T|)$ and until a positive optimal solution is obtained. Recall that by Theorem 5.2.1 the separation index is bounded above by $|S|$ and $|T|$.

Mixed Integer Program 5.4.1 (Deciding $sep(S,T) \leq m$**)**

$$\varepsilon^*_m = \max \epsilon \tag{5.7}$$

$$\begin{aligned}
\mathbf{s} \cdot \mathbf{A_i} - b_i &\leq 0 \qquad \forall \mathbf{s} \in S, i \in [m] && (5.8)\\
\mathbf{t} \cdot \mathbf{A_i} - b_i &\geq \epsilon - (1 - y_{it}) \qquad \forall \mathbf{t} \in T, i \in [m] && (5.9)\\
\sum_{i=1}^{m} y_{it} &\geq 1 \qquad \forall \mathbf{t} \in T && (5.10)\\
\mathbf{y} &\in \{0,1\}^{|T| \times m}\\
\mathbf{A} &\in \mathbb{R}^{m \times d} \qquad \mathbf{b} \in \mathbb{R}^m \qquad \epsilon \in \mathbb{R}
\end{aligned}$$

Propsition 5.4.1 *With Mixed Integer Program 5.4.1 we have* $sep(S,T) \leq m$ *if and only if* $\varepsilon^*_m > 0$*. In this case a separation with* m *inequalities is given by* $\{x : \mathbf{Ax} \leq \mathbf{b}\}$*.*

Proof. Constraint (5.8) assures that the obtained system is valid for any point $s \in S$. Constraint (5.9) assures that if $\epsilon > 0$ then for any $y_{it} = 1$ the i^{th} constraint in the obtained system is violated by all $\mathbf{t} \in T$. Note that it is always possible to scale $\mathbf{Ax} \leq \mathbf{b}$ such that the gap between the left hand side and the right hand side is at most 1 for any point $\mathbf{t} \in T$. By constraint (5.10), at least one inequality in the obtained system must be violated for any point $\mathbf{t} \in T$. ■

5.5 Computational Complexity

In this section we study the computational complexity of the separation index and the ip-separation index problems and show that both problems are hard for $\mathcal{NP}$.

The complexity proof of the separation problem is carried out by a two steps reduction. We first define a variant of the separation problem, with inequalities only, and show that it can be reduced into the separation problem. Then we show that the *hyper graph colorability problem* can be reduced into this problem. We start with the following definition,

Definition 5.5.1 ***An inequalities separation*** *of a pair of sets is a separation consists of inequalities only. The* ***inequalities separation index*** *(denoted by $sep^*(S,P)$) is the minimum number of inequalities in such a separation.*

Propsition 5.5.2 *If $sep(S,T) \geq 2$ then $sep(S,T) = sep^*(S,P)$*

Proof. First, note then any inequalities separation is also a separation and thus $sep^*(S,T) \geq sep(S,T)$. Conversely, if the minimal separation admits no inequality then it is also an inequality separation so consider a minimal separation with two inequalities or more and some equalities. By Theorem 5.1.9 we know that the equalities can be replaced by a single equality. Hence, it enough to consider a separation of the following shape,

$$\begin{aligned}
&a_{11}x_1 + \cdots + a_{1,d}x_d \leq b_1 \\
&a_{21}x_1 + \cdots + a_{2,d}x_d \leq b_2 \\
&\vdots \\
&a_{m1}x_1 + \cdots + a_{m,d}x_d \leq b_m \\
&a_1^{eq}x_1 + \cdots + a_d^{eq}x_d = b^{eq}.
\end{aligned} \tag{5.11}$$

Now we see that such a separation is equivalent to following inequalities separation

$$\begin{aligned}
&a_{11}x_1 + \cdots + a_{1,d}x_d + M \cdot (a_1^{eq}x_1 + \cdots + a_d^{eq}x_d - b^{eq}) \leq b_1 \\
&a_{21}x_1 + \cdots + a_{2,d}x_d - M \cdot (a_1^{eq}x_1 + \cdots + a_d^{eq}x_d - b^{eq}) \leq b_2 \\
&\vdots \\
&a_{m1}x_1 + \cdots + a_{m,d}x_d \leq b_m,
\end{aligned} \tag{5.12}$$

where M is large constant. Clearly for any point in aff(S) we have $a_1^{eq} + \cdots + a_d^{eq}x_d - b^{eq} = 0$ and thus the separation between S and aff$(S) \cap T$ is carried out as before. For points in $T \setminus$ aff(S) either $a_1^{eq} + \cdots + a_d^{eq}x_d - b^{eq} > 0$ or $a_1^{eq} + \cdots + a_d^{eq}x_d - b^{eq} < 0$ and so for large enough constant M either the first or the second inequalities is violated. Hence, $sep^*(S,T) \leq sep(S,T)$ is also established and we are done. ■

Propsition 5.5.3 *If $sep^*(S,T) \geq 3$ then $sep(S,T) = sep^*(S,P)$*

Proof. Assume by contradiction $sep^*(S,T) \geq 3$ and $sep^*(S,T) > sep(S,T)$. In such as case, by Proposition 5.5.2 we that $sep(S,T) \leq 1$. Consider such a separation

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1,d}x_d &\leq b_1 \\ a_1^{eq}x_1 + \cdots + a_d^{eq}x_d &= b^{eq}. \end{aligned} \tag{5.13}$$

Than it is equivalent to the following inequality separation

$$\begin{aligned} &a_{11}x_1 + \cdots + a_{1,d}x_d + M \cdot (a_1^{eq}x_1 + \cdots + a_d^{eq}x_d - b^{eq}) \leq b_1 \\ &a_1^{eq}x_1 + \cdots + a_d^{eq}x_d \geq b^{eq}. \end{aligned} \tag{5.14}$$

with only two inequalities and hence the contradiction. ∎

Theorem 5.5.4 *For any fixed $k \geq 2$, deciding whether $sep(S,T) \leq k$ for a given pair of sets $S, T \subset \mathbb{Z}^d$ is $\mathcal{NP}$-Complete.*

Proof. Let $H = (V, \mathcal{F})$ be a k-uniform hyper-graph with $V = [n]$ and $\mathcal{F} \subseteq \binom{V}{k}$ (that is, each element $F \in \mathcal{F}$ is a subset $F \subseteq V$ such that $|F| = k$). Construct two sets of matrices

$$S \equiv \left\{ \mathbf{s}_{F,j} \equiv \left(\sum_{i \in F} \mathbf{e}_i \right) \otimes \mathbf{e}_j : F \in \mathcal{F}, j \in [k] \right\} \subset \{0,1\}^{n \times k}$$

$$T \equiv \left\{ \mathbf{t}_i \equiv \mathbf{e}_i \otimes \left(\sum_{j=1}^{k} \mathbf{e}_j \right) : i \in [n] \right\} \subset \{0,1\}^{n \times k}.$$

In this proof we use bold lower case letters for vectors and matrices and upper case letters for sets of matrices. Recall that the *chromatic number* of a hyper-graph H, denoted by $\chi(H)$, is the smallest number of colors needed to color its vertices such that no hyper-edge is monochromatic.

Now, our proof follows from the fact that $sep^*(S,T) = \chi(H)$. To see this, suppose X is an hyperplane in $\mathbb{R}^{n \times k}$ such that $S \subset X^{\leq}$. Consider an edge $F \in \mathcal{F}$ then

$$\frac{1}{k} \sum_{j=1}^{k} \mathbf{s}_{F,j} = \frac{1}{k} \sum_{i \in F} \sum_{j=1}^{k} \mathbf{e}_i \otimes \mathbf{e}_j = \frac{1}{k} \sum_{i \in F} \mathbf{t}_i$$

is a common point in $\mathrm{conv}(S)$ and $\mathrm{conv}(T)$. Thus, $\{\mathbf{t}_i : i \in F\} \nsubseteq X^{>}$ and the same holds for each edge $F \in \mathcal{F}$.

Now, suppose $X_1, \ldots, X_c$ are hyperplanes separating T from S with $c = sep(S,T)$. Define a map $x : [n] \longrightarrow [c]$ to be $x(i) = \min\{j \in [c] : \mathbf{t}_i \in X_j^{>}\}$. We see that x is a coloring of H. Thus $\chi(H) \leq sep^*(S,T)$.

Conversely, suppose $x : [n] \longrightarrow [c]$ is a coloring of H. For all $r \in [c]$ put

$$h_r = \left(\sum_{i \in x^{-1}(r)} \mathbf{e}_i \right) \otimes \left(\sum_{j=1}^{k} \mathbf{e}_j \right) \in \{0,1\}^{n \times k}$$

We claim that the system of inequalities in matrices $\mathbf{q} \in \mathbb{R}^{n \times k}$

$$< \mathbf{h}_r, \mathbf{q} > \leq k-1 \qquad \forall r \in [c] \tag{5.15}$$

separates T from S. Indeed for all $i \in [r]$, $F \in \mathcal{F}$ and $\mathbf{t} \in [k]$,

$$\langle \mathbf{h}_r, \mathbf{s}_{F,t} \rangle = \left\langle \sum_{i \in x^{-}1(r)} \sum_{j=1}^{k} \mathbf{e}_i \otimes \mathbf{e}_j,\ \sum_{\mathbf{s} \in F} \mathbf{e}_s \otimes \mathbf{e}_t \right\rangle =$$

$$\sum_{i \in x^{-1}(r)} \sum_{j=1}^{k} \sum_{\mathbf{s} \in F} \langle \mathbf{e}_i, \mathbf{e}_s \rangle \cdot \langle \mathbf{e}_j, \mathbf{e}_t \rangle = |x^{-1}(r) \cap F| \leq k-1.$$

The last inequality is due to the fact that x is a coloring of H. On the other hand for each $i \in [n]$ let $r \equiv x(i)$ and we get,

$$\langle h_r, \mathbf{t}_i \rangle = \left\langle \sum_{t \in x^{-1}(r)} \sum_{j=1}^{k} \mathbf{e}_t \otimes \mathbf{e}_j,\ \mathbf{e}_i \otimes \sum_{s=1}^{k} \mathbf{e}_s \right\rangle = \sum_{t \in x^{-1}(r)} \sum_{j=1}^{k} \sum_{s=1}^{k} \langle \mathbf{e}_t, \mathbf{e}_i \rangle \cdot \langle \mathbf{e}_j, \mathbf{e}_s \rangle = k,$$

So $\mathbf{t}_i$ violates (5.15). Thus $sep(S,T) \leq c = \chi(H)$. We conclude that $\chi(H) = sep^*(S,T)$.

The reduction of the decision problem $sep^*(S,T) \leq k$ into $sep(S,T) \leq k$ is done as follows: If $sep(S,T) \geq 2$ then by Proposition 5.5.2 $sep^*(S,T) = sep(S,T)$. If $sep(S,T) \leq 1$ then $sep^*(S,T)$ may be either 1 or 2. However, deciding $sep^*(S,T) = 1$ can be done via linear programming (in the same way as in Linear Program 5.1.1) and $sep^*(S,T) = 2$ is concluded otherwise.

Now, as shown by Lovász (cf. [15]), deciding $\chi(H) \leq k$ is $\mathcal{NP}$-Complete for any fixed $k \geq 2$. Therefore, we conclude that deciding $sep^*(S,T) \leq k$ is also $\mathcal{NP}$-Complete for all fixed $k \geq 2$. . $\blacksquare$

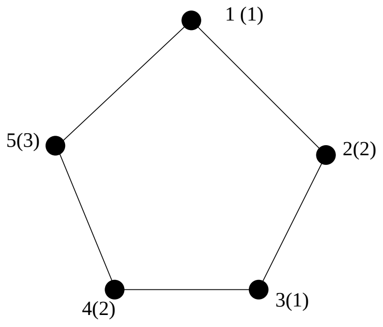

Figure 5.2: A C_5 graph with 3 coloring (colors in parentheses)

We demonstrate the construction of the proof on a 2-uniform hyper-graph (graph), $H = C_5$, the 5-gon, see Figure 5.2.

$$S = \left\{ \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}, \right.$$

$$\left. \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \right\}$$

$$T = \left\{ \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \right\}$$

Now by Theorem 5.5.4 $sep(S, T) = \chi(H) = 3$. Separation can be carried out by

$$\left\langle \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, X \right\rangle \le 1, \quad \left\langle \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}, X \right\rangle \le 1, \quad \left\langle \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}, X \right\rangle \le 1$$

Remark 5.5.5 It is clear from the proof that the separation decision problems for any fixed $k \ge 2$ remain $\mathcal{NP}$-Complete when the universe is restricted to $U = \{0, 1\}^d$.

We now consider the hardness of deciding $ip(S) \le k$. Naturally, for this problem the input assumed to be S [and not $(S, \bar{S})$]. Thus, it is not exactly a special case of the separation problem. For the hardness proof we use the concept of "Threshold Graph" introduced by Chvatál and Hammer [11].

Definition 5.5.6 *A graph $G = (V, E)$ is called* a Threshold Graph *if non-negative reals $w_v, v \in V$ and t exist such that $\sum_{v \in B} w_v \leq t$ if and only if $B \subseteq V$ is a stable set.*

An equivalent characterization of a Threshold Graph is as follow

Propsition 5.5.7 *[Mahadev and Peled 1995] A graph is a Threshold Graph if and only if there exist non-negative reals $w_v, v \in V$ and t such that for any pair of vertices u and v, $w_u + w_v > t \quad$ if and only if $uv \in E$.*

Proof. see [27] pages 10–13 ■

Definition 5.5.8 *The* Threshold Dimension, *$t(G)$, of a graph G is the minimum number k of threshold subgraphs $T_1, \ldots, T_k$ of G that cover the edge set of G, i.e., $E(T_1) \cup \cdots E(T_k) = E(G)$.*

Note that since every edge along with isolated vertices is a threshold subgraph, the threshold dimension is well-defined and bounded by the number of edges in the graph. Chvatál and Hammer [11] showed that the problem of determining the threshold dimension of a graph is NP-hard. Mahadev and Peled [27] later proved that for any fixed $k \geq 3$ deciding $t(G) \leq k$ is NP-Complete.

Remark 5.5.9 We note that the threshold dimension problem remains hard also for the case of connected graphs. Clearly, if $C_1, \ldots, C_k$ are connected component of G (in edges) and $E_1 \subseteq C_1, \ldots, E_k \subseteq C_k$ are threshold subgraphs then $\cup_{i=1}^{k} E_i$ is also a threshold subgraph of G. Hence, $t(G) = max_i t(C_i)$.

Theorem 5.5.10 *For any fixed $k \geq 3$, deciding whether $ip(S) \leq k$ for a given S is $\mathcal{NP}$-Complete.*

Proof. The proof is by reduction of the threshold dimension problem of connected graphs into the ip-separation problem. The characteristic vector of an edge (v, u) is the vector $e_u + e_v$ when e_i is the unit vector in the vertices space. Consider a graph $G = (V, E)$ given by the list of characteristic vectors of its edges. Let $S = \{\mathbf{x} \in \{0,1\}^{|V|} | supp(\mathbf{x}) \notin E, \mathbf{1^T x} = 2\}$. That is, S is the set of all characteristic vectors of all edges **not** included in G. We shall show that $ip(S) = t(G)$. Note that this construction can be carried out in polynomial time since, assuming connected graph the total number of non-edges at most $|E| + 1$ times the number of edges.

To see that $ip(S) \leq t(G)$, consider a graph $G = (V, E)$ with $t(G) = k$. Let $E_1, \dots, E_k \subseteq E$ be edge subsets such that for any $i \in [k]$ the subgraph (V, E_i) is a maximal threshold graph with threshold value b_i and vertices weights a_{ij} for all $j \in V$. Let $\mathbf{A} = \{a\}_{ij}$ and $\mathbf{b} = (b_1, \dots, b_k)$. By Proposition 5.5.7 we see that $\{\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}, \mathbf{1^T x} = 2\}$ is an IP-separation of S with k inequalities.

Conversely, we first show that $t(G) \leq sep^*(S, T)$ where T is the set of characteristic vectors of edges in the graph and $S = \{\mathbf{x} : \mathbf{1^T x} = 2\} \setminus T$ (the set of characteristic vectors of all "non-edges"). Consider an inequalities separation $\mathbf{Ax} \leq \mathbf{b}$ with k inequalities. For each $i \in [k]$ we set $c_i = \min\{\min_j a_{ij}, b_i/2\}$, $w_{ij} = a_{ij} - c_i$, $t_i = b_i - 2c_i$. Note that by our construction w_{ij} and t_i are non negative. Also, for any $\mathbf{x} \in \{0,1\}^{|V|}$ with $\mathbf{1^T x} = 2$, $\mathbf{x}$ is valid for $\mathbf{Ax} \leq \mathbf{b}$ if and only if it is valid for $\mathbf{Wx} \leq \mathbf{t}$. We see that for every $i \in [k]$, the weights $(w_{i1}, \dots, w_{i,|V|})$ and the threshold value t_i define a threshold subgraph of G with edge set $E_i = \{(\alpha, \beta) \in E : w_{i,\alpha} + w_{i,\beta} > t_i\}$ and vertex set V. Also, since all the characteristic vectors of edges are invalid for $\mathbf{Wx} \leq \mathbf{t}$ then any edge is contained in at least one of $E_1, \dots, E_2$. Now by Proposition 5.5.3 we have that if $sep^*(S, T) \geq 3$ then $sep^*(S, T) = sep(S, T)$ and so $t(G) \leq sep(S, T)$. Finally note that in this case $sep(S, T) = ip(S)$ since all the points in $S \cup T$ violates the equality $\mathbf{1x} = \mathbf{2}$ and so $t(G) \leq ip(S)$. ■

Example 5.5.11 Here we demonstrate the reduction presented in the proof of Theorem 5.5.10. Consider the C_4 graphs with $V = [4]$ and $E = \{\{1,2\}, \{2,3\}, \{3,4\}, \{4,1\}\}$. The threshold dimension of this graph is 2, obtained by dividing the edge set E into $T_1 = \{\{1,2\}, \{2,3\}\}$ and $T_2 = \{\{3,4\}, \{4,1\}\}$. with $A_1 = (2,4,2,0)$, $A_2 = (2,0,2,4)$, $b_1 = 5$, $b_2 = 5$. The only non-edges in this graph are $\{\{1,3\}, \{2,4\}\}$ so $S = \left\{(1,0,1,0)^T, (0,1,0,1)^T\right\}$

Indeed a separation of of S from $\bar{S}$ is given by the following system with 2 inequalities,

$$\mathbf{1x} = \mathbf{2}, \quad \mathbf{Ax} \leq \mathbf{b}$$

or explicitly

$$x_1 + x_2 + x_3 + x_4 = 2, \quad 2x_1 + 4x_2 + 2x_3 \leq 5, \quad 2x_1 + 2x_3 + 4x_4 \leq 5.$$

We note that the question whether the IP-Separation index decision problems for $k = 1$ and $k = 2$ are $\mathcal{NP}$-Complete is left open.

5.6 Examples of IP-Separation index for some Families of sets

In many situation, especially in combinatorial optimization, it is natural to consider a be family of binary sets of arbitrary dimension which posses some common uniform structure. It is then may be useful to make to following definitions regarding such families.

Let $\mathcal{F}$ be a family of binary sets in various dimensions and $\mathcal{F}_d$ be its sub-family of all members $F \in \mathcal{F}$ such that $F \subset \{0,1\}^d$. More formally, let $P(A) \equiv 2^A$ be the powerset of A; then $\mathcal{F}$ is a subset of $\cup_{d \geq 1} P(\{0,1\}^d)$ and $\mathcal{F}_d \equiv \mathcal{F} \cap P(\{0,1\}^d)$.

Definition 5.6.1 *The IP-Separation index of a family $\mathcal{F}$ as above is a function $ip_{\mathcal{F}} : \mathbb{N} \to \mathbb{N}$ defined by*

$$ip_{\mathcal{F}}(d) = \max \{ip(F) : F \in \mathcal{F}_d\}$$

Thus $\text{ip}_{\mathcal{F}}(d)$ is the "worst" IP-separation index of any member $F \in \mathcal{F}_d$.

Example 5.6.2 (Parity) *Let $P_d \subset \{0,1\}^d$ be the parity set i.e., the set of all the vectors in $\{0,1\}^d$ with even number of 1s. Let $\mathcal{P} = \{P_d : d \in \mathbb{N}\}$. By Example 5.3.9 we have that $ip_{\mathcal{P}}(d) = 2^{d-1}$.*

Example 5.6.3 (Perfect Matching) *Let M_n be the set of all characteristic vectors of perfect matchings in a complete graph K_n and let $\mathcal{M}$ be the family of all sets M_n. Since there are $\binom{n}{2}$ edges in K_n, then $M_n \subset \{0,1\}^{\binom{n}{2}}$. One can verify that*

$$M_n = \{\mathbf{x} \in \{0,1\}^{\binom{n}{2}} : \sum_{i=0}^{n-1} n^i \sum_{e \in \delta(v_i)} x_e = \sum_{i=0}^{n-1} n^i\} \equiv$$

$$\{\mathbf{x} \in \{0,1\}^{\binom{n}{2}} : \sum_{0\le i\le j\le n-1} (n^i + n^j)x_{ij} = \frac{n^n - 1}{n-1}\}$$

when the vertices of K_n are denoted by $v_0, v_1, \ldots, v_{n-1}$. This construction is based on the proof of 5.1.9. We conclude that $ip_{\mathcal{M}}(d) = 0$.

Definition 5.6.4 (Automorphisms of graphs) *An Automorphisms of graph $G = (V, E)$ is a vertex permutation $\pi : V \to V$ which preserve E, that is $\{\{\pi(i), \pi(j)\} : \{i, j\} \in E\} = E$.*

Recall that for each permutation $\pi : V \to V$, the corresponding **permutation matrix** is the $|V| \times |V|$ matrix X with $x_{ij} = 1$ if $\pi(i) = j$ and $x_{ij} = 0$ otherwise.

Definition 5.6.5 (Isomorphism group) *The isomorphism group of a graph G is the group $AUT(G)$ of permutation matrices that correspond to automorphism of G.*

Propsition 5.6.6 *Let $\mathcal{AUT}$ be family of all isomorphism groups of graphs. Then $ip_{\mathcal{AUT}}(d) = 0$.*

Proof. Using Theorem 5.1.9 it left to present an IP-formulation of this set by a system of equalities. Let $\mathbf{G}$ be the adjacency matrix of the graph, that is $G_{ij} = 1$ if $\{i, j\} \in E$ and 0 otherwise. It easy to check that if $\mathbf{X}$ as is a permutation matrix then the matrix $\mathbf{X}\mathbf{G}\mathbf{X}^T$ is the adjacency matrix of graph obtained by permuting G according to X. That is X is in the automorphism group of G if and only if $\mathbf{X}\mathbf{G}\mathbf{X}^T = \mathbf{G}$

$$AUT(G) = \{\mathbf{X} \in \{0,1\}^{n\times n} : \mathbf{G}\mathbf{X}^T - \mathbf{X}^T\mathbf{G} = \mathbf{0}, \mathbf{1}\mathbf{X} = \mathbf{1}, \mathbf{1}\mathbf{X}^T = \mathbf{1}\}.$$

The first type of equalities $\mathbf{G}\mathbf{X}^T - \mathbf{X}^T\mathbf{G} = \mathbf{0}$ obtained by a simple manipulation on $\mathbf{X}\mathbf{G}\mathbf{X}^T = \mathbf{G}$ and using the fact that for permutation matrix $\mathbf{X}^{-1} = \mathbf{X}^T$. The last two equalities assure that $\mathbf{X}$ is a permutation matrices.■

Definition 5.6.7 (Cuts of a complete graph) *Consider the family $\mathcal{C}$ of all sets of characteristic vectors of cuts in terms of edges in a complete graph K_n. Note that the number of edges in K_n is $d = \binom{n}{2}$ and thus $\mathcal{C}$ admits members only in dimensions $d = \binom{n}{2}$ for positive integers n.*

Propsition 5.6.8 *The IP-separation index $ip_{\mathcal{C}}(d) \leq 2d$.*

Proof. We provide a system with $2\binom{n}{2} = 2d$ inequalities that separates $\mathcal{C}$ from its complement.

$$\sum_{e \in \delta(i)} x_e + \sum_{e \in \delta(j)} x_e \leq (2 - x_{\{i,j\}}) \cdot n \qquad \forall \{i,j\} \in E \tag{5.16}$$

$$\sum_{e \in \delta(i)} x_e \geq -x_{\{1,i\}} \cdot n + \sum_{e \in \delta(1)} x_e \qquad \forall i \in V \setminus \{1\} \tag{5.17}$$

$$x_{\{1,i\}} + x_{\{1,j\}} \geq x_{\{i,j\}} \qquad \forall i,j \in V \setminus \{1\} : i \neq j \tag{5.18}$$

We first show that (5.16) - (5.18) are valid for any characteristic vector of a cut. Note that the edge set of a cut in K_n induces a complete bipartite graph $K_{\alpha,n-\alpha}$. To see why (5.16) is a valid inequality consider any edge of the cut then its end vertices are belong in different sides of the bipartite graphs. The vertex in the first side is connected to $n - \alpha$ vertices and the vertex in the second side connected to α vertices, that is for the case of $x_{i,j} = 1$ we have n on both sides of the inequality. Note that for $x_{ij} = 0$ the inequality is effectively vanished . Inequality 5.18) is valid since the degree of any vertices in the same side of the complete bipartite as vertex 1 is equal the degree of 1 and the inequality vanishes for vertices on the opposite side.

The see why (5.18) is a valid inequality for the set of characteristic vectors of cuts assume w.l.g that vertex 1 is on side A. Now, we always at one of the following four cases,

1. i and j are both in side A of the bipartite graph. That is $x_{ij} = x_{1,j} = x_{1,i} = 0$.
2. i is in side A and j is in side B of the bipartite graph. That is $x_{ij} = x_{1,j} = 1$ and $x_{1,i} = 0$.
3. j is in side A and i is in side B of the bipartite graph. That is $x_{ij} = x_{1,i} = 1$ and $x_{1,j} = 0$.
4. i and j are both in side B of the bipartite graph. That is $x_{1,j} = x_{1,i} = 1$ and $x_{ij} = 0$.

We see that (5.18) holds in any of these cases. Note that the zero vector that characterizes the empty cut is also valid for the above inequalities since the complete bipartite graph $K_{n,0}$ is an empty graph.

Conversely, assume $\mathbf{x} \in \{0,1\}^{\binom{n}{2}}$ admits (5.16) - (5.18). We show that $\mathbf{x}$ is a characteristic vector that induces a complete bipartite graph on V. Let us construct the following partition of V: $A = \{i \in V | x_{\{1,i\}} = 0\} \cup \{1\}$ and $B = \{i \in V | x_{\{1,i\}} = 1\}$. Note by (5.18) A is a stable set (i.e., there are no edge that connects its vertices). By (5.17) the degree of each vertex in A is at least $|B|$ but since A is a stable set it must be exactly $|B|$ and so each vertex of A is connected to all vertices of $|B|$. That is the degree of each vertex in B is at least $|A|$. Now, by (5.16) the degree of each vertex of B is at most $|A|$ and so it must be exactly $|A|$ and B must be a stable set. We conclude that the induced graph is complete bipartite graph on V and so it is a cut of the complete graph. ■

We note that the bound provided by Proposition 5.6.8 is not a tight bound. For example for $S = \{(1,1,0),(1,0,1),(0,1,1)\}$ the set off all cuts K_3 we have $sep(S) = 0$. A separation for this case is given by the following single equality $x_{\{1,2\}} + x_{\{1,3\}} + x_{\{2,3\}} = 2$. Also, for K_4 we are able to provide a separation with four inequalities namely,

$$\begin{array}{rrrrrrcr}
-4998x_1 & -2x_2 & +5002x_3 & +5000x_4 & -10000x_5 & -5004x_6 & \leq & 1 \\
12504x_1 & +12484x_2 & +25007x_3 & +24988x_4 & +37511x_5 & +37491x_6 & \leq & 1000000 \\
-2x_1 & +4998x_2 & -5002x_3 & -5000x_4 & +9998x_5 & -10000x_6 & \leq & 49950 \\
2x_1 & -4x_2 & -9996x_3 & -6x_4 & -9998x_5 & +10000x_6 & \leq & 1
\end{array}$$

Note this formulation obtained by solving a variant of Mixed Integer Program 5.4.1.

5.7 Discussion and Further Research

We study the concept of separation sets of vectors using a system of equalities and inequalities. We believe that this idea can be used as a basis for better design of Decision support system where binary decision and selection are to be made. In a sense our method is an extension of Logical Data Analysis theory as any logical representation with n terms can be converted to system of linear equalities and inequalities with at most n inequalities and one equality. Moreover usually a much smaller number of inequalities is needed.

This paper focused on some theoretical aspects of the separation. The practical use of this method for real life application is yet to be studied. There are some important points to be addressed in order to enhance the applicability of the .

First, more efficient algorithms and heuristic methods should be devised in order to solve the separation problem. One possible direction for the case of binary sets is to enhance Algorithm 5.2.1, see discussion in Section 5.2. It is also important to have an algorithm and bound based on $|S|$ for the case of sets of general integer vectors. For this end it may be useful to extend the concept of separation to allow separation using several systems of inequalities, where the points of S are those that are valid for at least one of the systems and the points of T violate all the systems. Using such a method it is possible to separate any pair of finite sets. Other interesting direction for further research is to consider separation using non-linear inequalities.

Finally, to promote our theoretical endeavor, it will be interesting to characterize the IP-Separability of some important and complex sets such as the set of all characteristic vectors of Hamiltonian cycles in a graph, or all cuts in a graph.

Bibliography

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, pages 391–401, 1988.

[2] G. Allon, D.P. Kroese, T. Raviv, and R.Y. Rubinstein. Solving the buffer allocation problem using the cross entropy method. *to be appear in Annals of Operations Research*, 2004.

[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation.* Springer, 1999.

[4] E. Babson, S. Onn, and R.R. Thomas. The Hilbert zonotope and a polynomial time algorithm for universal gröbner bases. *Advances in Applied Mathematics*, 30:529–544, 2003.

[5] D. Bertsimas and D. Gammarnik. Asymptotically optimal algorithm for job shop scheduling and packet routing. *Journal of Algorithms*, 33:296–318, 1999.

[6] D. Bertsimas and J. Sethuraman. From fluid relaxations to practical algorithms for job shop scheduling: the makespan objective. *Mathematical Programming*, 92:61–102, 2002.

[7] T. Boudoukh, M. Penn, and G. Weiss. Scheduling job shop with some identical or similar jobs. *Journal of Scheduling*, 4:177–199, 2001.

[8] G.H. Bradley. Transformation of an integer programs to knapsack problems. *Discrete Mathematics*, 1:29–45, 1971.

[9] H. Chen and A. Mandelbaum. Discrete flow networks: Bottleneck analysis and fluid approximation. *Mathematics of Operations Research*, 16:2:408–445, 1991.

[10] H. Chen and D. D. Yao. *Fundamentals of Queueing Networks: Performance, Asymptotics and Optimization.* Springer, 2001.

[11] V. Chvátal and P.L. Hammer. Aggregation of inequalities in integer programming. *Annals of discrete mathematics*, 1:145–162, 1977.

[12] J.G. Dai. and G. Weiss. Stability and instability of fluid models for certain re-entrant lines. *Mathematics of Operations Research*, 21:115–134, 1996.

[13] J.G. Dai and G. Weiss. A fluid heuristic for minimizing makespan in job-shops. *Operations Research*, 50:692–707, 2002.

[14] A. Deza and M. Deza. On the skeleton of dual cut polytope. *Report*, 1993.

[15] P. Duchet. *Handbook of combinatorics"*, chapter Hypergraphs, pages 381–432. Elsevier Science, Amsterdam, 1995.

[16] O. Ekin, P.L. Hammer, and A. Kogan. Convexity and logical analysis of data. *Theoretical Computer Science*, 244:95–116, 2000.

[17] L. Fleischer and J. Sethuraman. Approximately optimal control of fluid networks. 2003.

[18] S.B. Gershwin and J.E. Schor. Efficient algorithms for buffer space allocation. *Annals of Operations Research*, 93:117–144, 2000.

[19] Z. Drezner H. Gurnani and R. Akella. Capacity planning under different inspection strategies. *European Journal of Operational Research*, 89:302–312, 1996.

[20] N.G. Hall, T.E. Lee, and M.E. Posner. The complexity of cyclic shop scheduling problems. *Journal of Scheduling*, 5:307–327, 2002.

[21] P.L. Hammer. Partially defined boolean functions and cause-effect relationships. In *Int'l Conf. Multi-Attrubute Decision Making Via OR-Based Expert Systems*, April 1986.

[22] C. Hanen. Study of an np-hard cyclic scheduling problem: the periodic recurrent job shop. *European Journal of Operational Research*, 72:82–101, 1994.

[23] K. Kogan and T. Raz. Optimal allocation of inspection effort over a finite planning horizon. *IIE Transactions*, 34:515 527, 2002.

[24] T.E. Lee and M.E. Posner. Performance measures and schedule patterns in periodic job shops. *Operations Research*, 45:7291, 1997.

[25] G.F. Lindsay and A.B. Bishop. Allocation of screening inspection effort: a dynamic programming approach. *Management Science*, 10:342–352, 1965.

[26] L. Lovász and A. Schrijver. Cones of matrices and set-function and 0-1 optimization. *SIAM Journal of Optimization*, 1:166–190, 1991.

[27] N.V.R. Mahandev and U.N. Peled. *Threshold Graphs and Related Topics.* North-Holand, Elsiver, 1995.

[28] K. Mehlhorn and S. Näher. *Leda - a platform for combinatorial optimization and geometric computing.* Cambridge University Press, 1999.

[29] T. S. Mountford and B. Prabhakar. On the weak convergence of departures from an infinite series of $\cdot/m/1$ queues. *Annals of Applied Probability*, 5:121–127, 1995.

[30] B. L. Nelsson, J. Swann, D. Goldsman, and W. Song. Simple procedures for selecting the best simulated system when the number of alternatives is large. *Operations Research*, 49:950–963, 2002.

[31] S. Onn. Approximating oracle machines for combinatorial optimization. *SIAM Journal of Optimization*, 4:142–145, 1994.

[32] M. Penn and T. Raviv. Approximation procedure for the high multiplicity job shop and its extension. *Annual Meeting of the Operations Research Society of Israel*, 2001.

[33] M. Penn and T. Raviv. Solving the infinite planning horizon vehicle routing problem. *in preparation*, 2003.

[34] T. Raz. A survey of models for allocating inspections effort in multistage production system. *Journal of Quality Technology*, 18:239–247, 1986.

[35] T. Raz and M. Kaspi. Location and sequencing of imperfect operations in serial multi-stage production system. *International Journal of Production Research*, 29:1654–1659, 1991.

[36] R.T. Rockaffelar. *Convex Analysis.* Princeton University Press, 1970.

[37] B. Strumfels and R.R. Thomas. Variation of cost functions in integer programming. *Mathematical Programming*, 77:357–387, 1997.

[38] S. Suresh and W. Whitt. The heavy-trafic bottleneck phenomenon in open queueing networks. *Operations Research Letters*, 9:355–362, 1990.

[39] G.A. Vouros and H.T. Papadopoulos. Buffer allocation in unreliable production lines using a knowledge based system. *Computer & Operations Research*, 25:1055–1067, 1998.

[40] G. Weiss. A simplex based algorithm to solve separated continuous linear programs. *Mathematical Programming, forthcoming*, 2004.

[41] T. Yamada and R. Nakano. A genetic algorithm applicable to large-scale job-shop instances. In R. Männer and B. Manderick, editors, *Parallel instance solving from nature 2*, pages 281–290. North-Holand, Elsiver, 1992.

[42] B.J. Yum and E.D. McDowell. Optimal inspection policies in a serial production system including scarp, rework and repair: an MILP approach. *International Journal of Production Research*, 25:1451–1464, 1987.