View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Bilkent University Institutional Repository

THE LINE BALANCING ALGORITHM FOR OPTIMAL BUFFER ALLOCATION IN PRODUCTION LINES

A THESIS
SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING
AND THE INSTITUTE OF ENGINEERING
AND SCIENCE OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Ömer Selvi
September, 2002

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Murat Fadıloğlu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. İhsan Sabuncuoğlu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Erdal Erel

Approved for the Institute of Engineering and Science

Prof. Mehmet Baray
Director of Institute of Engineering and Science

Abstract

THE LINE BALANCING ALGORITHM
FOR OPTIMAL BUFFER ALLOCATION
IN PRODUCTION LINES

Ömer Selvi
M.S. in Industrial Engineering
Supervisor: Asst. Prof. Murat Fadıloğlu
September, 2002

Buffer allocation is a challenging design problem in serial production lines that is often faced in the industry. Effective use of buffers (i.e. how much buffer storage to allow and where to place it) in production lines is important since buffers can have a great impact on the efficiency of the production line. Buffers reduce the blocking of the upstream station and the starvation of the downstream station. However, buffer storage is expensive both due to its direct cost and the increase of the work-in-process inventories it causes. Thus, there is a trade-off between performance and cost. This means that the optimal buffer capacity and the allocation of this capacity have to be determined by analysis. In this thesis, we focus on the optimal buffer allocation problem. We try to maximize the throughput of the serial production line by allocating the total fixed number of buffer slots among the buffer locations and in order to achieve this aim we introduced a new heuristic algorithm called "Line Balancing Algorithm (LIBA)"applicable to all types of production lines meaning that there is no restriction for the distributions of processing, failure and repair times of any machine, the disciplines such as blocking, failure etc. and the assumptions during the application of LIBA in the line.

Keywords: Production Lines, Buffer, Optimal Buffer Allocation Problem

Özet

ÜRETİM HATLARINDA
OPTİMAL ARA DEPO PAYLAŞTIRIMI İÇİN
HAT DENGELEME ALGORİTMASI

Ömer Selvi
Endüstri Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticisi: Yar. Doç. Murat Fadıloğlu
Eylül, 2002

Üretim Hatlarında ara depo paylaştırımı günümüz endüstrisinde genellikle karşılaşılan önemli bir problemdir. Ara depoların üretim hattında etkili kullanımı yani ara depoların hangi miktarda ve nereye yerleştirileceği önemlidir çünkü ara depoların üretim hatlarının verimliliğinde büyük etkisi vardır. Ara depolar kendisinin önündeki ve kendisini takip eden istasyonun tıkanma ve aç kalma sıklıklarını azaltır. Ama direkt maliyetinden ve ara ürün miktarındaki artışa neden olmasından dolayı ara depo kullanımı pahalı bir yatırımdır. Bu yüzden, performans ve maliyet arasında endirekt bir ilişki vardır. Bu, optimal ara depo gereksinim miktarı ve bu miktarın paylaştırımı analiz ile belirlenmeli anlamına gelir. Bu çalışmada, optimal ara depo paylaştırım problemi üzerinde odaklanılmıştır. Toplam sabit ara depo miktarını mevcut ara depo lokasyonları arasında paylaştırımı yoluyla seri üretim hattının birim üretim miktarı maksimize edilmeye çalışılmış ve bu amaca ulaşmak için istasyonların işleme,bozulum ve onarım zamanları dağılımı, blokaj, bozulum vs. disiplini ve hattın varsayımları ne olursa olsun her türlü üretim hattına tatbik edilebilir "Hat Dengeleme Algoritması" adlı sezgisel bir algoritma geliştirilmiştir.

Anahtar Sözcükler: Seri Üretim Hatları, Ara Depo, Optimal Ara Depo Payalaştırım Problemi

Acknowledgement

I would like to express my gratitude to Asst. Prof. Murat Fadıloğlu for his supervision, suggestions and encouragement throughout the development of this thesis.

I am also indepted to Prof. İhsan Sabuncuoğlu and Prof. Erdal Erel for accepting to read and review this thesis and for their suggestions.

I would like to take this opportunity to thank Onur Boyabatlı, Çerağ Pinçe, Güneş Erdoğan, Savaş Çevik, Sezgin Işılak, Burhan Ürek, Ozan Pembe, Aydın Özçelik and Gökhan Çevik for their friendships, helps, morale supports and encouragements during my graduate life in Bilkent. I would also like to express my gratitude to Avni Sezer, Osman Sarıçam, Naci Yılmaz for their friendships.

My special thanks go to my family. This study is dedicated to them without whom it would not have been possible.

Contents

List of Tables

List of Figures

List of Charts

Chapter 1

INTRODUCTION

Production lines, also called *manufacturing flow lines, transfer lines, flow lines* or *serial production lines*, have been an important area of research ever since 1950's. Since flow lines can often be found throughout manufacturing industry (e.g. automobile industry), many researches have recognized the importance of the subject and contributed to it.

Let us first define manufacturing flow lines briefly. Manufacturing flow line systems consist of material, work areas, and storage areas. Material flows from work area to storage area, from storage area to the proceeding work area and so on. Material visits each storage and work area. There is an entry work area through which material enters and an exit work area through which it leaves the system. The work areas are usually called *machines* or *stations* and the storage areas are usually called *buffers*. Figure 1.1 illustrates an N-machine production line where M_i's stand for machines and B_i's are buffers.

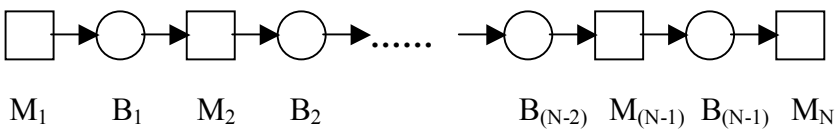

Figure 1. 1. *The N-machine production line*

Due to their diversity, complexity and inherent randomness in their behaviour, modelling and estimating the performances of manufacturing flow lines are difficult. Especially, the randomness inherent in production lines is what makes manufacturing flow lines difficult to analyze. The primary source of randomness is that the times parts spend in work areas are not deterministic. This randomness may be due to random processing times, random failure and repair events that occur on the stations, or both.

In serial production lines, one of the key questions that the designers face is the buffer allocation problem, i.e., how much buffer storage to allow and where to place it in the line. This is an important question since buffers can have a great impact on the efficiency of the production line. They reduce blocking in the upstream stations and the starvation in the downstream stations. Unfortunately, buffer storage is expensive both due to its direct cost and the increase of the work-in-process inventories it causes. Because of the trade-off between the performance and the cost, determination of the total buffer capacity and the allocation of the buffer capacities is an important problem.

The problem can be formulated in many different ways depending on the choice of the objective function. Objectives used in the literature are basically maximizing throughput, minimizing work-in-process, minimizing sojourn time and minimizing cost or maximizing profit based on the user defined cost or profit functions. In this thesis, we study the classical problem, which is known as “Optimal Buffer Allocation Problem (OBAP)” with the objective of maximizing production rate. We focus on the allocation of total fixed number of buffer slots among the buffer locations for the optimal production rate of the production line.

In the second chapter of our study, we give a brief background of the production lines. We provide a review of related research in the literature in Chapter 3. In Chapter 4, we present two related algorithms on the allocation of total fixed number of buffer slots for maximizing throughput and compare these algorithms in Chapter 5 where we also develop a heuristic algorithm. Finally, concluding remarks are made in Chapter 6.

Chapter 2

BACKGROUND

2. 1. Major Features and Classes of Production Lines

There are three major classes of manufacturing flow lines. These are;

1. Asynchronous Systems
2. Synchronous Systems
3. Continuous Systems

Asynchronous and synchronous systems are suitable for the manufacturing of discrete parts. The only difference between them is that all the operations and machine state changes in the line occur simultaneously as well as buffer levels in the synchronous systems. In asynchronous systems, the machines are not forced to start or stop their operations at the same instant. Even when machines have fixed, equal operation times, the presence of buffers between them allows them to start and stop independently, as long as the intermediate buffers are neither empty nor full. In some applications, the operation times may be random. Finally, uncertain failure and repair times can lead the unsynchronized operation times. Unlike asynchronous systems, in synchronous systems, all machines are forced to start and stop their operations at the same time.

The feature distinguishes continuous systems from the others is that the material is treated as continuous rather than discrete. That is, instead of discrete parts moving from buffer to machine and vice versa at specific instants, there is a fluid that is transferred continuously. Continuous systems are naturally the production systems in which the material processed is a fluid rather than discrete entities (e.g. chemical processing).

A machine is said *blocked* if the processed part on it cannot be put to the downstream buffer and *starving* if it is idle and there is no part to be processed in the upstream buffer. The function of a buffer is to decouple machines. If a machine is subject to a disruption (a failure or a long operation time), the machine upstream can still operate until the upstream buffer fills up and the machine downstream can still operate until the downstream buffer becomes empty. The larger the buffers of the line, the longer before the filling or emptying occur, and the larger is the production rate. Pairs of machines that have no storage space between them have the greatest coupling; and infinite buffers, or storage areas that are never filled, have the least (Infinite buffers allow coupling when they become empty).

In real life, since all buffers have finite capacity, blocking may occur. There are two types of blocking;

1. Blocking After Service (BAS)
2. Blocking Before Service (BBS)

BAS, also called *type-1 blocking*, *manufacturing blocking*, *production blocking*, *transfer blocking* or *non-immediate blocking*, occurs at the instant of completion of a part on the machine, if downstream buffer is full. In that case, the part stays on the machine until a space is available in the downstream buffer. During this time, the machine is prevented from working and it is said to be blocked. When a space becomes available in the downstream buffer, the part is immediately transferred to the downstream buffer and the machine can start processing another part, if any.

In BBS, also called *type-2 blocking*, *communication blocking*, *service blocking* or *immediate blocking*, machine can start processing only if there is a space available in the downstream buffer. Otherwise, it has to wait until a space becomes available. BBS is further classified according to whether the position (space) on the machine may be

occupied or not while the machine is blocked. These two cases are *Blocking Before Service with Position Occupied* (BBS-PO) if the space on the machine is used during the blockage, and *Blocking Before Service with Position Non-Occupied* (BBS-PNO) if the space on the machine is not used during the blockage.

In some systems, machines are prone to failures. In the literature, generally two types of failures are considered. These are;

1. Operation Dependent Failures (ODFs)
2. Time Dependent Failures (TDFs)

ODFs are failures that are related to the processing of parts and thus can only occur when the machine is working. The machine is working means that the machine is up (operational) and it is not idle. On the other hand, TDFs are not related to the processing of part and thus can occur at any time, including the time when the machine is idle.

When failure occurs, the machine cannot process any material, so the upstream buffer cannot lose material and the downstream buffer cannot gain material. Systems in which machines can fail are called *Flow Lines with Unreliable Machines* (FLUMs) and systems in which machines do not fail are called *Flow Lines with Reliable Machines* (FLRMs). In FLRMs, all the randomness is due to the variability of the processing times, while, in FLUMs, randomness is due to both varying processing times and failures.

Material arrives at and leaves from the flow line in a variety of different ways. It is always possible for raw material to be unavailable, or removal of finished goods may be delayed in real life. Such systems are *non-saturated systems*. On the other hand, in the literature, it is almost always assumed that the first machine is never starved and the last is never blocked. Such systems are called *saturated systems*. However, it is possible to model a non-saturated system with a saturated system by adding a non-starving initial machine as the arrival process and a final machine that is never blocked (means it has infinite capacity downstream buffer) as the departure process to the line. Hence, the second machine of the model corresponds to the first machine of the real system and the machine just before the last machine of the model corresponds to the last machine of the real system.

2. 2. General Results Pertaining to Production Rate of a Production Line

Several measures of performance are of interest when analyzing flow lines. The most important one is the production rate, P, which is the average number of parts leaving the system per unit time.

The production rate of a line is limited in two ways. First, the throughput can be no greater than that of the machine with the smallest isolated production rate. The *isolated production rate* of a machine is the rate that it would operate at if it were not in a system with other machines and buffers. When the machines have different isolated production rates, their capacities except the lowest are largely wasted. Second, the unsynchronized disruptions that cause buffers to be empty or full also waste machine capability. Buffers become empty or full because machines fail or take long time to process material at different times. If all machines could be perfectly synchronized, not only in performing operations, but also in failing and getting repaired, buffers would not affect flow. It is the lack of synchronization that causes machines to be starved or blocked, and thus to lose the opportunity to work.

A fundamental relationship of flow lines is the *conservation of flow* which states that all machines have the same production rates, that is

$$P_1 = P_2 = \ldots\ldots = P_N = P.$$

Conservation of flow holds for FLRMs and also for FLUMs provided that there is no scrapping of parts. Conservation of flow can be established by using sample path approach. The *sample path* behaviour of any flow line can be described by means of recursive equations. These equations are defined as the *evolution equations* of the flow line.

2. 2. 1. Buffer Issues

The production rate increases monotonically as the buffer capacities increase. This is *monotonicity property*. Consider two flow lines, L_1 and L_2, which have identical machines but with different buffer capacity vectors K_1 and K_2. The capacity of each

buffer in L_2 is at least as large as the corresponding buffer in L_1. That is, $K_1 \leq K_2$. Then the production rate of the flow line satisfies

$$P(K_1) \leq P(K_2).$$

The production rate of a flow line in which one buffer is infinite can be obtained by decomposing the line into two sub-lines from this infinite capacity buffer. Let L^a be the part of line L that consists only of the first i machines and the first (i – 1) buffers and similarly let L^b be the part of line L that consists only of the last (N - i) machines and the last (N - i - 1) buffers where the buffer location B_i has infinite capacity. Let P^a and P^b be the production rates of lines L^a and L^b respectively. Then, the production rate, P_{inf} , of the line with the infinite buffer is

$$P_{inf} = \min(P^a , P^b).$$

By combining the above result with the monotonicity property, we obtain the following upper bound for the production rate of the line where there is no infinite capacity buffer:

$$P \leq \min(P^a , P^b)$$

By applying this decomposition several times, we see that the production rate of a flow line is bounded by the isolated production rate of the machine that has the smallest isolated production rate as given below:

$$P \leq \min(\rho_i) \quad i = 1 \text{ to } N$$

Tighter upper bound on the production rate of the original line can be derived from the decomposition approach and given as

$$P \leq \min(P^{i,i+1}) \quad i = 1 \text{ to } (N-1)$$

where $P^{i,i+1}$ is the production rate of the two-machine flow line consisting of M_i, B_i , M_{i+1} .

This upper bound can be useful since the production rates of the two-machine flow lines can be calculated exactly in most cases.

The monotonicity property can also be used to obtain the following lower bound on the production rate of the original line:

$$P^0 \leq P$$

where P^0 is the production rate of the flow line with no intermediate storage.

2. 2. 2. Reversibility and Duality

Consider a flow line L^r, which is obtained from flow line L by reversing the flow of parts. The first machine in L^r is the same as the last machine in L. More generally, M_i in line L^r is the same as M_{N-i+1} in line L. Also buffers are reversed in line L^r. Then the production rate of the reversed line L^r is the same as the production rate of original line L if both lines' blocking mechanism is BAS. This is *reversibility property*. Proof of this property is based on the comparison of the sample paths of the two systems again using the evolution equations.

Consider now the case of BBS. In that case, there is a much stronger equivalence between the two systems (L^r and L). This equivalence is based on the concept of job/hole (or part/hole) *duality*. The idea is that in line L, whenever a part moves in one direction, a hole (empty space) moves in the other direction. In the case of BBS, it is easy to check that the behaviour of parts in the reversed system is the same as the behaviour of holes in the original system. Indeed, starvation in the reversed system corresponds to blocking in the original system and vice-versa. As a result, the steady-state distribution of parts in the reversed line is exactly the same as the steady-state distribution of holes in the original line. This equivalence especially implies that these two systems have the same production rate.

The concept of job/hole duality still makes sense in the case of BAS. However, the behaviour of parts in the reversed system is no longer the same as the behaviour of holes in the original system.

In this chapter we give the brief background of the production lines. It is worth to give this background since all researches on the production lines in the literature use any of these classes and features of production lines as the framework while modelling them and introduce new derivations by basically using the general results pertaining to

flow lines. Therefore, the content of this chapter will help the reader understand the next chapter easily where we will introduce the researches on production lines in the literature.

Chapter 3

LITERATURE SURVEY

Over the years, a large amount of research has been devoted to the analysis of production lines. This body of research can be classified as evaluative and generative. Evaluative studies focused on the performance evaluation of the production lines such as production rate, average WIP and average sojourn time in the system. Generative studies dealt with the optimization of these performance measures of the production lines. Since there is vast amount of work on production lines, we will only deal with the ones that are directly related to our problem, *buffer allocation*. However, the review of production lines written by Gershwin and Dallery[8] can be given as a guide to the readers who are interested in finding about evaluative studies.

In serial production lines, one of the key questions that the designers face is the buffer allocation problem, i.e., how much buffer storage to allow and where to place it in the line. This is an important question because buffers can have a great impact on the efficiency of the production line. They compensate blocking of the upstream stations and the starvation of the downstream stations. Unfortunately, buffer storage is expensive both due to its direct cost and the increase of the work-in-process inventories it causes. Therefore, there is a trade-off between performance and cost. Thus, the

determination of buffer capacity requirement and the allocation of the buffer capacities is an important issue.

While solving the buffer allocation problem, determination of the objective and the assumptions of the models that are worked on are also important for some reasons such as the tractability, fidelity to reality etc. Objectives that were used in the literature are basically maximizing throughput, minimizing work-in-process, minimizing sojourn time and minimizing cost or maximizing profit based on the user defined cost or profit functions. Minimization of WIP and average sojourn time are positively correlated, meaning that minimization of one produces the minimization of the other, while these objectives are negatively correlated with the objective of maximizing throughput. There are also multi-objective studies aiming to achieve two or more objectives, which were stated above, at the same time. You can see various studies with these objectives in the proceeding section. Unless otherwise stated, all proceeding researches used the basic assumption stated below due to simplicity and tractability;

1. The first machine is never starved and the last machine is never blocked (Saturated Systems).
2. All random variables (processing times, uptimes, downtimes) are independent random variables.
3. The transfer through the buffers takes zero time.
4. Manufacturing blocking (BAS) is the blocking criterion, meaning that any machine can pass the completed part as long as a buffer space is available (or, when no buffer exists, the downstream machine is idle).
5. Failures are operation dependent failures (ODFs) meaning that any machine can fail only when it is processing a part. In other words, a machine can not fail when it is idle (starved or blocked).
6. When a failure occurs, the part stays on the machine; it can be reworked when the machine is up again; the work resumes exactly at the point it stops(no scrapping of parts)

Conway et al.[5] analyzed both balanced and unbalanced serial lines with stations having uniform and exponential processing times via simulation and reported a number of useful generalizations about the effect of buffers on serial lines. These can be summarized as follows:

- ***Diminishing returns*:** Throughput increases at a decreasing rate when successive buffer are placed at a single buffer site, or when successive sets of buffers are placed at all sites.
- ***Non-concavity*:** Throughput increases in a non-concave fashion when successive buffers are placed optimally.
- ***Sufficiency of small numbers*:** For lines with low coefficient of variations, small numbers of buffers at each site are sufficient to recover most of the throughput lost to stochastic interference.
- ***Bowl-phenomenon*:** Buffers should be allocated evenly to all sites if possible, with any remaining buffers allocated symmetrically around the centre of the line.
- ***Reversibility principle*:** Any line has the same throughput as its mirror image.
- ***Decomposition principle*:** A single buffer should be placed where an unlimited buffer would be most effective.
- ***Built-up property*:** The optimal allocation of (n+1) buffers can be built upon the optimal allocation of n without moving any of the first n.

Anderson and Moodie[2] analyzed the balanced production lines with normal and exponential operation times to estimate the coefficients for buffer locations that satisfies the optimal production cost modeled. Multi-product production lines with equal storage capacity for all buffer locations were considered. Anderson and Moodie derived mathematical expression for the operation cost of the line for both cases: normal and exponential service times, and developed minimum cost buffer models from these expressions. Transient behavior of the line was also considered in order to observe whether it is beneficial to control the buffer capacities during this period or not. However, it was observed that there was no cost advantage in controlling the inventory during the transient period.

In his study, Helber[12] defined the problem of buffer space allocation in production lines as an investment problem. A model was developed and solution techniques were described that could be used to determine buffer allocations that maximize the expected net present value of the investment, including machines, buffers and inventory. Several examples of flow lines as well as assembly / disassembly

systems and flow lines with rework loops were analyzed. Optimal buffer allocation was determined via gradient algorithm based on the assumption of concavity of the function of the expected net present value of the investment with respect to the buffer capacity vector. Basic result of this study was that as product quality in a system with a rework loop improves, an optimally designed system can receive more buffer spaces and may use more inventory.

A flow-shop type production line where the stations were subject to breakdown was studied by Altiok and Stidham[1]. The objective was to find the allocation of inter-stage buffer capacities that maximizes the total profit. The stations, which are modeled as single-server queuing systems, had completion time distribution of two-stage Coxian type. After a standard transformation to a phase-type state representation, the new system gave rise to a Markov chain. The balance equations for this chain were solved by successive approximations to find the steady-state probability distribution of the number of items at each station, once the buffer capacities were given. A search procedure has been employed to find the optimal buffer capacities.

Seong et al.[36] studied the same objective function as Altiok and Stidham[1] with general linear constraints on buffer sizes and continuous-type product assumption. Operation times were assumed to be deterministic and equal, while the repairs and failures were exponentially distributed. They solved the problem with a gradient projection algorithm.

While allocating the fixed total number of buffers among intermediate buffer locations optimally, Andijani and Anwarul[3] considered and investigated the trade-off between three conflicting objectives: maximizing the average throughput rate, minimizing the average WIP and minimizing the average system time. They used lines with three and four identical reliable machines with exponential and uniform service times. Stochastic system simulation was used to generate and construct an efficient set of buffer allocations which maximizes the average throughput rate and minimizes both the average WIP and the average system time. Based on these simulation results, Analytical Hierarchy Process (AHP) was utilized to identify the most preferred allocation. The objective of this process was to find, for the line, the best buffer allocation solution to the trade-off between the three conflicting objectives stated.

Papadopoulos et.al.[25] also tried to allocate the fixed-number of buffers, servers and fixed amount of workload at the same time as well as individually and in couples in order to maximize the throughput with minimum average WIP by using simulated annealing approach. Decision variables considered were the sizes of the buffers placed between successive workstations of the lines, the number of servers operating parallel allocated to each workstation and the amount of workload allocated to each workstation. The study was extended up to 60 stations with 120 buffer capacity and 120 servers, and it was observed that the approach worked very well compared with complete enumeration whenever possible as well as produced near-optimal configurations for relatively large lines in reasonable time.

So[38] studied the buffer allocation problem with the objective of minimizing the average work-in-process (WIP) subject to a minimum required throughput and a constraint on the total buffer space. Both the balanced and the unbalanced lines up to five reliable machines were considered in this study. Exponential and non-exponential (Erlang-2, Coxian etc.) operation times were assumed in balanced lines while only exponential operation times were taken into account in the unbalanced lines. So's results showed that the optimal strategy of allocating buffer size for this problem exhibited a rather interesting pattern that was different from the buffer allocation problem of maximizing the throughput subject to a constraint on the total buffer space. Specifically, monotonically increasing allocations, where an increasing amount of buffer space is assigned toward the end of the line, were shown to be optimal for the most cases investigated. Furthermore, empirical results obtained in this study suggest that when the line is unbalanced, the slowest operations should be assigned to the beginning of the line to provide the best throughput and the average work-in-process trade-off. On the basis of these results, a good heuristic for selecting the optimal buffer allocations for minimization of work-in-process inventory while achieving minimum required throughput with constant total buffer space was developed.

Papadopoulos and Vidalis[28] worked on the same optimization problem, minimizing the work-in-process inventory while achieving the required throughput with constant total buffer capacity. However, they only focused on the short reliable balanced production lines with Erlang-k ($k \geq 2$) operation times. More specifically, they studied the average WIP and throughput for all the ordered buffer allocations of a certain total number of buffer slots among the intermediate buffer locations. The

vectors of the buffer allocations were classified systematically into equivalence classes, something that facilitated a lot in the analysis of the evolution of the average WIP and the throughput as a function of these ordered allocations. Papadopoulos and Vidalis' results were very similar to the ones So[38] derived. According to these results, each buffer location takes at least as much buffer slot as the preceding one for the required throughput levels small relative to the theoretical maximum throughput that is attained when the buffer slots at hand are placed in order to maximize the throughput of the whole line. However, while the desired throughput level is increasing to the theoretical maximum level, buffer slots are transferred to the inner locations gradually resulting in well-known bowl phenomenon. Also a heuristic algorithm was proposed to find the optimal buffer allocation (OBA), which reduces the search space by 50% compared to enumeration in this study.

Another study on the OBA in order to minimize WIP by Kim and Lee[20] proposed an efficient heuristic algorithm. This algorithm named MNS (Modified Non-SEVA) is the modified version of the Non-SEVA algorithm (Non-Standard Exchange Vector Algorithm) which was originally proposed by Seong et al.[35] for the throughput maximization problem. However, since some useful structural properties such as monotonicity and concavity which hold for the throughput function and are the basic assumptions of the Non-SEVA algorithm, do not hold for the average WIP. Therefore, Kim and Lee used the results of Seong et al.[35] in order to obtain a initial solution which close to the global optimum. Kim and Lee worked on the unreliable production lines with up to ten machines. The failure rate, the repair rate and the production rate of each machine were obtained from the same uniform distributions. In order to compare the efficiency of MNS in the computational tests, also another heuristic, which was based on one buffer assignment at a time, was proposed called Simple Heuristic Algorithm (SHA) in this study. The two algorithms were compared with the solutions obtained by enumeration for short lines with up to 3 or 4 machines and compared against each other for longer lines with 8 and 10 machines where complete enumeration is inefficient. For all cases, MNS outperformed SHA in terms of average WIP levels with reasonable number of iterations. MNS also gave average WIP levels very close to the optimal solution achieved by enumeration for the cases where enumeration technique was used.

Papadopoulos and Vouros[29] presented a prototype model management system (MMS) for the design and operation of manufacturing systems. The model management system classifies different models according to the type of the manufacturing system to which they apply and according to the particular technique employed. The system comprises three different techniques, namely, analytical, simulation and artificial intelligence (AI) based techniques for production lines. The first two are evaluative methods, whereas the last one is a generative (optimization) method that solves the buffer allocation problem in a production line. Papadopoulos and Vouros studied on both balanced and unbalanced production lines with reliable stations having exponential operation times aiming to minimize average WIP by allocating fixed number of buffer capacity. First contribution of this work was that the development of a flexible MMS, which provides a simple and intelligible framework for classifying different, modeling techniques, enables the interaction among these models and does not restrict the developers to follow a particular model development task. Second one is the development of a knowledge based system, called Advisor System for Buffer Allocation (ASBA), which solves the buffer allocation problem in the production lines with very satisfactory results.

Papadopoulos and Vouros[30] also introduced ASBA2, a knowledge based system that solves the buffer allocation problem in production lines as an extension of ASBA. ASBA allocates buffer space in reliable both balanced and unbalanced production lines, aiming at reducing average WIP subject to a given total buffer space and a required throughput. However, ASBA2 aims to extend the functionality of ASBA to unreliable, balanced and unbalanced production lines and allocates the fixed total buffer space in order to achieve the objective of maximizing throughput. The results showed that ASBA2 allocates the buffers very close to the optimal ones in a computationally efficient way by using specific types of knowledge.

Hillier and So[15] also provided a study of the effect of machine breakdowns and inter-stage storage on the efficiency of production lines. Based on the results they obtained, Hillier and So developed a simple heuristic method to estimate the amount of storage space required to compensate for the decrease in throughput due to machine breakdowns. The study focused on four and five machine production lines with again operation times from two-stage Coxian distribution. Hillier and So used Coxian distribution for operation times due to useful interpretation of this distribution. Stage 1

can be interpreted as corresponding to the normal service for an item at a machine, whereas Stage 2 corresponds to downtime at the station for whatever reason (e.g. breakdown of the machine) that interrupts this service where the probability of having Stage 2 corresponds to the probability that the service is interrupted by down time. Therefore, their model can be used to study the effect of breakdowns on the allocation of storage space in a production line. First basic result of this study was that the throughput of the production line is inversely proportional to the coefficient of variation of the operation times meaning that increase in coefficient of variation will reduce the throughput of the line. Secondly, percentage increase in the throughput achieved by adding one extra unit of buffer space decreases as the buffer capacities increase. Lastly, while the throughput of a line depends heavily on the average amount of downtime during one service, the mean length of downtimes can affect the throughput significantly for fixed average amount of downtime during one service; smaller mean length of downtimes gives higher throughput than larger mean length of downtimes.

Yamashita and Altiok[41] were concerned with finding the minimum total buffer number required and its allocation for a desired throughput in both balanced and unbalanced production lines with three and five stations having phase-type processing times. One significant difference of this study from others was that the capacity of each buffer was assumed to be bounded above by a constant value, say C_i. They have implemented a dynamic programming algorithm that uses a decomposition method to approximate the line throughput at every stage.

Lutz et. al.[21] addressed the problem of buffer location and the storage size in a manufacturing lines. The question was what buffer sizes should be employed and where the buffers should be located. Hence, the objectives of Lutz et.al. were to determine the minimum number of storage spaces needed and the allocation of these storage spaces among the buffers, so as to maximize the overall throughput of the line. To achieve these objectives, simulation-search heuristic procedure based on tabu search was developed. Simulation was used to model the manufacturing process and the tabu search was used to guide the search to overcome the problem of being trapped at local optimal solutions. The procedure employs a Swap Search routine and a Global Search routine. With the Swap Search routine, the procedure identifies good performing buffer profiles and determines the maximum output level for any given storage level. With the Global Search routine, the procedure can locate promising neighborhood of buffer

profiles quickly. The procedure is capable of modeling a variety of manufacturing processes with a variety of scheduling policies and dispatching rules.

Park[31] presented characteristics of the buffer design problem associated with the production lines and discussed some drawbacks related to the optimization methods thus far his study applied to the buffer allocation problem. An efficient two-phase heuristic method, using a dimension reduction strategy and a buffer utilization-based beam search method, was developed to minimize total buffer storage required while satisfying a desired throughput rate in unreliable balanced production lines with stations having deterministic processing times, and geometric failure and repair times. While Phase I attempts to accelerate the finding of an initial solution by reducing the combinatorial search dimension to one, Phase II reduces the total buffer storage required as much as possible while maintaining a desired throughput rate.

Gershwin and Schor [9] described efficient algorithms for determining how buffer space should be allocated in a flow line. They considered unreliable lines with deterministic operation times. Two problems were analyzed: a primal and a dual problem. The goal of the primal problem is to minimize total buffer space required for the line to meet or exceed a given average production rate, and the goal of dual problem is to maximize the production rate achievable with a given total buffer space. The dual problem is solved by means of a gradient method, and the primal problem is solved using the dual solution. It was also showed how buffer allocation problems with profit maximization objective could be solved by using essentially the same algorithms.

Sheskin[37] studied the allocation of buffer spaces in systems like Gershwin and Schor[9]: those with unreliable machines with equal deterministic processing times. In addition, he assumed time-dependent failures. A decomposition method was used to produce numerical results for small systems with small buffer capacities. These results led to some rules of thumb on the allocation of buffer spaces to maximize production rate.

Soyster et.al.[39] used the same model with Sheskin[37] to study the maximization of production rate subject to general linear inequality constraints on buffer sizes. They approximated the production rate for small systems and used an integer programming package to find optimal allocation of buffer spaces.

El-Rayah[6] attempted to study the effect of unequal allocation of fixed number buffer storage and the imbalance in the operation time variabilities on the throughput and average WIP. He simulated balanced lines in terms of mean operation times where operation times were assumed to be normal. In the first section of the study, he only investigated the effect of unequal buffer allocation on the output rate and average WIP of the lines up to four machines which were also CV-balanced (where all machines have same coefficient of variation). On the basis of simulation results for this section, El-Rayah concluded that the output rate of a production line where the buffer capacity is allocated equally cannot be significantly improved by deliberately unbalancing buffer allocation. However, if imbalance is unavoidable, throughput is maximized by assigning larger buffer capacities to the middle buffer locations and smaller capacities to the end buffer locations on the line. He also observed from the results of the first section that increasing order of buffer capacities encourages the reduction of average WIP significantly while affecting the throughput in the decreasing direction so this knowledge should be taken into account while the objective is to minimize WIP. In the second section of his study, El-Rayah investigated the effect of imbalance of operation time variabilities on the output rate of the balanced production lines in terms of mean operation times with no storage buffers and he observed that bowl phenomenon holds meaning that assigning stations with more variable operation times to the ends of the line while assigning the ones with less variable operation times to the middle of the line in order to maximize output rate.

Hillier and So[14] studied the effect of coefficient of variation of operation times on the optimal allocation of storage space in production lines. They worked on both μ-balanced and CV-balanced lines with operation times having two-stage Coxian type distributions and considered the throughput as the only performance measure. Their study showed that the optimal buffer allocation depends on the degree of variability in the operation times. Specifically, the results showed that the inverted bowl effect is more pronounced with higher variability in the operation times. Higher variability meaning increasing coefficient of variations generally increases the imbalance in the optimal allocation.

Powell[32] provided a detailed study of the unbalanced three-station serial lines with reliable stations having log-normal processing times with the objective of maximizing throughput. In this study, imbalances in both means and variances were

considered. The study established a rule, Alternation Rule, for buffer allocation in unbalanced lines. It was observed that the optimal sequential allocation of buffers to lines in which one station had a higher mean or variance was to place the first buffer next to the bottleneck, but then to place subsequent buffers alternately at the two available sites. In effect, this rule suggests that a balanced allocation is optimal unless the imbalance in processing times is extreme. Powell also observed that imbalances in means have stronger effect than imbalances in variances, so that when a line is unbalanced in both senses one can buffer the bottleneck with the high mean in preference to that with the high variance unless the imbalances are extreme.

Chow[4] pursued a simple and practical solution for the optimal allocation of buffers with the objective of maximizing throughput. He adopted an approach similar to that used in Anderson and Moodie[2] except that the operation times were not necessarily identical and the number of stations in the line could be arbitrary. At the end, Chow constructed a dynamic programming procedure for buffer design for optimal throughput which generates results that consolidates the bowl-phenomenon.

Yamashina and Okamura[42] dealt with the role of buffer stocks in multi-stage transfer lines by presenting computer simulation results. Lines with unreliable stations were investigated. Breakdown and repair times were assumed to have geometric distributions. It was also assumed that breakdown results in the destruction or damage of the production unit at the affected stage so that the production unit must be removed from the line as scrap. Yamashina and Okamura observed that bowl phenomenon, which was stated for balanced lines with reliable stations, also holds for the balanced lines with unreliable stations. They also obtained the result that uniform buffer storage capacity allocation does not guarantee the optimum allocation even for balanced identical lines, but this postulate may be accepted for balanced identical lines in the sense that the throughput for uniform capacity allocation does not differ very much from the throughput for optimum allocation. It was also shown that an N-stage line should be designed such that the lowest stage production rate occurs in the N^{th} stage, the second lowest in the first stage, the third lowest in the $(N-1)^{th}$ stage, the fourth lowest in the second stage and so on, to maximize the line throughput. This study also demonstrates that the total buffer capacity should be allocated such that the difference between the production rates of the stages on either side of a storage point is minimized

and the production rate of the stages before the storage point is slightly greater than that of the stages following the storage point.

Papadopoulos and Spinellis[23] described a simulated annealing approach for solving the buffer allocation problem with the objective of maximizing throughput for fixed amount of buffer slots in reliable production lines with exponential operation times. Performance of the simulated annealing approach was evaluated by comparing the results of it with the results of complete enumeration whenever practical for short lines and the results of the reduced enumeration which is widely used in literature for the cases of longer lines. Obtained throughput rates by the simulated annealing approach were quite close to the solutions obtained by complete and reduced enumerations. However, evaluated configurations in simulated annealing approach nearly did not change while asymptotically increasing in complete enumeration and reduced enumeration with respect to the increase in the total buffer slots that will be allocated among the buffer locations. For this reason, simulated annealing approach is superior over both enumerative techniques for the lines with large total buffer slots.

Papadopoulos and Spinellis[24] broaden their research[23] by also taking genetic algorithm into account near simulated annealing and obtained interesting results. Genetic algorithm showed similar properties to simulated annealing. However, it gave slightly worse throughput rates than simulated annealing approach with less evaluative configurations where the difference between the throughput rates decreased with the increasing total buffer capacity and line length. The most interesting result that makes genetic algorithm superior to simulated annealing was that the number of evaluative configurations for simulated annealing is increasing linearly but with significantly higher rate than the case of genetic algorithm with respect to the increase in the number of stations in the line.

Hillier et.al.[16] investigated the problem of the optimal allocation of fixed to total buffer capacity for maximizing the throughput of the whole line. They used enumeration on balanced lines with identical exponential service times. Their conclusion was that storage bowl phenomenon holds meaning that interior buffer locations are given preferential treatment (more buffer slots) over the end buffer locations. The other key conclusion of this study was the hypothesis that, when the total amount of storage space also is a decision variable, the overall optimal solution

commonly follows a storage bowl phenomenon whereby the allocation of buffer storage space fits an inverted buffer pattern meaning that optimal allocation would have one additional storage space at each of the internal buffer locations.

Hillier[13] investigated the hypothesis in Hillier et al.[16] with a simple cost model including a linear revenue function and a linear cost per buffer space. The objective was to maximize profit with the total buffer space being decision variable in this study. Hillier worked with balanced and unbalanced four- and five-stations production lines with a single bottleneck in terms of mean processing times. Exponential, Erlang-2 and Erlang-4 processing times were used. Hillier observed that inverted bowl phenomenon was typically optimal for balanced lines but shape became more and more pronounced with larger numbers of buffer spaces. However, in unbalanced lines the buffer space pattern deviates from the bowl pattern by reducing the number of buffer spaces in buffer locations that are not adjacent to the bottleneck station. Also it was stated that the processing time variability measured by coefficient of variation was shown to have very little impact on the pattern of buffer space allocation while the total number of buffer spaces was significantly affected by (being roughly proportional) coefficient of variation of processing times.

Ho et.al.[17] presented a design algorithm based on the gradient vector of the throughput with respect to the buffer sizes, and aiming to maximize throughput via allocation of fixed amount of buffer capacity in transfer lines. They studied the effect of allocating an additional buffer space at a certain location along the line and predicted the improvement in the production rate. Proceedingly, they introduced simulation-based gradient algorithm which solves the buffer allocation problem for unreliable lines having Markovian property effectively.

Gurkan[11] used simulation-based optimization, sample path optimization, to find the optimal buffer allocation in serial production lines where machines were subject to random breakdowns and repairs in contrast to deterministic operation times. Gurkan's objective was to maximize throughput with given total buffer capacity but she used fluid-type single product instead of discrete-type. Gurkan decided to work with continuous type production line instead of discrete-type since continuous type line simulation are substantially faster than discrete type line simulations meaning considerable increase in computational efficiency, the approximations of discrete

product transfer lines via continuous product transfer lines are quite accurate and she interested in optimizing systems of large size. Obtained results showed that her method performed quite well even for very long lines.

Papadopoulos and Vidalis[26] dealt with the optimal fixed amount of buffer allocation problem with the objective of maximizing throughput in balanced production lines with reliable workstations having exponential or Erlang-k (k = 2,3,4) processing times. They presented two basic design rules that were extracted for the optimal buffer allocation in these types of lines using enumeration and developed a search technique that gives the optimal buffer allocation very fast.

Papadopoulos and Vidalis[27] also investigated the optimal buffer allocation giving the maximum throughput in short (with 3,4,5,6 and 7 stations) production lines with unreliable stations balanced in mean processing times. Repair and failure times were assumed to be exponential whereas operation times were assumed to have Erlang-k (k = 1,2,4 and 8) distribution. They answered the critical questions such as the effect of the distribution of the service and repair times, the availability of the stations and the repair rates on the optimal buffer allocation and the throughput of these types of lines. Papadopoulos and Vidalis also confirmed the validity of reversibility property for unreliable lines in this work.

Powell and Pyke [34] studied the problem of buffering reliable serial lines with moderate variability and a single bottleneck in terms of processing time for the maximization of throughput. Processing times were assumed to have log-normal distribution. Their analysis showed that bottleneck station drew buffers toward itself, but the optimal allocation was dependent on the location and the severity of the bottleneck, as well as the number of buffers available. It was also observed that relatively large imbalances in mean processing times are required to shift the optimal buffer allocation away from an equal allocation and line length appeared to have a relatively small effect on the optimal allocation with a given bottleneck. Furthermore, in severely unbalanced lines, throughput appeared to be insensitive to the allocation of buffers. Based on these results, Powell and Pyke suggested that equal buffer allocations might be optimal except in severely unbalanced lines.

Jafari and Shanthikumar[19] also aimed to solve the problem of allocation of given total buffer storage with the objective of maximizing throughput subject to local buffer storage constraints (i.e. buffer slots no more than C_i could be assigned to the buffer location B_i) in transfer lines. They worked on the synchronized transfer lines with unreliable stations having geometric up- and down-times. It was also assumed that when station i breaks down, the part being processed by it is either scrapped with probability β_i or it will be completed with probability 1- β_i , at the end of the cycle where the station is repaired. Jafari and Shanthikumar presented a heuristic solution which was based on dynamic programming and an approximate procedure to compute the production rate of the transfer line and which was producing quite reasonable results.

It is worth to conclude this chapter with summary of researches on the optimal buffer allocation in the literature. Below table named Table 3.1 gives the related researches with their objectives and types of lines on which they focus on briefly.

AUTHOR	TYPES OF LINES	OBJECTIVE	THROUGHPUT ESTIMATION METHOD
Conway et al.[5]	Balanced, Unbalanced	Analyzing the effect of buffers on serial lines	Simulation
Anderson and Moodie[2]	Balanced	Analyzing the optimal production cost modeled	Simulation
Helber[12]	Several examples of flow lines	Maximizing net present value of the investment	Simulation
Altiok and Stidham[1].	Unreliable	Maximizing the total profit	Analytical approximation
Seong et al.[36]	Unreliable	Maximizing the total profit	
Andijani and Anwarul[3]	Balanced Reliable	Maximizing throughput, minimizing WIP and minimizing time in system	Simulation
So[38]	Balanced and Unbalanced Reliable	Minimizing WIP	Exact analytical solutions(Markovian)
Papadopoulos and Vidalis[28]	Balanced Reliable	Minimizing WIP	Exact analytical solutions(Markovian)
Kim and Lee[20]	Unreliable	Minimizing WIP	Analytical approximation
Papadopoulos and Vouros[29]	Balanced, Unbalanced	Minimizing WIP	Exact analytical solutions(Markovian), Simulation
Papadopoulos and Vouros[30]	Balanced and Unbalanced Unreliable	Maximizing throughput	Exact analytical solutions(Markovian), Simulation
Hillier and So[15]	Unreliable	Analyzing the effect of breakdowns and buffers on the efficiency of line	Exact analytical solutions(Markovian)
Yamashita and Altiok[41]	Balanced, Unbalanced	Minimizing total buffer slots for desired throughput	Dynamic programming algorithm
Lutz et. al.[21]	Several examples of flow lines	Minimizing total buffer slots for desired throughput	Simulation
Park[31]	Balanced Unreliable	Minimizing total buffer slots for desired throughput	Analytical approximation
Gershwin and Schor [9]	Unreliable	Minimizing total buffer slots for desired throughput	Analytical approximation
Sheskin[37]	Unreliable	Maximizing throughput	Analytical approximation
El-Rayah[6]	Balanced	Analyzing the effect of unequal allocation of buffers on throughput and WIP	Simulation
Hillier and So[14]	Balanced	Analyzing the effect of CV of operation times on OBA of buffer	Exact analytical solutions(Markovian)
Powell[32]	Unbalanced Reliable	Maximizing throughput	Simulation
Chow[4]	Balanced, Unbalanced	Maximizing throughput	Analytical approximation, Simulation
Yamashina and Okamura[42]	Unreliable	Maximizing throughput	Simulation
Papadopoulos and Spinellis[23],[24]	Reliable	Maximizing throughput	Analytical approximation
Hillier et.al.[16]	Balanced	Maximizing throughput	Exact analytical solutions(Markovian)
Hillier[13]	Balanced, Unbalanced	Maximizing profit	Exact analytical solutions(Markovian)
Ho et.al.[17]	Unreliable	Maximizing throughput	Simulation
Gurkan[11]	Unreliable	Maximizing throughput	Simulation
Papadopoulos and Vidalis[26]	Balanced Reliable	Maximizing throughput	Exact analytical solutions(Markovian)
Papadopoulos and Vidalis[27]	Balanced Unreliable	Maximizing throughput	Exact analytical solutions(Markovian)
Powell and Pyke [34]	Reliable	Maximizing throughput	Simulation
Jafari and Shanthikumar[19]	Unreliable	Maximizing throughput	Analytical approximation

Table 3. 1. *Summary of Literature Survey*

Chapter 4

TWO RELATED ALGORITHMS

Most of the studies, reviewed in the previous chapter, about the optimal buffer allocation in production lines with the objective of maximizing throughput do not solve the problem directly. Instead, some generalizations and intuitive ideas about the characteristic of the optimal buffer allocation or the effects of some parameters (i.e. repair rate, failure rate etc.) on it are introduced. On the other hand, the ones that solve the optimal buffer allocation problem are not applicable to all types of production lines. These types of studies focus on the specific production lines (i.e. balanced lines) or production lines with special features such as reliable machines etc. However, on the contrary to these studies, Seong et.al.[35] and Powell and Harris [33] introduced new heuristic algorithms applicable to all types of production lines.

4. 1. Standard and Non-Standard Exchange Vector Algorithms (SEVA and Non-SEVA)

Seong et al.[35] worked on unbalanced lines with unreliable machines having exponential failure and repair times whereas operation times are deterministic or exponential with different rates. They focused on the optimal buffer allocation problem (OBAP) with the objective of maximizing throughput with the concavity assumption of

objective function. In their study, OBAP, a non-linear integer-programming problem, is presented as below mathematical structure;

$$\text{OBAP:} \quad \max_{\underline{K}} E(\underline{K})$$

$$\text{s.t.} \quad e^T \underline{K} = C$$

$$K_j \text{ is a non-negative integer}$$

where $E(\underline{K})$ is the throughput with $\underline{K}$,
e is a unit column vector,
$\underline{K}$ is a buffer allocation (= (K_1 , …….. , $K_{(N-1)}$)^T), and
C is the fixed total buffer capacity available.

Two different versions of the heuristic algorithm for solving OBAP based on the idea of the *local search* are presented in this study. Namely, first of all, it is needed to define a specific neighborhood with respect to a given solution. The best solution in this defined neighborhood is determined and becomes the next solution. This process is repeated until no better solution is found. The process of defining the specific neighborhood is called the *line segment selection* and the process of finding and moving to the best solution in the neighborhood is called the *point search*.

The line segment selection yields a line segment L which is specified by two integer vectors $\underline{L}^1$, $\underline{L}^2$ and two integer parameters θ_1 and θ_2 ($\theta_1 < \theta_2$) as below:

$$L (\underline{L}^1, \underline{L}^2 , \theta_1 , \theta_2) = \{ \underline{L} \mid \underline{L} = \underline{L}^1 + \theta \underline{L}^2 , \theta = \theta_1 , \theta_1 + 1, \dots , \theta_2 \}$$

where $\underline{L}^1$, $\underline{L}^2$, θ_1 and θ_2 are selected in such a way that all points in the set L ($\underline{L}^1$, $\underline{L}^2$, θ_1 , θ_2) are within the feasible region.

In the point search, an optimization problem given below is solved:

$$\max E(\underline{L})$$

$$\text{s.t.} \quad \underline{L} \text{ in } L (\underline{L}^1, \underline{L}^2 , \theta_1 , \theta_2)$$

which is denoted by **PS**(E: $\underline{L}^1$, $\underline{L}^2$, θ_1 , θ_2).

At this point, it is worthwhile to explain the point search and the line segment selection in more detail:

The point search is a process of finding the best integer solution among a set of integer solutions defined on a straight line. In **PS**(E: $\underline{L}^1$, $\underline{L}^2$, θ_1 , θ_2), there are (θ_2 - θ_1 + 1) integer vector points in L. If (θ_2 - θ_1 + 1) is less than or equal to 4, objective function is evaluated at each point and the optimum solution can be obtained. However, if (θ_2 - θ_1 + 1) is greater than 4, solving this problem becomes equivalent to finding an interval containing 4 consecutive integer vector points defined by an integer value θ^* satisfying the following conditions:

$$E\,(\,\underline{L}^1 + \theta^*\underline{L}^2\,) \;\le\; E\,(\,\underline{L}^1 + (\theta^* + 1)\underline{L}^2\,)$$

$$E\,(\,\underline{L}^1 + (\theta^* + 2)\underline{L}^2\,) \;\ge\; E\,(\,\underline{L}^1 + (\theta^* + 3)\underline{L}^2\,)$$

Such an interval can be found in O(log M) time where M (= θ_2 - θ_1 + 1) is the number of integer vector points on the line segment L by using modified bisecting method. Optimum solution is among the one ($\underline{L}^1 + (\theta^* + 1)\underline{L}^2$) , ($\underline{L}^1 + (\theta^* + 2)\underline{L}^2$) with higher objective value of $E(\underline{L})$.

Throughout the algorithm, since the sum of the components of each solution is equal due to the fixed amount of total given buffer slots, moving from one solution to the other can be considered as the movement along an integer directional vector $\underline{h}$ whose entries sum up to zero meaning that $e^T\underline{h} = 0$. Such vector is defined as “exchange vector”. If a certain exchange vector yields the better production rate, it is called “improving exchange vector”. Hence, the line segment selection can be thought of as a process of choosing a line segment along an improving exchange vector.

Two propositions are presented below that are necessary to develop heuristic algorithms, which differ only in the line segment selection procedure and that are based on the interesting properties of the feasible region K which is given as

$$\mathrm{K} = \{\underline{K} \mid e^T\underline{K} = C,\ \underline{K} \ge \underline{0}\ ,\ \underline{K} \in R^{(\mathrm{N}-1)},\ K_j \text{ integer for all } j = 1, \ldots, (\mathrm{N}-1)\}.$$

***Proposition I*:** *For an arbitrary pair of points* $\underline{K}^1$ *and* $\underline{K}^2$ *in the set* **K**, $\underline{K}^2 - \underline{K}^1$ *can be represented as a unique integer linear combination of vectors* $\{\underline{X}^1, \ldots, \underline{X}^{(\mathrm{N}-2)}\}$ *defined as*

$$X_j^i = \begin{cases} 1 & \text{if } j = i \\ -1 & \text{if } j = i + 1 \\ 0 & \text{otherwise} \end{cases} \qquad \text{for all } i = 1, \ldots, (N-2).$$

Proposition II: *The vector* $\underline{Z}$ *satisfies* $e^T\underline{Z} = 0$, *if and only if* $\underline{Z}$ *is a linear combination of vectors* $\{\underline{X}^1, \ldots, \underline{X}^{(N-2)}\}$.

$\underline{X}^i$ is called "standard exchange vector" since it represents an exchange between two adjacent buffers, i.e. i^{th} and $(i+1)^{th}$ buffers.

4. 1. 1. Standard Exchange Vector Algorithm (SEVA)

Proposition I gives the basis for setting up a line segment selection procedure used for developing the first algorithm, Standard Exchange Vector Algorithm (SEVA). The basic derivation from Proposition I is the simple fact that all possible exchanges among the buffers can be represented as a unique linear combination of standard exchange vectors. In other words, any exchange among buffer allocations can be achieved by a set of exchanges between adjacent pairs of buffers. This idea is used for developing SEVA.

In the Figure 4.1, it can be seen how SEVA proceeds for 4-machine production line with three buffer locations where totally seven buffer slots will be allocated. The initial solution is $\underline{K}^{0,0}$ (= $\underline{K}^0$). At this point, two point search procedures are performed, generating $\underline{K}^{0,1}$ and $\underline{K}^{0,2}$. Then $\underline{K}^{0,2}$ is assigned to $\underline{K}^1$(= $\underline{K}^{1,0}$) and again two more point search procedure is applied, yielding $\underline{K}^{1,1}$ and $\underline{K}^{1,2}$ where $\underline{K}^{1,2}$ is set to the third point $\underline{K}^2$ (= $\underline{K}^{2,0}$) and the algorithm proceeds.

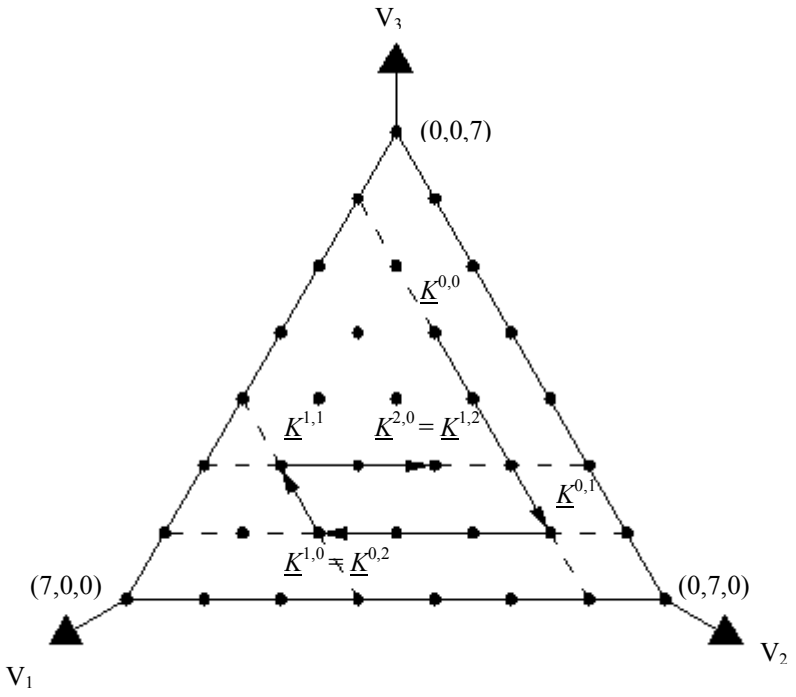

Figure 4. 1. *The illustration of SEVA*

SEVA can be summarized as follows;

Step 0: (Initialization) Set m = 0.

Choose an initial feasible allocation $\underline{K}^0$, which is in the feasible region **K**.

Step 1**:** Set $\underline{K}^{m,0} = \underline{K}^m$.

Step 2**:** For i = 1, … , (N-2) , set $\underline{K}^{m,i}$ to be an optimal solution to

PS(E: $\underline{K}^{m,(i-1)}$, $\underline{X}^i$, θ^i_1 , θ^i_2)

where $\theta^i_1 = -K_i^{m,(i-1)}$ and $\theta^i_2 = K_{(i+1)}^{m,(i-1)}$.

Step 3**:** Set $\underline{K}^{(m+1)} = \underline{K}^{m,(N-2)}$.

Step 4**:** (Termination)

If $| E(\underline{K}^{(m+1)}) - E(\underline{K}^m) | < \varepsilon$ (ε is set to 10^{-6} in the applications), then STOP.

Otherwise set m = m + 1 and go to *Step 1*.

4. 1. 2. Non-Standard Exchange Vector Algorithm (Non- SEVA)

The number of elements of the feasible region **K** becomes huge with the dramatic increase in the total fixed amount of buffer slots *C* and the number of machines N. In such cases, SEVA might have to go through too many point search procedures and each point search procedure can be slowed down significantly due to the large number of integer vector points on each selected line segment. Based on the observation that the selection of a non-standard exchange vector pointing toward the region with better solutions can improve the efficiency of the algorithm, second algorithm called Non-Standard Exchange Vector Algorithm (Non-SEVA) is developed.

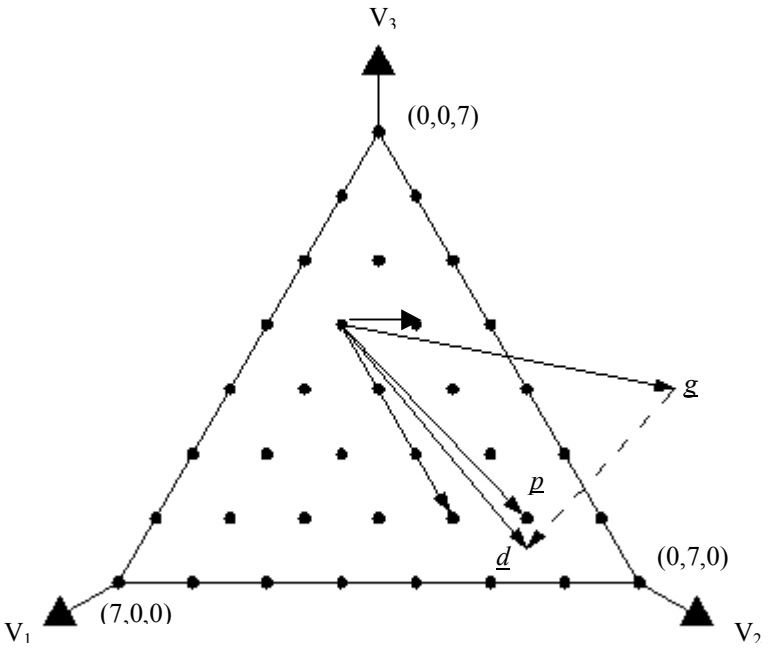

Figure 4. 2. *The procedure how to obtain non-standard integer exchange vector approximating the gradient vector in Non-SEVA*

Proposition II, which is the consequence of Proposition I, is the basis for Non-SEVA. Proposition II implies a simple fact, that an arbitrary exchange vector can be represented as a linear combination of the standard exchange vectors, which is used to develop heuristic procedure for finding a good non-standard integer exchange vector approximating the gradient of the throughput function.

Non- SEVA starts with a point in the feasible region **K** as SEVA does and improving non-standard exchange vector is obtained which is the approximation of the gradient of the throughput. However, since the objective function is not differentiable, pseudo-gradient $\underline{g}$ is obtained which is the approximation of the gradient of the throughput function first by finite differencing:

$$\underline{g}(\underline{K}^m) = (g(K_1^m),....,g(K_{(N-2)}^m))^T$$

$$g(K_j^m) = \frac{E(\ldots, K_{(j-1)}^m, K_j^m + 1, K_{(j+1)}^m, \ldots) - E(\ldots, K_{(j-1)}^m, K_j^m - 1, K_{(j+1)}^m, \ldots)}{(K_j^m + 1) - (K_j^m - 1)}$$

$$= \frac{E(\ldots, K_{(j-1)}^m, K_j^m + 1, K_{(j+1)}^m, \ldots) - E(\ldots, K_{(j-1)}^m, K_j^m - 1, K_{(j+1)}^m, \ldots)}{2}$$

The projection of pseudo-gradient $\underline{g}$ on the hyper plane $e^T\underline{Z} = 0$ is a non-standard exchange vector (not necessarily an integer vector) satisfying $e^T\underline{d} = 0$ and it is given as

$$\underline{d} = (I - e(e^Te)^{-1}e^T)\,\underline{g}. \qquad (*)$$

Due to Proposition II, $\underline{d}$ can be represented as a linear combination of the standard exchange vectors satisfying

$\widetilde{\underline{d}} = S * \underline{d} = \alpha_1 \underline{X}^1 + \alpha_2 \underline{X}^2 + \ldots + \alpha_{(N-2)} \underline{X}^{(N-2)}$, where S is the scale factor.

By rounding off α_i's to get γ_i's, we get an integer vector $\underline{p}$ approximating the pseudo-gradient vector $\underline{g}$:

$$\underline{p} = \gamma_1 \underline{X}^1 + \gamma_2 \underline{X}^2 + \ldots + \gamma_{(N-2)} \underline{X}^{(N-2)}$$

The illustration of how Non-SEVA obtains a non-standard integer exchange vector can be seen in Figure 4.2.

Two different procedures for the round-off are developed. One is for making "big" steps and the other is for making "small" steps.

In the big step round-off procedure, γ_i's are defined to be as the following:

For a given allocation $\underline{K}$, calculate $\underline{d}$ using the equation (*).
Let d_j be the j[th] entry of $\underline{d}$:

$$d_{\min} = \min_{d_j \neq 0,\, j=1,\ldots,(N-1)} \{|d_j|\}$$

$$\tilde{d} = \frac{1}{d_{\min}} \underline{d} \qquad (**)$$

$$\alpha_j = \sum_{k=1}^{j} \tilde{d}_k \quad \text{for all j = 1, … ,(N-2)}$$

Finally, γ_i's are determined by rounding off α_i's.

In the small step round-off procedure, γ_i's are defined to be as the following:

For a given allocation $\underline{K}$, calculate $\underline{d}$ using the equation (*).
Let d_j be the j[th] entry of $\underline{d}$:

$$d_{\max} = \max_{j=1,\ldots,(N-1)} \{|d_j|\}$$

$$\tilde{d} = \frac{1}{d_{\max}} \underline{d} \qquad (***)$$

$$\alpha_j = \sum_{k=1}^{j} \tilde{d}_k \quad \text{for all j = 1, … ,(N-2)}$$

Finally, γ_i's are determined by rounding off α_i's.

While Non-SEVA yields a significant improvement, the point search procedure along the non-standard exchange vectors are performed making "big" steps. However, if the reverse is the case meaning that Non-SEVA does not yield a significant improvement, making "small" steps is invoked during the point search along the non-standard exchange vectors.

The systematic representation of Non-SEVA is as follows:

Step 0: (Initialization)

Set m=0. Choose an initial feasible allocation $\underline{K}^m$, which is in feasible region **K**.

Step 1: (Big Step)

Obtain a non-standard exchange vector $\underline{p}$ from $\underline{K}^m$ by using equation (**).

If $\underline{p} = \underline{0}$ then go to *Step 3*.

Otherwise, let $\underline{K}^{m+1}$ be the optimal solution to **PS**(E: $\underline{K}^m$, $\underline{p}$, θ , θ_2)

where

$$\theta_2 = \min_{j=1,\ldots,(N-1)} \{\hat{p}_j\}$$

and

$$\hat{p}_j = \begin{cases} \dfrac{C - K_j^m}{p_j} & \text{if} \quad p_j > 0 \\ 1 - \dfrac{K_j^m}{p_j} & \text{if} \quad p_j < 0 \\ \infty & \text{if} \quad p_j = 0. \end{cases}$$

Step 2: If $| E(\underline{K}^{(m+1)}) - E(\underline{K}^m) | < \delta$ (δ is set to 10^{-4} in the applications) then go to *Step 3*.

Otherwise, set m = m + 1 and go to *Step 1*.

Step 3: (Small Step)

Obtain a non-standard exchange vector $\underline{p}$ from $\underline{K}^m$ by using equation (***).

If $\underline{p} = \underline{0}$ then stop here. $\underline{K}^m$ is optimal solution.

Otherwise, let $\underline{K}^{m+1}$ be the optimal solution to **PS**(E: $\underline{K}^m$, $\underline{p}$, 0 , θ_2) where θ_2 is defined as in *Step 1*.

Step 4: (Termination)

If | E ($\underline{K}^{(m+1)}$) - E ($\underline{K}^{m}$) | < ε (ε is set to 10^{-6} in the applications) then there exists no improvement and $\underline{K}^{m}$ is optimal solution. Stop here.

Otherwise set m = m + 1 and go to *Step 1*.

Same balanced initial allocation procedure which is proposed by Hillier and So[14] is used for both SEVA and Non-SEVA in the study of Seong et.al[35]. According to this procedure, the initial allocation, $\underline{K}^{0}$, is set as follows:

$$K_j^0 = \left\lfloor \frac{C}{(N-2)} \right\rfloor \quad \text{for} \quad j = 1, \ldots, (N-2)$$

$$K_{(N-1)}^0 = C - \sum_{j=1}^{N-2} K_j^0 \qquad \text{where } \lfloor x \rfloor \text{ is the largest integer that does not exceed } x.$$

4. 2. Simple Search Algorithm (SSA)

Powell and Harris [33] developed an efficient simple search algorithm for determining the optimal allocation of a fixed amount of buffer capacity giving the maximum throughput in both balanced and unbalanced serial production lines with reliable stations having log-normal processing times. Simulation was used to obtain the throughput for every allocation considered in the algorithm. Simple Search Algorithm is based on two important observations grasped in the execution of Non-SEVA. First observation is that information on the throughput gradient at any point in the feasible region is tedious to determine since the central difference approximation to the gradient requires two simulations for each of the (N-1) buffer locations. Second one is that the estimated gradient may not suggest a useful search direction either because it leads out of the feasible region or because its projection onto the feasible region is not itself an integer vector, so information is lost in approximating the projected gradient with an integer vector. For these reasons, a simple search procedure that maintains, at each stage, a collection of feasible buffer allocations that are sorted in order of throughput values is introduced. The search direction is determined by moving from the point with the lowest throughput in the current candidates to the one with the highest throughput.

The Simple Search Algorithm (SSA) has its origins in the sequential search procedures given by Spendley and Hext [40] and Nelder and Mead [22]. The Spendley-Hext algorithm starts with a set of candidate solutions that form a regular simplex. Then, this algorithm identifies a search direction by moving from the centre of the simplex out through the face opposite the worst candidate solution. A new candidate called *reflection* is identified in this search direction while the old worst solution is discarded and the procedure starts again. The shape of the simplex does not change from stage to stage, so it may move slowly even when the gradient is steep. In the Nelder-Mead algorithm, the centroid of all solutions in the simplex except the worst is determined and the search direction is the one from the worst solution through the centroid and out beyond the simplex. However, in contrast to Spendley-Hext algorithm, the algorithm introduced by Nelder and Mead accelerates when the gradient is steep and decelerates when it flattens out. While adapting these algorithms to the Simple Search Algorithm, two problems are faced: to ensure that the search can move quickly when the current candidate allocations are far from the optimal and to ensure that the new candidate allocation that is determined from the search direction is feasible meaning that it is an integer vector with entries sum up to total fixed number of buffer slots.

Firstly, an initial candidate allocation $K = (K_1 , K_2 , \dots , K_{(N-1)})$ is selected. This selection is done based on the studies of Powell[32] and Powell and Pyke[34]. This initial allocation will be balanced or as close to balanced as possible since these two studies show that balanced allocations tend to be optimal except for highly unbalanced lines. Then, (N-2) additional allocations from the closest neighbours to K_1 are generated by transferring one unit of buffer slot from the buffer location with the largest capacity to the each of other buffer locations successively to form the initial simplex. These candidates are sorted from the best to the worst according to their throughput values estimated via simulation. In order to find the search direction, the Spendley-Hext reflection procedure is adapted and, after some experimentation, a reflection procedure that computes the difference between the double of the buffer allocation vector with the best throughput and the buffer allocation vector with the worst throughput is introduced. By this way the second one of the pre-stated problems faced during the adaptation is overcome. The resulting allocation is always an integer vector with entries sum up to the total fixed number of buffer slots and can be expected to lie in the

direction of improving throughput. To make these more understandable, let the vector (0,0,0,0,5) be the initial buffer allocation for six station production line with the 5 buffer slots. Then the generated neighbours that form the initial simplex with the initial buffer vector (0,0,0,0,5) become (1,0,0,0,4), (0,1,0,0,4), (0,0,1,0,4), (0,0,0,1,4). Say that the best allocation is (0,1,0,0,4) and the worst is (0,0,0,0,5). By using these, new allocation, the reflection point, is obtained as follows:

$$2*(0,1,0,0,4) - (0,0,0,0,5) = (0,2,0,0,3)$$

If the estimated throughput of the new allocation is better than the current worst, it is replaced with the worst allocation and the search procedure begins again on the new simplex. If the other is the case meaning that the estimated throughput of the new allocation is worse than the current worst allocation, the search is restarted by generating an initial simplex via the same method (transferring one buffer slot from the buffer location with highest capacity to each of the other buffer locations successively) around the best of the current allocations.

The simplex usually grows as it proceeds, in the sense that new allocations are farther and farther away from the existing allocations. This feature of the simplex solves the first pre-stated problem during the adaptation procedure since it allows the algorithm to accelerate when a good direction is identified.

The reflection may be infeasible. In other words, the reflection may have negative entries. This is undesirable situation since any buffer location cannot have negative capacity. If this is the case, reflection is produced from the second worst allocation in the current candidates. If again an infeasible reflection is obtained by the second worst, the third worst allocation is used in the reflection procedure and this goes on until a feasible reflection is achieved. If no feasible reflection is achieved by all candidates, restarting option is employed which is operated in the case of reflection with less throughput than the current worst allocation in the simplex.

An important feature of this algorithm is that the simulation run length was adjusted during the implementation of the algorithm to save simulation run time when high precision in throughput estimates was not needed, and to ensure the adequate precision when it was needed. In the first step, based on some judgment and experience, a minimum (also initial) run length R_{min} is determined in a way that the search never

uses inappropriately short runs and a maximum run length R_{max} is determined in a way that the search will not continue long after an optimal or near optimal allocation is found. In other words, R_{max} is chosen to balance the trade-off between accuracy level and the desire for short runs.

Secondly, in adjusting the simulation run lengths, the *height*, H, of the simplex is computed in order to estimate how close the simplex to the optimal solution at each step of the algorithm by

$$H = (P_{best} - P_{worst}) / P_{best} .$$

In the above equation, P_{best} and P_{worst} are the estimated throughput values of the best and the worst of the current candidates respectively.

In the next step, a run length constant k is chosen (empirically a value of 100-200 for k performs well) and run length R is set to

$$R = k / H^2 .$$

The closer the simplex is to the optimum, the smaller the height of the simplex H is expected to be and as a consequence of this the larger the run length R is. However, this calculated R value is not directly used in the next step. Since large errors in throughput estimates may make H either increase or decrease substantially, and thus lead to inappropriately large changes in R, a weighted average of the previous run length and the current value of R is taken at each stage of the algorithm and the larger of this weighted sum and current value R is selected as the current run length. By this way, failing to decline or incline in run length in need of less or more precision respectively is prevented.

Two conditions must hold simultaneously in order to stop the Simple Search Algorithm. These conditions are

1. The current reflection must have a lower throughput value than the worst allocation.
2. Te best allocation must be the initial point in the simplex from which (N-2) neighbours are generated.

Since it is probable for the algorithm to stop at inappropriately short run length, an additional condition of doubling the run length at each iteration until R_{max} is reached is integrated as the third stopping condition.

The Simple Search Algorithm can be summarized as below:

Step 0: Choose an initial candidate. (Balanced or as close to balanced as is practical)

Step 1: Generate (N-2) adjacent candidate solutions from the current best candidate to form a new simplex. (Initial simplex is formed from the initial candidate)

Step 2: Simulate all candidate solutions. (Simulation run length is R_{min} for initial simplex)

Step 3: Sort candidate solutions by their estimated throughput values.

Step 4: Determine the feasible reflection. (Reflection procedure was given before)

If no feasible reflection is found
    Go to the Step 1.
Else
    Simulate reflection point.
    If the throughput of reflection is better than the worst's
        Replace the worst candidate with the reflection.
        Calculate the run length. (Technique was stated before)
        Go to Step 2.
    Else
        If the best candidate is same as the initial candidate
            If the run length is less than R_{max} .
                Double run length.
                Go to the Step 2.
            Else
                Stop.
                Initial vertex in the final simplex is the optimum.
            Else
            Go to Step 1.

Chapter 5

LINE BALANCING ALGORITHM (LIBA)

5. 1. Introduction

Before introducing our algorithm for solving OBAP, it is worthwhile to state some important observations related to production lines. Let L be the N-machine production line with (N-1) buffers as depicted in Figure 5.1 that we are trying to allocate the total fixed number of buffer slots among the buffer locations with the objective of maximizing throughput.

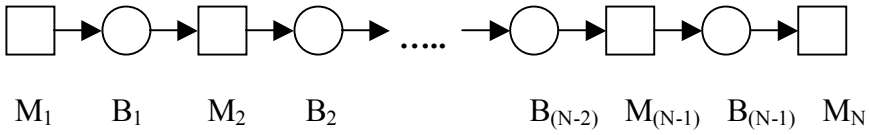

Figure 5. 1. *The N-machine production line L*

Now, let L_1, L_2 be two independently operating sub-lines obtained by dividing the line L into two from the buffer location B_i**.** Then L_1 and L_2 are given as below:

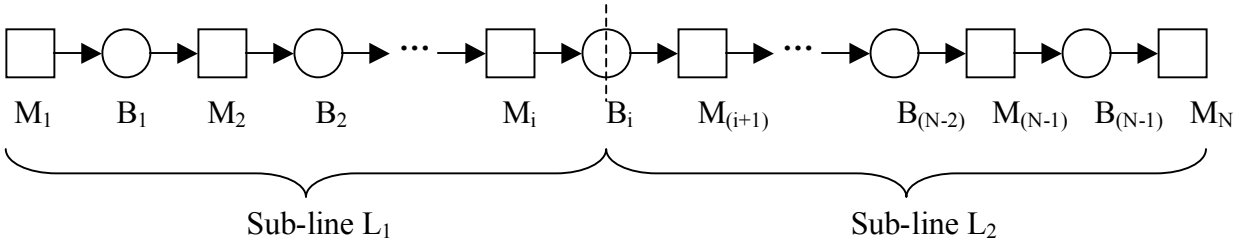

Figure 5. 2. *Two sub-lines L_1 , L_2 obtained by decoupling L from the buffer i*

The output of L_1 is the input of L_2 , so if the throughput of L_1 is less than the throughput of L_2 , L_2 will starve and if the opposite is the case meaning that the throughput of L_1 is more than the throughput of L_2 , L_1 will be blocked where both situations are undesirable. To be able to cope with this unbalance problem between two sides of buffer B_i of the whole line L, we should increase the throughput of the slower sub-line while not decreasing the throughput of the faster sub-line under the value of the throughput of the slower one, since we know in advance that the throughput P of the whole line L is bounded by the minimum of the throughput of these two sub-lines and given as

$P \leq \min(P^1 , P^2)$ where P^1 and P^2 are the production rates of L_1 and L_2 respectively.

The way of achieving this goal is to transfer buffer slots from the buffer locations belonging to faster sub-line to the buffer locations belonging to slower sub-line. Applying this procedure, we may increase the production rate of the slower sub-line with the possibility of decreasing the production rate of the faster one. Hence, we decrease the difference between the production rates of the two sub-lines, which results in obtaining more balanced line around the buffer where two sub-lines are separated. In addition to this, upper limit for the throughput of the whole line is raised to higher value, since the minimum of P^1 and P^2 increases. The idea of separating the whole line from a buffer location and obtaining more balanced line around it by the buffer slot transfer from the faster sub-line to the slower one is defined as "*Buffer Centric Line Balance"*.

By implementing the *Buffer Centric Line Balance* concept to each buffer location consecutively, we expect to obtain a more balanced line with increasing production rate at each step. However, it may not be possible to obtain a production line with exact balance around each buffer location simultaneously, which is an interesting property of production lines. There may most probably be imbalance around any other buffer location even though we achieve an exact balance around any specific one. To make this property more apparent, think of the below line:

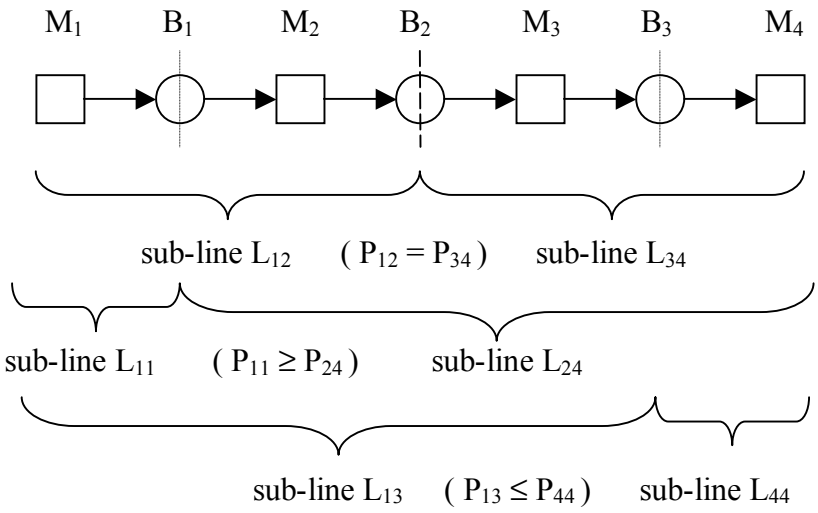

Figure 5. 3. *Illustration of the property of imbalance around any other buffer locations even though the exact balance around any specific one in production lines*

In the above line, although we have an exact balance around the buffer location B_2 ($P_{12} = P_{34}$), we may not reach an exact balance around the other buffer locations B_1 ($P_{11} \geq P_{24}$) and B_3 ($P_{13} \leq P_{44}$) due to the fact that the expectation of decrease in the production rate of the line when adding a new machine to it. Based on this fact, the production rate of the sub-line L_{11} is expected to be greater than the production rate of the sub-line L_{12}($P_{11} \geq P_{12}$) while the production rate of the sub-line L_{24} is expected to be less than the production rate of the sub-line L_{34}($P_{24} \leq P_{34}$). Consequently, the production rate of the sub-line L_{11} is expected to be greater than the production rate of the sub-line L_{24} due to the exact balance around buffer location B_2 ($P_{11} \geq P_{12} = P_{34} \geq P_{24}$). Therefore, exact balance around the buffer location B_2 may most probably produce an imbalance around the buffer location B_1 as well as B_3, which can be shown by the same logic.

Based on this property and the observations introduced, we develop a new algorithm called "*Line Balancing Algorithm (LIBA)*". The logic behind this algorithm is the minimization of the sum of the production rate differences between two sub-lines obtained by dividing the whole line around each buffer location. Thereby, we aim to increase the throughput of the whole line. The correctness of this logic is supported by the Table A.4 given in the Appendix A.4. In Table A.4, total imbalance and throughput value for each feasible allocation for 3-station production line with total fixed number of buffer slots equal to 15 which is studied in Seong et.al.[35]. The way of decreasing the total imbalance is to break the initial line into two sub-lines and to apply the buffer

slot transfer from faster side to slower side that will improve the throughput of the line we focus on. It should be kept in mind that transfer of the buffer slots from the faster side to slower side of the line does not always improve the efficiency. The reason is that to make the whole line more balanced around any arbitrary buffer, will usually increase the imbalance around some other buffers, which may also result in increase in the sum of imbalance in the whole line. Hence, we continue to find the improving transfers during the execution of the algorithm until no more improving transfer condition is reached.

5. 2. The Algorithm

First of all, throughput of the whole line is obtained. Then the line is divided into two sub-lines from the buffer location in the centre of the line. However, if the number of the machines is odd, there will be two central buffer locations since there is even number of buffer locations in the whole line. In this case, the one towards the end of the line, in other words the right one, is selected. This buffer location is called the main division buffer. In the second step, production rates of the two sub-lines are evaluated and the one with higher production rate is determined as the potential buffer slot giver, while the slower one is determined as the potential buffer slot receiver. After determining the potential giver and potential receiver sides of the line, determination of the buffer locations that are receiver and the giver is the order that we follow. While determining these locations, bisection technique is used. This bisection can be visualized as below figure:

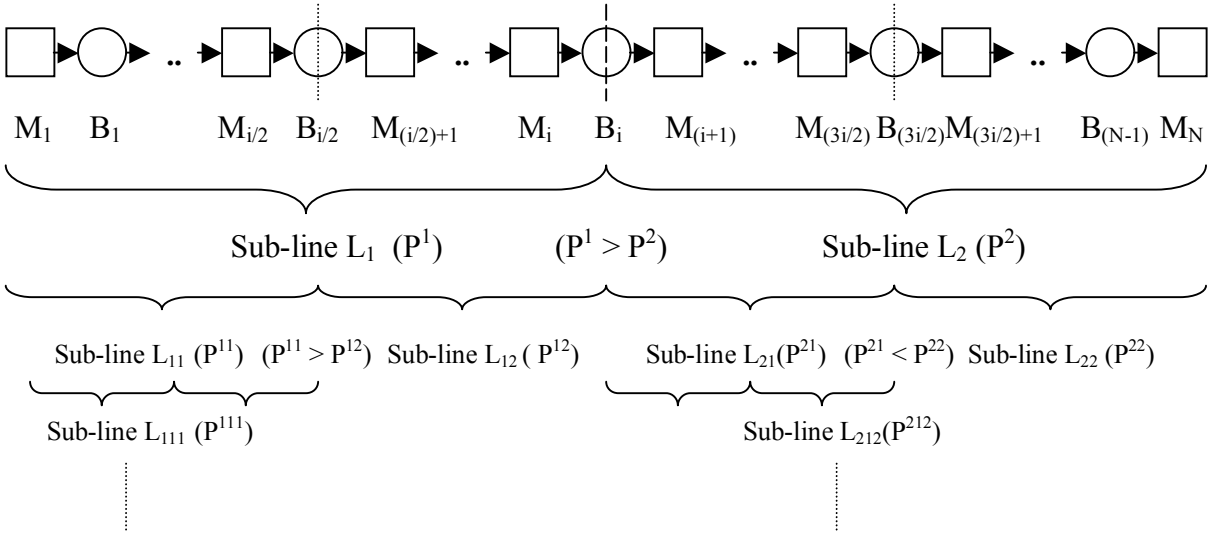

Figure 5. 4. *Bisection procedure for determination of potential giver and receiver*

For determining the potential giver buffer location, the initial sub-line with higher throughput will again be divided into two independently working sub-lines from the central buffer location of the initial sub-line. Among the sub-lines of the initial sub-line, the one with higher rate is again decoupled from its centre and this procedure will go until we obtain two sub-lines with at most one buffer location. Among these final sub-lines, the one with higher rate is potential giver. If this line has two machines, the buffer location between these two machines is the one we are looking for. However, if this line has only one machine, the potential giver buffer location is the final division buffer location, which is just in front of this machine.

Same procedure used while determining the potential giver buffer location is applied to determine the potential receiver buffer location. Decoupling from the center of obtained sub-lines with lower rates continues until reaching the final sub-lines that have at most one buffer location and potential receiver is assigned to the buffer location between the machines if the lower rate final sub-lie has two machines or to the final division buffer location which is just in front of the machine if the lower rate final sub-lie has only one machine.

Buffer slot transfer between these two determined buffer locations has the highest likelihood to improve the efficiency of the whole line since decoupling sequence is based on the observation that we mentioned as the basis of our algorithm. This basis is to decrease the unbalance between two sub-lines obtained. This procedure supports the imbalance reduction between each sub-line pair formed. To see this more clearly, let's take N-machine production line with N equal to 8:

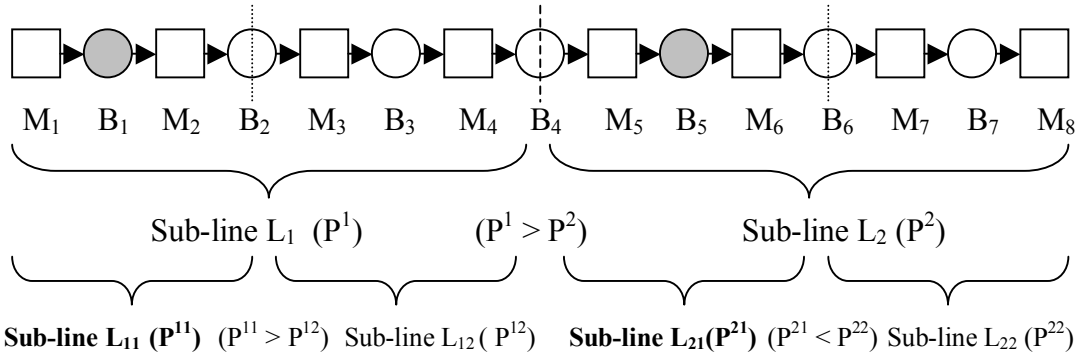

Figure 5. 5. *Bisection of 8-machine production line until its final sub-lines*

As can be seen from a simple example given above in Figure 5.5, shaded buffer locations are potential giver and potential receiver. B_1 is the potential giver and B_5 is the potential receiver. By the buffer slot transfer from B_1 to B_5, we hope to decrease the unbalance between the sub-lines L_{11} and L_{12} for the sub-line L_1 as well as the unbalance between the sub-lines L_{21} and L_{22} for the upper sub-line L_2 since the production rate of the sub-lines L_{11} may decrease while the production rate of the sub-line L_{12} remains the same and the production rate of the sub-lines L_{21} may increase while the production rate of the sub-line L_{22} remains the same. Moreover, the unbalance between the upper sub-lines L_1 and L_2 is expected to be decreased, since the possible decrease in the production rate of L_1 is accompanied with the possible increase in the production rate of the L_2 with this buffer slot transfer. Hence, this transfer option has the highest likelihood to improve the throughput of the whole line due to the expectation of achieving maximum imbalance reduction.

After determining the potential giver and receiver buffer locations, buffer slots are transferred from the potential giver to the potential receiver until no improvement is achieved in the efficiency of the whole line and we restart the algorithm from the same main division buffer. However, if any transfer does not increase the throughput of the whole line, potential receiver is changed to just previous division buffer and it is checked whether there will be increase in the throughput of the whole line. If this transfer, from the initial potential giver to the new potential receiver, improves the rate of the whole line, buffer slots is transferred again until no improvement is achieved. On the other hand, if no improvement in the throughput of the whole line is achieved, the potential receiver is assigned to the next division buffer, which is just before the existing potential receiver. This goes on until the first division point, which is the main division buffer, in the slower part of the line. If there is still no improvement in the throughput of the whole line, the potential giver is changed to the just previous division buffer and potential receiver is assigned to the initial potential receiver buffer location. If, again, no improvement in the efficiency of the whole line is achieved for this potential giver after applying the same potential receiver sequence, the potential giver is changed to the division buffer for the one upper sub-line. This sequence of changing potential giver continues until the first division point, which is the main division buffer, in the faster part of the line.

To make it more understandable, consider the 8-machine production line example again. Transfer sequence will be as the below order until any improvement of overall throughput is reached:

1. B_1 to B_5
2. B_1 to B_6
3. B_1 to B_4
4. B_2 to B_5
5. B_2 to B_6
6. B_2 to B_4
7. B_4 to B_5
8. B_4 to B_6

If there is still no improvement in throughput although all transfer options are checked for that main division buffer (it is B_4 in our example), we change the main division buffer to one left buffer location (B_3 in our case) and restart bisection. If, again, we cannot increase the production rate for this main division buffer, we will change the main division buffer to one right of the first main division buffer. The next main division buffer will be two left of the first one and then two right of it if no improvement is the case. Changing of the main division buffer can be seen as an oscillation around the center of the whole line. The order of the main division points in our 8-machine case given in Figure 5.5 is

$$B_4 \longrightarrow B_3 \longrightarrow B_5 \longrightarrow B_2 \longrightarrow B_6 \longrightarrow B_1 \longrightarrow B_7 .$$

Up to this point, we assume that each sub-line pair has unequal production rates. When opposite situation, meaning that the production rates of formed sub-lines are equal, occurs around the main division buffer, we progress to the next main division buffer. On the other hand if it occurs around any division buffer obtained during bisection process, we stop the bisection at that division point and determine that division buffer as first potential giver if it appears in the faster side of the line or first potential receiver if it occurs in the slower side.

The stopping criterion for this heuristic algorithm is the failure in the improvement of the overall throughput for (N-1) consecutive main division buffer since we return to the same point where we restart the algorithm.

5. 2. 1. Initial Allocation Procedure

Initial allocation may be any feasible point in our feasible region. The feasible region is given below as the one stated in the study of Seong et al. [35];

$$\mathbf{K}=\{\underline{K} \mid e^T\underline{K} = C,\ \underline{K} \geq \underline{0}\ ,\ \underline{K} \in R^{(N-1)},\ K_j \text{ integer for all } j = 1, \ldots, (N-1)\}$$

The number of elements of the above feasible region namely the number of all feasible allocations of C buffer slots among the (N-1) buffer locations in an N-machine serial production line is given by the formula below:

$$\binom{N+C-2}{C} = \frac{(N+C-2)!}{(N-2)!(C)!}$$

In our heuristic algorithm, based on many trials, we have observed that we might reach different final allocations by starting with different initial allocations. We also observed that initial allocation has an effect on the number of iterations (# iterations is equal to the number of N-machine throughput estimation in our case) during the implementation of the algorithm. Therefore, we came to the conclusion that we should try to start with a good initial point, which is close to global optimum, in order to reach this global optimum with less iteration.

After comprehending the importance of efficient initial allocation, we focused on how we succeed in realizing our purpose. Before proceeding to the stage of determination of efficient initial allocation, it may be useful to state some relationships for production lines that we benefit from:

$$\boldsymbol{\lambda_i} = \frac{1}{T_i} \qquad \boldsymbol{\mu_i} = \frac{1}{MTTF_i} \qquad \mathbf{r_i} = \frac{1}{MTTR_i}$$

$$\mathbf{e_i} = \frac{MTTF_i}{(MTTF_i + MTTR_i)} = \frac{r_i}{(\mu_i + r_i)}$$

$$\boldsymbol{\rho_i} = \lambda_i * e_i = \frac{\lambda_i * r_i}{(\mu_i + r_i)}$$

where $\mathbf{T_i}$: average processing time of M_i

$\boldsymbol{\lambda_i}$: processing rate of M_i

$\mathbf{MTTF_i}$: mean time to failure for M_i

$\mathbf{MTTR_i}$: mean time to repair for M_i

$\boldsymbol{\mu_i}$: average failure rate of M_i

$\mathbf{r_i}$: average repair rate of M_i

$\mathbf{e_i}$: efficiency of M_i in isolation

$\boldsymbol{\rho_i}$: production rate of machine M_i in isolation

The *isolated efficiency*, e_i, is the average fraction of time that M_i would be operational if it were operated in isolation (never starved or blocked). This quantity is also seen as the *availability* of M_i. (Note that $e_i = 1$ for reliable machines and $e_i < 1$ for unreliable machines.)

In addition to these relationships, we make use of the set of generalizations given by Freeman [7] in his study. These generalizations can be paraphrased as follows:

1. Avoid extreme buffer allocation, even in highly unbalanced lines.
2. Allocate more buffer capacity to the station with highest mean downtime.
3. Allocate more buffer capacity between a bad and a mediocre station than a bad and a good station.
4. The optimum allocation of buffers to the various potential buffer locations is relatively unaffected by the total number of buffers available.
5. The end of the line is more critical than the front, so if a bad station is located toward the end it should get more of the available buffer capacity than if it is toward the front.

Despite the early date of this study and its limited scope, these rules have been borne out studies except the last one, which violates the reversibility principle.

It can be deduced from the generalization 4 that each buffer location should have capacities whose ratios to each other are constant. In other words, any change in the amount of buffer slots that will be allocated should not affect the capacity ratios of buffer locations to one another. Based on this derivation, as an initial step of the initial allocation procedure, we assign criticalities to each buffer location, that are independent on any change in the amount of buffer slots allocated. In the second step, we determine

the initial capacities of all buffer locations according to their criticalities. By this way, relative capacities of buffer locations will be independent of the total number of buffer slots available meaning that any change in the total number of buffer slots available will not change the ratio of capacities of any of two buffer locations.

During the process of the determination of these criticalities for each buffer location, we use the information that the capacity K_i of any buffer location B_i between two machines M_i and M_{i+1} is inverse proportional with the production rate of these machines in isolation ρ_i and ρ_{i+1}. Therefore, the criticality of any buffer location that will be determined is the monotonically decreasing function of the production rates of adjacent machines in isolation. We have three candidates for the criticality function initially. These are

$$\mathbf{f_1(\rho_i , \rho_{i+1})} = \frac{1}{\min\{\rho_i , \rho_{i+1}\}}$$

$$\mathbf{f_2(\rho_i , \rho_{i+1})} = \frac{1}{\rho_i} + \frac{1}{\rho_{i+1}}$$

$$\mathbf{f_3(\rho_i , \rho_{i+1})} = \frac{1}{\rho_i + \rho_{i+1}} .$$

While selecting the most efficient one among these functions, we took the first three of the above generalizations.

First of all, the first function $f_1(\rho_i , \rho_{i+1}) = 1 / \min\{\rho_i , \rho_{i+1}\}$ fails to conform the third generalization while the others conform. To make this clear, consider the example below:

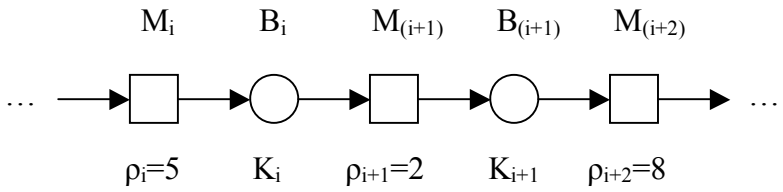

Figure 5. 6. *Illustration of determination of criticality function*

According to the criticality function $f_1(\rho_i , \rho_{i+1}) = 1 / \min\{\rho_i , \rho_{i+1}\}$, both buffer location B_i and $B_{(i+1)}$ will have equal criticalities since

$$\min\{\rho_i=5 \, , \, \rho_{i+1}=2 \} = \min\{ \rho_{i+1}=2 \, , \, \rho_{i+2}=8 \} = 2$$

$$f_1(\rho_i=5, \rho_{i+1}=2) = \frac{1}{\min\{5,2\}} = \frac{1}{2} = f_1(\rho_{i+1}=2, \rho_{i+2}=8) = \frac{1}{\min\{2,8\}}.$$

Hence, our first candidate is eliminated since it contradicts with the third generalization, which favors more buffer slots to B_i than to B_{i+1}. For the second and third candidates, this generalization holds which can be proved by below inequalities:

$$f_2(\rho_i=5,\rho_{i+1}=2) = 1/5 + 1/2 = 7/10 \; > \; f_2(\rho_{i+1}=2,\rho_{i+2}=8) = 1/2 + 1/8 = 5/8$$

$$f_3(\rho_i=5,\rho_{i+1}=2) = \frac{1}{(5+2)} = \frac{1}{7} \; > \; f_3(\rho_{i+1}=2,\rho_{i+2}=8) = \frac{1}{(2+8)} = \frac{1}{10}$$

These remaining candidates conform to the second generalization. However, the second one contradicts with the first generalization. To see this, let's rewrite these functions:

$$\mathbf{f_2(\rho_i \, , \, \rho_{i+1})} = \frac{1}{\rho_i} + \frac{1}{\rho_{i+1}} = \frac{1}{(\lambda_i * e_i)} + \frac{1}{\rho_{i+1}}$$

$$= \frac{1}{\lambda_i * (\frac{MTTF_i}{MTTF_i + MTTR_i})} + \frac{1}{\rho_{i+1}}$$

$$= \frac{(MTTF_i + MTTR_i)}{(\lambda_i * MTTF_i)} + \frac{1}{\rho_{i+1}}$$

$$\mathbf{f_3(\rho_i \, , \, \rho_{i+1})} = \frac{1}{(\rho_i + \rho_{i+1})} = \frac{1}{(\lambda_i * e_i) + \rho_{i+1}}$$

$$= \frac{1}{(\lambda_i * (\frac{MTTF_i}{MTTF_i + MTTR_i})) + \rho_{i+1}}$$

$$= \frac{(MTTF_i + MTTR_i)}{(\lambda_i * MTTF_i) + (\rho_{i+1} * (MTTF_i + MTTR_i))}$$

As mean downtime (Mean Time To Repair) increases production rate in isolation ρ decreases which results in increase in both criticality function candidates. Increase in criticality function means increase in both side buffer location of that machine which complies with the second generalization.

However, as mean downtime (MTTR) goes to infinity, the second candidate f_2 diverges to infinity since it is linear function of mean downtime (MTTR). Increase in mean downtime also means increase in imbalance. Since the second candidate supports the extreme allocation in the sense of high imbalance, it becomes against the first generalization.

On the other hand, as mean downtime (MTTR) goes to infinity the third criticality function candidate f_3 converges, so this candidate is consistent with the first generalization since it does not support extreme allocation despite high imbalance.

Since the third candidate is the only one that conforms to all generalizations, among these three candidates, it is the most efficient one. This derivation is supported by the results we obtained in the cases that we worked on so we select this candidate as the criticality function in our initial allocation procedure.

Based on this derivation, we determine the relative criticality RC_i of any arbitrary buffer location B_i by defining the criticalities cr_i as below:

$$cr_i = f_3(\rho_i , \rho_{i+1}) \quad \rightarrow \quad cr_i = \frac{1}{(\rho_i + \rho_{i+1})}$$

$$RC_i = \frac{cr_i}{cr_1 + cr_2 + \ldots + cr_{(N-1)}}$$

After determining these criticalities, we calculate the amount of buffer slots KI_i that will be allocated initially to the buffer location B_i by multiplying the relative criticality RC_i by the total fixed number of buffer slots allocated *C*:

$KI_i = RC_i * C$ for i = 1,2, … ,(N-1)

However, KI_i's may not be integer. If this is the case, we first assign the largest integer value that is smaller than KI_i's (integer part of KI_i's) as the capacity of each

buffer location. The remaining buffer slots are assigned to the buffer locations according to decimal part of their KI_i values in decreasing order by starting from the highest until the last remaining buffer slot allocation is completed. During the second part of this allocation process pertaining to the remaining buffer slots equality of the decimal parts may be faced. If such an exception arises, buffer location with higher integer part in KI_i value has higher priority. In the situation of exact equality of KI_i values meaning that the equality of both decimal and integer parts is the condition, higher priority is given to the buffer location closer to the centre of the system. Moreover, exact equality of KI_i values and equal closeness to the centre of the line at the same time is the case; the buffer location closer to the end of the line has higher priority for the remaining buffer slot.

5. 2. 2. A Simple Example for Line Balancing Algorithm (LIBA)

In order to understand how LIBA works, we consider the flow line consisting of four machines that are prone to failure. Processing, failure and repair times for each machine have independent exponential distributions. This is the case on which is worked in the study of Seong et.al[35]. Processing, failure and repair rates for each machine are given in the table below:

	Processing rate(λ_i)	Failure rate(μ_i)	Repair rate(r_i)
MC#1	3.7	0.07	0.17
MC#2	1.5	0.11	0.37
MC#3	1.1	0.49	0.78
MC#4	3.0	0.19	0.50

Table 5. 1. *Processing, failure and repair rates for each machine*

From the above table, availabilities and production rates in isolation for all machines are computed:

	Availability (e_i)	Prod.rate in isolation (ρ_i)
MC#1	0.70833	2.62083
MC#2	0.77083	1.15625
MC#3	0.61417	0.67559
MC#4	0.72464	2.17391

Table 5. 2. *Availabilities and production rates in isolation for all machines*

By using production rates in isolation (ρ_i's) obtained in the above table, we determine the initial allocation of buffer slots among the buffer locations. This process is illustrated in the Table 5.3 given below:

	cr_i	KI_i	Integer part of KI_i	**Remaining buffer slots**	**Initial allocation**
Buffer 1	0.264755	2.279239	2		2
Buffer 2	0.545899	4.699577	4	1	5
Buffer 3	0.350938	3.021184	3		3
Total	1.161592	10	9	1	10

Table 5. 3. *Initial buffer allocation where* $cr_i = 1/(\rho_i + \rho_{i+1})$

Even though we showed that last one that we use is the most efficient among the three pre-stated candidates of criticality function, we determine the initial allocation by means of the other two criticality function candidates in order to show that our selected candidate is superior and introduce extra examples to the initial allocation procedure. These procedures are given in Table 5.4 and Table 5.5 respectively:

	cr_i	KI_i	Integer part of KI_i	**Remaining buffer slots**	**Initial allocation**
Buffer 1	0.864865	2.260944	2		2
Buffer 2	1.480186	3.869528	3	1*	4
Buffer 3	1.480186	3.869528	3	1**	4
Total	3.825238	10	8	2	10

Note: ** First Remaining Buffer Slot*

*** Second Remaining Buffer Slot*

Table 5. 4. *Initial buffer allocation where* $cr_i = 1 / min\{\rho_i, \rho_{i+1}\}$

	cr_i	KI_i	Integer part of KI_i	**Remaining buffer slots**	**Initial allocation**
Buffer 1	1.246423	2.253253	2		2
Buffer 2	2.345051	4.239326	4		4
Buffer 3	1.940186	3.507421	3	1	4
Total	5.531661	10	9	1	10

Table 5. 5. *Initial buffer allocation where* $cr_i = (1/\rho_i) + (1/\rho_{i+1})$

We estimate the throughput for each of these initial allocations via simulation. We simulate the system for 45000 parts. Since we want to estimate the steady-state throughput value for any buffer allocation, observations during the warm-up period will have an effect of increasing bias in the estimated throughput value. Therefore, we should decide the length of warm-up period and we should not take the observations pertaining to this period into account during the estimation of the throughput. While predicting the warm-up period, we should achieve two conflicting objectives. The more data we discard belonging to the warm-up period, the less bias we will have in the estimation of throughput. On the other hand, the more data we discard belonging to the warm-up period, the more variability we will have in our estimated throughput value. Hence, we should compromise between reducing bias and increasing variance for our throughput estimation.

During the estimation of the warm-up period we use the simplest and the most practical and the most popular technique named Cumulative Moving Averages Technique and we come to a conclusion of the warm-up period of 5000 parts.

We use Replication–Deletion Technique while obtaining the estimation of throughput and variance of this estimation since it is the best-suited technique for the systems with minimal warm-up period. This technique is also very simple and enables us to direct use of statistical procedures such as constructing confidence intervals for the estimation of throughput values. While estimating the throughput value and its variance, we have two sources of observations: individual observations within each replication and mean of these individual observations for each replication. Although the observations within a given replication are dependent, averages of these observations are independent of each other and it is reasonable to assume that these average values have normal distributions based on the Central Limit Theorem. By means of these results, we can construct confidence interval for the throughput. Let X_i denote the mean of individual observations in i^{th} replication for throughput. We can compute the sample mean X_{bar} and sample variance S^2 of X_i's and X_{bar} from n replications as follows:

$$X_{bar} = \sum_{i=1}^{n} \frac{X_i}{n}$$

$$S^2(X) = \sum_{i=1}^{n} \frac{(X_i - X_{bar})^2}{(n-1)}$$

$$S^2(X_{bar}) = \frac{S^2(X)}{n}$$

Since X_i's are normally distributed, the half-width, h, of the 100(1-α)% CI for throughput is given as

$$h = t_{(n-1),1-(\alpha/2)} * S(X_{bar}).$$

Consequently 100(1-α)% CI for throughput is centered at X_{bar} and it is given as

$$X_{bar} \pm t_{(n-1),1-(\alpha/2)} * S(X_{bar}).$$

At this point, we should decide the number of replications. The smaller the number of replications is, the fewer the amount of data we discard belonging to the warm-up period. However, as n decreases, the degrees of freedom for t- statistics will be smaller, resulting in a larger value for t-statistics and an increasing half-width for the confidence interval. The next chart, named Chart 1, which is the graph of t-statistics versus the degrees of freedom for α = 0.05 confidence level that we used in our cases, shows this trade-off obviously:

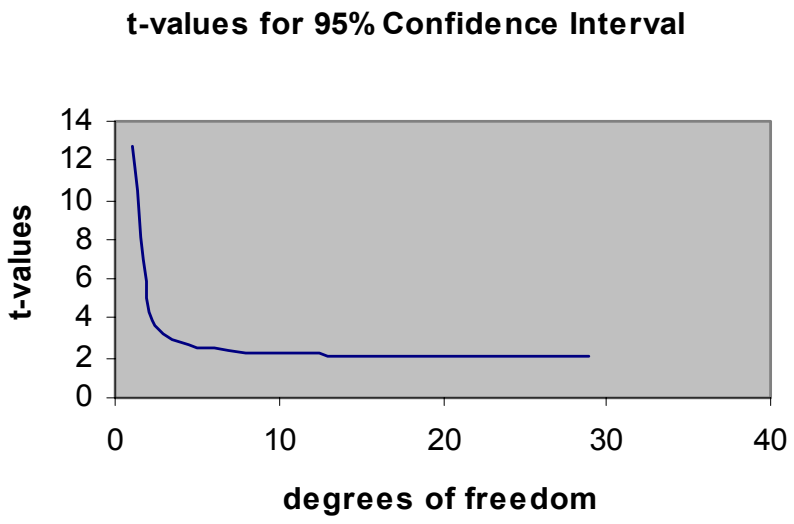

Chart5. 1. *t-statistics versus the degrees of freedom for α = 0.05 confidence level*

As can be easily seen from the above chart, the most reasonable number of replications to use is 10, since improvements in the t-statistics diminish beyond this point. Based on this fact, we choose the number of replications as 10.

After explaining techniques that we use and the reasons why we use these techniques, we return to the throughput estimation for the candidates for the comparison step and we see that the third candidate gives the largest throughput value. These throughput values are given in the below table with 95% confidence levels:

Case	**Average(X_{bar})**	**Half- width(h)**	**(h / X_{bar}) %**	**X_{bar} - h**	**X_{bar} + h**
(2,4,4)	0,6457656	0,002151472	0,33316606	0,643614128	0,64791707
(2,5,3)	0,6490498	0,001997834	0,30780903	0,647051966	0,65104763

Table 5. 6. *Throughput values and related computations for the candidate initials*

To reach more healthy results, we also use *paired-t test* for two allocation alternatives. Below table gives the throughput values for two alternatives and the throughput differences between them for each replication in the simulation:

Replication Number	**Throughput of Allocation (2,5,3)**	**Throughput of Allocation (2,4,4)**	**Throughput Difference(D_i)**
1	0.645809	0.642097	0.003712
2	0.650929	0.647573	0.003356
3	0.647366	0.644907	0.002459
4	0.651672	0.648316	0.003356
5	0.644689	0.640807	0.003882
6	0.651590	0.647887	0.003703
7	0.646329	0.643114	0.003215
8	0.651822	0.649662	0.002160
9	0.648823	0.645130	0.003693
10	0.651469	0.648163	0.003306

Table 5. 7. *Throughput values and differences for two initial allocation alternatives for each replication*

Based on the values given Table 5.7, we obtain the paired-t confidence interval for the throughput differences as below:

Average(X_{bar})	**Half- width(h)**	**(h / X_{bar}) %**	**X_{bar} - h**	**X_{bar} + h**
0.0032842	0.000402048	12.24189878	0.002882	0.003686

The superiority of the third candidate can also be seen from the *paired-t confidence interval* for the throughput differences for each replication. All these findings support our design for initial allocation procedure.

Moreover, it is worth to state that our estimation of throughput is extremely accurate, since the half-width value is very small itself as well as relative to the estimated throughput value. This can be observed from the column representing the half-width and its percent ratio to the estimated throughput. Due to this reason, with contentment, we use the estimated throughput as the exact value of throughput without focusing on the confidence intervals in the cases we study.

During the implementation of LIBA, we compare the throughput values for any allocation with each other in order to decide whether to proceed to the better solution or not. In these steps, to achieve more convenient comparisons of alternative allocations and more accurate results, we use *common random numbers*. This means that random numbers generated for the same operations in the line for each solution will be same. More clearly, consecutive processing, repair and failure times of any machine will be equal to each other for every alternative allocation during the simulation. Consequently, the difference between the alternatives comes only from the capacities assigned to the buffer locations, which enables us to reach healthier comparisons.

Now, let's return to our line with initial allocation of (2,5,3) and throughput $P^*(2,5,3) = 0.6490498$. Execution of the algorithm is given step by step as;

1. Decompose the line into two sub-lines from the buffer location B_2 :

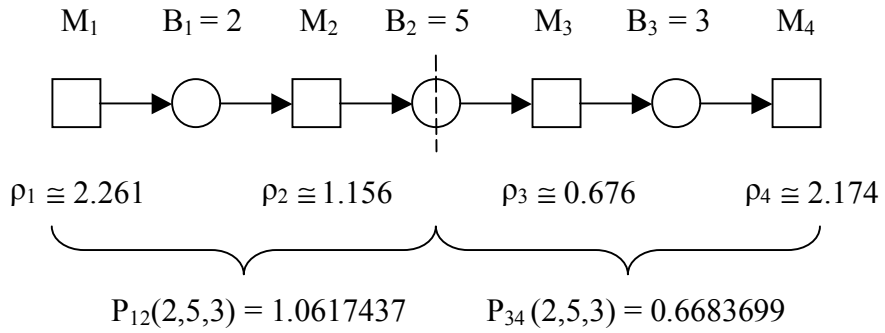

Figure 5. 7. *Initial decomposition of the line during the execution of LIBA*

1.1. $P_{12}(2,5,3) \geq P_{34}(2,5,3)$

1.1.1. Try to transfer from B_1 to B_3.

$P(1,5,4) = 0.649035 < P^*(2,5,3) = 0.6490498$ not an improving direction.

1.1.2. Try to transfer from B_1 to B_2.

$P(1,6,3) = 0.651497 > P^*(2,5,3) = 0.6490498$ improving direction, continue to transfer from B_1 to B_2.

$P(0,7,3) = 0.653132$ improving direction.

Cannot continue to transfer from B_1 to B_2 since no buffer slot in B_1.

New point is (0,7,3) with $P^*(0,7,3) = 0.653132$.

2. Decompose the line from the buffer location B_2:

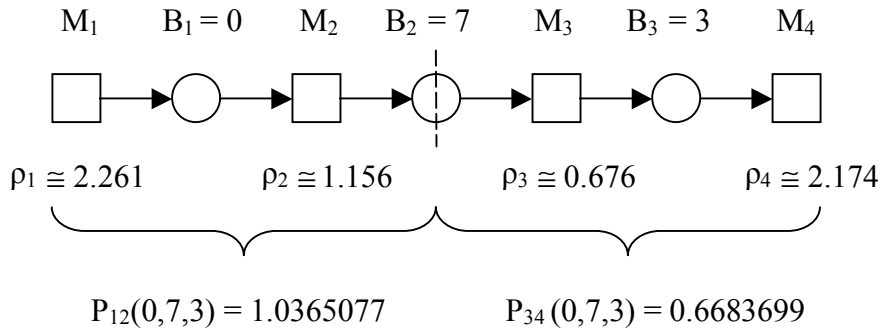

Figure 5. 8. *Restart of LIBA from the second buffer location B_2*

2.1. $P_{12}(0,7,3) \geq P_{34}(0,7,3)$

2.1.1. Transfer from B_1 to B_3 is infeasible.

2.1.2. Transfer from B_1 to B_2 is infeasible.

2.1.3. Try to transfer from B_2 to B_3.

$P(0,6,4) = 0.651336 < P^*(0,7,3) = 0.653132$ not an improving direction.

No improvement for the main buffer division point B_2.

3. Decompose the line from the main division buffer location B_1:

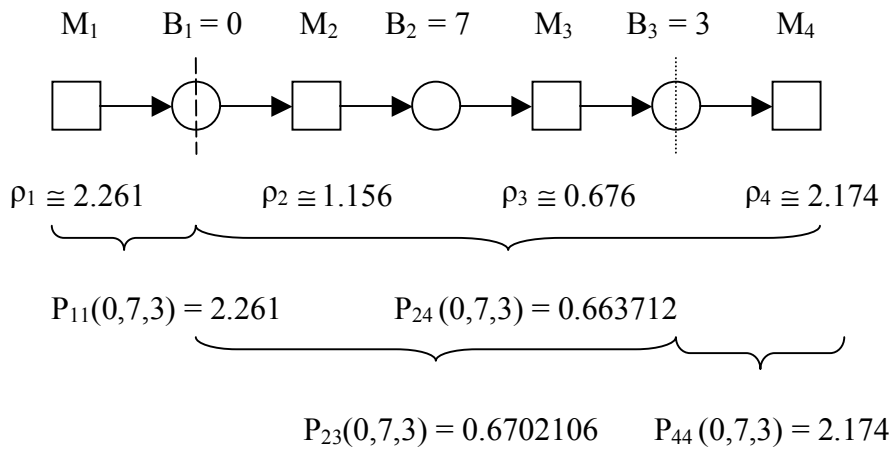

Figure 5. 9. *LIBA proceeds to the first buffer location*

3.1. $P_{11}(0,7,3) \geq P_{24}(0,7,3)$

3.1.1. $P_{23}(0,7,3) \leq P_{44}(0,7,3)$

3.1.1.1. Transfer from B_1 to B_2 is infeasible.

3.1.1.2. Transfer from B_1 to B_3 is infeasible.

No improvement for the main buffer division point B_1 .

4. Decompose the line from the main division buffer location B_3:

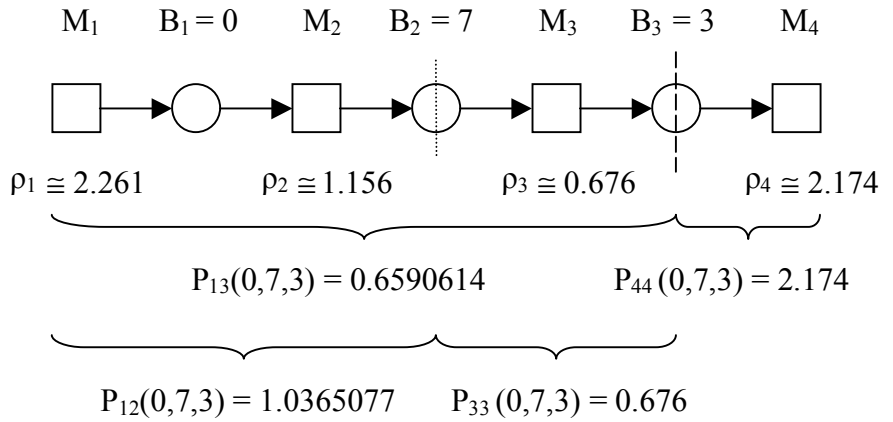

Figure 5. 10. *Termination of LIBA*

4.1. $P_{44}(0,7,3) \geq P_{13}(0,7,3)$

4.1.1. $P_{12}(0,7,3) \geq P_{33}(0,7,3)$

4.1.1.1. Try to transfer from B_3 to B_2 .

$P(0,8,2) = 0.651612 < P^*(0,7,3) = 0.653132$ not an improving direction.
No improvement for the main buffer division point B_3.

5. Stop since no improvement is achieved for every main division point consecutively and we return to the same point where we restarted.

6. Our optimal solution is (0,7,3) with throughput $P^*(0,7,3) = 0.653132$.

There are 66 feasible points in this example and we obtained the throughput values of all these feasible points. Results can be seen in the Appendix A.5. By the algorithm, we reached the global optimum with 5 iterations. Number of iterations in our case is the number of N-machine simulations where N is the number of machines in the whole line.

5. 3. Comparison of Algorithms

Any buffer allocation in an N-machine production line with (N-1) buffer locations and total fixed number of buffer slots C can be described by a vector $\underline{K} = (\underline{K}_1, \underline{K}_2, \ldots, \underline{K}_{(N-1)})$ with non-negative integer entries summing up to the total buffer slots C. Thus, any alternative solution satisfies the equality $C = \sum_{i=1}^{(N-1)} \underline{K}_i$. The set of all points in (N-1)-dimensional space for which this equality holds is an (N-2)-dimensional hyperplane. Since the buffer allocations must be non-negative integers, the feasible region, K, is set to the integer lattice including the vectors whose entries are non-negative on this hyperplane:

$$\mathbf{K} = \{\underline{K} \mid e^T\underline{K} = C,\ \underline{K} \geq \underline{0}\ ,\ \underline{K} \in R^{(N-1)},\ K_j \text{ integer for all } j = 1, \ldots, (N-1)\}$$

The algorithms (SEVA, Non-SEVA, SSA and LIBA) are all based on the idea of *local search*. In other words, the algorithms define a specific neighborhood with respect to the given solution, find the best solution in this neighborhood and move to this solution. This procedure repeats until no better solution is obtained. Since the sum of the components of each solution is equal due to the fixed amount of total given buffer slots, moving from one solution to the other can be considered as the movement along an integer directional vector $\underline{h}$ whose entries sum up to zero meaning that $e^T\underline{h} = 0$.

Such vector is defined as "exchange vector" in Seong et.al.[35]. If a certain exchange vector yields the better production rate, it is called "improving exchange vector".

All algorithms that we focus on define a neighborhood and determine the best solution in this neighborhood. The difference between the algorithms comes from the determination of the neighborhood and the selection procedure of improving exchange vectors through which the algorithms will proceed. The process of defining the specific neighborhood in SEVA, Non-SEVA and LIBA is based on the selection of line segment along an improving exchange vector and containing at least one or more integer solutions. This process is defined as *line segment selection* in Seong et.al[35]. However, in SSA, the specific neighborhood from the existing neighborhood is defined either by the transfer of one buffer slot from the largest capacity buffer location to each of the other buffer locations for the best allocation in the existing neighborhood or by replacing the worst allocation with the one obtained by subtracting it from twice the best one in the existing neighborhood. The new neighborhood has (N-1) allocations where they do not have to lie along the same line as in SEVA, Non-SEVA and LIBA. SSA also differs from SEVA, Non-SEVA and LIBA in the selection procedure of improving exchange vector. In SSA, the improving exchange vector is directly set to the twofold of the vector obtained by the subtraction of the worst allocation from the best one in the existing neighborhood. However, SEVA, Non-SEVA and LIBA select a line segment firstly along the improving exchange vector and extend the improving exchange vector until maximum throughput is achieved along the line segment.

Although the line segment selection concept is valid for all SEVA, Non-SEVA and LIBA, the procedures of selecting line segments are different. Both SEVA and LIBA are based on the concept of the buffer slots transfer between two different buffer locations, while there may be multiple buffer slot transfer between more than two buffer locations in Non- SEVA simultaneously. Although the transfer of buffer slots between two different buffer locations is the case for both SEVA and LIBA, SEVA uses two adjacent buffer locations for the transfer whereas LIBA does not have to use adjacent buffer locations for this transfer. The buffer slot transfer occurs in only one direction from the pre-determined giver to the pre-determined receiver in LIBA. However, in contrast to LIBA, the buffer slot transfer can be done in two directions, between two adjacent buffer locations in SEVA.

Like SEVA, the improving exchange vector in LIBA has only two non-zero entries which differ in sign only. If we re-introduce the standard exchange vectors stated in Seong et.al.[35], we can show the improving exchange vector in LIBA as a linear combination of standard exchange vectors as

$$\underline{P} = \begin{cases} (-1)*Q*(\underline{X}^{i} + \underline{X}^{(i+1)} + \ldots + \underline{X}^{(j-1)}) & \text{if } i < j \\ Q*(\underline{X}^{i} + \underline{X}^{(i+1)} + \ldots + \underline{X}^{(j-1)}) & \text{if } i > j \end{cases}$$

where Q stands for the amount of buffer slot transferred and B_i ,B_j are the giver and receiver buffer locations respectively.

5. 3. 1. Numerical Results

After mentioning the logical and methodological differences between the algorithms presented, the numerical implementation will be the next step in the comparison of the relative efficiencies. To see the efficiency of Line Balancing Algorithm (LIBA), we first focus on the cases that are worked in the study of Seong et.al[35]. There are 18 cases in this study. The first eight of these cases are synchronous production lines with deterministic processing times while the others are asynchronous with independent exponential processing times. In all cases, production lines consist of unreliable machines with independent exponential failure and repair times. All related data including the number of machines N, total fixed number of buffer slots *C* and processing, failure and repair rates for each machine are given in the Table A.6 given in the Appendix A.6.

First of all, we determine the upper bound for throughput for each case by obtaining throughput of each line with the assumption of infinite buffer in each buffer location. Then, we determine the initial allocations and the throughput of each line with these initial allocations. Throughput values for each line with initial allocations determined by the initial allocation procedure of LIBA and infinite buffer can be seen in the Table 5.8 given below:

CASE	Throughput with Initial Allocation	Throughput with Infinite Buffer	Efficiency Percentage
1	0.5806027	0.7341491	79.0851205
2	0.6633516	0.6657005	99.6471536
3	0.6451765	0.7650795	84.3280339
4	0.1725131	0.4382678	39.3624857
5	0.6495863	0.6623969	98.0660236
6	0.5664700	0.5698288	99.4105600
7	0.6637694	0.6662338	99.6300998
8	0.6270106	0.6313999	99.3048304
9	0.6490498	0.6754868	96.0862300
10	0.6359135	0.6453851	98.5324111
11	0.3459187	0.5974446	57.8997115
12	0.9629590	1.1461459	84.0171395
13	1.1207174	1.1210134	99.9735953
14	0.4749513	0.4982646	95.3211005
15	0.6755334	0.6759900	99.9324546
16	1.2726538	1.2754591	99.7800557
17	0.9786912	0.9819490	99.6682312
18	0.5423842	0.5424653	99.9850497

Table 5. 8. *Throughput values and efficiency of initial allocations determined by LIBA initial allocation procedure*

The last column in the above table shows the percent ratio of the throughput with our initial allocation to the one with infinite buffer. This ratio displays the yield of the line with the allocation of given total fixed number of buffer slots with the proposed technique. In half of the cases (i.e. case 2,6,7,8,13,15,16,17,18) above the 99% of throughput value for infinite buffer slots for each buffer locations are satisfied. Due to this reason, to work with these cases will not contribute us about the efficiency of our algorithm since increase in the throughput value can be insignificant and so it is not worth to implement the algorithm for a negligible increase. Therefore, we skip these cases except Case 2 in the comparison step. We hold the second case, since we want to see whether LIBA finds better allocation despite the great efficiency of its initial allocation procedure. The results can be seen in the Appendix A.7.

Although throughput values for the lines are evaluated by the algorithms developed by Glassey and Hong[10] and Hong et.al.[18] in the study of Seong et.al[35], we do not use the given throughput values directly. Instead, we determine the throughput values for given optimal allocations via simulation. The results for LIBA as well as SEVA and Non-SEVA in the Table A.7 in the Appendix A.7, are obtained via

simulation with Replication-Deletion Technique. We use 10 replications for each simulation and 45000 parts for each replication for all cases we focused on. Surprisingly, in all cases, we decide that 5000 parts as a warm-up period is the most suitable value.

In order to make healthy comparison, we also execute LIBA with initial allocation determined by the procedure presented in Seong et.al[35]. Final allocations and optimal throughputs obtained by starting with the initial allocation determined by the procedure of Seong et.al[35] are also given in the Appendix A.7 with the title LIBA 1 in the sub rows. The last column with the title ITERATION in the Appendix A.7 denotes the number of N-machine throughput evaluation during the execution of the algorithms. The number of iterations is the measure of time spent during the execution of the algorithm and so it is another comparison field for the performance of the algorithms.

With the expectation of decreasing the number iterations during the execution of LIBA, we introduce a step size concept, where step size stands for the number of buffer slots that will be transferred during each transfer. Empirically, we decide the step size, w, as

$$w = \left\lceil \frac{C}{5(N-1)} \right\rceil$$

where $\lceil x \rceil$ means the smallest integer that is greater than x.

Step sizes for the cases we worked on are given in the below table:

CASE	1	2	3	4	5	9	10	11	12	14
N	3	4	5	10	5	4	4	5	5	6
C	15	30	12	47	110	10	30	10	15	130
w	2	2	1	2	6	1	2	1	1	6

Table 5. 9. *Step sizes for the cases studied*

N: *Number machines in the line*
C: *Total fixed number of buffer slots that will be allocated*
w : *Step size*

When we determine the giving and receiving buffer location, we transfer w number of buffer slots from the giver to the receiver. If there is no increase in the

throughput, instead of transferring w, we transfer w_1 (= $\lfloor w/2 \rfloor$) amount of buffer slots where $\lfloor x \rfloor$ means the largest integer that is smaller than x. If, again, we could not reach an improvement by this transfer, we reduce the transfer amount from w_1 to w_2 (= $\lfloor w_1/2 \rfloor$). This reduction procedure for the transfer amount goes on with w_{i+1} equal to $\lfloor w_i/2 \rfloor$ in the $(i+1)^{th}$ trial until an improvement is reached in the throughput value for the candidate giver and receiver buffer locations or the transfer amount becomes zero. By this way, we hope to proceed in the increasing direction faster and reach the optimum with less iteration, which improves the performance and speed of the algorithm.

Before proceeding to the comparison of LIBA with SEVA and Non-SEVA, it is worth to mention how the performance of and the results of LIBA if effected by distinct initial allocations. LIBA may reach to different optimal solutions with different initial allocation. This can be observed from the Appendix A.7. There are only three cases (cases 1, 9 and 11) that LIBA achieves to the same final allocation with different initials. Surprisingly, it is proven by complete enumeration that these final allocations are the global optima. The only difference between LIBA with its original allocation, LIBA 2, and LIBA with initial allocation of Seong et.al[35], LIBA 1, is the number of iterations for reaching the optimal values for these three cases. LIBA with its original allocation, LIBA 2, reaches the optimal with less iteration. The number of iterations for LIBA 2 is approximately the half of the number of the iterations done in LIBA 1. For five cases, LIBA 2 gives better results than LIBA 1, whereas LIBA 1 is better than LIBA 2 in remaining two cases that we study. Since the initial allocation of LIBA 1 gives worse throughput than the initial allocation of LIBA 2 in all cases we study and despite this fact in some cases LIBA 1 finds better solutions, we can conclude that the performance of LIBA is dependent on the initial allocation. In addition to this, LIBA may reach better solutions with worse initial allocation. However, the likelihood of reaching better final allocation with better initial allocation is higher than with the worse one. Interestingly, in all these seven cases better solutions, independently of whichever LIBA finds, are achieved with more iteration. Hence, we can say that the more the number of iteration, the better solution LIBA obtains.

When we analyze the results in the Appendix A.7, we observe that LIBA 2 finds better solutions than both SEVA and Non-SEVA in most of the cases. There are only

four cases (Case 1, 3, 9, 11)that the equality of solutions occurs for LIBA 2, SEVA and Non-SEVA where it is shown by complete enumeration that three (Case 1, 9, 11)of these four solutions are already global optima. Briefly, among the cases we worked on, there is no one that SEVA or Non-SEVA could overcome LIBA 2. Hence, it can be said that LIBA 2 is superior to SEVA and Non-SEVA in terms of optimal throughput value.

On the other hand, there are three cases (Case 2, 10, 12) that LIBA 1 has worse solution than both SEVA and Non-SEVA. For the cases 1, 9 and 11, LIBA 1 also reaches the global optimum solution as SEVA, Non-SEVA and LIBA 2. In the remaining four cases, LIBA 1 finds better results than SEVA and Non-SEVA. Hence, it can be misleading to claim that LIBA 1 is superior to SEVA and Non-SEVA or vice versa as we do for LIBA 2.

However, the optimal throughput values are not the only comparison criterion. The numbers of iterations done for reaching these optimal values are also important. Therefore, we should take the number of iterations for attaining the optimal allocations into account during the comparison of the algorithms. From this point of view, LIBA for both initial allocation procedures uses less number of iterations than both SEVA and Non-SEVA for less complex cases. The complexity of the case is direct proportional to the number of machines in the line and the number of buffers slots that will be allocated, meaning that when the number of machines or the total fixed number of buffer slots or both increases, solving the allocation problem for that line becomes more complex. For more complex problems, the number of iterations increases for LIBA whatever the initial allocation is. This situation is obvious for the cases 4, 5 and 14. SEVA and Non-SEVA reaches their optimal solutions with less iteration. However, these are the cases that both LIBA 1 and LIBA 2 find better solutions than SEVA and Non-SEVA and it should be denoted that the number of iterations for LIBA for achieving the throughput at least equal to the optimal throughput values of SEVA and Non-SEVA is approximately the half of the number of iterations for LIBA to achieve its optimal. Therefore, the number of iterations necessary for LIBA to achieve at worst the same solution with SEVA or Non-SEVA and the number of iterations for SEVA and Non-SEVA for optimal solution are approximately the same except Case 4. When we sum up all these findings, we can conclude that LIBA is superior to both SEVA and

Non-SEVA for less complex cases, while this superiority diminishes as the complexity increases.

We also check how much increase is obtained by LIBA until reaching to the optimum. Below tables, Table 5.10 and Table 5.11 give these increases in the percentage form:

CASE	Throughput with Initial Allocation	Throughput with Final Allocation	Percent Increase
1	0.5806027	0.5806027	0
2	0.6633516	0.6648211	0.2215266
3	0.6451765	0.6458731	0.1079705
4	0.1725131	0.1764558	2.2854497
5	0.6495863	0.6498637	0.0427041
9	0.6490498	0.6531318	0.6289196
10	0.6359135	0.6381569	0.3527860
11	0.3459187	0.3479317	0.5819287
12	0.9629590	0.9727568	1.0174680
14	0.4749513	0.4794121	0.9392121

Table 5. 10. *Increase in the throughput value in LIBA 2*

CASE	Throughput with Initial Allocation	Throughput with Final Allocation	Percent Increase
1	0.5666616	0.5806027	2.4602161
2	0.6630851	0.6631803	0.0143571
3	0.6218006	0.6459400	3.8821770
4	0.1703881	0.1765216	3.5997232
5	0.6015092	0.6497500	8.0199605
9	0.6281393	0.6531318	3.9788149
10	0.6277183	0.6375757	1.5703541
11	0.3391255	0.3479317	2.5967378
12	0.9573224	0.9630241	0.5955888
14	0.4400651	0.4793024	8.9162490

Table 5. 11. *Increase in the throughput value in LIBA 1*

All the percent increases in the throughput values seem reasonable for LIBA 1 except two cases (Case 2 and 12). However, the optimal solutions found in these cases, are over the 99 percent of the optimal values obtained by LIBA 2, SEVA and Non-SEVA. Moreover, the number of iterations for reaching the optimum in these cases is very small compared to the ones pertaining to the other algorithms in question. On the other hand, interestingly, the percent increases in the throughput values in LIBA 2 are very small. Based on this result and the knowledge that LIBA 2 attains throughput at

least equal to the ones reached by SEVA and Non-SEVA, we also examine the ratio of the throughput values for initial allocations determined by our own procedure to the optimal throughput values of SEVA and Non-SEVA. These ratios can be seen from the percentage column of Table A.8 in the Appendix A.8.

Surprisingly, the percent ratios are very high meaning that we can attain allocation with throughput very close to the optimal throughput of SEVA and Non-SEVA. Moreover, as occurred in Case 14, we can reach an allocation with higher throughput than the optimal value of SEVA and Non-SEVA via the initial allocation procedure of LIBA itself. By combining the results related to the percent increases in the throughput values and the percent ratios of throughput values to the optimal throughputs for SEVA and Non-SEVA, we can claim that the initial allocation procedure that we introduced is very powerful and it can attain very good allocations close to the optimal solutions by itself without implementing any algorithm.

After comparing the performance of LIBA with SEVA and Non-SEVA, we also want to see the relative efficiency of LIBA with respect to the Simple Search Algorithm (SSA) of Powell and Harris[33]. Even though they developed a new heuristic algorithm for optimal buffer allocation in their study, Powell and Harris[33] basically focused on some characteristics of the production lines such as the effects of bottleneck stations on the optimal buffer allocation and bowl phenomenon instead of demonstrating the performance their algorithm. Hence, the cases that were studied in Powell and Harris[33] were selected according to this goal. Due to this reason, instead of studying the all cases in Powell and Harris[33], we selected a small sample of cases which are more likely to help us to make a healthier comparison and applied LIBA to this sample.

We studied six cases from Powel and Harris[33]. As mentioned before, Powell and Harris[33] worked with serial production lines with reliable stations having independent log-normal processing times. The first three of these cases consists of four machines with a single bottleneck in the third machine, while the last three cases consist of six machines with a single bottleneck in the fourth machine. The table given below summarizes the related data including the number of machines N, total fixed number of buffer slots, *C*, that will be allocated:

CASE	1	2	3	4	5	6
N	4	4	4	6	6	6
C	3	6	9	5	10	15

Table 5. 12. *Data of the cases that we study in Powell and Harris[33]*

NOTE: *All Machines are reliable having lognormal processing times with* ***mean*** *= 1 and* ***standard deviation*** *= 0.5 unless otherwise stated*

** Mean of MC#3 in first three cases is equal to 1.25*

** Mean of MC#4 in last three cases is equal to 1.25*

In contrast to Seong et.al.[35], throughput values is the only comparison criterion for SSA and LIBA since Powell and Harris[33] did not mention any other one such as the number of iteration for these cases. It should also be noted that our initial allocation procedure gave the same initial allocations for all cases with SSA so we do not need to execute LIBA two times as we did in SEVA and Non-SEVA.

CASE	METHOD	ALLOCATION	THROUGHPUT
1	**SSA**	(1,1,1)	0.751123
	LIBA	(1,1,1)	0.751123
2	**SSA**	(1,3,2)	0.785928
	LIBA	(1,3,2)	0.785928
3	**SSA**	(2,3,4)	0.794758
	LIBA	(1,5,3)	0.796687
4	**SSA**	(1,1,1,1,1)	0.743104
	LIBA	(1,1,1,1,1)	0.743104
5	**SSA**	(1,2,3,2,2)	0.784251
	LIBA	(1,2,3,2,2)	0.784251
6	**SSA**	(1,3,4,5,2)	0.796078
	LIBA	(2,3,3,5,2)	0.797372

Table 5. 13. *Optimal allocations with estimated throughput values via simulation for SSA and LIBA*

In the above table, the allocations with the estimated throughput values for both algorithms for each case are given. When we analyze the results, we observe that SSA and LIBA both find the same solutions for the cases 1,2,4,5 where it is verified by complete enumeration that these solutions are global optima. However, LIBA finds

better solutions than SSA in the third and sixth cases. It is worth to state that the lines in first three cases are the same. The only difference among these three cases is the total number of buffer slots, *C*, to be allocated. The same situation also holds for the last three cases meaning that they have the same system with different total number of buffer slots. Case 3 and Case 6 are the ones having the most total number of buffer slots available among the first and last three cases respectively. There is no case that SSA reaches better solution than LIBA. As a result, we can say that LIBA is superior to SSA in terms of optimal throughput value and this superiority becomes more apparent as the number of buffer slots to be allocated increases.

Chapter 6

CONCLUSION

Buffer allocation is a challenging design problem in serial production lines that is often faced in the industry. Effective use of buffers (i.e. how much buffer storage to allow and where to place it) in production lines is important since buffers can have a great impact on the efficiency of the production line. Buffers reduce the blocking of the upstream station and the starvation of the downstream station. However, buffer storage is expensive both due to its direct cost and the increase of the work-in-process inventories it causes. Thus, there is a trade-off between performance and cost. This means that the optimal buffer capacity and the allocation of this capacity have to be determined by analysis.

In this thesis, we studied the optimal buffer allocation problem. The objective was to maximize the throughput of the serial production line by allocating the total fixed number of buffer slots among the buffer locations and in order to achieve this aim we introduced a new heuristic algorithm called "Line Balancing Algorithm (LIBA)"applicable to all types of serial production lines meaning that there is no restriction for the distributions of processing, failure and repair times of any machine, the disciplines such as blocking, failure etc. and the assumptions during the application of LIBA in the line.

The aim of LIBA is to make the line more balanced. To obtain more balanced line. LIBA tries to minimize the total imbalance, which is equal to the sum production

rate differences of the sub-lines obtained by dividing the whole line into two from each buffer location, by buffer slot transfer between only two different buffer locations.

Although LIBA can be started by any arbitrary initial allocation of buffer slots among the buffer locations, we observed that LIBA may reach different final allocations with different initials. Based on this observation, we also integrated to LIBA a new efficient initial allocation procedure which conforms to generalizations about optimal buffer allocation in order to reach better solutions.

To see the power of LIBA, we applied it to some of the cases where SEVA, Non-SEVA and SSA had applied. Even though in some cases LIBA obtains worse solutions than SEVA and Non-SEVA with same initial allocations determined by procedure in the study of Seong et. al.[35], it outperforms SEVA and Non-SEVA in all cases studied with its original initial allocation procedure except the cases where every algorithm finds the global optima.

Besides the optimal throughput value, the number of iterations done for reaching the optimal solution is another comparison criterion. The number of iterations is the number of throughput estimation of the whole line for achieving the optimal solution. From this point of view, LIBA for both initial allocation procedures uses less number of iterations than both SEVA and Non-SEVA for less complex cases. The complexity of the case is proportional to the number of machines in the line and the number of buffers slots that will be allocated, meaning that when the number of machines or the total fixed number of buffer slots or both increases, solving the allocation problem for that line becomes more difficult. For more complex problems, the number of iterations increases for LIBA whatever the initial allocation is. Briefly, we came to a conclusion that LIBA is superior to both SEVA and Non-SEVA for less complex cases, while this superiority diminishes as the complexity increases.

As a final observation, it is worth to state that the initial allocation procedure we introduced is extremely useful. Interestingly, as an indicator of this power, improvement in the production rate of the line after the execution of LIBA did not exceed 2.3 % for all cases that we study. In addition to this, we obtained approximately same production rates with SEVA and Non-SEVA optimal values, even better results in some cases, by only implementing our initial allocation procedure. Therefore, we can

say that our initial allocation procedure by itself, without the execution of LIBA, can be enough for the systems where the effect of small increase in the throughput is negligible.

In contrast to SEVA and Non-SEVA, throughput values is the only criterion which we take into account during the comparison of SSA and LIBA since Powell and Harris[33] did not mention any other one such as the number of iteration for these cases in their study. In all cases studied, LIBA also outperforms SSA in terms of throughput values except the cases where both algorithms find the global optima. The cases that LIBA gives better solutions are the ones having the most total number of buffer slots available among the first and last three cases respectively. Therefore, we can say that LIBA is superior to SSA in terms of optimal throughput value and this superiority becomes more observable as the number of buffer slots that will be allocated increases.

Finally, although it is the case in the industry, we observed that currently available algorithms as well as LIBA do not consider the production lines with stations consisting of parallel machines having individual upstream and/or downstream buffers. It should be possible to adapt currently available algorithms to or new algorithms can be introduced for these types of production systems.

Assembly / Disassembly (A/D) operations, which are also the parts of the manufacturing systems, have been neglected so far. Unlike the machines in the flow lines, A/D operations have more than one upstream and downstream buffer including different part types. Starvation or blocking of these operations occurs when one of these upstream/downstream buffers is empty. Therefore, more investigation is needed on this issue to see how it affects the performance measure of interest and to provide optimal buffer allocation generalizations or algorithms of such systems. We believe in that our algorithm can easily be extended to these types of settings.

Buffer issues may be investigated more. The general assumption on the behaviour of buffers is that transfer times of the parts both from machines to buffers and from buffers to machines are zero. However, in real life, these transfer times are not zero. There are not many studies investigating this issue. Another assumption on the buffer behaviour is the perfect reliability of buffers. However, buffers may be prone to failure as machines. This issue has also been neglected. Few papers have been

published on this issue. Although our algorithm can be readily used by incorporating these changes in the simulation of the production line, more investigation is required on this issue.

BIBLIOGRAPHY

[1] Altiok, T. And Stidham, S.Jr., "The allocation of inter-stage buffer capacities in production lines" IIE Transactions, Vol.15, No.4, pp.292-299 (1983).

[2] Anderson, D.R. and Moodie, C.L., "Optimal buffer storage capacity in production line systems" International Journal of Production Research, Vol.7, No.3, pp.233-240 (1969).

[3] Andijani, A.A. and Anwarul, M., "Manufacturing blocking discipline: A multi-criterion approach for buffer allocations" International Journal of Production Economics, 51, pp. 155-163 (1997).

[4] Chow, W., "Buffer capacity analysis for sequential production lines with variable process times" International Journal of Production Research, Vol.25, No.8, pp.1183-1196 (1987).

[5] Conway, R.W. , Maxwell, W.L. , McClain, J.O. and Thomas, L.J., "The role of work-in-process inventories in serial production lines" Operations Research, 36, pp.229-241 (1988).

[6] El-Rayah, T.E., "The effect of inequality of inter-stage buffer capacities and operation time variability on the efficiency of production line systems" International Journal of Production Research, Vol.17, No.1, pp.77-89 (1979).

[7] Freeman, M.C., "The effects of breakdowns and inter-stage storage on production line capacity" Journal of Industrial Engineering, 15, pp.194-200 (1964).

[8] Gerschwin, S.B. and Dallery, Y., "Manufacturing flow line systems: a review of models and analytical results" Queuing Systems, 12, pp.3-94 (1992).

[9] Gershwin, S.B. and Schor, J.E., "Efficient algorithms for buffer space allocations" Annals of Operations Research, 93, pp.117-144 (2000).

[10] Glassey, C.R. , Hong, Y., "Analysis of behavior of an unreliable n-stage transfer line with (n-1) inter-stage storage buffers" International Journal of Production Research, Vol.31, No.3, pp.519-530 (1993).

[11] Gurkan, G., "Simulation optimization of buffer allocations in production lines with unreliable machines" Annals of Operations Research, 93, pp.177-216 (2000).

[12] Helber, S., "Cash-flow-oriented buffer allocation in stochastic flow lines" International Journal of Production Research, Vol.39, No.14, pp.3061-3083 (2001).

[13] Hillier, M.S., "Characterizing the optimal allocation of storage space in production line systems with variable processing times" IIE Transaction, 32, pp.1-8 (2000).

[14] Hillier, F.S. and So, K.C., "The effect of the coefficient of variation of operation times on the allocation of storage space in production line systems" IIE Transactions, Vol.23, No.2, pp.198-206 (1991).

[15] Hillier, F.S. and So, K.C., "The effect of machine breakdowns and inter-stage storage on the performance of production line systems" International Journal of Production Research, Vol.29, No.10, pp.2043-2055 (1991).

[16] Hillier, F.S., So, K.C. and Boling, R.W., "Notes: Toward characterizing the optimal allocation of storage space in production line systems with variable processing times" Management Science, Vol.39, No.1, pp.126-133 (1993).

[17] Ho, Y.C., Eyler, M.A. and Chien, T.T., "A gradient technique for general buffer storage design in a production line" International Journal of Production Research, Vol.17, No.6, pp.557-580 (1979).

[18] Hong, Y., Glassey, C.R., Seong, D., "The analysis of a production line with unreliable machines and random processing times" IIE Transactions, Vol.24, No.1, pp.77-83 (1992).

[19] Jafari, M.A. and Shanthikumar, J.G., "Determination of optimal buffer storage capacities and optimal allocation in multi-stage automatic transfer lines" IIE Transactions, Vol.21, No.2, pp.130-135 (1989).

[20] Kim, S. and Lee, H.J., "Allocation of buffer capacity to minimize average work-in-process" Production Planning&Control, Vol.12, No.7, pp.706-716 (2001).

[21] Lutz, C.M., Davis, K.R. and Sun, M., "Determining buffer location and size in production lines using tabu-search" European Journal of Operational Research, 106, pp. 301-316 (1998).

[22] Nelder, J.A., Mead, R., "A simplex method for function minimization" Computer Journal, 7, pp.308-321. (1965).

[23] Papadopoulos, C.T. and Spinellis, D.D., "A simulated annealing approach for buffer allocation in reliable production lines" Annals of Operations Research, 93, pp. 373-384 (2000).

[24] Papadopoulos, C.T. and Spinellis, D.D., "Stochastic algorithms for buffer allocation in reliable production lines" Mathematical Problems in Engineering, Vol.5, Iss.6, pp. 441-458 (2000).

[25] Papadopoulos, C.T. , Spinellis, D.D. and Smith, J.M., "Large production line optimization using simulated annealing" International Journal of Production Research, Vol.38, No.3, pp.509-541 (2000).

[26] Papadopoulos, H.T. and Vidalis, M.I., "Optimal buffer storage allocation in balanced reliable production lines" International Transactions in Operational Research, Vol.5, No.4, pp.325-339 (1998).

[27] Papadopoulos, H.T. and Vidalis, M.I., "Optimal buffer allocation in short mu-balanced unreliable production lines" Computers & Industrial Engineering, 37, pp.691-710 (1999).

[28] Papadopoulos, H.T. and Vidalis, M.I., "Minimizing WIP inventory in reliable production lines" International Journal of Production Economics, 70, pp.185-197 (2001).

[29] Papadopoulos, H.T. and Vouros, G.A.,"A model management system (MMS) for the design and operation of production lines" International Journal of Production Research, Vol.35, No.8, pp.2213-2236 (1997).

[30] Papadopoulos, H.T. and Vouros, G.A., "Buffer allocation in unreliable production lines using a knowledge based system" Computers and Operations Research, Vol.25, No.12, pp.1055-1067 (1998).

[31] Park, T., "A two-phase heuristic algorithm for determining buffer sizes of production lines" International Journal of Production Research, Vol.31, No.3, pp.613-631 (1993).

[32] Powell, S.G., "Buffer allocation in unbalanced three-station serial lines" International Journal of Production Research, Vol.32, No.6, pp.2201-2217 (1994).

[33] Powell, S.G. and Harris, J.H., "An algorithm for optimal buffer placement in reliable serial lines" IIE Transaction, 31, pp.287-302 (1999).

[34] Powell, S.G. and Pyke, D.F., "Allocation of buffers to serial production lines with bottlenecks" IIE Transactions, 28, pp.18-29 (1996).

[35] Seong, D. , Chang, S.Y. and Hong, Y., "Heuristic algorithms for buffer allocation in a production line with unreliable machines" International Journal of Production Research, Vol.33, No.7, pp.1989-2005 (1995).

[36] Seong, D. , Chang, S.Y. and Hong, Y., "An algorithm for buffer allocation with linear resource constraints in a continuous flow production line" Asia-Pacific Journal of Operational Research, Vol.17, Iss.2, pp.169-180 (2000).

[37] Sheskin, T.J., "Allocation of inter-stage storage along an automatic production line" AIIE Transactions, 8(1), pp.146-152 (1976).

[38] So, C.K., "Optimal buffer allocation strategy for minimizing work- in-process inventory in unpaced production lines" IIE Transactions, 29, pp .81-88 (1997).

[39] Soyster, A.L. , Schmidt, J.W. and Rohrer, M.W., "Allocation of buffer capacities for a class of fixed cycle production lines" AIIE Transactions, 11(2), pp.140-146 (1979).

[40] Spendley,W. , Hext, G.R., "Sequential application of simplex designs in optimization and evolutionary operations" Technometrics, 4, pp.441-461 (1962).

[41] Yamashita, H. and Altiok, T., "Buffer capacity allocation for a desired throughput in production lines" IIE Transactions, 30, pp.883-891 (1998).

[42] Yamashina, H. and Okamura, K., "Analysis of in-process buffers for multi-stage transfer line systems" International Journal of Production Research, Vol.21, No.2, pp.183-195 (1983).

APPENDIX

A. 1. The Pseudo-Code LIBA

Line Balancing Algorithm (LIBA) can be given in a systematic way as below:

Step 0: Start with an initial allocation of buffer slots.

Step 1: Evaluate the production rate P of line L.

Step 2: Set "k = 0" and "N = #machines in line L".

Step 3: (Initial decoupling of the whole from the main division buffer location)

3.1. If "N is even" then

3.1.1. Decouple the line into two sub-lines L_1 and L_2 from the buffer location $(N/2)+k$, which is the main division buffer $B_{(N/2)+k}$.

3.2. If "N is odd" then

3.2.1. Decouple the line into two sub-lines L_1 and L_2 from the buffer location $(\frac{(N+1)}{2})+k$, which is the main division buffer $B_{(\frac{(N+1)}{2})+k}$.

Step 4: Evaluate the production rates P^1 and P^2 of L_1 and L_2.

Step 5: (Proceeding to the next main division buffer location in case of throughput equality around the existing one)

5.1. If "$P^1 = P^2$" then

5.1.1. If "$k \geq 0$" then

5.1.1.1. If "N is even" and "k = $(N/2)-1$" then

5.1.1.1.1. Set "k = 0".

5.1.1.1.2. Return to Step 3.

5.1.1.2. Set "k = - (k+1)".

5.1.1.3. Return to Step 3.

5.1.2. If "k < 0" then

5.1.2.1. If "N is odd" and "k = - $(\frac{(N+1)}{2}-1)$ " then

5.1.2.1.1. Set "k = 0".

5.1.2.1.2. Return to Step 3.

5.1.2.2. Set "k = -k".

5.1.2.3. Return to Step 3.

Step 6: (Determination of the potential giver and receiver initial sub-lines)

6.1. If "$P^1 > P^2$" then

6.1.1. L_1 is the potential giver line with N_1 machines.

6.1.2. L_2 is the potential receiver line with N_2 machines.

6.1.3. Set "$N_1(1) = N_1$" and "$L_1(1) = L_1$".

6.1.4. Set "$N_2(1) = N_2$" and "$L_2(1) = L_2$".

6.2. If "$P^1 < P^2$" then

6.2.1. L_1 is the potential receiver line with N_1 machines.

6.2.2. L_2 is the potential giver line with N_2 machines.

6.2.3. Set "$N_1(1) = N_2$" and "$L_1(1) = L_2$".

6.2.4. Set "$N_2(1) = N_1$" and "$L_2(1) = L_1$".

6.3. Go to Step 7.

Step 7: (Determination of the potential buffer slot giver candidate sequence)

7.1. Set "I =1".

7.2. While "$I \leq \left[\frac{\ln(N_1(1)-1)}{\ln 2}-1\right]$" do

7.2.1. If "$N_1(I)$ is even" then

7.2.1.1. Decouple the line $L_1(I)$ into two sub-lines $L_1(I,1)$ and $L_1(I,2)$ with $N_1(I,1)$ and $N_1(I,2)$ machines respectively from the buffer location $N_1(I)$, which is the I^{th} division buffer location.

7.2.2. If "$N_1(I)$ is odd" then

7.2.2.1. Decouple the line $L_1(I)$ into two sub-lines $L_1(I,1)$ and $L_1(I,2)$ with $N_1(I,1)$ and $N_1(I,2)$ machines respectively from the buffer location $\frac{(N_1(I)+1)}{2}$, which is the I^{th} division buffer location.

7.2.3. Evaluate the production rates $P_1(I,1)$ and $P_1(I,2)$ of the sub-lines $L_1(I,1)$ and $L_1(I,2)$ respectively.

7.2.4. If "$P_1(I,1) > P_1(I,2)$" then

7.2.4.1. Set "$N_1(I+1) = N_1(I,1)$" and "$L_1(I+1) = L_1(I,1)$".

7.2.5. If "$P_1(I,1) < P_1(I,2)$" then

7.2.5.1. Set "$N_1(I+1) = N_1(I,2)$" and "$L_1(I+1) = L_1(I,2)$".

7.2.6. If "$P_1(I,1) = P_1(I,2)$" then

7.2.6.1. Set initial potential giver to the I^{th} division buffer location of line $L_1(I)$.

7.2.6.2. Go to Step 8.

7.2.7. Set "I = I +1".

7.3. If "$N_1(I) = 2$" then

7.3.1. Set initial potential giver to the buffer location of the line $L_1(I)$.

7.3.2. Go to Step 8.

7.4. If "$N_1(I) = 1$" then

7.4.1. Set initial potential giver to the buffer location, which is the I^{th} division buffer location just in front of the machine.

7.4.2. Go to Step 8.

Step 8: (Determination of the potential buffer slot receiver candidate sequence)

8.1. Set "J =1".

8.2. While "$J \le \left[\frac{\ln(N_2(1)-1)}{\ln 2} - 1\right]$" do

8.2.1. If "$N_2(J)$ is even" then

8.2.1.1. Decouple the line $L_2(J)$ into two sub-lines $L_2(J,1)$ and $L_2(J,2)$ with $N_2(J,1)$ and $N_2(J,2)$ machines respectively from the buffer location $N_1(J)$, which is the J^{th} division buffer location.

8.2.2. If "$N_2(J)$ is odd" then

8.2.2.1. Decouple the line $L_2(J)$ into two sub-lines $L_2(J,1)$ and $L_2(J,2)$ with $N_2(J,1)$ and $N_2(J,2)$ machines respectively from the buffer location $\frac{(N_2(J)+1)}{2}$, which is the J^{th} division buffer location.

8.2.3. Evaluate the production rates $P_2(J,1)$ and $P_2(J,2)$ of the sub-lines $L_2(J,1)$ and $L_2(J,2)$ respectively.

8.2.4. If "$P_2(J,1) > P_2(J,2)$" then

8.2.4.1. Set "$N_2(J+1) = N_2(J,1)$" and "$L_2(J+1) = L_2(J,1)$".

8.2.5. If "$P_2(J,1) < P_2(J,2)$" then

8.2.5.1. Set "$N_2(J+1) = N_2(J,2)$" and "$L_2(J+1) = L_2(J,2)$".

8.2.6. If "$P_2(J,1) = P_2(J,2)$" then

8.2.6.1. Set initial potential receiver to the J^{th} division buffer location of the line $L_2(J)$.

8.2.6.2. Go to Step 9.

8.2.7. Set "J = J +1".

8.3. If "$N_2(J) = 2$" then

8.3.1. Set initial potential receiver to the buffer location of the line $L_2(J)$.

8.3.2. Go to Step 8.

8.4. If "$N_2(J) = 1$" then

8.4.1. Set initial potential giver to the buffer location, which is the J^{th} division buffer location just in front of the machine.

8.4.2. Go to Step 8.

Step 9: (Determination of transfer locations)

9.1. If "improvement occurs with transfer between existing potential giver and receiver" then

9.1.1. Transfer the buffer slots from the potential giver to potential receiver until no improvement.

9.1.2. Set "M = 0".

9.1.3. Go to Step 3.

9.2. Set "$J_{max} = J$".

9.3. While "I > 0" and "no improvement with transfer between existing potential giver and receiver" do

9.3.1. Set potential giver to the I^{th} division buffer location of the line $L_1(I)$.

9.3.2. While "J > 0" and "no improvement with transfer between existing potential giver and receiver" do

9.3.2.1. Set potential receiver to the J^{th} division buffer location of the line $L_2(J)$.

9.3.2.2. If "improvement occurs with transfer between existing potential giver and receiver" then

9.3.2.2.1. Transfer the buffer slots from the potential giver to potential receiver until no improvement.

9.3.2.2.2. Set "M = 0".

9.3.2.2.3. Go to Step 3.

9.3.2.3. If "no improvement with transfer between existing potential giver and receiver" then

9.3.2.3.1. Set "J = J – 1".

9.3.3. If "J = 0" then

9.3.3.1. Set potential receiver to the main division buffer location.

9.3.3.2. If "improvement occurs with transfer between existing potential giver and receiver" then

9.3.3.2.1. Transfer the buffer slots from the potential giver to potential receiver until no improvement.

9.3.3.2.2. Set "M = 0".

9.3.3.2.3. Go to Step 3.

9.3.3.3. If "no improvement with transfer between existing potential giver and receiver" then

9.3.3.3.1. Set "$J = J_{max}$".

9.3.3.3.2. Set potential receiver to the J^{th} division buffer location of the line $L_2(J)$.

9.3.3.3.3. Set "I = I – 1".

9.4. If "I = 0" then

9.4.1. Set potential giver to the main division buffer location.

9.4.2. While "J > 0" and "no improvement with transfer between existing potential giver and receiver" do

9.4.2.1. Set potential receiver to the J^{th} division buffer location of the line $L_2(J)$.

9.4.2.2. If "improvement occurs with transfer between existing potential giver and receiver" then

9.4.2.2.1. Transfer the buffer slots from the potential giver to potential receiver until no improvement.

9.4.2.2.2. Set "M = 0".

9.4.2.2.3. Go to Step 3.

9.4.2.3. If "no improvement with transfer between existing potential giver and receiver" then

9.4.2.3.1. Set "J = J – 1".

9.4.3. Set "M = M + 1"

9.4.4. If "$k \geq 0$" then

9.4.4.1. If "N is even" and "$k = (N/2) - 1$" then

9.4.4.1.1. Set "k = 0".

9.4.4.1.2. Return to Step 3.

9.4.4.2. Set "k = - (k+1)".

9.4.4.3. Return to Step 3.

9.4.5. If "$k < 0$" then

9.4.5.1. If "N is odd" and "$k = -\left(\frac{(N+1)}{2} - 1\right)$" then

9.4.5.1.1. Set "k = 0".

9.4.5.1.2. Return to Step 3.

9.4.5.2. Set "k = - k".

9.4.5.3. Return to Step 3.

A.2. General Model frame for the simulation of the production lines in SIMAN V

```
BEGIN;

CREATE;

ASSIGN:M=ILK+1;

ENTRY  QUEUE,INITIAL;
          SCAN:NQ(M-1).EQ.0;
          DUPLICATE:1,ENTRY;
          ROUTE:0,M;

STATION,STATIONSET;

QUEUE,M-1;

SCAN:(NQ(M+numstat).EQ.0).and.(NR(M).EQ.0).and.(MR(M).NE.0);

SEIZE:M;

ASSIGN:PROCESSTIME(M)=1/PROCESS_RATE(M):
          FAILTIME(M)=EXPO(1/FAILURE_RATE(M),M-1);

REPAIR IF:(MR(M).EQ.0).AND.(NQ(M+1+(2*numstat)).EQ.1);
             DELAY:EXPO(1/RERAIR_RATE(M),M-1);
             ALTER:M,1;
             DISPOSE;
          ENDIF;

WHILE:PROCESSTIME(M).GT.FAILTIME(M);
    DELAY:FAILTIME(M);
    RELEASE:M;
    ALTER:M,-1;
    ASSIGN:PROCESSTIME(M)=PROCESSTIME(M)-FAILTIME(M);
    ASSIGN:FAILTIME(M)=EXPO(1/FAILURE_RATE(M),M-1);
    DUPLICATE:1,REPAIR;
    QUEUE,M+1+(2*numstat);
    SEIZE:M;
ENDWHILE;
```

```
IF:PROCESSTIME(M).LE.FAILTIME(M);
   DELAY:PROCESSTIME(M);
ENDIF;

RELEASE:M;

IF:BUFFERCAPACITY(M).EQ.0;
   QUEUE,M+numstat;
   SCAN:(NR(M+1).EQ.0).and.(NQ(M+1+numstat).EQ.0).and.(MR(M+1).NE.0);
ENDIF;

IF:BUFFERCAPACITY(M).EQ.1;
   IF:NQ(M).EQ.1;
      ASSIGN:HESITATE(M)=0.0000000001;
   ELSEIF:NQ(M).EQ.0;
      ASSIGN:HESITATE(M)=0;
  ENDIF;
  QUEUE,M+numstat;
  SCAN:(NQ(M).LT.BUFFERCAPACITY(M));
  DELAY:HESITATE(M);
ENDIF;

IF:BUFFERCAPACITY(M).GT.1;
   QUEUE,M+numstat;
   SCAN:(NQ(M).LT.BUFFERCAPACITY(M));
ENDIF;

IF:M.EQ.MEMBER(STATIONSET,SON+1);
   ROUTE:0,EXITSYSTEM;
ENDIF;

ROUTE:0,M+1;

STATION,EXITSYSTEM;

ASSIGN:PART=PART+1:
            RATE=PART/TNOW;

IF:PART.GT.WARMUP;
   ASSIGN:CSUM=CSUM+RATE;
ENDIF;

IF:PART.EQ.MC(1);
   WRITE,AVGRATE:CSUM/(PART-WARMUP);
ENDIF;
```

```
COUNT:OUTPUT,1;

DISPOSE;

END;
```

A.3. Experimental frame of the production line given in Seong et.al.[35] as Case 9 for the simulation in SIMAN V

```
BEGIN;THESIS,CASE 9,SEONG ET.AL.[35];

VARIABLES:BUFFERCAPACITY(11),1000,1,5,4,0,0,0,0,0,0,1000:

      FAILURE_RATE(12),1000,0.07,0.11,0.49,0.19,0,0,0,0,0,0,1000:
      REPAIR_RATE(12),1000,0.17,0.37,0.78,0.5,0,0,0,0,0,0,1000:
      PROCESS_RATE(12),1000,3.7,1.5,1.1,3,0,0,0,0,0,0,1000:
      PROCESSTIME(12):FAILTIME(12):HESITATE(12):
      PART:RATE:CSUM:
      numstat,10:WARMUP,5000:
      ILK,1: SON,4;

STATIONS:MC0:MC1:MC2:MC3:MC4:MC5:MC6:
            MC7:MC8:MC9:MC10:EXITSYSTEM;

SETS:STATIONSET,MC0,MC1,MC2,MC3,MC4,MC5,MC6,MC7,MC8,MC9,MC10;

QUEUES:BUFFER0:BUFFER1:BUFFER2:BUFFER3:BUFFER4:BUFFER5:
          BUFFER6:BUFFER7:BUFFER8:BUFFER9:BUFFER10:
          DUMMY1:DUMMY2:DUMMY3:DUMMY4:DUMMY5:
          DUMMY6:DUMMY7:DUMMY8:DUMMY9:DUMMY10:
          RESUME0:RESUME1:RESUME2:RESUME3:RESUME4:RESUME5:
          RESUME6:RESUME7:RESUME8:RESUME9:RESUME10:INITIAL;

RESOURCES:MACHINE0:MACHINE1:MACHINE2:MACHINE3:
              MACHINE4:MACHINE5:MACHINE6:MACHINE7:
              MACHINE8:MACHINE9:MACHINE10;

FILES:AVGRATE,"out.txt",SEQ,FOR;

COUNTERS:OUTPUT,45000,YES;

REPLICATE,10;

END;
```

A.4. Behaviour of throughput with respect to total imbalance

CASE	Buffer #1	Buffer #2	Subline #1 MC#1	Subline #2 MC#2 - MC#3	Absolute imbalance around buffer #1	Subline #1 MC#1 - MC#2	Subline #2 MC#3	Absolute imbalance around buffer #2	Total Imbalance	Throughput
1	**15**	**0**	0.7510105	0.6040874	0.1469231	0.6644307	0.7526061	0.0881754	**0.2350985**	**0.5666616**
2	**14**	**1**	0.7510105	0.6108226	0.1401879	0.6621405	0.7526061	0.0904656	**0.2306535**	**0.5706369**
3	**13**	**2**	0.7510105	0.6169557	0.1340548	0.6596564	0.7526061	0.0929497	**0.2270045**	**0.5735585**
4	**12**	**3**	0.7510105	0.6226406	0.1283699	0.6569585	0.7526061	0.0956476	**0.2240175**	**0.5757271**
5	**11**	**4**	0.7510105	0.6278978	0.1231127	0.6540009	0.7526061	0.0986052	**0.2217179**	**0.5781502**
6	**10**	**5**	0.7510105	0.6326509	0.1183596	0.6508111	0.7526061	0.1017950	**0.2201546**	**0.5792758**
7	**9**	**6**	0.7510105	0.6373042	0.1137063	0.6475239	0.7526061	0.1050822	**0.2187885**	**0.5802479**
8	**8**	**7**	0.7510105	0.6416683	0.1093422	0.6438669	0.7526061	0.1087392	**0.2180814**	**0.5805494**
9	**7**	**8**	0.7510105	0.6455683	0.1054422	0.6400218	0.7526061	0.1125843	**0.2180265**	**0.5806027**
10	**6**	**9**	0.7510105	0.6487522	0.1022583	0.6360769	0.7526061	0.1165292	**0.2187875**	**0.5801709**
11	**5**	**10**	0.7510105	0.6529079	0.0981026	0.6318851	0.7526061	0.1207210	**0.2188236**	**0.5794884**
12	**4**	**11**	0.7510105	0.6560910	0.0949195	0.6269482	0.7526061	0.1256579	**0.2205774**	**0.5782482**
13	**3**	**12**	0.7510105	0.6588160	0.0921945	0.6218415	0.7526061	0.1307646	**0.2229591**	**0.5760097**
14	**2**	**13**	0.7510105	0.6612480	0.0897625	0.6158844	0.7526061	0.1367217	**0.2264842**	**0.5740194**
15	**1**	**14**	0.7510105	0.6639242	0.0870863	0.6096652	0.7526061	0.1429409	**0.2300272**	**0.5711139**
16	**0**	**15**	0.7510105	0.6665807	0.0844298	0.6030325	0.7526061	0.1495736	**0.2340034**	**0.5673847**

Table A.4. *Behaviour of throughput with respect to total imbalance*

A.5. Throughput values for all feasible allocations in the sample problem given as Case 9 in Seong et. al.[35]

BUFFER LOCATION #1	BUFFER LOCATION #2	BUFFER LOCATION #3	PRODUCTION RATE
0	7	3	0.6531318
0	8	2	0.6516116
1	6	3	0.6514968
0	6	4	0.6513359
1	7	2	0.6506884
2	6	2	0.6491286
2	5	3	0.6490498
1	5	4	0.6490348
0	5	5	0.6474089
3	5	2	0.6466561
0	9	1	0.6459260
2	4	4	0.6457656
3	4	3	0.6456720
1	8	1	0.6455040
2	7	1	0.6445656
1	4	5	0.6443031
4	4	2	0.6430809
3	6	1	0.6430230
0	4	6	0.6419335
3	3	4	0.6406850
4	5	1	0.6403728
4	3	3	0.6403363
2	3	5	0.6393603
5	3	2	0.6379511
5	4	1	0.6369806
1	3	6	0.6368844
0	10	0	0.6336940
0	3	7	0.6336293
1	9	0	0.6335532
2	8	0	0.6329160
4	2	4	0.6327535
5	2	3	0.6324114
3	7	0	0.6321167
6	3	1	0.6318294
3	2	5	0.6315741
4	6	0	0.6304804
6	2	2	0.6299246
2	2	6	0.6294050
5	5	0	0.6281393
1	2	7	0.6262134
6	4	0	0.6246663
7	2	1	0.6237570
0	2	8	0.6220849
5	1	4	0.6203103
6	1	3	0.6198679
7	3	0	0.6195772
4	1	5	0.6193777
7	1	2	0.6171390
3	1	6	0.6170826
2	1	7	0.6146056
8	2	0	0.6119668
8	1	1	0.6113383
1	1	8	0.6106461
0	1	9	0.6056569
6	0	4	0.6005731
7	0	3	0.6000120
9	1	0	0.5999668
5	0	5	0.5998935
4	0	6	0.5981102
8	0	2	0.5975358
3	0	7	0.5956128
2	0	8	0.5921852
9	0	1	0.5920033
1	0	9	0.5875122
0	0	10	0.5815684
10	0	0	0.5810414

Table.A.5. *Throughput values for all feasible allocations in the sample problem given as Case 9 in Seong et. al.[35]*

A.6. Processing, failure and repair rates for production lines in Seong et.al.[35]

CASE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
N	3	4	5	10	5	5	8	9	4	4	5	5	5	6	8	9	10	10
C	15	30	12	47	110	200	110	155	10	30	10	15	115	130	125	200	310	315

MC#1	λ_1	1	1	1	1	1	1	1	1	3,7	3	1,2	2,8	2,6	3	1	2,5	2,7	2,4
	μ_1	0	0,4	0,2	0	0,1	0,1	0	0,3	0,1	0,4	0,1	0,3	0,1	0,3	0,3	0,2	0,1	0,4
	r_1	0,1	0,7	0,7	0	0,3	0,2	1	0,7	0,2	0,5	0,3	0,6	0,4	0,2	0,5	0,7	0,8	0,5
MC#2	λ_2	1	1	1	1	1	1	1	1	1,5	1	1	1,7	3	1	3,6	1,5	1,8	1,7
	μ_2	0	0,2	0,2	0	0,1	0,2	0,1	0,2	0,1	0,3	0,3	0,4	0,2	0,5	0,2	0,1	0,2	0,2
	r_2	0,1	0,8	0,7	0	0,4	0,3	0,9	0,6	0,4	0,6	0,5	0,8	0,4	0,5	0,5	0,6	0,3	0,6
MC#3	λ_3	1	1	1	1	1	1	1	1	1,1	2	3	2,5	3,4	1,2	1,7	2,8	2,1	2,8
	μ_3	0	0,2	0,2	0	0,2	0,3	0,1	0,4	0,5	0,4	0,5	0,5	0,2	0,1	0,2	0,3	0,3	0,3
	r_3	0,1	0,7	0,7	0	0,4	0,4	0,9	0,7	0,8	0,5	0,2	0,8	0,6	0,3	0,6	0,8	0,5	0,5
MC#4	λ_4		1	1	1	1	1	1	1	3	3,6	2	3,4	4,7	1,8	1,4	3,6	2,3	2,2
	μ_4		0,1	0,2	0	0,2	0,2	0	0,2	0,2	0,5	0,4	0,4	0,2	0,2	0,3	0,2	0,2	0,4
	r_4		0,6	0,7	0	0,5	0,4	1	0,5	0,5	0,4	0,3	0,9	0,5	0,1	0,5	0,8	0,5	0,5
MC#5	λ_5			1	1	1	1	1	1			1,8	1,9	1,5	1,5	2,8	2,1	1,6	2,1
	μ_5			0,2	0	0,2	0,1	0,2	0,2			0,2	0,1	0,1	0,3	0,2	0,1	0,3	0,3
	r_5			0,7	0	0,4	0,3	0,7	0,6			0,1	0,7	0,3	0,2	0,5	0,7	0,8	0,5
MC#6	λ_6				1			1	1						2	2,7	1,9	2,7	2,5
	μ_6				0			0,2	0,1						0,4	0,4	0,1	0,3	0,4
	r_6				0			0,6	0,5						0,3	0,5	0,6	0,7	0,4
MC#7	λ_7				1			1	1							1,6	2,7	1,5	1,1
	μ_7				0			0,3	0,4							0,3	0,3	0,1	0,3
	r_7				0			0,5	0,7							0,7	0,8	0,6	0,5
MC#8	λ_8				1			1	1							1,2	3	1,5	1,3
	μ_8				0			0,1	0,3							0,2	0,2	0,2	0,3
	r_8				0			0,9	0,8							0,4	0,5	0,6	0,5
MC#9	λ_9				1				1								2	1,2	1,6
	μ_9				0				0,2								0,3	0,1	0,3
	r_9				0				0,5								0,6	0,6	0,5
MC#10	Λ_{10}				1													2,6	0,8
	μ_{10}				0													0,3	0,2
	R_{10}				0													0,4	0,5

Table A.6. *Processing, failure and repair rates for production lines in Seong et.al.[35]*

λ_i: *processing rate for machine i*

μ_i: *failure rate for machine i*

r_i: *repair rate for machine i*

C: *total fixed number of buffer slots that will be allocated*

N: *number of machines in the production line*

NOTE: *Processing times are deterministic for the cases from Case#1 to Case#8*

Processing times are exponential for the cases from Case# 9 to Case#18

Failure and Repair times are all exponential for all cases

A.7. Optimal allocations with estimated throughput values via simulation for SEVA, Non-SEVA and LIBA

CASE	N	*K*	METHOD	ALLOCATION	THROUGHPUT	ITERATION
1	3	15	A	(7,8)	0.5806027	8
			B	(7,8)	0.5806027	8
			C	(7,8)	0.5806027	8
			LIBA 1	(7,8)	0.5806027	7
			LIBA 2	(7,8)	0.5806027	4
2	4	30	A	(20,8,2)	0.6646476	48
			B	(19,9,2)	0.6647446	25
			C	(20,8,2)	0.6646476	29
			LIBA 1	(19,11,0)	0.6631803	4
			LIBA 2	(18,9,3)	0.6648211	15
3	5	12	A	(2,4,4,2)	0.6458731	34
			B	(2,4,4,2)	0.6458731	25
			C	(2,4,4,2)	0.6458731	30
			LIBA 1	(3,3,4,2)	0.6459400	19
			LIBA 2	(2,4,4,2)	0.6458731	17
4	10	47	A	(0,3,8,8,9,8,8,3,0)	0.1756255	333
			B	(0,5,5,10,10,5,5,5,2)	0.1752594	41
			C	(0,4,7,8,9,8,6,4,1)	0.1757494	58
			LIBA 1	(0,4,8,8,8,9,5,5,0)	0.1765216	407
			LIBA 2	(0,5,6,9,8,9,4,6,0)	0.1764558	274
5	5	110	A	(22,27,38,23)	0.6494134	141
			B	(22,28,36,24)	0.6497027	36
			C	(22,28,36,24)	0.6497027	43
			LIBA 1	(19,35,34,22)	0.6497500	77
			LIBA 2	(23,30,35,22)	0.6498637	81
9	4	10	A	(0,7,3)	0.6531318	26
			B	(0,6,4)	0.6513359	16
			C	(0,7,3)	0.6531318	24
			LIBA 1	(0,7,3)	0.6531318	8
			LIBA 2	(0,7,3)	0.6531318	5
10	4	30	A	(11,16,3)	0.6380394	53
			B	(11,16,3)	0.6380394	37
			C	(11,16,3)	0.6380394	24
			LIBA 1	(10,18,2)	0.6375757	23
			LIBA 2	(10,16,4)	0.6381569	28
11	5	10	A	(1,3,4,2)	0.3479317	31
			B	(1,3,4,2)	0.3479317	48
			C	(1,3,4,2)	0.3479317	48
			LIBA 1	(1,3,4,2)	0.3479317	24
			LIBA 2	(1,3,4,2)	0.3479317	15
12	5	15	A	(4,6,3,2)	0.9719334	74
			B	(4,6,4,1)	0.9708824	25
			C	(4,6,4,1)	0.9708824	25
			LIBA 1	(4,7,4,0)	0.9630241	8
			LIBA 2	(4,7,3,1)	0.9727568	14
14	6	130	B	(26,45,26,16,17)	0.4611404	58
			C	(23,38,47,14,8)	0.4448229	83
			LIBA 1	(15,36,23,42,14)	0.4793024	177
			LIBA 2	(15,31,28,42,14)	0.4794121	178

Table A.7. *Optimal allocations with estimated throughput values via simulation for SEVA, Non-SEVA and LIBA*

A: *SEVA*
B: *Non-SEVA with both big and small steps*
C: *Non-SEVA with only small steps*
LIBA 1: *LIBA with initial allocation determined in Song et.al.[35]*
LIBA 2: *LIBA with initial allocation determined by its original procedure*

NOTE: *SEVA was not applied to Case 14*

A.8. Efficiency evaluation of initial allocation procedure of LIBA

CASE	N	*K*	METHOD	ALLOCATION	THROUGHPUT	PERCENTAGE
1	3	15	A	(7,8)	0.5806027	100
			B	(7,8)	0.5806027	100
			C	(7,8)	0.5806027	100
			initial	(7,8)	0.5806027	-
2	4	30	A	(20,8,2)	0.6646476	99.81
			B	(19,9,2)	0.6647446	99.79
			C	(20,8,2)	0.6646476	99.81
			initial	(11,10,9)	0.6633516	-
3	5	12	A	(2,4,4,2)	0.6458731	99.89
			B	(2,4,4,2)	0.6458731	99.89
			C	(2,4,4,2)	0.6458731	99.89
			initial	(3,3,3,3)	0.6451765	-
4	10	47	A	(0,3,8,8,9,8,8,3,0)	0.1756255	98.23
			B	(0,5,5,10,10,5,5,5,2)	0.1752594	98.43
			C	(0,4,7,8,9,8,6,4,1)	0.1757494	98.16
			initial	(5,5,5,5,6,6,5,5,5)	0.1725131	-
5	5	110	A	(22,27,38,23)	0.6494134	100.03
			B	(22,28,36,24)	0.6497027	99.98
			C	(22,28,36,24)	0.6497027	99.98
			initial	(27,27,28,28)	0.6495863	-
9	4	10	A	(0,7,3)	0.6531318	99.38
			B	(0,6,4)	0.6513359	99.65
			C	(0,7,3)	0.6531318	99.38
			initial	(2,5,3)	0.6490498	-
10	4	30	A	(11,16,3)	0.6380394	99.67
			B	(11,16,3)	0.6380394	99.67
			C	(11,16,3)	0.6380394	99.67
			initial	(10,12,8)	0.6359135	-
11	5	10	A	(1,3,4,2)	0.3479317	99.42
			B	(1,3,4,2)	0.3479317	99.42
			C	(1,3,4,2)	0.3479317	99.42
			initial	(2,3,2,3)	0.3459187	-
12	5	15	A	(4,6,3,2)	0.9719334	99.08
			B	(4,6,4,1)	0.9708824	99.18
			C	(4,6,4,1)	0.9708824	99.18
			initial	(4,5,3,3)	0.9629590	-
14	6	130	B	(26,45,26,16,17)	0.4611404	102.99
			C	(23,38,47,14,8)	0.4448229	106.77
			initial	(22,27,25,31,25)	0.4749513	-

Table A.8. *Efficiency evaluation of initial allocation procedure of LIBA*

A: *SEVA*

B: *Non-SEVA with both big and small steps*

C: *Non-SEVA with only small steps*

Initial: *Initial allocation determined by our own procedure*

N: *Number of machines in the production line*

K: *Total fixed number of buffer slots that are to be allocated*

NOTE: *SEVA was not applied to Case 14*