MULTIPLE PART-TYPE SCHEDULING IN FLEXIBLE ROBOTIC CELLS

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

Gül Didem Batur

May, 2009

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.

Assoc. Prof. Dr. Oya Ekin Karaşan (Advisor)

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. Dr. Selim Aktürk

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. Dr. Serpil Erol

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

MULTIPLE PART-TYPE SCHEDULING IN FLEXIBLE ROBOTIC CELLS

Gül Didem Batur
M.S. in Industrial Engineering
Supervisor: Assoc. Prof. Dr. Oya Ekin Karaşan
May, 2009

This thesis considers the scheduling problem arising in two-machine manufacturing cells which repeatedly produce a set of multiple part-types, and where transportation of the parts between the machines is performed by a robot. The cycle time of the cell depends on the robot move sequence as well as the processing times of the parts on the machines. For highly flexible CNC machines, the processing times can be adjusted. As a result, this study tries to find the robot move sequence as well as the processing times of the parts on each machine that minimize the cycle time. The problem of determining the best cycle in a 2-machine cell is first modeled as a travelling salesman problem. Then, an efficient 2-stage heuristic algorithm is constructed and compared with the most common heuristic approach of Longest Processing Time.

Keywords: Robotic cell, CNC, flexible manufacturing systems, controllable processing times.

ÖZET

ESNEK ROBOTİK HÜCRELERDE ÇOKLU PARÇA TİPİ ÇİZELGELEMESİ

Gül Didem Batur
Endüstri Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Oya Ekin Karaşan
Mayıs, 2009

Bu tez çalışması tekrarlı olarak çoklu parça tiplerinden oluşan bir set üreten ve makinalar arası parça taşımalarının bir robot tarafından gerçekleştirildiği iki makinalı imalat hücrelerinde ortaya çıkan çizelgeleme problemini göz önüne almıştır. Hücrenin döngü zamanı robotun hareket dizisi yanısıra makinalardaki parçaların işleme zamanlarından etkilenmektedir. Yüksek esnekliğe sahip CNC makinalarında, makinalardaki işleme zamanları herhangi bir sırayla belirlenebilmektedir. Sonuç olarak, bu çalışma çevrim zamanını en aza indiren robotun hareket dizisiyle birlikte her makinadaki parçaların işleme zamanlarını bulmaya çalışmaktadır. İki makinalı bir hücrede en iyi çevrim zamanını belirleme problemi ilk olarak Gezgin Satıcı Probleminin özel bir şekli olarak modellenmiştir. Daha sonra, etkin bir 2 aşamalı sezgisel algoritma geliştirilmiş ve yaygın olarak kullanılan bir sezgisel olan En Uzun İşlem Zamanı algoritmasıyla karşılaştırılmıştır.

Anahtar sözcükler: Robotik hücre, CNC, esnek imalat sistemleri, kontrol edilebilir işleme zamanları.

To my dearest family...

Acknowledgement

I would like to express my most sincere gratitude to my advisor and mentor, Assoc. Prof. Dr. Oya Ekin Karaşan for all the trust and encouragement during my graduate study. She has been supervising me with everlasting interest and great patience for this research and has helped me to shape my future research career.

I am also grateful to Prof. Dr. Selim Akürk for his invaluable guidance, remarks and recommendations not only for this thesis but also for my future career. His understanding and sympathy was a great encouragement for me.

I am also indebted to Prof. Dr. Serpil Erol for accepting to read and review this thesis and for her invaluable suggestions.

I would like to thank to my friends Tuğçe Akbaş and Yüce Çınar for all the memories we shared over the last three years. Without their academic and most importantly morale support things would have been much more difficult.

I also would like to thank to Ali Gökay Erön and Sıtkı Gülten for their camaraderie and for being there whenever I needed. Furthermore, I would also like to thank to Hatice Çalık, Merve Çelen and Emre Uzun, and my officemates Zeynep Aydın and İhsan Yanıkoğlu for their closest conversations and for providing me a friendly environment to work.

I would like to thank all my friends again for their intimacy and positive mood in every moment of my graduate study.

Last but not the least, I would like to express my warm thanks to my family. Even from kilometers away, without their patience, understanding and help this study would never have been completed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The point of origin of this research is the increase in the level of automation in manufacturing industries. In order to be successful in today's highly competitive world, increasing productivity is an essential factor for manufacturing systems. Recent technological improvements opened new perspectives for industries. Since setup times are reduced to improve flexibility, as stated by Kamoun et al. [24], material handling time and cost become bottleneck, and efficient material handling becomes very important. Today, robots are used for this purpose extensively in automotive, electrical, electronic and mechanical engineering industries. Browne et al. [9] states that robots are installed in order to;

1- Reduce labor cost,

2- Increase output,

3- Provide more flexible production system,

4- Replace people working in dangerous or hazardous conditions,

5- Compensate for local shortage of skilled labor and,

6- Achieve more consistent control of quality.

A manufacturing cell consisting of a number of machines and a material handling robot is called a robotic cell. The efficient use of such cells necessitates the tackling of some important and challenging problems. Robotic cells can process lots that contain different types of parts. Generally, parts of different types have different processing times for a given machine. Multiple part-type problems are harder than their identical part-type counter problems even for small number of machines. The term Minimal Part Set (MPS) defines the set of parts containing the relative proportions of the part types the same as the relative proportions of the demand. The problem of interest is to find the robot move sequence and the part input sequence for the MPS that jointly minimize the cycle time. Within the scope of multiple part-type production, the decisions to be made include finding the robot move cycle and the part sequence that jointly minimize the production cycle time or the average steady-state cycle time. Additionally, within this literature, the problem of sequencing parts and robot moves in a robotic cell which is a flow-line manufacturing system and where the robot is used to feed machines in the cell is dealt with.

Since this thesis focuses on robotic cells of two identical machines which are capable to process all the parts, both the literature on the robotic cell scheduling when all of the machines are identical and working in parallel and the one on the flowshop systems which assume that each part being processed passes through the same sequence of locations from the input buffer (I), through machines $M_1, \ldots, M_m$ and finally into the output buffer (O) play important roles. Another relation between the studies on the robotic cell scheduling in parallel machine systems and the current study is that; setup operations considered by these researchers may be viewed as a kind of robot operation, like the travel time from its existing point to the related machine and the loading/unloading times.

One of the main points considered in this study is the flexibility property of the robotic cells. CNC machines possess operational flexibility and process flexibility by definition. Browne et al. [9] defined process flexibility as the ability to handle a mixture of operations. That is, if a part requires different operations

such as drilling, milling, etc., process flexibility states that one CNC can handle all of these operations. On the other hand, operational flexibility is defined as the ability to interchange the ordering of several operations for each part type. That is, the processing sequence of operations required for a part type can be changed.

In the current literature, the allocation of operations to machines is assumed to be fixed. However, the workstations in a robotic cell are predominantly CNC machines and they possess operational and process flexibility by definition. By the help of these two type of flexibilities, we are able to adjust processing times. Therefore, we assume that the operations constituting each part may be processed in any order. Furthermore, flexibility properties make it possible to allocate every operation on any one of the two machines. As the allocation of the operations changes, the processing times on the machines also change accordingly [4]. In this thesis we considered the problem of finding the optimal robot move cycle while simultaneously allocating the operations to the machines in order to jointly minimize the cycle time. Our results showed that allocation gave better results by using the machine capacities more effectively. Different from the existing literature, we do not assume the process of a part to compose of a number of operations. Instead, we consider the total processing time to be composed of unit times.

Another decision is related to the way that we observe the production of parts. We have two choices for this decision; we can consider the makespan problem or the cyclic production. Dawande et al. [13] showed that cyclic schedules which repeat a fixed sequence of robot moves indefinitely are the only ones that need to be considered in order to maximize the long-term average throughput. Thus, we choose to consider cyclic schedules.

The remainder of this thesis is organized as follows: In the following chapter, an extensive review of the literature is provided. In Chapter 3, the notation

and basic assumptions pertinent to this study are introduced and the problem is defined. Chapter 4 presents the proposed mathematical model. In Chapter 5 the algorithms developed will be defined and two specific algorithms will be distinguished. In Chapter 6 the proposed algorithms will be compared with each other and with the classical Longest Processing Time algorithm results considered in the existing robotic cell scheduling literature. Chapter 7 explains the contributions obtained by this study and presents the concluding remarks and future research directions.

Chapter 2

Literature Review

As manufacturers implement larger and more complex robotic cells, more sophisticated models and algorithms are required to optimize such systems. To meet this demand, there have been many studies. Some date as far back as the late 1970s, but the majority have been performed since 1990. Given the increasing importance of automated manufacturing, in this chapter, we review the literature related to this thesis under the headings of identical parts case, multiple parts case, parallel machines, and flexibility in robotic cells. Crama et al. [12], Lee et al. [30] and Dawande et al. [14] also provide surveys in this area.

2.1 Identical Parts

The identical parts robotic cell scheduling problem is simpler than its multiple parts counterpart since the part sequencing problem vanishes in identical parts case. A systematic study of the problem of finding optimal sequences of parts and robot moves to maximize the long-term throughput was started by Sethi et al. [36]. He set the agenda for most subsequent studies on cells. This paper can be considered as the initiation of the robotic cell scheduling literature. In this study, the objective was to maximize the throughput or in other words minimize the cycle time. One of the problems considered in this study was one

part type problem with two machines. For this problem they proved that the optimal solution is a 1-unit cycle. Since there are a total of two feasible 1-unit cycles in a 2-machine cell, they determined the regions of optimality for each of these cycles by comparing the cycle times of these two cycles with each other. Another problem considered in the paper was one part type problem with three machines. Determining the sequence of robot moves constituting a 1-unit cycle that minimizes the cycle time was considered. In this problem, only 1-unit cycles were considered since the analysis of the problem without this restriction was difficult and perhaps intractable. As a solution to this problem, a decision tree was constructed in order to determine the optimal policy. It was also proved that the number of one-part cycles in the m-machine case is exactly $m!$.

Crama et al. [10] considered the problem of a robotic cell with m machines. They showed that, when there is only one type of part to be produced and considering only 1-unit cycles, the problem could be solved in (strongly) polynomial time, when the number of machines is viewed as an input parameter of the problem. This generalized previous results established by Sethi et al. [36]. An algorithm was given which computes the cycle time of a schedule. Lastly, a dynamic programming approach was presented that solves the identical parts cyclic scheduling problem with the restriction that one unit is produced in each cycle in $O(m^3)$ time where m is the number of machines in the cell. Another result of that study was the derivation of the upper and the lower bounds on the optimal cycle time.

Hall et al. [20] considered 3 machine cells producing single part-types and proved that, the repetition of 1-unit cycles dominates more complicated policies that produce two units. The validity of the conjecture of Sethi et al. [36] for 3 machine robotic flowshops was established by Crama et al. [11]. Brauner and Finke [7] simplified this proof. In a later study, Brauner and Finke [8] proved that 1-unit cycles do not necessarily yield optimal solutions for cells of size four or large. They presented examples of such cases.

Dawande et al. [15] considered a different case of identical parts robotic cell scheduling problem. They studied the problem of finding the optimal robot move cycle that minimizes the cycle time in an m machine robotic cell. However, they considered only the 1-unit cycles. Differing from the literature, they assumed the robot travel time between any pair of machines to be constant, which was referred to as *constant travel time robotic cells*. They provided a polynomial time algorithm for finding an optimal 1-unit cycle.

Geismar et al. [16] considered the robotic cells consisting of a number of stages served by a single robot. Each part is processed at each stage; each part follows the same order through these stages. They investigated an important and intuitive subclass of cycles, called blocked cycles, and developed a general expression for their cycle times. They also derived a formula that determines in constant time how many parallel machines are needed for each stage in order for the cell to meet a specified throughput for this common case. Additionally, they identified instances in which the use of parallel machines would be a wasted expense.

One study with no-wait constraints is the study of Kats and Levner [25]. Such systems are required in some manufacturing systems such as plastic molding, electroplating and steel manufacturing where material handling is mainly done by an automated material handling device such as AGV's, hoists or robots. They considered an m-machine identical parts robotic cell scheduling problem with the objective of finding the 1-unit robot move cycle that minimizes the cycle time. They assumed that any machine may occur more than once in the processing sequence of parts. A polynomial algorithm which solves the problem in $O(K^5)$ was presented. Here K is the number of processing stages in the part's production. If the re-entrance constraint is relaxed, they showed that the same algorithm has complexity $O(m^4)$, where m is the number of machines. In a latter study, Levner et al. [31] considered the same problem without re-entrance constraints. They presented an algorithm which in turn improved the complexity of the previous one to $O(m^3 log m)$.

2.2 Multiple Parts

Since we focus on multiple part-type production in this study, related literature is essential for us. Within the literature of this subject, Sethi et al. [36] solved the part sequencing problem associated with two possible cycles in a 2-machine cell producing multiple-part types. They attempted to minimize the cycle time of an MPS. They showed that for one cycle the problem is trivial, whereas for the other cycle it is equivalent to a special case of the traveling salesman problem (TSP), which can be solved optimally using the polynomial time algorithm of Gilmore and Gomory [17].

Stern and Vitner [38] considered 2-machine multiple parts problem with the objective of minimizing the makespan, when the transportation time between the machines is job dependent. They showed that the problem is equivalent to an asymmetric travelling salesman problem and is NP-hard in the strong sense. For the same problem, if the transportation time between the machines is not job dependent, an $O(n^3)$ procedure that solves the problem based on the known Gilmore and Gomory [17] algorithm, was proposed by Kise et al. [26].

Again for the 2-machine multiple parts problem, Logendran and Sriskandarajah [32] considered three different layouts and established optimal robot sequences for these layouts. The problem of determining the optimal sequence of multiple part types was shown to be equivalent with a 2-machine no-wait flow shop problem, and was solved by Gilmore and Gomory's algorithm [17]. Besides the analysis of a single MPS, production of multiple MPSs was also given.

Hall et al. [20] considered the scheduling of operations in a robotic cell of two or three machines. For multiple part-type problems in a two-machine cell, they provided an efficient algorithm that simultaneously solves the robot move and part sequencing problems. Cycle time minimization problem in the multiple part-types case was shown to become one of deciding which parts to process under

the two possible robot move cycles, S_1 and S_2 and how to switch from one cycle to another, while simultaneously choosing the optimal part sequence. An $O(n^4)$ time algorithm which solves this problem optimally was also provided, where n defines the number of parts considered. They also proved that, for 3 machine cells producing single part-types, the repetition of 1-unit cycles dominates more complicated policies that produce two units.

Hall et al. [21] also considered the scheduling of operations in a manufacturing cell which is served by a robot. In a three machine cell producing multiple part-types, they proved that in two out of the six potentially optimal robot move cycles for producing one unit, the recognition version of the part sequencing problem is unary NP-complete. The general part sequencing problem not restricted to any robot move cycle in a three machine cell was shown to be unary NP-complete. They showed that in steady-state production of multiple-part types, the cell returns to the same state after producing either one or two MPSs.

Aneja and Kamoun [5] considered the problem of minimizing the long run average time in a 2-machine robotic cell like Hall et al. [20] and improved their algorithm which finds the optimal robot move sequence and the part input sequence. They modeled the problem as a special case of TSP and provided an algorithm of complexity *O(n logn)*.

Sriskandarajah et al. [37] developed criteria to assess the complexity of the part scheduling problem in larger ($m \geq 4$) cells. The part sequencing problems associated with the robot move cycles were classified into the following categories;

- sequence independent,
- capable of formulation as a traveling salesman problem (TSP), but polynomially solvable,
- capable of formulation as a TSP and unary NP-hard,

- unary NP-hard, but not having TSP structure.

As a consequence of this classification, they proved that the part sequencing problems associated with exactly $2m - 2$ of the $m!$ available robot cycles are polynomially solvable. The remaining cycles had associated part sequencing problems which were unary NP-hard.

Sriskandarajah et al. [37] proved that 3-machine multiple parts problem is intractable. Thus, Kamoun et al. [24] considered the scheduling problems arising in robot-served manufacturing cells in which the machines are configured in a flowshop that repetitively produces a family of similar parts. They developed and tested several heuristics for the general three- and four-machine cell, multiple-part type problems. They also identified and tested computationally effective heuristics for the general problem of determining both the robot move cycle and the part sequence. They considered the objective of minimizing the average steady-state cycle time for the repetitive production of minimal part sets (MPSs). They also studied how to design robotic cells for efficient performance by grouping machines into cells, identifying good part sequences, and providing appropriate size buffers between cells, in a larger manufacturing system. Suggestions for extending those approaches to larger cells were also provided.

Kamalabadi et al. [23] implemented the particle swarm optimization (PSO) algorithm for solving a cyclic multiple part-type 3-machine robotic cell problem.

2.3 Parallel Machines

Some researchers studied scheduling or sequencing problems in robotic cells or cells with server when all of the machines are identical and working in parallel. These researchers considered the problem of scheduling jobs with setup operations, where processing of a job is assumed to be performed on any one of the (mostly two) machines, since machines are identical. They did not need to make

any definitions as 'single' or 'multiple' part-types, and by taking different 'setup time' or 'processing time' constraints into account, they tried to construct some algorithms or show the problems defined are binary or unary NP-hard.

Koulamas et al. [28], Koulamas [27], Kravchenko et al. [29] and Hall et al. [22] studied the subject dealing with parallel machines for which each job requires a setup to be carried out, immediately prior to its processing, by a single server, with the processing executed unattended, and include some complementary results. Koulamas et al. [28] proposed a look-ahead heuristic for an environment with continuously arriving jobs for parallel machine scheduling with a common server. Koulamas [27] showed that the problem of minimizing forced idle time, which is defined as the amount of time when some machine is idle due to the unavailability of the server, in a 2-machine parallel system with single server is unary NP-hard and they gave local search and beam search algorithms.

Kravchenko et al. [29] presented a pseudopolynomial time algorithm for the case of two machines when all setup times are equal to one and showed that the problem is strongly NP-complete. In the more general case, Hall et al. [22] presented a complexity analysis for the problem where a robot is shared among several pieces of equipment for tool change and part setup purposes, under common scheduling objective functions. In the same paper, two heuristics of makespan scheduling are analyzed. Morton et al. [34] also considered the problem of an external but common server.

Abdekhodaee et al. [1] solved the equal length jobs case exactly using a sorting algorithm. Some heuristics for the general case were proposed and their performances were compared to published results. Abdekhodaee et al. [2] considered the computational complexity of two further special cases, namely equal processing time and equal setup time problems, and tested the performance of various heuristics for these cases. Abdekhodaee et al. [3] also considered the problem of minimizing the makespan for the manufacture of a given set of jobs, scheduling

two operation non-preemptable jobs on two identical semi-automatic machines, where a single server is available to carry out the first (or setup) operation. They built on the earlier work of regular problems, to deal with the general case. The results of this study together with the earlier work in Abdekhodaee et al. [1] and Abdekhodaee et al. [2] appeared to provide comprehensive heuristic solutions to the general problem.

2.4 Flexibility In Robotic Cells

Dealing with the robotic cell scheduling problem of identical parts, the studies of Akturk et al. [4], Gultekin et al. [18], and Gultekin et al. [19] considered the 'flexibility' properties. Akturk et al. [4] considered the two machine, identical parts robotic cell scheduling problem with operational flexibility. With this definition of the problem, each part has a number of operations to be processed and the problem is to allocate these operations to the machines and is to find the corresponding robot move cycle that jointly minimize the cycle time. The main result of the paper is that the optimal robot move cycle is not necessarily a 1-unit cycle as Sethi et al. [36] prescribes, but a 2-unit robot move cycle may also be optimal for some parameter inputs and they provided the regions of optimality for each robot move cycle. Gultekin et al. [18] dealt with the two machine, identical parts robotic operation allocation problem with tooling constraints. The operation allocation flexibility is said to be a direct consequence of assuming that the machines in the robotic cell are CNC machines as is the case for machining operations. It is assumed that some operations can only be processed on the first machine while some others can only be processed on the second machine due to tooling constraints. Remaining operations can be processed on either machine. The problem is to find the allocation of the remaining operations on the machines and the corresponding robot move cycle that jointly minimize the cycle time. As a solution to this problem, they proved that the optimal solution is either a 1-unit or a 2-unit cycle. They presented the regions of optimality for these robot move cycles. They showed that the study of Sethi et al. [36] becomes a special case of this one. Gultekin et al. [19] considered an m-machine robotic cell used for metal

cutting operations. The machines used in such manufacturing cells are CNC machines which are highly flexible. As a consequence, each part is assumed to be composed of a number of operations and each machine is assumed to be capable of performing all of the required operations of each part. They investigated the productivity gain attained by the additional flexibility introduced by the CNC machines. They defined a new class of robot move cycles, namely the pure cycles, which resulted from the flexibility of the machines and proved that, the set of pure cycles dominates all flowshop type robot move cycles. The results showed that these proposed cycles are not only simple and practical but also perform very efficiently as well. Assuming each machine has the capability of performing all of the operations of a part, flexibility leads to the possibility of new cycles [19].

2.5 Summary

Due to the increase in the level of automation in manufacturing industries, the use of computer controlled machines and automated material handling devices has become essential. As can be seen from the previous sections, there have been many studies related to the subject of optimizing robotic systems. Most of these studies assume that each part being processed passes through the same sequence of locations or that all the machines are identical and work in parallel. These types of assumptions in fact make works easier as there is no need for the decision of robot move sequence any more. On the other hand, some other studies mentioned that it is possible to allocate every operation of any part on any one of the two machines. However, these studies focussed on identical parts whereas we have extended this fact on multiple parts case. Considering all these studies together, we focus on flexible robotic manufacturing cells consisting of CNC machines and producing multiple part types. As far as we know, this is the first study to consider allocation possibility in multiple part-type robotic cell scheduling literature.

Chapter 3

Notation, Assumptions and Problem Definition

In this chapter, we define the notations which are used throughout this thesis and give a formal definition of our problem.

Throughout this thesis we focus on flexible robotic manufacturing cells consisting of CNC machines. Robotic cells can process lots that contain different types of parts. Generally, these parts have different processing times on a given machine. In accordance with just-in-time manufacturing, the relative proportions of the part types in each lot should be the same as the relative proportions of the demand. Consequently, researchers focus on cycles which contain the minimal part set (MPS) that has these same proportions. For example, if the demand for a company's three products is divided so that product A has 40 %, product B has 35 %, and product C has 25 %, the MPS has 20 parts: 8 of product A, 7 of product B, and 5 of product C. In many applications, robotic cells are used in repetitive or cyclic production of MPSs. In multiple part scheduling problem, an MPS, which is the smallest possible set of parts having the same proportions as the overall production target, is produced repetitively. The objective is to

minimize the average time to produce one MPS in a cyclic production environment. Throughput rate is defined to be inverse of that average time. In general, the cell processes k different part-types. In one MPS, r_i parts of type i are produced, where $i = 1, \ldots, k$. The total number of completed parts in a cycle is $n = r_1 + \ldots + r_k$. An *MPS cycle* is a cycle during which the MPS parts enter the system at input, get processed, leave the system at output, and the system returns to the same initial state. An MPS cycle can be defined by specifying the sequence in which the MPS parts enter the cell from input, and the schedule of operations to be performed on those parts within the cell. The sequence of MPS parts is called an *MPS part sequence* (or, simply a part sequence). An *MPS robot move sequence* (or simply a robot move sequence) is a sequence of the robot activities performed during an MPS cycle. The challenge in finding an MPS cycle with the minimum cycle time is twofold. Namely, two decisions need to be made: (a) choosing a robot move sequence, and (b) determining a part sequence. The objective of optimally scheduling the parts in a robotic cell is to find the MPS cycle with the minimum cycle time. This requires a *simultaneous* determination of a part sequence and a robot move sequence so as to minimize the cycle time [14].

Based on the definitions of operational and process flexibilities which state that one CNC can handle all of the operations of a part type and the processing sequence of these operations can be changed; we consider an in-line robotic cell of two identical machines which are capable of performing all the required processes. Figure 3.1 is given in order to represent such cells. Each part is assumed to have known processing times to be performed. By taking the advantage of flexibility property, we claim that robot may choose either to perform all the processing of a part completely on any one of the machines or to share the total time among the machines. In order to use robotic cell systems efficiently, problems including the scheduling of the robot moves and the determination of the machines to perform processing of each part should be solved. We try to find the parts to be processed on the machines by allocating the processes to them and finding the robot move cycle which will jointly minimize the cycle time. Throughout this study, we assume the processing times of parts to be integer valued.

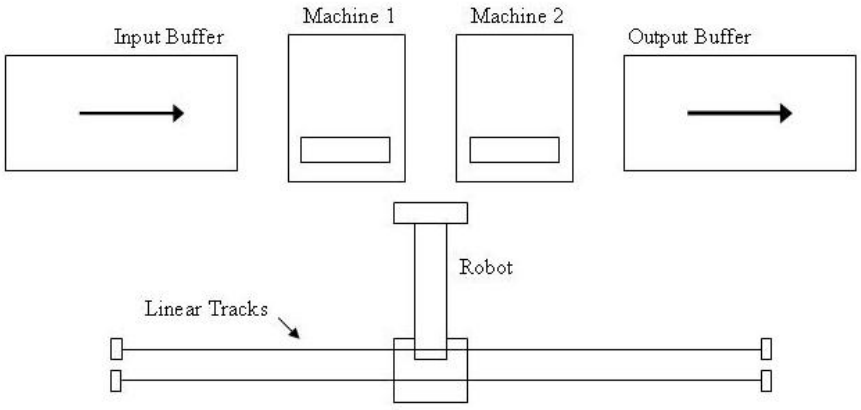

Figure 3.1: In-line robotic cell layout

Basic assumptions for our study that are common for most of the studies in the literature are as follows:

- All data are deterministic.
- Parts are always available at the input buffer and there is always an empty place at the output buffer.
- No buffer storage exists between the machines, each part is either on a machine or being handled by the robot.
- Neither the robot nor the machines can be in possession of more than one part at any time.
- The robot and the processing machines never experience breakdown and never require maintenance. Setup times are assumed to be negligible.
- No preemption is allowed in the processing of any operation.

As is mentioned above, our focus is on multiple part-type production. Thus, we need to solve both the problems of scheduling of parts and sequencing of robot moves for robotic cells. Researchers assume the allocated processing time values

on each machine to be constant and for given processing times seek for the optimum part sequence and robot move cycle minimizing the cycle time. However, as is originally considered by Gultekin et al. [18], we do not make such an assumption, meaning that allocation constitutes our third main problem.

Though we have multiple part-types, two identical parts belonging to the same part-type might have different allocations. Therefore; throughout this study, each part in the MPS is treated independently due to the allocation possibility.

Parameters related to this study are:

n : Total number of parts to be produced in the MPS.

P_j : Processing times of each part to be produced, $j = 1, \ldots, n$.

ϵ : Load/unload times of machines by the robot. Consistent with the literature we assume that loading/unloading times for all machines are the same.

δ : Time taken by the robot to travel between two consecutive machines. The robot travel time is assumed to be additive. That is, travelling from machine s to machine m is equal to $|s - m|\delta$, where $s, m \in \{0, 1, 2, 3\}$. We take I, the input buffer, to be machine M_0 and O, the output buffer, to be machine M_3.

Against this background, the objective to minimize is the long run average cycle time required for the repetitive production of one or more minimal part sets. While solving this problem, we determine the allocated processing times of parts on the machines together with the waiting and blocked times of machines for the parts. These values will be formally defined in forthcoming chapters.

For single part-type production, researchers have already considered allocation and flexibility properties; while for multiple part type production these topics have not been studied yet. Let us consider an example which will shed light

on our assumptions of process and operational flexibility for multiple part-type production.

Example 1 *Assume that we have* 3 *parts to be completed with corresponding processing times:* $P_1 = 87$, $P_2 = 84$, $P_3 = 57$. ϵ *and* δ *are 1 and 2 time units, respectively.*

If we assume that all the processes of a job need to be carried out on any of the two machines as in a parallel machines system, the optimal sequence given by the Gantt-chart in Figure 3.2 occurs. Under this restrictive assumption, an optimal solution of 172 (which is obtained by the mathematical model to be explained in the next section) is obtained. In this figure, R represents the movements of robot and 1, 2 are used to define the first and second machines, respectively. In the initial state of the system, machine 1 is empty and machine 2 is already loaded with part 2.

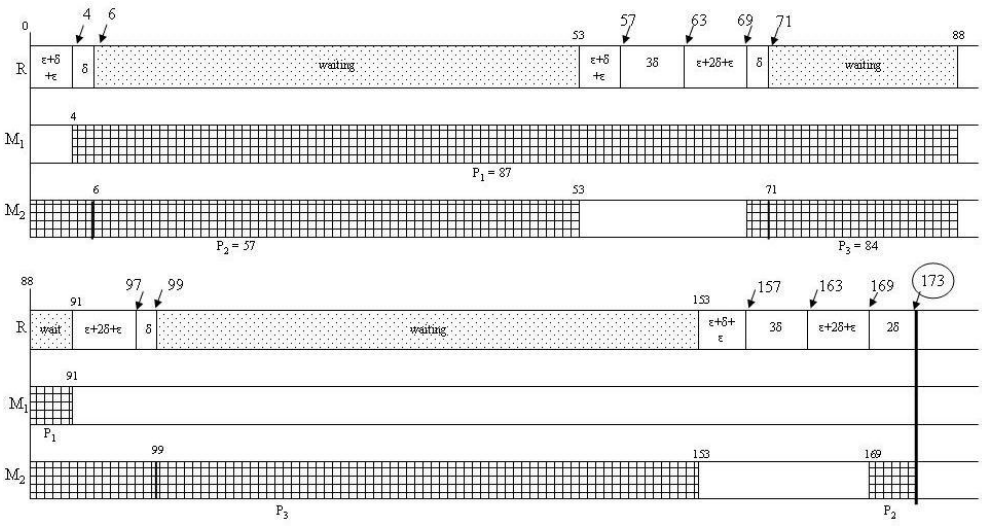

Figure 3.2: Example 1 without allocation

As a potential allocation scenario, we can share out the processing time of only one of the jobs on the machines, whereas we process all of the other jobs still completely on any one of the two machines. As can be seen from Figure 3.3, the main difference here is the processing time of the first part which is shared

among the two machines as the first one has 57 and the second one has 30 units of time. This approach gives a better result by a smaller cycle time and the new solution equals to 142.

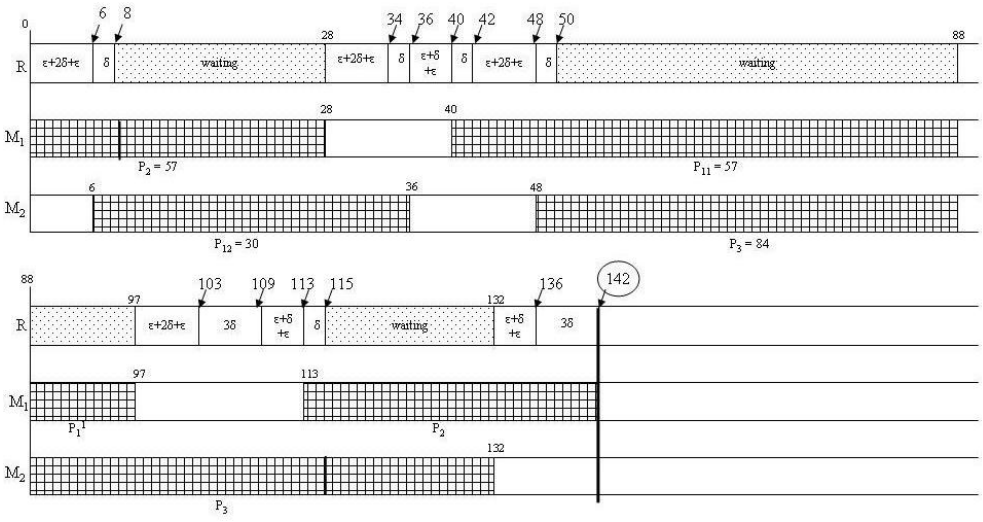

Figure 3.3: Example 1 with allocation

When we allow allocation, we cause the robot to have extra travel and load/unload movements. This decision results with the sum of δ and ϵ values getting higher and it may be thought that this would give a longer cycle time value. However, since the waiting times decrease and the machine capacities are used more effectively, a smaller cycle time is possible to be obtained as is realized for Example 1.

In summary, the improvement attained via allocation is approximately 17 %. We achieved that much gain only by changing processing time allocations for one part; without decreasing the processing time of parts, changing the machining conditions, or using a faster robot. It is important to notice that this improvement is achieved at no additional cost.

Chapter 4

Mathematical Model

As is mentioned before, in this study, we do not fix the operations to machines. Our aim is to find a solution that gives both the processing times on machines (i.e. which parts are processed on which machines and their corresponding processing times) and the order (i.e. the sequence) with which these parts are going to be processed. We claim that it is unnecessary to assume that either all the operations of a job are to be processed by one of the machines or the operations are to be totally shared by two machines. The solution that we are looking for should define the movements of the robot exactly; giving the part to be carried/loaded/unloaded together with the related machine.

In this chapter, we show how to model this problem as a special travelling salesman problem (TSP) in which the distance matrix consists of decision variables as well as parameters. A basic definition used in this formulation is the following one:

Definition 1 *Node i_k identifies the epoch that part i is on station k. The input buffer is denoted as station 1, first and second machines are denoted as stations 2 and 3 respectively, and the output buffer is denoted as station 4. During an MPS cycle, the machines may need to be visited twice, one for loading and one for unloading of the same part; since the robot may perform some other activities*

rather than to wait in front of the machine during the time that the part is being processed. Therefore, stations 5 and 6 are also created as the copies of stations 2 and 3, respectively, in order to account for any potential cyclic solution.

Movements of the robot are defined between the nodes. Two types of movements are possible; in the first one a part is carried from a station to another, whereas in the second one robot leaves a part on a station and goes to the other one to pick up a new part.

For the formulation, we have an arc set A and a node set N, which are represented as follows:

$$N = \{i_k : \ \ i = 1, \ldots, n, \ k = 1, \ldots, 6 \},$$
$$A = \{(i_k, j_l) : \ i_k, j_l \in N \text{ and the movement from node } i_k, \text{ where part } i \text{ is at station } k, \text{ to node } j_l, \text{ where part } j \text{ is at station } l, \text{ is possible}\}.$$

Throughout this thesis, we will use the notation k' to represent the copy of node k as is given by the following table;

k	k'
2	5
3	6
5	2
6	3

Table 4.1: Relation between stations k and k'.

In order to formulate the problem as a TSP, where a node is created for each of the six stations shown in Figure 4.1, we use the following parameters and variables:

Parameters:

n : Total number of parts to be produced in the MPS.

P_j : Processing times of parts to be produced, $j = 1, \ldots, n$.

ϵ : Load/unload times of machines by the robot.

δ : Time taken by the robot to travel between two consecutive machines.

$Cost_{i_k,j_l}$: Total time needed for the movement prescribed by the robot activity from where part i is at station k to where part j is at station l.

$$Wait_{i_k,j_l} : \begin{cases} 1 & \text{if there exists a potential waiting time for the movement of the robot from node } i_k \text{ to node } j_l, \\ 0 & \text{otherwise.} \end{cases}$$

Decision variables:

$$y_{i_k,j_l} : \begin{cases} 1 & \text{if robot goes from node } i_k \text{ to node } j_l, \\ 0 & \text{otherwise.} \end{cases}$$

P_{i_k} : Processing time of part i on station k, $k = 2, 3, 5, 6$.

$start_{i_k}$: Time that processing of part i is started on station k.

$comp_{i_k}$: Time that robot completed the activity related to node i_k.

w_{i_k} : Robot waiting time for part i in front of the station k.

$$z_{i_k} : \begin{cases} 1 & \text{if start time of processing of part } i \text{ on station } k \text{ is considered before its completion time within a cycle,} \\ 0 & \text{otherwise.} \end{cases}$$

$m2_{i_k}$: The part with which machine 2 is loaded, when robot is at node i_k.

$m3_{i_k}$: The part with which machine 3 is loaded, when robot is at node i_k.

For the variables $m2_{i_k}$ and $m3_{i_k}$, we say that they both are in the set $\{0, 1, \ldots, n\}$. When the variable equals to 0, it means that the machine is empty at that moment; whereas its equivalence to any value between 1 and n means that the machine is loaded with that particular part at node i_k.

C : Cycle time value.

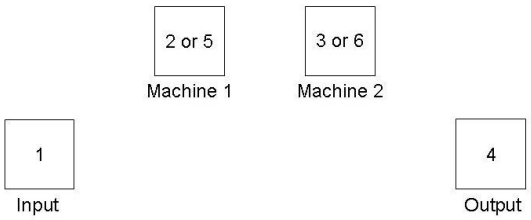

Figure 4.1: Stations used in the model

For this problem, we need to force the feasibility conditions which indicate that a machine that is already loaded cannot be loaded again and a machine that is already empty cannot be unloaded. Besides, some movements are unreasonable; for example, a part cannot be taken from the input buffer and left to the output buffer without any processing, cannot be taken from the output buffer, or cannot be left on the input buffer. All the possible movements are determined due to these considerations and some movements are forbidden within the cycle.

$Cost_{i_k,j_l}$ values represent the total time spent by the robot in going from node i_k to node j_l. For example, a movement from node i_1 to node i_2 corresponds to the situation that robot takes a part from the input buffer (ϵ), carries it to the second station which is the first machine (δ) and loads the part (ϵ); which make a total of $2\epsilon + \delta$. As it is mentioned before, we may represent two different situations according to the equivalence of parts i and j; because of this reason, cost values also differ from each other. For our problem, related $Cost_{i_k,j_l}$ values

are shown by Tables 4.2 and 4.3, where the first one corresponds to the states on which $i = j$ and the second one corresponds to the states on which $i \neq j$. Moreover, the movements with costs indicated by X's cannot be performed.

	(j_1)	(j_2) / (j_5)	(j_3) / (j_6)	(j_4)
(i_1)	X	$2\epsilon + \delta$	$2\epsilon + 2\delta$	X
(i_2) / (i_5)	X	X	$2\epsilon + \delta$	$2\epsilon + 2\delta$
(i_3) / (i_6)	X	$2\epsilon + \delta$	X	$2\epsilon + \delta$
(i_4)	3δ	X	X	X

Table 4.2: Costs of movements performed for the same part.

	(j_1)	(j_2) / (j_5)	(j_3) / (j_6)	(j_4)
(i_1)	X	X	X	X
(i_2) / (i_5)	δ	X	δ	X
(i_3) / (i_6)	2δ	δ	X	X
(i_4)	3δ	2δ	δ	X

Table 4.3: Costs of movements performed for different parts.

Waiting time is one of the main ingredients in the cycle time calculation. It occurs when the robot is ready to unload when processing of the currently loaded part has not been completed yet. This value equals to zero if the processing of the part is completed when the robot arrives in front of this machine to unload it or to the remaining processing time. It is represented as follows:

$$w_{j_k} = max\ \{0, P_{j_k} - v_{j_k}\},\ j = 1, \ldots, n,\ k = 1, \ldots, 6. \tag{4.1}$$

where j is the part loaded on station k and v_{jk} is the total activity time of the robot in between just after loading the machine corresponding to station k by part j and arriving in front of the same machine to unload it.

$Wait_{i_k,j_l}$ values are used to see for which movements robot may need to wait for part i in front of station k. As can be guessed, for the movements of different

parts (i.e. $i \neq j$) no waiting time will be observed, thus we will only define the states on which $i = j$. Waiting is possible for part i among the movements given by Table 4.4. For example, when we choose to go from node i_2 to node i_3, we will take part i from machine 2 and load it to machine 3, for which we may need to wait until the processing of the part on machine 2 is completed.

	(i_1)	(i_2) / (i_5)	(i_3) / (i_6)	(i_4)
(i_1)	0	0	0	0
(i_2) / (i_5)	0	0	1	1
(i_3) / (i_6)	0	1	0	1
(i_4)	-	-	-	-

Table 4.4: Waiting possibilities of movements performed for the same part.

After defining the parameters and variables of the model, we are now ready to proceed with our model. We will start by forcing the y variables to take on proper values. We need to ensure that when there is an incoming arc to a node, there must also be an outgoing arc from it. This fact is guaranteed by the following equation:

$$\sum_{j_l:(j_l,i_k)\in A} y_{j_l,i_k} = \sum_{j_l:(i_k,j_l)\in A} y_{i_k,j_l},\ \forall\ i_k \in N. \tag{4.2}$$

In our model, we allow some nodes not to be visited. Thus, the assignment constraints of a TSP become the following in our special case:

$$\sum_{j_l:(j_l,i_k)\in A} y_{j_l,i_k} \leq 1,\ \forall\ i_k \in N, \tag{4.3}$$

$$\sum_{i_k:(j_l,i_k)\in A} y_{j_l,i_k} \leq 1,\ \forall\ j_l \in N. \tag{4.4}$$

For this problem, we also need to satisfy all the parts to be processed in the system. This requires all the parts to be taken from the input buffer and left to the output buffer exactly once as is defined by Equation 4.5:

$$\sum_{j_l} y_{i_1,j_l} = 1,\ \forall\ i,\ \text{and}\ \sum_{j_l} y_{j_l,i_4} = 1,\ \forall\ i. \tag{4.5}$$

Without loss of generality, we refer node 1_1 as the starting point for this problem, meaning that system is assumed to start when robot is in front of the input buffer and ready to take part 1. In order to satisfy this fact, the following equation is used;

$$comp_{1_1} = 0. \tag{4.6}$$

We use the following Miller-Tucker-Zemlin [33] type constraints which help us calculate completion times of nodes according to the movements, costs and waitings, where $j_l \neq 1_1$ due to equation 4.6:

$$comp_{j_l} \geq comp_{i_k} + cost_{i_k,j_l} \cdot y_{i_k,j_l} - M(1 - y_{i_k,j_l}) + wait_{i_k,j_l} \cdot w_{i_k} \; \forall \; i_k, \; j_l. \tag{4.7}$$

This equation ensures that when an arc from node i_k to node j_l is used, completion time of node j_l is determined by the sum of completion time of node i_k and cost of the movement from node i_k to node j_l, and the waiting time value for the process history of part i at station k if such a waiting time is a possibility.

We have the following, processing time related equation, which states that all the processing of a part should be completed:

$$P_{i_2} + P_{i_3} = P_i, \text{ where } P_{i_2} = P_{i_5} \text{ and } P_{i_3} = P_{i_6}, \; \forall \; i. \tag{4.8}$$

The reason of the equivalence of processing time values corresponding to the nodes i_2, i_5 and i_3, i_6 is the equivalence of the stations 2, 5 and the stations 3, 6, respectively.

In order to define the relation between the machines and the processing times, we use the following equation. Since processes are performed only on the machines (not the buffers), these equations are constructed for only stations 2 and 3:

$$P_{i_k} \leq P_i \sum_{(j_l):(i_k,j_l)\in A \text{ or } (i_{k'},j_l)\in A} (y_{i_k,j_l} + y_{i_{k'},j_l}), \; i_k \in N \text{ s.t. } k = \{2,3\}. \tag{4.9}$$

Equation 4.9 states that when node i_k is visited on a tour, a processing with a value of at most the total processing time of part i can be performed on station k.

Besides, if node i_k is not visited throughout a solution, right hand side of equation 4.9 will be zero and no processing will be performed on station k, i.e. $P_{i_k} = 0$.

We define 'start' and 'comp' variables as the beginning of the processing on a station and the time that related movement is performed on a node, respectively. The beginning time of a part equals to the time that it is loaded on the related station. This value can be represented by either $start_{i_k}$ or $start_{i_{k'}}$ for part i according to the choice of whether it is loaded on station k or on k', respectively. Since these two stations are the copies of each other, we have the following result;

$$start_{i_k} = start_{i_{k'}}. \tag{4.10}$$

As can be understood from the above explanations, 'start' values are determined according to the 'comp' variables by the following equations, where $l \neq 4$ and $l \neq k$, $l \neq k'$:

$$\begin{aligned} start_{i_k} &\geq comp_{i_k} - M(1 - y_{i_l,i_k}),\ i_k \in N \text{ s.t. } k = \{2,3\}, \\ start_{i_k} &\geq comp_{i_{k'}} - M(1 - y_{i_l,i_{k'}}),\ i_k \in N \text{ s.t. } k = \{2,3\}. \end{aligned} \tag{4.11}$$

Equation set 4.11 shows that if there is an arc from node i_l to node i_k or i'_k, meaning that if part i is carried from station l to k or k'; processing of part i on machine corresponding to station k or k' starts at the time when the part is left on the machine which is equal to the time when the movement is performed. Since we cannot take any part from the output buffer, l cannot be equal to station 4.

Considering the relation between 'start' and 'comp' values again, by the help of the Equation set 4.11, we can say that 'start' value will be set equal to either the completion time for node i_k or the completion time for node $i_{k'}$; meaning that $start_{i_k} = comp_{i_k}$ or $start_{i_k} = comp_{i_{k'}}$.

In order to calculate waiting times we use another variable associated to each node which is represented by z_{i_k}. This variable helps to understand whether the

loading or unloading of a part is performed sooner in a given cycle. Figure 4.2 represents the two possible situations. Figure 4.2-a corresponds to the situation that system starts when part i is not loaded on any machine whereas in 4.2-b station k is loaded by part i.

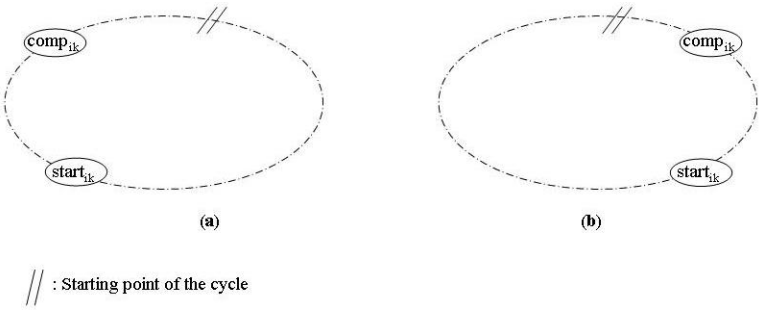

Figure 4.2: Choices on load/unload sequence

We determine the value of z by the following equations:

$$comp_{i_k} \geq start_{i_k} + P_{i_k} - M(1 - z_{i_k}),\ \forall\ i_k \in N,$$
$$start_{i_k} \geq comp_{i_k} + 4\epsilon + 6\delta - M z_{i_k},\ \forall\ i_k \in N. \tag{4.12}$$

In these equations $z_{i_k} = 1$ corresponds to $comp_{i_k} \geq start_{i_k} + P_i$, which gives Figure 4.2-a, whereas $z_{i_k} = 0$ corresponds to $start_{i_k} \geq comp_{i_k} + 4\epsilon + 6\delta$, which gives Figure 4.2-b. In this case, an already loaded part is to be unloaded and then loaded again, the value $4\epsilon + 6\delta$ which represents the total time that passes between the unload and load of a machine, is used.

For Equations 4.11 and 4.12, M represents a big number that the start or completion times cannot be more than this value. Since we need $4\epsilon + 6\delta$ units of time for each part to be loaded and unloaded on a machine, $M = \sum_i P_i + n(4\epsilon + 6\delta)$ can be used.

As it is mentioned before, waiting times occur when a part's processing is not completed by the time the robot arrives to unload it. In this formulation, waiting time is again calculated according to the start and completion times of nodes (i.e.

$start_{i_k}$ and $comp_{i_k}$ values) together with z_{i_k} variables:

$$\begin{aligned} w_{i_k} &\geq P_{i_k} - (comp_{i_k} - start_{i_k}) - M(1 - z_{i_k}),\ \forall\ i_k \in N, \\ w_{i_k} &\geq P_{i_k} - (C - start_{i_k} + comp_{i_k}) - M z_{i_k},\ \forall\ i_k \in N. \end{aligned} \tag{4.13}$$

As can be seen from Equation set 4.13, waiting time equals to $\max\{0, P_{i_k} - (comp_{i_k} - start_{i_k})\}$ if $z_{i_k} = 1$ and to $\max\{0, P_{i_k} - (C - start_{i_k} + comp_{i_k})\}$ otherwise. These results can be followed from Figure 4.2. For example, in the second case, total time passed from $start_{i_k}$ to $comp_{i_k}$ equals to the sum of $C - start_{i_k}$ and $comp_{i_k}$. Therefore, we compare this value with the total processing time. In order to determine for which movements we need to include the waiting time in the cycle time calculation, $Wait_{i_k,j_l}$ parameters are used.

Another set of equations is constructed in order to define the relation between the movements and the parts loaded on the stations. Considering the movement (i_k, j_l), there are two cases that can be observed, which are explained as follows:

Case 1. Robot activity is related to different parts, i.e. $i \neq j$.
This situation refers to the activity that the robot loads part i on station k and travels to station l for part j. Such a movement does not cause any part carriage. Therefore, no difference in terms of the loaded parts is observed for the machines.

Case 2. Robot activity is performed for the same part, i.e. $i = j$.
This situation refers to the activity that the robot takes a part from station k and travels to station l to load it. In order such a movement to exist, the following conditions should be satisfied;

- The machine corresponding to station l, which is the one to be loaded, needs to be empty at node i_k, for $l \neq 4$ since there is always an empty space in the output buffer by assumption.
- The machine corresponding to station k which is the one to be unloaded becomes empty at node j_l, for $k \neq 1$ since there is always some parts in the input buffer by assumption.

- This type of a movement results with no difference in terms of the loaded parts for the machines corresponding to the stations rather than k, k' and l, l'.

Since stations 1 and 4 correspond to input and output buffers, respectively; we check these relations only for stations 2, 5 and 3, 6 which correspond to the first and second machines, respectively again. We can show the equations that are used to satisfy these cases in the following classes:

Case 1. If a movement from node i_k to j_l exists and such a movement does not cause any difference in terms of the loaded parts on machines, $m2_{i_k} = m2_{j_l}$ and $m3_{i_k} = m3_{j_l}$ should be satisfied. These results are obtained by the following equations;

$$\begin{aligned} -m2_{i_k} + m2_{j_l} &\leq n(1 - y_{i_k,j_l}), \\ m2_{i_k} - m2_{j_l} &\leq n(1 - y_{i_k,j_l}), \end{aligned} \tag{4.14}$$

$$\begin{aligned} -m3_{i_k} + m3_{j_l} &\leq n(1 - y_{i_k,j_l}), \\ m3_{i_k} - m3_{j_l} &\leq n(1 - y_{i_k,j_l}). \end{aligned} \tag{4.15}$$

Case 2. If a part (part i) is carried from station k to station l;

- Machine corresponding to the station l (either 2^{nd} or 3^{rd} machine) needs to be empty. This means that its related value must be equal to 0 when robot is at the previous station.

$$m2_{i_k} \leq n(1 - y_{i_k,i_l}) \text{ or } m3_{i_k} \leq n(1 - y_{i_k,i_l}), \tag{4.16}$$

- Similar to the previous situation, machine corresponding to the station k (either 2^{nd} or 3^{rd} machines) becomes empty. This means that its related value must be equal to 0 when robot is at the next station.

$$m2_{i_l} \leq n(1 - y_{i_k,i_l}) \text{ or } m3_{i_l} \leq n(1 - y_{i_k,i_l}), \tag{4.17}$$

- In this case, no difference is observed for the other machine. Such a solution refers to equation set 4.14 or 4.15 according to the possibilities of $k, l \neq 2$ or $k, l \neq 3$, respectively.

Our aim in the whole formulation is to find the minimum cycle time value. In order to define this, we use the following equation:

$$C \geq comp_{i_k} + y_{i_k,1_1}.cost_{i_k,1_1}. \tag{4.18}$$

By the help of this constraint, we see the completion time of the last node which is equal to the completion time of the cycle. The reason of using node 1_1 here is that without loss of generality, we assume it as the starting point of the travel.

As a result we have the following mixed integer linear programming formulation:

$$\begin{aligned} \min \quad & C \\ \text{Subject to} \quad & (4.2) - (4.18). \end{aligned}$$

Besides, we can obtain the solutions when no-allocation is allowed in the above model only by adding the following constraint:

$$y_{i_2,i_3} + y_{i_2,i_6} + y_{i_3,i_2} + y_{i_3,i_5} = 0, \ \forall \ i, \tag{4.19}$$

which makes the transportation of parts between the machines impossible. Additionally, it is worthwhile to mention that one can easily adapt this model to an m-machine case, by redefining the nodes with respect to the additional stations.

Optimal solutions that correspond to the above TSP formulation for Example 1 are given by Figures 4.3 and 4.4, which correspond to the Gantt-charts given by Figures 3.2 and 3.3, respectively. These representations include the arcs giving the minimum cycle time value. The values written on the arcs are the waiting and travel time values between the nodes respectively.

As is mentioned before, node 1_1 is the starting point for both of the examples. The values written on the arcs represent the waiting and travel time values between the nodes respectively. We can obtain all the necessary information according to optimal y values leading to the assignments and sequences. The reader

can understand this relation comparing the figures along with the Gantt-charts given before. For example; in Figure 4.3, system starts when the first machine is empty and the second one is loaded by part 2. At time 4, robot loads the first machine by part 1 and goes to the second machine at time 6. As can be seen from Figure 3.2, processing of part 2 is completed at this moment and robot waits until time 53. At time 57, part is unloaded and robot goes to the input buffer at time 63. Part 3 is taken from input buffer and loaded on second machine at time 69. Robot now goes to the first machine to unload part 1 and waits until time 91, since the processing is not completed. Further steps can also be observed in the same manner.

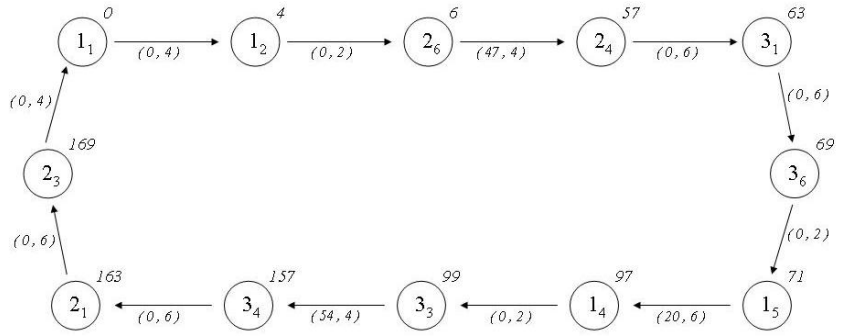

Figure 4.3: TSP representation of example 1 without allocation

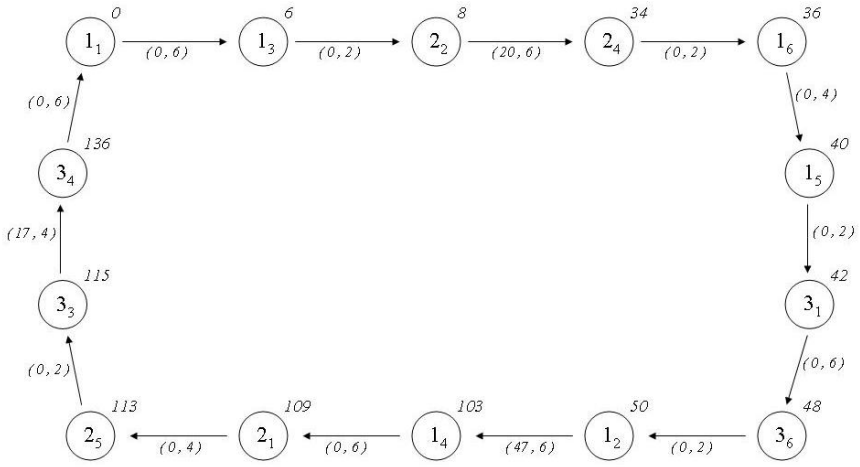

Figure 4.4: TSP representation of example 1 with allocation

Comparing Figures 4.3 and 4.4, we can see that some additional nodes are included in the latter one. This difference between the two solutions arise from the choice of allocation. As it was mentioned with Equation 4.19, on the one for which no allocation is allowed, movements of the same part between stations 2 and 3 (and their copies) cannot be used; whereas there is not such a restriction for the allocated solution. As can be seen from Figure 3.3, part 1 is processed on both of the two machines. Since there needs to be a carriage from machine 2 to machine 1, which corresponds to a movement from station 6 to station 5; all of the nodes 1_3, 1_6, 1_5 and 1_2 are used in Figure 4.4.

The TSP is a well known NP-Hard problem. The formulation above is more general than the classical TSP formulation and requires a great amount of computational effort even if the number of machines in the cell is small. Consequently, we focussed our attention on heuristic approaches and composed the algorithms defined in Chapter 5.

Chapter 5

Solution Procedure

The scope of this chapter is to explain the heuristic approaches that are constructed. Throughout the solution procedure we consider *machine*s instead of *station*s. From now on, notation m will be used instead of k, where $m = 1$ corresponds to $k = 2$ and 5 and $m = 2$ corresponds to $k = 3$ and 6.

In order to solve the scheduling problem of multiple part-types, we propose a two-stage algorithm in which the second stage works as an improvement approach on the first one. In the first stage, we find a solution trying to minimize blocked times and in the later one we try to improve the solution by changing the allocated processing time values (i.e. P_{j_1} and P_{j_2}) of one of the parts.

Blocked time is an important variable for this study. Different from the waiting time value, blocked time occurs when processing is completed before the robot arrives to the machine. It is represented as follows:

$$b_{j_m} = max\ \{0, v_{j_m} - P_{j_m}\},\ j = 1, \ldots, n,\ m = 1, 2. \tag{5.1}$$

where j is the part loaded on machine m and v_{j_m} is the total activity time of the robot in between just after loading machine m by part j and arriving in front of the machine to unload it.

Equation 4.1 which represents the waiting time value as the positive difference between the processing time of the part and the time that robot arrives to machine, can now be rewritten as the following:

$$w_{j_m} = max\ \{0, P_{j_m} - v_{j_m}\},\ j = 1, \ldots, n,\ m = 1, 2. \tag{5.2}$$

Considering equation 5.2 together with equation 5.1 which represents the blocked time value; it can be seen that there is a strong relation between waiting and blocked times. For any part, waiting time is observed on a machine if its processing has not been completed by the time the robot arrives to the machine to unload it; whereas blocked time is observed if the part's processing is completed before the arrival of the robot. Thus, the following equation holds for all parts: $w_{j_m} \cdot b_{j_m} = 0$, where j is the part loaded on machine m.

In our solution methodology, we first find a solution without any allocation, as in a parallel machine system. In this initial one, we try to minimize blocked times. Then we search for improvements by altering allocated solutions. With the help of the constraints to be defined in subsection 5.2, we determine a part that can be processed on both of the machines. As will be clarified, only one such part is enough. At the end of this stage, we have a new solution for which we know the parts to be processed on each of the machines, their sequences and related processing time values.

In parallel machine systems, the most popular approach to minimize the makespan is the Longest Processing Time (LPT) rule [35]. This rule orders the jobs in the order of decreasing processing times. Whenever a machine is freed, the largest job ready at the time will begin processing. This algorithm is a heuristic which schedules the longest jobs first so that no one large job will "stick out" at the end of the schedule and dramatically lengthen the completion time of the last job. Time complexity of LPT is known to be O(n logn) [6]. In order for the reader to see the solutions obtained by this approach, we use the following example for which no allocation is allowed.

Example 2 *Assume that we have* 6 *parts to be completed with corresponding processing times:* $P_1 = 97$, $P_2 = 123$, $P_3 = 4$, $P_4 = 18$, $P_5 = 20$, $P_6 = 26$. ϵ *and* δ *are both 4 units of time. Related Gantt-chart of the LPT solution is given by Figure 5.1.*

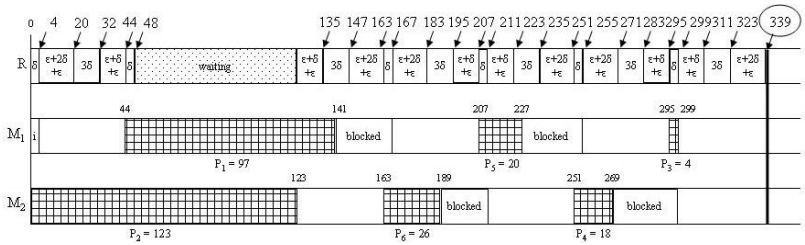

Figure 5.1: Gantt-chart of LPT approach for Example 2

.

In this example, system starts by processing the longest part and continues with the next longest one at each step until no more parts are left. It is possible to use either 1-MPS or more MPS's in order to reach to a steady state which defines a cyclic solution. LPT helps to provide equal work-loads for machines. As is mentioned before, our problem environment is also a special case of parallel machines systems. Therefore, for the rest of this study, we will compare our results with the ones obtained by LPT.

5.1 Stage 1: Construction Algorithm

In this section, we will explain our first algorithm which we call as 'Stage 1'. In this stage, we ignore the possibility of allocation for all the parts to be produced. The main aim is to minimize blocked times, by which we will be able to obtain a good assignment as well as the sequence. The importance of blocked time values arise from the importance of equal work-loads on machines; when we minimize blocked times on both of the two machines, it is more likely to obtain a fair assignment. We also checked the solutions obtained when waiting time values are considered as the first parameter and noticed that better solutions are

obtained for almost all examples minimizing the blocked times. However, considering waiting times as a second parameter, we also assure that waiting times will also be equally loaded. One of the main assumptions of this stage is that we consider only non-delay schedules. In a non-delay schedule, the machine to load is always the one that is just unloaded. This fact can be explained in other words as; no inserted idle times, which occur whenever a machine is deliberately kept idle in the face of waiting jobs, are allowed in such systems. The reason of this assumption is that no better solutions are expected to be obtained when a machine is kept idle although it can be loaded and some process can be performed on that time interval. The exception of the decision on machine selection is the first two parts considered, since we start to solve the problem when the system is empty.

Step-by-step representation of the Construction Algorithm, denoted as Stage 1, is as follows;

Step 1. Load the input buffer with an MPS.

pos $= 0$, $F_1 = F_2 = 0$,

Step 2. **If** the first machine is empty, as in line (8) of Algorithm 1,

{

Take the first part of the MPS from input buffer (ϵ),

Go to the first machine (δ),

Load the part on the machine (ϵ). (pos $= 1$ at this situation.)

Go to Step 2.

}

Else if the second machine is empty, as in line (16) of Algorithm 1,

{

Go to the input buffer (posδ).

}

Else

{

Determine the machine to unload (machine m) according to the finish times of the machines (F_1 and F_2), as described in lines (20)-(24) of Algorithm 1

Go to machine m $((\text{pos} - m)\delta)$,

Wait if the processing is not completed yet (w_{i_m}),

Unload the part (ϵ),

Go to the output buffer $((3 - m)\delta)$,

Put the part on the output buffer (ϵ),

Go to the input buffer for the new part (3δ).

}

Step 3. Determine the part to be loaded according to the possible blocked and waiting time values, with the Algorithm 2.

Step 4. Take the part from input buffer (ϵ),

Go to the machine m $(m\delta)$,

Load the part (ϵ). (Position of the robot is $\text{pos} = m$ at this situation.)

Step 5. If all the parts are assigned,

Go to Step 6,

Else

Go to Step 2

Step 6. If end-up policy is not satisfied,

Load a new MPS, increase the number of loaded MPSs by one,

Go to Step 2,

Else

Go to Step 7.

Step 7. Repeat the sequence and schedule determined so far until two successive results are found to be the same.

Step 8. Report cycle time value, total blocked and waiting time values of machines and blocked and waiting time values of parts for the solution obtained.

As it is given in Step 1 of the Algorithm Stage 1, Construction Algorithm starts when both machines are empty and all the parts in an MPS are ready to be processed. At this initial state, robot is in front of the input buffer. At Step 2, since the first machine is empty, the first part (according to the order in the input buffer) is taken from the input buffer and loaded on this machine. This movement is not a restrictive assumption since the solution obtained when we

solve the problem assigning the part on the second machine instead of the first is the mirror image of the one that is considered with the first machine. Going back to Step 2, next machine is selected to be the second one, since it is empty. At this point, which refers to Step 3, the part to load is determined according to the possible blocked and waiting times caused by the remaining parts. The word 'possible' indicates that at any decision point, blocked and waiting time values of each part left in the input buffer are calculated as if it is the one that is going to be loaded next. Comparing these values we first check blocked times and pick the one causing smallest blocked time value. In the event of equivalence of blocked times, we compare waiting times. If waiting times are also the same, we pick any one of the equally resulting parts randomly. After loading the determined part on machine 2, at Step 5 we check if all the parts are assigned or not. If not, we go back to Step 2 for the assignments of the remaining part(s). Since both the machines are not empty we consider the last 'else' part of Step 2 and from now on the decision of the machine to unload is taken according to the times that the loaded parts are to be completed. We know this value, since we know both the time that the part is loaded and its processing time. After unloading the selected machine, we determine the part to load as in Step 3 again. Machine and part selections and load/unload steps are repeated as long as there are parts in the input buffer, as it is mentioned in Step 5. When all the parts in set S are assigned and sequenced by Algorithm 1, checking the end-up criteria which will be explained below; algorithm either calculates the cycle time value repeating the assignments and sequences obtained so far, or loads a new set and goes on the part selection steps for this new set.

Since we try to reach to a steady-state cyclic solution, end-up conditions for the algorithm are either to reach to a state that the first part is assigned to the same machine while other machine's state is also same as the previous or to repeat the algorithm for a given number of MPS's. 'Given number of MPS's' means that we load input buffer with the same MPS for a few times and run the algorithm trying to reach to a state that was observed before. If we can find such a point at the end of Algorithm 1, algorithm stops without repeating the given number of

MPS's; but if we could not find such a steady solution even after a given number of MPS sets, algorithm does not search any more and take the resulting state as the starting one. As can be guessed, the state that we look for reaching for the second time is the load of the first part on the same machine; since load of it is the starting point of each set. This end-up policy can be seen from Figure 5.2. In chart 'a', system returns to its initial state after loading only one MPS, whereas in chart 'b' initial state is catched after two loads of the same MPS. In chart 'b', if we take 1 as the maximum MPS number, i.e. we assume that we cannot load the set more than once, our system would stop at point '1' and we would obtain the solution given by chart 'c' instead of the one in 'b'. In consequence of this policy, we obtain solutions producing either 1 or higher number of MPS's (mostly $2-MPS$). However, in our solutions, a $2-MPS$ solution does not refer to a solution of 2 times the part number in $1-MPS$; we use this term when we load a second MPS after the first one is completed while these two sets do not use the same sequences. This means that our choice of $2-MPS$ does not cause any need for extra inventory.

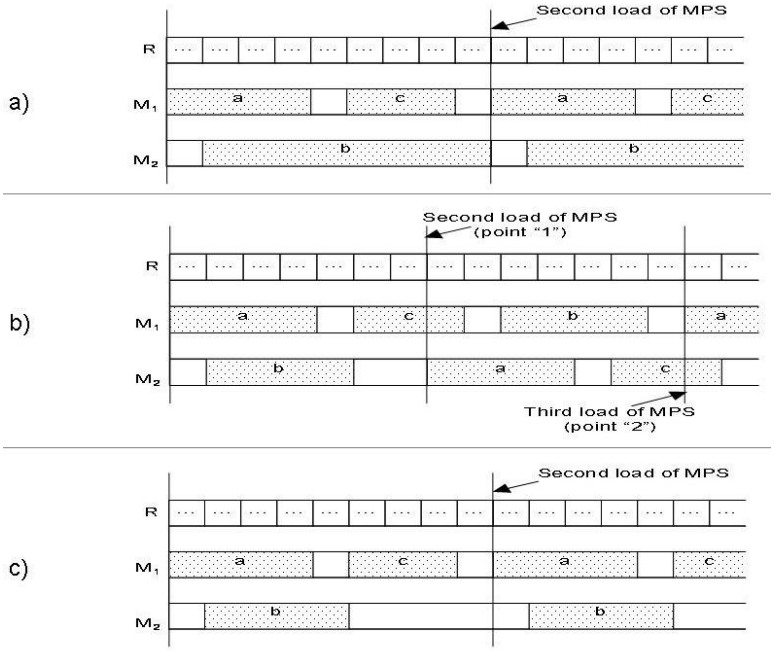

Figure 5.2: End-up policy

We use the following additional notations throughout the proposed algorithms;

t_{now} : Current time of the system following the activities of the robot.

S : Set of parts available in the input buffer.

S_1, S_2 : Set of parts assigned to machines 1 and 2 respectively.

m : Selected machine, where m=0,1,2,3.

pos : Current position of the robot, where pos=0,1,2,3.

s_{j_m} : Starting time of the processing of part j on machine m, where $j = 1, \ldots, n$ and $m = 1, 2$.

F_m : Completion time of the currently loaded part on machine m, where $m = 1, 2$.

These new notations are strongly related to the ones given in Section 4. We use t_{now} following the robot activities which were defined by $comp_{i_k}$ values, and keeping the result cumulatively. s_{j_m} corresponds to $start_{j_k}$ values where notation m of machines is used instead of notation k of stations. F_m is simply the sum of the start time and the processing time of the currently loaded part on machine m.

Pseudo-codes of the two algorithms, where the first one describes Stage 1 applied for the first set of parts and the second one the part selection procedure, are given by Algorithms 1 and 2.

Lemma 1 *Time complexity of the Construction Algorithm is* $O(n^2)$.

Proof.
Throughout the algorithm, we assign the parts on machines according to some calculations which are previously defined. Starting from the second part, we check all the remaining parts for this decision. In other words; we consider $n-1$ possibilities for the second part, $n-2$ possibilities for the third one, ... and

2 possibilities for the $(n-1)^{th}$ part. Since the sum of these values equals to n^2-n-1, complexity equals to $O(n^2)$. □

Algorithm 1 First MPS Solution

Input: $S=\{1,\ldots,n\}$, ϵ, δ, $P_1,\ldots,P_n$.
Output: S_1, S_2, w_{j_1}, w_{j_2}, b_{j_1}, b_{j_2}.
$S_1=\emptyset$, $S_2=\emptyset$,
$w_{j_1}=0$, $w_{j_2}=0$,
$b_{j_1}=0$, $b_{j_2}=0$,
$t_{now}=0$, pos $=0$, $F_1=0$, $F_2=0$,
while $S\neq\emptyset$ **do**
  if $S_1=\emptyset$ **then**
    $m=1$,
    $S_1=S_1\bigcup\{1\}$, $S=S-\{1\}$,
    $t_{now}=t_{now}+2\epsilon+\delta$,
    pos $=1$,
    $s_{1_1}=t_{now}$, $F_1=s_{1_1}+P_1$,
    $current_1=1$,
  end if
  if $S_2=\emptyset$ **then**
    $m=2$,
    $t_{now}=t_{now}+\text{pos}\delta$,
  else
    if $F_1\leq F_2$ **then**
      $m=1$,
    else
      $m=2$,
    end if
    $t_{now}=t_{now}+|\text{pos}-m|\delta$,
    $w_{current_{m\,m}}=\max\{0,F_m-t_{now}\}$,
    $t_{now}=t_{now}+w_{current_{m\,m}}+\epsilon+(3-m)\delta+\epsilon+3\delta$,
  end if
  PartSelect,
  $best\leftarrow$ Part returned by **PartSelect**,
  $t_{now}=t_{now}+2\epsilon+m\delta$,
  $S_m=S_m\bigcup\{best\}$, $S=S-\{best\}$,
  pos $=m$,
  $s_{best_m}=t_{now}$, $F_m=s_{best_m}+P_{best}$,
  $current_m=best$
end while

Algorithm 2 PartSelect

$\bar{B} = M$, $\bar{W} = M$,
for $i \in S$ **do**
  $t_{now} = t_{now} + 2\epsilon + m\delta$,
  $\text{pos} = m$,
  $s_{i_m} = t_{now}$, $F_m = s_{i_m} + P_i$,
  if $F_m \leq F_{(3-m)}$ **then**
    $w_{i_m} = \max\{0, P_i - t_{now}\}$,
    $t_{now} = t_{now} + w_{i_m} + 4\epsilon + 6\delta + \delta$,
    $b_{i_{|1-m|}} = \max\{0, t_{now} - F_{(3-m)}\}$
    $B = b_{i_m}$, $W = w_{i_m}$,
  else
    $t_{now} = t_{now} + \delta$,
    $\text{pos} = (3 - m)$,
    $b_{i_{|1-m|}} = \max\{0, t_{now} - F_{(3-m)}\}$,
    $t_{now} = t_{now} + \max\{0, F_{(3-m)} - t_{now}\} + 4\epsilon + 6\delta + \delta$,
    $b_{i_m} = \max\{0, t_{now} - F_m\}$
    $B = b_{i_{|1-m|}} + b_{i_m}$, $W = w_{i_m}$,
  end if
  if $B < \bar{B}$ **then**
    $\bar{B} = B$,
    $\bar{W} = W$,
    $j = i$,
  else if $B = \bar{B}$ **then**
    if $(W < \bar{W})$ **then**
      $\bar{B} = B$,
      $\bar{W} = W$,
      $j = i$,
    end if
  end if
end for
return j

5.1.1 Numerical Example For Construction Algorithm

We will use Example 2 in order to ensure that the algorithm is clearly understood. As is mentioned before, algorithm starts by loading the first part on the

first machine; thus robot loads P_1 on the first machine, which will take a total of $2\epsilon + \delta = 2(4) + 4 = 12$ units of time. Time of the system is 12 and robot position is pos = 1.

Since second machine is empty, it is chosen to be the machine to load and robot goes to the input buffer (posδ). Time of the system is $12 + 4 = 16$ and robot position is pos = 0. Start and finish times of the part are as follows:

$$s_{i_m} = s_{1_1} = t_{now} = 12,$$
$$F_m = F_1 = t_{now} + P_i = t_{now} + P_1 = 12 + 97 = 109,$$

Now we need to determine which part to be loaded, thus we will calculate possible blocked and waiting time values for the remaining part (i.e. parts 2 and 3). These observations can also be followed from Figure 5.3.

a) If we select part 2 ($P_2 = 123$), following situations will be observed:
Robot will take the part from input buffer (ϵ), go to the second machine (2δ), load the machine (ϵ); which will make a total of $2\epsilon + 2\delta = 2(4) + 2(4) = 16$ unit time and system time will be $16 + 16 = 32$. Robot position is pos = 2. Start and finish times of the part are as follows:
$s_{i_m} = s_{2_2} = t_{now} = 32,$
$F_m = F_2 = s_{i_m} + P_i = s_{2_2} + P_2 = 32 + 123 = 155,$
Since both the machines are loaded and $F_1 \leq F_2$, unload will be performed for the first machine. Robot will go to the machine ($|\text{pos} - m|\delta = |2 - 1|\delta = \delta$), wait until processing of the part is completed (i.e. until $F_1 = 109$, meaning 73 unit times of wait), take the part (ϵ), go to the output buffer ($|\text{pos} - m|\delta = |1 - 3|\delta = 2\delta$), and leave the part ($\epsilon$); these will make a total of $w_{1_1} + 2\epsilon + 3\delta = 73 + 2(4) + 3(4) = 93$ and system time will be $32 + 93 = 125$. Robot position will be pos = 3.
Robot will then go to the input buffer ($|\text{pos} - m|\delta = |3 - 0|\delta = 3\delta$), take a new part (which has not been determined yet) (ϵ), go to the first machine (δ), load the part (ϵ). System time will increase by $2\epsilon + 4\delta = 2(4) + 4(4) = 24$ and will be $125 + 24 = 149$.

Although we do not know the processing time of the newly loaded part, we check at this point if there occurred any waiting or blocked time for machine 2. Robot goes to the second machine which will make system time $149 + \delta = 149 + 4 = 153$ and this time is compared with the finish time of the already loaded part. F_2 is known to be 155, so we have $|155 - 153| = 2$ units of waiting time at this moment.

As a result, this choice will cause 75 units of waiting and 0 units of blocked time.

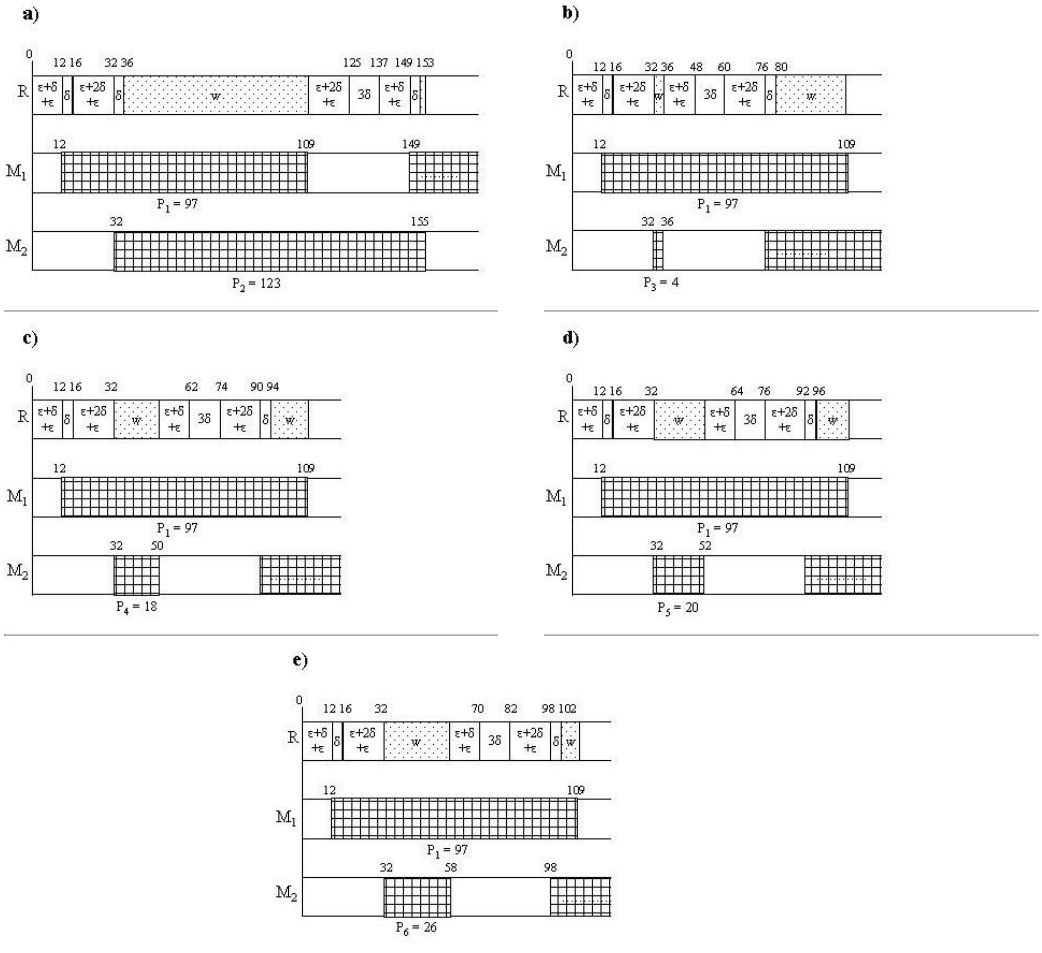

Figure 5.3: Part selection for Example 1 in Stage 1

With the help of similar observations for all the remaining part, we obtain

the following results:

b) If we select part 3 ($P_3 = 4$), this choice will cause $4 + 29 = 33$ units of waiting and 0 units of blocked times.

c) If we select part 4 ($P_4 = 18$), this choice will cause $18 + 15 = 33$ units of waiting and 0 units of blocked times.

d) If we select part 5 ($P_5 = 20$), this choice will cause $20 + 13 = 33$ units of waiting and 0 units of blocked times.

e) If we select part 6 ($P_6 = 26$), this choice will cause $26 + 7 = 33$ units of waiting and 0 units of blocked times.

Since all the possible blocked and waiting time values are determined to be the same, robot will select part 3 randomly and take the part from input buffer, go to the second machine, load the machine, wait until processing of the part is completed, take the part, go to the output buffer, leave the part and go to the input buffer for the new part. This point is again a decision point for the algorithm. According to the possible blocked and waiting time values again, this time robot picks part 2 and loads on the second machine. These part selection steps continues until all the parts are assigned. By the time we have finished the assignments of all the parts in the first MPS, we have also reached to a state that was observed before; so we end-up the algorithm having obtained the sequence and the schedule. Now we will determine the cycle time value of this result by repeating the assignments and the order until we reach to a steady solution. All the movements and processes of this example can be seen from Figure 5.4. The time that passed until the end-up point is $295 - 12 = 283$. We repeat the determined sequence and scheduled movements and reach the same state at time 590, for which $590 - 295 = 295$ units of time have passed. Third repetition finished at time 885, by which we have counted again $885 - 590 = 295$ units of time. Thus, the solution is reported as 295.

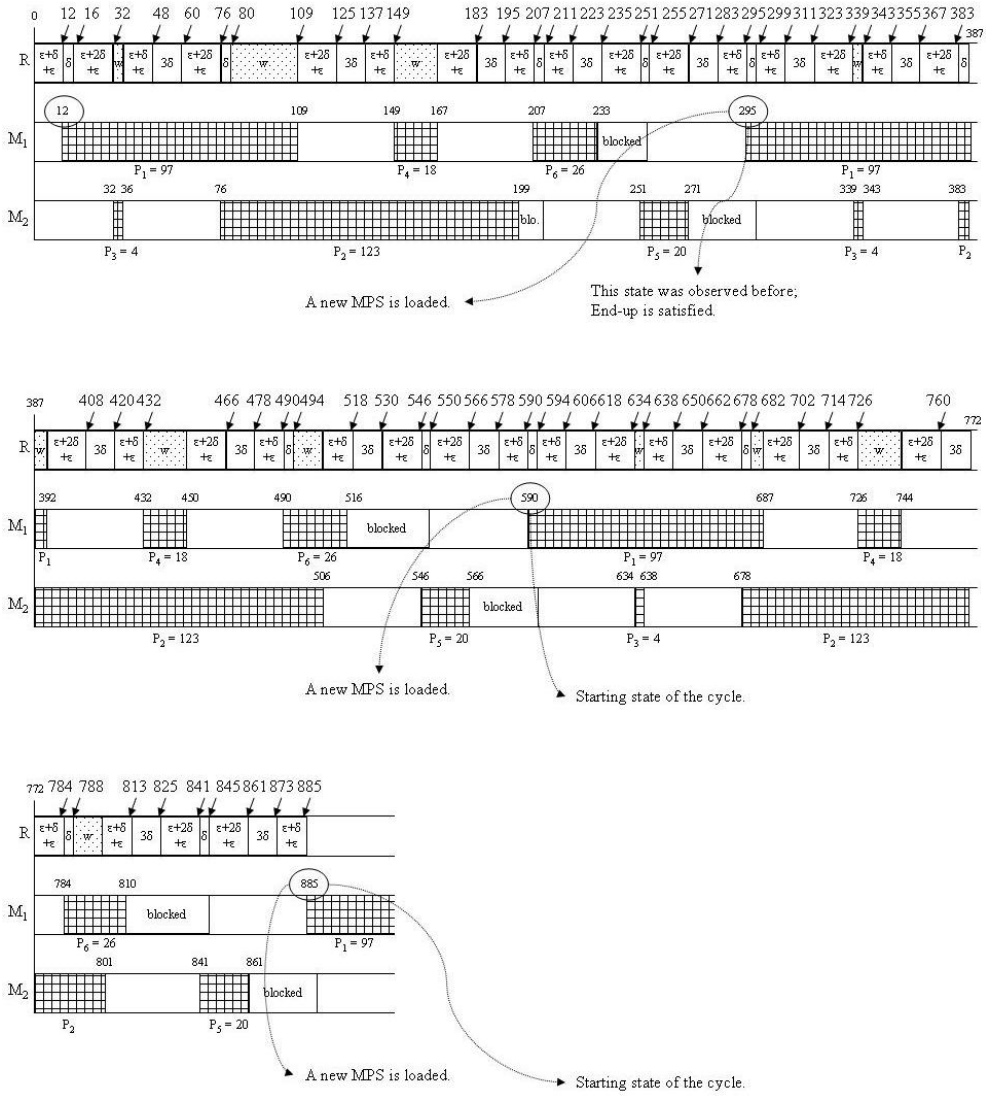

Figure 5.4: Gantt-chart of Stage 1 for Example 2

.

5.2 Stage 2: Allocation Algorithm

As is mentioned before, one of the main topics of this study is allowing allocation. Although, for the first stage, we assume the parts to be processed on only one of the machines; for this second stage, we try to find better solutions by letting one part to be processed on both machines. In this section, we give the

steps of this algorithm.

In order to reach our goal, we first decide on which part to allocate and then determine the processing time values on each machine for this allocated part. For the first decision, looking at the total blocked time values of machines (B_m) from the solution obtained in stage 1, we select the machine that has larger blocked time as the one that extra processing time will be added (machine x). The part to allocate (part i) will be the one that causes the largest blocked time value on machine x among the parts processed on the other machine (machine y in the initial solution, where $y \neq x$). Letting b^*_{jm} be the blocked time values observed on machine m in the same time interval with the processing and unload of part j, we pick part a to allocate where $B_a = \max\{b^*_{jm}\},\ m \neq x$. This part selection policy can be seen from Figure 5.5.

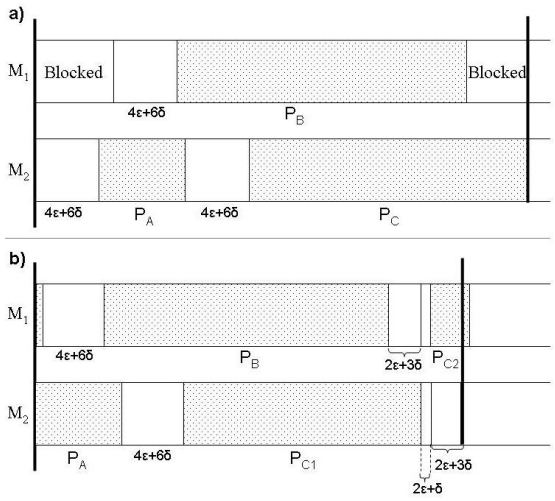

Figure 5.5: Part selection policy for Stage 2

In the first chart of the figure, part C is the one causing the biggest blocked time (which is also the only blocked time) on the first machine, thus it is chosen to be allocated. After loading the part on the second machine and unloading the first machine as before we unload part C without waiting all the processing is completed and load it on the first machine to perform the rest of its processing. As can be seen, allocation shortened the cycle time value filling the blocked time

observed in the initial solution.

After choosing the part, we should determine how we allocate the processes on machines. We know the total processing time of part i which is equal to P_i and need to calculate P_x which determines total processing time to be allocated on machine x. Before explaining the constraints defined for this aim, we will show the notations and define the important values for allocation problem.

Additional notations used at this stage:
Parameters:
P_j : Processing times of parts to be produced, where $j = 1, \ldots, n$.
B_d : The difference between total blocked times of the machines.
B_a : Longest blocked time value observed on machine x.
Decision variable:
P_x : Processing time to be allocated on machine x, which is completed by machine y in the 'stage 1' solution.

$4\epsilon+4\delta$ and $2\epsilon+4\delta$ are two important values for the allocation problem. $4\epsilon+4\delta$ is the extra time for machine x since the machine was not supposed to work for that part in the solution obtained by Stage 1. When we allocate a part, following movements will be performed immediately after the unload of machine x: go to machine y from the output buffer (δ or 2δ, for $x = 1$ or $x = 2$ respectively), unload the part to be allocated from machine y (ϵ), go to machine x (δ), load the part on machine x (ϵ), unload the part (ϵ), go to the output buffer (2δ or δ, for $x = 1$ or $x = 2$ respectively), load the part on output buffer((ϵ)).

$2\epsilon + 4\delta$ is a twofold important value. First, it represents the time that passes between the load of the allocated part on machine y and the time that robot comes back to unload it after unloading the previous assigned part on machine x which means to get machine x ready for the allocated part [travel time to machine x from machine y (δ), unload of part from machine x (ϵ), travel time to

output buffer (2δ if machine x is the first one or δ if machine x is the second one), load of part on output buffer (ϵ), travel time to machine y (2δ if machine y is the first one or δ if machine y is the second one)]. Its second meaning is to be the time that passes between the load of the allocated part on machine x and the time that robot comes back to unload it after loading the next assigned part on machine y [travel time to input buffer from machine x (δ if machine x is the first one or 2δ if machine x is the second one), unload of part from input buffer (ϵ), travel time to machine y (δ if machine y is the first one or 2δ if machine y is the second one), load of part on machine y (ϵ), travel time back to machine x (δ)].

We can now define the necessary conditions used to find out how we should perform allocation.

Lemma 2 *Allocation is useful if and only if there exists a P_x value satisfying the following conditions:*

$$\begin{aligned} &\textit{i)}\ P_x \leq B_d - (4\epsilon + 4\delta), \\ &\textit{ii)}\ \max\{0, P_i - P_x - 2\epsilon - 4\delta\} + \max\{0, 2\epsilon + 4\delta - P_x\} \leq B_a - P_x, \\ &\textit{iii)}\ 0 \leq P_x \leq P_i. \end{aligned} \tag{5.3}$$

Where P_i is total processing time of the chosen part, P_x is the time we need to determine, B_d is the difference of total blocked times of the machines, B_a is the blocked time on machine x that is caused by the chosen part.

Proof.
For the first equation, the significance of $4\epsilon + 4\delta$ arises from the fact that it is a newly added time for machine x which can be seen as the fixed cost of allocation for machine x. The sum of this value and the allocated processing time on machine x (P_x) should not be more than the blocked time difference (i.e. the excess of blocked time on machine x). In such a case, cycle time value of machine x in the new solution obtained becomes longer than the previous, which is a worse solution. This value can be followed from Figure 5.5, as the sum of $2\epsilon + 3\delta$ and $2\epsilon + \delta$. We observe from the same figure that this condition needs to be considered determining P_x values as the cycle time can be shortened only

if the sum $P_x + 4\epsilon + 4\delta$ is less than the blocked time value.
For the second equation, it was mentioned before that $2\epsilon + 4\delta$ is the time needed for unload of machine x and travel to machine y for the allocated part. If the process left on machine y is not completed by the time robot arrives the machine, in other words, if processing time value is larger than $2\epsilon + 4\delta$; there will occur waiting time for machine y which is equivalent to blocked time for machine x. In a similar manner, $2\epsilon + 4\delta$ also represents the time needed for taking the allocated part from machine y to machine x and coming back to this machine after loading a new part on machine y. If the process assigned to machine x is already completed by the time robot arrives the machine, in other words, if processing time value is smaller than $2\epsilon + 4\delta$; there will occur blocked time for machine x. Considering these explanations, we can say that left side of the equation gives us the total blocked time value observed on machine x for the allocated part after allocation is performed. We know that B_a is the blocked time caused by the chosen (to be allocated) part and P_x is the allocated time value. So, the right side of the equation gives us the available time that can be used for other movements after part is assigned. These observations can be seen from Figure 5.6. This equation satisfies that, the blocked times occuring after allocation is performed should not be more than the initial solution. □

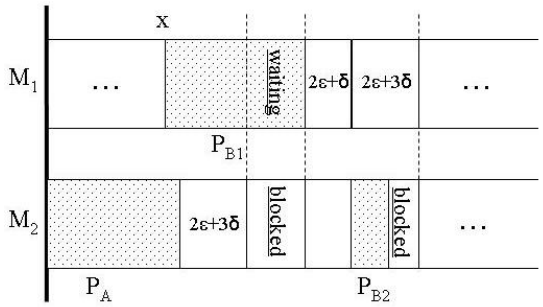

Figure 5.6: Effect of the value $2\epsilon + 3\delta$ on allocation

By the conditions defined in Lemma 1, we obtain a range for P_x value. After this point, we tried two ways to determine the exact value among the ones satisfying the conditions;
i) We take the one in the middle of the range obtained,

ii) We take the one closest to $2\epsilon + 4\delta$, in order to obtain minimum blocked time value for machine x.
In most cases, both of these two choices gave the same result; however, since we obtained better results by the second one for few cases, we use the value $2\epsilon + 4\delta$.

If there does not exist any part that satisfies the Lemma 1, there is no need to perform an allocation. Else, we follow the same assignments obtained in Stage 1 until part j is loaded on machine y. Once part j is loaded (with a processing time value of $(P_j - P_x)$ instead of P_j); we go to machine x, get prepared for part j by unloading the machine and without loading the next assigned part, we go to machine y, wait if necessary, unload this machine and take the allocated part to machine x (with a processing time value of P_x). After this point, we go back to the assignments of Stage 1 again; go to the input buffer, take the next assigned part for machine y, go to the machine and load the part. Using the same machine selection policy described for Stage 1, and following the assignments obtained in Stage 1; we perform load/unload activities for all the parts. In order to find a steady solution, the sequences determined so far are repeated until two successive results are found to be the same.

For the reader to get the insight of the Allocation Algorithm, we also give a step-by-step representation:

Step 1. Determine machines x and y according to the machines' blocked time values obtained in Stage 1.

Step 1.1. Calculate I_d value, which represents the difference of blocked times of machines.

Step 2. Determine the part to be allocated according to the blocked time values of the parts processed on machine x in the solution of Stage 1.

Step 3. Determine the processing time value to be allocated on machine x according to Lemma 2.

Step 4. If $P_x > 0$

Go to Step 5,

Else

End the algorithm without allocation,

Step 5. Perform the assignments of part j.

Step 5.1. Follow the same assignments obtained in Stage 1 until part j is loaded on machine y.

Step 5.2. After part j is loaded (with a processing time value of $(P_j - P_x)$), go to machine x, wait if necessary, unload the machine, go to machine y, wait if necessary, unload the machine, go to machine x, load the part on the machine (with a processing time value of P_x).

Step 6. Following the assignments of Stage 1 again; go to the input buffer, take the next assigned part for machine y, go to the machine and load the part.

Step 7. Using the same machine selection policy described for Stage 1, and following the assignments obtained in Stage 1; perform load/unload activities for all the parts.

Step 8. Repeat the sequence and schedule determined so far until two successive results are found to be the same.

Step 9. Report cycle time value, total blocked and waiting time values of machines and blocked and waiting time values of parts for the solution obtained.

Lemma 3 *Time complexity of the Allocation Algorithm is $O(n)$.*

Proof.

In this algorithm, we determine the part to allocate according to the conditions in Lemma 2. Since the necessary calculations are performed once for each of the n parts, they take a total of O(n) time. □

One of the main points for this stage is that; we are not guaranteed that we will be able to find a better solution with an allocation. Besides, the success of this method depends mostly on the choice of assignments and sequence. Now, we will apply the recently defined steps of the allocation algorithm on an example.

5.2.1 Numerical Example For Allocation Algorithm

Similar to stage 1, we consider Example 2, for which Gantt-charts of LPT and Stage 1 solutions are given in Section 5 by Figures 5.1 and 5.4, respectively.

Considering Stage 1 of the example, total blocked time value of machine 1 is 34 and of machine 2 is 28. Since the difference between the machines equals to 6 and $(4\epsilon + 4\delta)$ equals to 32, due to the first condition in Lemma 2, allocation is impossible.

We now implement the Allocation Algorithm on the LPT solution of Example 2. As can be determined from Figure 5.1, total blocked time value of machine 1 is 98 and of machine 2 is 52. Since the first one has more blocked time than the second one, we set the first machine as machine x and the other one as machine y. As is mentioned, part to be loaded is determined according to the longest blocked time value on machine x, which is part 4 causing 44 units of blocked time with processing time of 18. Now we can start the calculations. I_d value equals to $|98 - 52| = 46$ and the first condition in Lemma 1 is determined as follows:

$$P_x \leq B_d - (4\epsilon + 4\delta) = 46 - (4(4) + 4(4)) = 14, \tag{5.4}$$

We have obtained our first constraint as $P_x \leq 14$.

For the second condition, we will consider $P_i - P_x - 2\epsilon - 4\delta$ and $2\epsilon + 4\delta - P_x$ values. We solve this equation as follows:

$$\max\{0, P_i - P_x - 2\epsilon - 4\delta\} + \max\{0, 2\epsilon + 4\delta - P_x\} \leq B_a - P_x, \tag{5.5}$$

$$\max\{0, 18 - P_x - 2(4) - 4(4)\} + \max\{0, 2(4) + 4(4) - P_x\} \leq 44 - P_x, \tag{5.6}$$

$$\max\{0, -6 - P_x\} + \max\{0, 24 - P_x\} \leq 44 - P_x. \tag{5.7}$$

Since the two 'max' values in Equation 5.7 give different results for different values of the allocated time, we will check for the following ranges on P_x value:

- $\max\{0, -6 - P_x\}$ gives the solution 0 for all P_x values.
- If $P_x \leq 24$; then $0 + 24 - P_x \leq 44 - P_x$, which gives $24 \leq 44$ and since this is true, our second valid interval is $P_x \leq 24$.
- If $P_x \geq 25$; then $0 + 0 \leq 44 - P_x$, which gives $P_x \leq 44$ as our third valid interval.

We know that $P_x \leq 14$ from the first equation of Lemma 1, combining this one with the newly obtained intervals, we get the following interval: $0 \leq P_x \leq 14$.

After obtaining the interval, we will take the value closest to $2\epsilon + 4\delta$ which is equal to 24, as P_x. We have obtained P_x value as 14. We can follow the next steps from the Gantt-chart given by Figure 5.7.

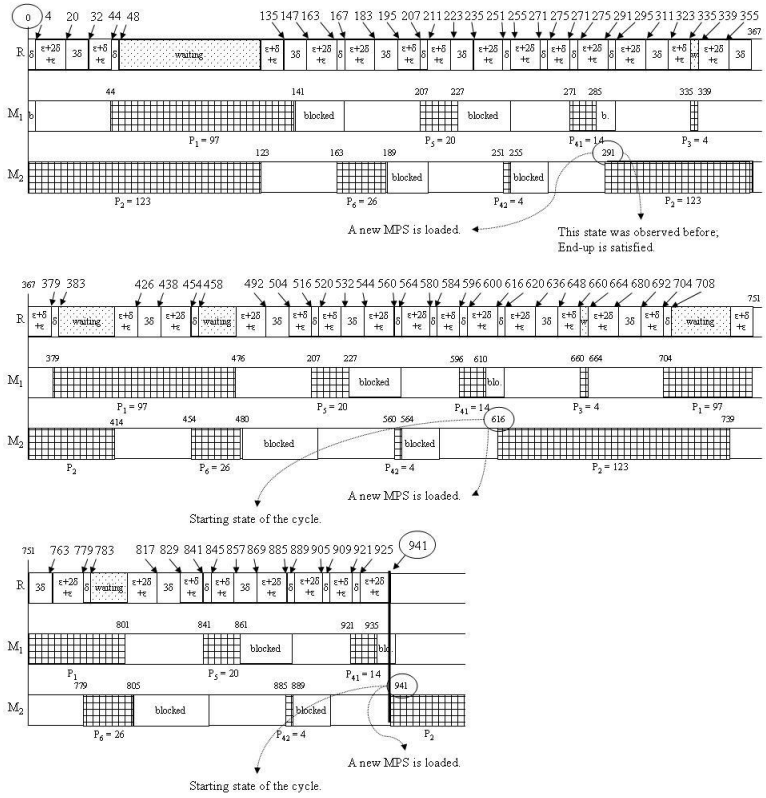

Figure 5.7: Gantt-chart of Stage 2 on LPT for Example 2

As can be seen from Figure 5.7, we start the system as was determined by LPT solution assigning parts 1, 3 and 5 on machine 1, and 2, 6 on machine 2. When the turn comes to part 4, which is to be allocated, after loading it on its already assigned machine (machine 2), robot unloads the other machine (machine 1) and comes back to take the part for loading on the other machine. From now on, part 4's processing continues on this machine and robot goes to input buffer for the next assigned part. End-up policy used in this approach is also the same as previous; for this example, steady state is obtained at time 291. The time passed until that moment is 291-0=291. Repeating the same movements once more, system arrives to the same state at time 616, and 325 units of time passed until this point. On the third repetition, same state is reached at time 941, which needed 325 units of time again. Since the last two results are the same, algorithm ends with the solution of 325, where it was 339 in LPT.

We obtained an improvement of approximately 3% in this example. As it was mentioned before, this gain caused no cost. We only changed the assignments of some operations for one of the parts and completed the production of the same set of parts in a cycle time of 3% smaller than before.

Chapter 6

Computational Experiments

In this chapter, we attempt to verify the significance of our results by comparing the algorithms described so far with the LPT approach. In order to be able to summarize the results, we have run the algorithm on many different cases. We first observed that five factors which are the five basic parameters of this study, play important roles on the solutions obtained. In order to grade these classes, we run extensive trials for each of the criteria. As a result of these studies, we obtained the factors and factor levels shown in Table 6.1.

Factor	Number of Levels	Low	Medium	High
Number of Parts	3	≤ 10	$11 \leq \ldots \leq 99$	$100 \leq \ldots \leq 800$
Average Processing Times	2	≤ 50	-	$51 \leq \ldots \leq 4800$
Range of Processing Times	3	≤ 139	$140 \leq \ldots \leq 149$	$150 \leq \ldots \leq 640$
Load/Unload Times	3	$1, 2, 3$	$4, 5, 6$	$7, 8, 9$
Travel Times	3	$1, 2, 3$	$4, 5, 6$	$7, 8, 9$

Table 6.1: Factors and their corresponding levels

Using 1,2 and 3 for low, medium and high factor levels respectively, we define the cases in the order of number of parts, average processing times, range of processing times, epsilon and delta values.

Throughout this study, we use the following equation in order to calculate the improvements:

$$\% \text{ deviation from LPT} = \frac{\text{C(LPT)} - \text{C(Stage1)}}{\text{C(LPT)}} \cdot 100 \ , \tag{6.1}$$

where 'C(LPT)' is the cycle time obtained by the LPT approach and 'C(Stage1)' is the one obtained by the Stage1 approach.

Number of replications that we run our algorithm for each of the above factor levels is 10, i.e. we checked $(3 \cdot 2 \cdot 3 \cdot 3 \cdot 3 = 162)$*(10)=1620 instances. Related data are generated randomly according to the given levels. Considering the solutions that are obtained when allocation is not allowed firstly, we can assert that our algorithm performs better than LPT approach. For all the examples compared for these two, only 3 out of 1620 randomly generated runs give better results with LPT than our heuristic; for all other runs our algorithm give either better or same results. Average gain of our algorithm is 3,3%, where this value goes up to 28,3% for a case with high part number, high mean, medium range, low epsilon and low delta levels. Average improvements for the factor levels are given in Table 6.2. One of the main parameters for the problem is the range of processing times; for medium or especially, high ranges, our gain is 7% or 8% on the average. Another aspect of this argument can be seen from the results given by Table 6.2; for these two levels, we were able to obtain a benefit of 28,3%. Similarly, using medium values of epsilon and delta, we obtained a benefit of 2,78% and 3,05%, respectively.

Factor	Factor Level	Average	Minimum	Maximum
Part #	LOW	4,32 %	0,60 %	24,30 %
	MEDIUM	2,16 %	0,00 %	14,50 %
	HIGH	1,54 %	0,00 %	28,30 %
Mean	LOW	2,12 %	0,00 %	19,40 %
	HIGH	3,23 %	0,00 %	28,30 %
Range	LOW	0,80 %	0,00 %	10,30 %
	MEDIUM	2,72 %	0,00 %	28,30 %
	HIGH	4,49 %	2,00 %	24,30 %
Epsilon	LOW	2,62 %	0,10 %	28,30 %
	MEDIUM	2,78 %	0,00 %	24,30 %
	HIGH	2,62 %	0,00 %	18,20 %
Delta	LOW	2,82 %	0,20 %	28,30 %
	MEDIUM	3,05 %	0,20 %	17,90 %
	HIGH	2,14 %	0,00 %	19,40 %

Table 6.2: Observations for Stage 1 - a

As is mentioned before, our algorithm gave smaller cycle time values for almost all examples and factor combinations. Although average values on Table 6.2 seem to be close to each other, higher improvements are mostly observed for combinations of either medium or high epsilon and delta values. This statement means that our algorithm performs better when the load/unload times of the robot and the travel times between the machines are between 4% and 9% of the average processing time of the parts in an MPS. The logic behind this result is the fact that, different from the LPT approach, construction algorithm takes the robot movements into account. When activities of the robot are ignored in a schedule, it becomes bottleneck and this behaviour results with worse solutions for higher epsilon and delta values. Range is another important factor for this study. The larger the range is, the more sufficient solutions are obtained by our approach. Having closer processing time values with the help of a decrease in the range, we become able to get the important weakness of LPT under control. Also, for those with small number of parts, we get better solutions than others.

FactorCombination	*Average*	*FactorCombination*	*Average*
11313	7,40 %	11211	6,25 %
11331	7,16 %	11221	6,42 %
11332	7,02 %	11312	5,55 %
12222	7,24 %	11321	5,65 %
12231	7,02 %	11322	6,49 %
12313	7,58 %	11323	5,08 %
12322	7,62 %	12212	6,06 %
12323	7,56 %	12213	6,54 %
22232	8,60 %	12223	5,75 %
32231	8,90 %	11232	6,23 %
		12312	5,38 %
		12321	5,67 %
		12331	6,01 %
		12332	6,79 %
		12333	6,72 %
		22312	6,17 %
		22313	6,07 %
		22322	6,90 %
		22331	5,85 %
		22332	5,54 %
		32322	5,27 %
		32331	5,15 %

Table 6.3: Observations for Stage 1 - b

Although average processing time is not as effective as the other parameters, by giving close results on average; similar to epsilon and delta values, higher percentages are observed mostly for its high levels as can be seen from Table 6.3.

When we come to comparisons on Stage 2, we used the same five factor levels. Similar to the previous one, we have run the algorithm on different combinations and observe that the previously defined ones are still valid for comparisons.

One of the basic points here is that, not every solution could be improved by allocation. As a result of this fact, out of 1620 runs, only 235 of LPT solutions and 34 of heuristic solutions can be improved. Average improvements observed for these two approaches are 1,23% and 1,66% respectively.

Factor	Factor Level	Number*	Improvement**
Part #	LOW	42	3,39 %
	MEDIUM	131	1,03 %
	HIGH	62	0,20 %
Mean	LOW	76	1,58 %
	HIGH	159	1,07 %
Range	LOW	20	0,82 %
	MEDIUM	87	0,50 %
	HIGH	128	1,80 %
Epsilon	LOW	78	1,30 %
	MEDIUM	77	1,15 %
	HIGH	80	1,25 %
Delta	LOW	48	0,92 %
	MEDIUM	74	1,26 %
	HIGH	112	1,43 %

* Number of improved solutions out of 1620/(number of levels of the factor) instances.

** Average improvement percentage for the given number of improved solutions.

Table 6.4: Observations for Stage 2 over LPT

Factor	Factor Level	Number*	Improvement**
Part #	LOW	3	6,65 %
	MEDIUM	21	1,27 %
	HIGH	9	0,37 %
Mean	LOW	16	2,02 %
	HIGH	17	1,35 %
Range	LOW	5	2,40 %
	MEDIUM	23	1,74 %
	HIGH	5	0,44 %
Epsilon	LOW	5	2,50 %
	MEDIUM	10	1,55 %
	HIGH	18	1,51 %
Delta	LOW	1	0,30 %
	MEDIUM	3	3,03 %
	HIGH	29	1,38 %

* Number of improved solutions out of 1620/(number of levels of the factor) instances.

** Average improvement percentage for the given number of improved solutions.

Table 6.5: Observations for Stage 2 over Stage 1

For the observations of this second algorithm Tables 6.4 and 6.5 are used. Allocation Algorithm is first applied over the solutions obtained by LPT rule

and the improvements observed by this stage are given by Table 6.4. We then apply the same algorithm over the Construction Algorithm and gave the related results by Table 6.5. As is observed from these tables, LPT approach is more suitable for allocation. Although improvement percentages are higher when Stage 2 is applied over Stage 1, number of improved solutions are significantly higher for LPT results. In a similar manner to the previous observations, we search for the most effective factors and factor levels for the problem. As can be seen from Tables 6.4 and 6.5 again, low number of parts were able to react better to allocation for both LPT and heuristic approaches, even if its number is smaller compared to other classes. Most important factors for allocation are determined to be epsilon and delta values. Due to the values ($2\epsilon + 4\delta$, $4\epsilon + 4\delta$, etc.) which are used in Lemma 1, this was an expected behaviour for the problem. Allocation is affected mostly by ϵ and δ values. When we pay attention to these two parameters, we can see that the results for which allocation is determined to be possible, are observed mostly for the combinations of medium and high epsilon and delta values. The reasoning here can be explained by the fact that the system is more likely to have longer waiting and more importantly blocked times as the load/unload times and the travel times get higher values. Since we try to shorten the cycle time filling the blocked times observed in the initial solution, the more blocked times the system has the more possible to response positive to allocation.

For the same problem, we checked the possibility of allocating two parts instead of one. We tried to allocate a second part after the first one is allocated but could not find any P_x value satisfying the conditions defined by Lemma 1 for none of the 1620 solutions. This was also an expected result, since we define the constraints trying to take the advantage of blocked times and to balance the work-loads on the machines as much as possible.

We also checked the results obtained by choosing the part causing the second largest blocked time value on machine x among the ones processed on machine y, to allocate. For the LPT solutions, 85 out of 1620 changed; as 41 of them gave the same result with different part and processing time values and 44 of them

could not perform allocation while they were before. Similarly, for the initial algorithm's solutions, 38 out of 1620 changed; as 23 of them gave the same result with different part and processing time values, 14 of them could not perform allocation while they were before and only 1 of them gave a smaller solution with a percentage of 18 %. This was another expected solution and we can see this fact checking the second constraint given in Lemma 1. By selecting a part with a smaller blocked time value, we decrease the right hand side of this equation and obtain a smaller range for P_x value. This decision causes the possibility of allocation to decrease. Besides, for the ones that are able to perform allocation, since other constraints in the lemma remain the same, trying to reach the best possible P_x value, same solutions as before are obtained.

Chapter 7

Conclusion

In this study, our primary focus is on the scheduling problem arising in two-machine manufacturing cells which repeatedly produces a set of multiple part-types, and where transportation of the parts between the machines is performed by a robot. The machines in a robotic cell are predominantly CNC machines which are capable of performing all the required processes of parts. In such systems, the machines and the robot can interact on a real time basis by the help of the cell controller. Since our subject is on multiple part scheduling problem, the term MPS, which is the smallest possible set of parts having the same proportions as the overall production target, is used. The objective considered is to minimize the average time to produce one MPS in a cyclic environment. The challenge here is twofold, meaning that we need to make two decisions: (a) choose a robot move sequence, (b) determine a part sequence. CNC machines are highly flexible. Taking the advantage of this property, different from the existing literature, we claim that robot may choose either to perform all the processing of a part completely on any one of the machines or to share them among the machines. As a result of this thought, allocation becomes our third question.

The solution that we look for defines the movements of the robot exactly; giving the part to be carried/loaded/unloaded together with the related machine. We modeled this problem as a special travelling salesman problem (TSP) in which

the distance matrix consists of decision variables as well as parameters. The formulation obtained is more general than the classical TSP formulation and requires a great amount of computational effort even if the number of machines in the cell is small. Consequently, we focussed our attention on heuristic approaches. We first constructed an algorithm which does not allow allocation for parts. In this stage, we try to minimize blocked times which occur when processing is completed before the robot arrives to the machine. We compared the results obtained with this algorithm with the well-known LPT (longest processing time first) approach, and noticed that our algorithm outperformed its results for almost all the examples on a set of randomly generated problems. Then we considered allocation case and let the robot to load a selected part on both of the machines consecutively. Although there would be extra load/unload and travel times, decreasing the waiting times and using the machine capacities more effectively, allocation gave better results for some specific cases.

As far as the authors know, this is the first study to consider allocation possibility in multiple part-type robotic cell scheduling literature. There is a room for improvement and future research for this study. One possible topic may be not to assume each machine to be capable of performing all the processes of a part. Considering the fact that the tool magazines of the CNC machines have a limited capacity, processing times can be viewed as the sum of relevant operations such that some of these operations can only be processed on the first or second machine and the remaining ones are to be allocated to these two machines. Another topic is to consider cycles producing higher number of units. Proving the validity of the conjecture about 1- or 2-unit cycles being optimal is important. The results may also be extended to m-machine cells. In such a case, the complexity of the problem increases dramatically since the number of feasible 1-unit cycles in an m-machine cell is exactly $m!$. One may also consider the technological differences between the machines, studying the non-identical machine case for this problem.

Bibliography

[1] A. H. Abdekhodaee, A. Wirth. Scheduling parallel machines with a single server: some solvable cases and heuristics. *Computers & Operations Research*, $29 : 295 - 315, 2002$.

[2] A. H. Abdekhodaee, A. Wirth and H. S. Gan. Equal processing and equal setup time cases of scheduling parallel machines with a single server. *Computers & Operations Research*, $31 : 1867 - 1889, 2004$.

[3] A. H. Abdekhodaee, A. Wirth and H. S. Gan. Scheduling two parallel machines with a single server: the general case. *Computers & Operations Research*, $33 : 994 - 1009, 2006$

[4] M.S. Akturk, H. Gultekin and O.E. Karasan. Robotic cell scheduling with operational flexibility. *Discrete Applied Mathematics*, $145(3) : 334 - 348, 2005$.

[5] Y.P. Aneja and H. Kamoun. Scheduling of parts and robot activities in two-machine robotic cell. *Computers & Operations Research*, $26 : 297 - 312, 1999$.

[6] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. Scheduling Computer and Manufacturing Process. *Springer. 2nd printing*, $3 - 540 - 41931 - 4, 2001$.

[7] N. Brauner and G. Finke. On the conjecture in robotic cells: new simplified proof for the three-machine case. *INFOR*, $37(1) : 20 - 36, 1999$.

[8] N. Brauner and G. Finke. On cycles and permutations in robotic cells. *Mathematical and Computer Modeling*, $34 : 565 - 591, 2001$.

[9] J. Browne, J. Harhen, J. Shivnan. *Production Management Systems, Addison-Wesley, New York*, 1996.

[10] Y. Crama and J. Van de Klundert. Cyclic scheduling of identical parts in a robotic cell. *Operations Research*, $45(6) : 952 - 965, 1997$.

[11] Y. Crama and J. Van de Klundert. Cyclic scheduling in 3-machine robotic flow shops. *Journal of Scheduling*, $4 : 35 - 54, 1999$.

[12] Y. Crama, V. Kats, J. Van de Klundert and E. Levner. Cyclic scheduling in robotic flowshops. *Annals of Operations Research*, $96 : 97 - 124, 2000$.

[13] M. Dawande, H. N. Geismar and S.P. Sethi. Dominance of cyclic solutions and challenges in the scheduling of robotic cells. *SIAM Review*, $47(4) : 709 - 721, 2005$.

[14] M. Dawande, H. N. Geismar, S.P. Sethi and C. Sriskandarajah. Sequencing and scheduling in robotic cells: Recent developments. *Journal of Scheduling*, 8(5):387–426, 2005.

[15] M. Dawande, C. Sriskandarajah and S.P. Sethi. On throughput maximization in constant travel-time robotic cells. *Manufacturing and Service Operations Management*, $4(4) : 296 - 312, 2002$.

[16] H. N. Geismar, M. Dawande and C. Sriskandarajah. Robotic Cells With Parallel Machines: Throughput Maximization in Constant Travel-TimeCells. *Journal of Scheduling* $7 : 375 - 395, 2004$.

[17] P.C. Gilmore and R.E. Gomory Sequencing in one state-variable machine: a solvable case of the travelling salesman problem. *Operations Research*, $12 : 655 - 679, 1964$.

[18] H. Gultekin, M.S. Akturk and O.E. Karasan. Robotic cell scheduling with tooling constraints. *European Journal of Operational Research*, $174 : 777 - 796, 2006$.

[19] H. Gultekin, O. E. Karasan and M. S. Akturk. Pure Cycles in Flexible Robotic Cells. *Computers and Operations Research*, 2007, doi: 10.1016/j.cor.2007.10.007.

[20] N.G. Hall, H. Kamoun and C. Sriskandarajah. Scheduling in robotic cells: classification, two and three machine cells. *Operations Research*, 45(3) : $421 - 439, 1997$.

[21] N.G. Hall, H. Kamoun and C. Sriskandaraj. Scheduling in robotic cells: Complexity and steady state analysis. *European Journal of Operational Research*, 1997.

[22] N. G. Hall, C.N. Potts, C. Sriskandarajah. Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, $102 : 223 - 43, 2000$.

[23] I. N. Kamalabadi, S. Gholami, and A. H. Mirzaei. Considering a cyclic multiple-part type three-machine robotic cell problem. *IEEM, IEEE International Conference*, $704 - 708, 2007$.

[24] H. Kamoun., N.G. Hall, C. Sriskandarajah. Scheduling in robotic cells: heuristics and cell design. *Operations Research 47*, $6 : 821 - 835$, 1999.

[25] V. Kats and E. Levner. A strongly polynomial algorithm for no-wait cyclic robotic flowshop scheduling. *Operations Research Letters*, $21 : 171 - 179, 1997$.

[26] H. Kise, T. Shioyama, and T. Ibaraki. Automated two machine flowshop scheduling: a solvable case. *IIE Transactions*, $23 : 80 - 87, 1993$.

[27] C.P. Koulamas. Scheduling two parallel semiautomatic machines to minimize machine interference. *Computers and Operations Research*, $23(10) : 945 - 56, 1996$.

[28] C.P. Koulamas and M.L. Smith. Look-ahead scheduling for minimizing machine interference. *International Journal of Production Research*, $26 : 1523 - 33, 1988$.

[29] S. A. Kravchenko and F. Werner Parallel Machine Scheduling Problems with a single server. *Mathematical and Computer Modelling*, $26(12) : 1 - 11, 1997$.

[30] C.Y. Lee, L. Lei and M. Pinedo. Current trends in deterministic scheduling. *Annals of Operations Research*, $70 : 1 - 41, 1997$.

[31] E. Levner, V. Kats and V.E. Levit. An improved algorithm for cyclic flow-shop scheduling in a robotic cell. *European Journal of Operational Research*, $97 : 500 - 508, 1997$.

[32] R. Logendran and C. Sriskandarajah. Sequencing of robot activities and parts in two-machine robotic cells. *International Journal of Production Research*, $34 : 34 - 47, 1996$.

[33] C. Miller, A. Tucker and R. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, $7(4) : 326329, 1960$.

[34] T.E. Morton and D.W. Pentico. Heuristic scheduling systems: with applications to production systems and Project management. *New York:Wiley*, 1993.

[35] M. Pinedo. Scheduling: Theory, Algorithms, and Systems. *Prentice Hall*, 2002.

[36] S.P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz and W. Kubiak. Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems*, $4 : 331 - 358, 1992$.

[37] C. Sriskandarajah, N.G. Hall and H. Kamoun. Scheduling large robotic cells without buffers. *Annals of Operations Research*, $76 : 287 - 321, 1998$.

[38] H.I. Stern and G. Vitner. Scheduling parts in a combined production-transportation work cell. *Journal of the Operational Research Society*, $41 : 625 - 632, 1990$.

VITA

Gül Didem Batur was born on June 9, 1984 in Samsun, Türkiye. She received her high school education at Samsun Atatürk Anadolu Lisesi, Türkiye. She has graduated from the Department of Industrial Engineering, Gazi University in June 2006 and joined the Department of Industrial Engineering at Bilkent University as a teaching assistant in September 2006. From that time to present she worked with Assoc. Prof. Dr. Oya Ekin Karaşan for her graduate study at the same department.