# ENERGY EFFICIENT SCHEDULING IN FLOW-SHOP AND PARALLEL MACHINE ROBOTIC CELLS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÇİYA AYDOĞAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

SEPTEMBER 2021

Approval of the thesis:

**ENERGY EFFICIENT SCHEDULING IN FLOW-SHOP AND PARALLEL MACHINE ROBOTIC CELLS**

submitted by **ÇİYA AYDOĞAN** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _______________

Prof. Dr. Esra Karasakal
Head of Department, **Industrial Engineering** _______________

Prof. Dr. Sinan Gürel
Supervisor, **Industrial Engineering, METU** _______________

**Examining Committee Members:**

Assoc. Prof. Dr. Seçil Savaşaneril Tüfekci
Industrial Engineering, METU _______________

Prof. Dr. Sinan Gürel
Industrial Engineering, METU _______________

Assist. Prof. Dr. Sakine Batun
Industrial Engineering, METU _______________

Assist. Prof. Dr. Serhat Gül
Industrial Engineering, TEDU _______________

Assist. Prof. Dr. Gülşah Karakaya
Business Administration, METU _______________

Date: 10.09.2021

| | |
|---|---|
| Name, Surname: | ÇİYA AYDOĞAN |
| Signature : | |

## ABSTRACT

## ENERGY EFFICIENT SCHEDULING IN FLOW-SHOP AND PARALLEL MACHINE ROBOTIC CELLS

AYDOĞAN, ÇİYA
M.S., Department of Industrial Engineering
Supervisor: Prof. Dr. Sinan Gürel

September 2021, 108 pages

Robotic cell scheduling studies mostly focus on increasing the throughput of the cell. Therefore, those studies consider cycle time or makespan minimization objectives. However, energy-efficient and environmentally sensitive manufacturing operations become more important and receive attention in recent studies in the literature. In this thesis, we study two robotic cell scheduling problems with machines and a material handling robot. We consider robot energy consumption as an objective to minimize while maximizing the throughput. The problem is to find efficient solutions for these objectives. Two scheduling environments are studied: a two machine flow-shop robotic cell and a two parallel machine robotic cell. In a flow-shop scheduling environment, parts are processed on both machines, but in a parallel machine scheduling environment, parts are processed on one of two machines. In both robotic cells, a robot performs all handling and loading-unloading operations.

The robot consumes energy during its moves. We assume that the energy consumption can be formulated as a convex nonlinear function of robot's speed. We try to find the optimal robot move sequence and optimal speed of the robot to minimize energy consumption and makespan (or cycle time). We also make other scheduling decisions such as part sequencing and machine-part assignment. For the flow-shop robotic cell scheduling problem we propose a mathematical model that finds efficient solutions for energy consumption and cycle time objectives. We solve the model using mixed integer second-order conic programming (MISOCP) reformulation. For the parallel machine robotic cell, similarly, we propose a mathematical model and its MISOCP reformulation to find efficient solutions. We also propose alternative neighborhood search algorithms based on Simulated Annealing. For both problems, we test the computational performance of proposed solution approaches and present energy saving achieved by robot speed control strategy.

Keywords: Robotic cell, Scheduling, Non-identical parts, Nonlinear optimization, Energy consumption, Robot speed control

# ÖZ

## AKIŞ TİPİ VE PARALEL MAKİNELİ ROBOTİK HÜCRELERDE ENERJİ TASARRUFLU ÇİZELGELEME

AYDOĞAN, ÇİYA
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Sinan Gürel

Eylül 2021 , 108 sayfa

Robotik hücre çizelgeleme problemleri çoğunlukla hücrenin üretim hızını artırmaya odaklanır. Ancak, son zamanlarda enerji tasarrruflu ve çevreye duyarlı üretim operasyonları daha önemli hale geldi ve yakın zamanda yapılan çalışmalarda daha fazla ilgi çekmekte. Bu tezde, iki robotik hücre çizelgeleme problemi ele alındı. Bir robotik hücrede makineler ve elleçleme robotu yer alır. Bu tezde robot enerji tüketimini minimize etme hedefini üretim hızını maksimize etme hedefi ile birlikte ele aldık. Bu problemlerde bu iki hedef için etkin çözümler bulmayı amaçladık. Önce birden fazla parça üreten iki makineli akış tipi robot hücresi, daha sonra ise iki paralel makineli robot hücresi ele alındı. Akış tipi çizelgelemede parçalar her iki makinede de sabit bir sırada işlenirken paralel makineli çizelgelemede parçalar iki makineden birinde işlenir. Her iki robot hücresinde de tüm taşıma ve yükleme-boşaltma işlemleri bir robot tarafından gerçekleştirir.

Robot taşıma işleri sırasında enerji tüketir. Bu çalışmada enerji tüketiminin robot hızına göre dışbükey doğrusal olmayan bir fonksiyon olarak formüle edilebileceğini varsayıyoruz. Bu tezde robot enerji tüketimini ve maksimum işbitim (ya da çevrim süresi) hedeflerini minimize etmek için en iyi robot hareket dizisini, robot hareket hızlarını bulmayı amaçlıyoruz. Aynı zamanda parça sırası ve makine-parça atama kararları da veriliyor. Akış tipi robotik hücresi için, çevrim süresi ve enerji tüketimi hedefleri için etkin çözümleri bulan bir matematiksel model önerdik. Bu modeli karışık tamsayılı ikinci derece konik programlama problemi olarak çözdük. Paralel makineli robotik hücre çizelgeleme problemi için, enerji tüketimi ve işbitim zamanı hedeflerini minimize edecek çözüm yöntemleri önerdik. İlk olarak bir matematiksel model geliştirdik ve konik gösterimini çözdük. Daha sonra Tavlama Benzetimi kullanan komşuluk arama algoritmaları önerdik. Her iki problem için önerilen çözüm yöntemlerinin hesaplama performansını test ettik. Ayrıca robot hız kontrol yaklaşımının bu problemlerde sağladığı enerji tasarrufunu gösterdik.

Anahtar Kelimeler: Robotik hücre, Çizelgeleme, Özdeş olmayan parça, Doğrusal olmayan optimizasyon, Enerji tüketimi, Robot hız kontrolü

To my mother

# ACKNOWLEDGMENTS

First of all, I would like to express my special appreciation and thanks to Prof. Sinan Gürel for his guidance, encouragement, positive attitude. He has been very kind and helpful. Without him, I wouldn't be able to complete this study.

I would also like to express my gratitude and admiration to my committee members for their valuable feedbacks, comments, and suggestions.

My deepest gratitude goes to my parents, sisters, and brother. They have been the most influential people in my life.

I am also highly thankful to our faculty members for their contributions to my personal and professional development. I am also indebted to my high school teacher Naime Çıldır and primary school teacher Serap Üstündağ.

I would also like to thank my dear friends from İlkyar student club, our department, and METU for their motivation and encouragement.

## TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| MIP | Mixed Integer Programming |
| SOCP | Second-order Conic Programming |
| MISOCP | Mixed Integer Second-order Conic Programming |
| SA | Simulated Annealing |
| GA | Genetic Algorithm |
| TS | Tabu Search |

# CHAPTER 1

# INTRODUCTION

The efficiency of production systems has always been an important issue. Efficiency of a production system can be defined as the production of goods at the lowest possible cost. Costs can include material cost, labor cost, time cost, energy cost, etc. Recently, energy efficient manufacturing has been a rising topic with increasing ecological awareness due to global warming, and rising energy prices (Bunse et al., 2011). There is an increasing trend in energy consumption around the world (*Statistical Review of World Energy, 70th edition*, 2021). According to U.S. Energy Information Administration, the industry sector has the highest level of energy consumption in the world. Therefore, energy efficiency in industry is an important research area.

With technological developments, the use of industrial robots in various industries has increased. These industrial robots are used in loading-unloading operations, transportation, warehousing, quality control, etc. In the last ten years, usage of the industrial robots has increased significantly (*World Robotics Report International Federation of Robotics*, 2020). Engelmann (2009) estimated that the energy consumption of industrial robots is approximately 8% of total energy consumption in the automotive industry. In industry, cost reduction and sustainability can be achieved by reducing the energy consumption of the robots. In addition, by doing this, damages to the environment due to energy consumption such as $CO_2$ emission and environmental pollution are reduced. In this thesis, we aim to find energy efficient operation schedules in two types of manufacturing cells with a material handling robot.

Robotic cells are production environments that include a number of machines, one or more material handling robots which are programmable. Robotic cells can have buffers to stock materials/parts in production. In the robotic cells, material handling robots perform loading-unloading operations and handling operations between machines/buffers while the machines process parts. Parts produced in a cell can be identical or non-identical. Identical parts have the same processing times and operations. Non-identical parts have different processing times from each other.

In this thesis, we consider two machine robotic cells with one material handling robot for producing non-identical parts. We assume the cells under consideration have one input buffer and one output buffer. The parts are in the input buffer at the beginning, and the finished parts are stocked in the output buffer. First, we consider a flow-shop manufacturing cell where each part is first processed on the first machine (machine 1) and then on the second machine (machine 2). Machines have different operations on parts. Buffers and machines are located linearly as can be seen in Figure 1.1. The robot moves between buffers/machines to provide the flow of parts from the input buffer to the output buffer.

Figure 1.1: Flow-shop manufacturing cell layout

I
Input Buffer
M1
Machine 1
M2
Machine 2
O
Output Buffer

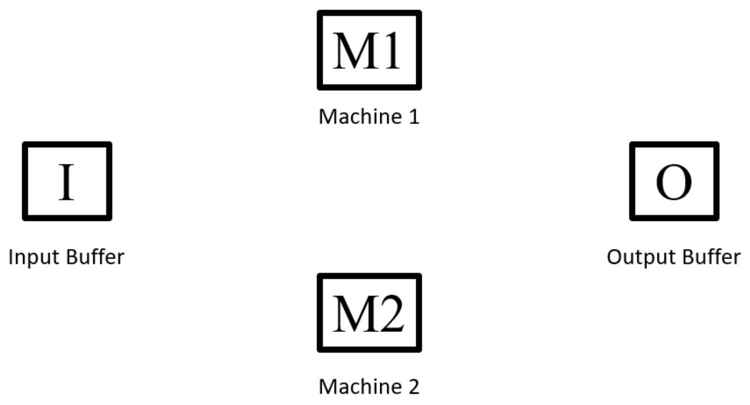
In the flow-shop manufacturing environment, the robot has certain move sequences which are called cycles. We consider a cyclic scheduling problem as we need to schedule a minimum part set for a given set of parts. There are two possible cycles ($S_1$ and $S_2$). In each cycle, a part is processed on machine 2, the next part is processed on machine 1. So, in a cycle, we process two parts. One part is finished and delivered to the output buffer, the other part is partly finished and loaded on machine 2 at the end of a cycle. From a scheduling perspective, two decisions have to be made. Part sequencing and cycle type selection ($S_1$ or $S_2$) for each robot move position.

During a cycle, the robot does some operations like moving between machines/buffers and loading/unloading parts on machines. While moving between two locations in the cell, the robot moves at a certain speed and consumes energy. In this thesis, different than the studies which consider the same robotic cell, we assume that robot speed is controllable. We assume that the robot's energy consumption in a move can be represented as a function of its speed and also depends on the distance traveled and the technical specifications of the robot. As the speed of the robot increases, energy consumption increases, and cycle time may decrease. Therefore, along with the aforementioned scheduling decisions, we need to make robot speed (or move time) decisions in this scheduling environment.

As we consider minimizing energy consumption and cycle time objectives at the same time and these two objectives conflict, we need to find efficient solutions for the problem. To this end, we develop a mathematical model which makes part sequencing, cycle type selection, and robot speed decisions to minimize energy consumption while achieving a certain cycle time (i.e. throughput) level. We perform a computational study on this model and present the results.

Second, we consider a two parallel machine robotic cell scheduling problem. In this environment, the parts are processed on one of two parallel machines. But, the speed of processing of each machine can be different, i.e. the processing time of a part on a machine can be different than the processing time of that part on the other machine. A layout in which the machines are located between the buffers is considered, as given in Figure 1.2. The robot carries a part from the input buffer to one of the machines, and then to the output buffer.

Figure 1.2: Two parallel machine manufacturing cell layout



In this problem, we again consider robot energy consumption objective. The scheduling objective to minimize is the makespan of given set of parts. The problem is to find the robot moves, speed of the robot at these moves, part sequence, and machine-part assignment so that both objectives are minimized. However, the two objectives are conflicting, because while makespan can be improved by speeding up a robot, the energy consumption objective gets worse.

For this problem, we first defined robot routes. In a route, the robot follows certain move sequences. Then, the problem can be defined as finding a sequence of robot routes rather than a sequence of robot moves. In each route, a part starts its process on a machine so we use as many routes as the number of parts in the schedule. Representing the problem using robot routes helps to develop a mathematical model and heuristic algorithms for the problem. In order to find efficient solutions by solving the mathematical model, we used $\epsilon$-constraint approach and minimized energy consumption subject to a given makespan level. For the larger instances, we designed three neighborhood search algorithms which find efficient solutions.

## 1.1 Contributions of the thesis

Cycle time minimization in a two machine flow-shop robotic scheduling problem (fixed robot speed) is already studied in the literature. There is a polynomial-time algorithm which solves the makespan minimization problem as a special case of the traveling salesman problem. However, when robot speed control is introduced and energy consumption objective is considered in this problem, new solution approaches are needed. In this thesis, we develop a mathematical model which finds efficient solutions. The model can solve problem instances up to 20 parts. Therefore, as we need to solve this problem for a minimum part set, we believe that the model is sufficient for practical size problems.

For the two parallel machine robotic cell problem, to the best of our knowledge, there are no studies that consider makespan minimization or energy consumption objectives, separately or together. We need to make part sequencing, machine-part assignment and robot move scheduling (sequence of moves and speed) decisions at the same time. By introducing robot route definitions, we are able to formulate both makespan minimization and energy consumption problems, using off the shelf branch and bound solvers. By using the $\epsilon$-approach we find efficient solutions for the two objectives. For solving large problem instances, we developed neighborhood search algorithms. These algorithms can be used to minimize makespan or to find efficient solutions for the problem.

In summary, this thesis contributes to the robotic cell scheduling literature, by introducing robot speed control strategy in flow-shop robotic cell and by studying a new robotic cell scheduling problem with parallel machines and a material handling robot with scheduling and energy consumption considerations.

The rest of the thesis is organized as follows. Chapter 2 reviews the studies about flow-shop robotic cell scheduling and parallel machine scheduling with common servers. In Chapter 3, we give problem definition, mathematical model, the analysis of the mathematical model, and computational results for the problem on flow-shop manufacturing robotic cell. For the parallel machine robotic cell, the problem definitions, mathematical models, algorithms, their analyses, and numerical study are given Chapter 4. We conclude the study in Chapter 5.

# CHAPTER 2

# LITERATURE REVIEW

In this thesis, we study two different scheduling environments: a flow-shop manufacturing cell and a parallel machine manufacturing cell. In flow-shop scheduling, each part is processed on each machine visiting the machines in a fixed order. In parallel machine scheduling each part is processed on one of the parallel machines. We assume that an input and an output buffer exists in each case. Also, we assume a material handling robot does all handling operations (carrying, loading, unloading) between input buffer-machines-output buffer in both cases. In Section 2.1, we present a review of the studies on scheduling problems in robotic cells with a flow-shop environment. Then, in Section 2.2 we present the literature on robotic scheduling problems in a parallel machine environment.

## 2.1 Studies on Flow-Shop Robotic Cells

In this section, we present a review of the studies on robotic cell scheduling with flow-shop production. The pioneering study in robotic cell scheduling problem is by Sethi et al. (1992) and robotic cell scheduling problems receive the attention of researchers since the early 1990s. Robotic cell scheduling problems can be classified according to problem characteristics. Dawande et al. (2005) provided a classification scheme and a literature survey. The problems can be grouped based on the number of machines in the cell, i.e, if a cell contains $m$ machines, then it is called $m$-machine robotic cell. Also, the number of robots, robot types (i.e., single or dual gripper), and variety of parts (i.e., identical or multiple) are used while classifying these problems. Early studies on robotic cell scheduling problems focus on material handling robot with a single gripper, so only one part can be carried at a time. Later, various problems in which the different robot and the production cell characteristics were considered. The vast majority of the studies in the literature aim to find schedules (robot moves and part sequences) that minimize the cycle time, which equivalently implies maximizing the throughput of the cell.

Robotic cells can be designed to produce identical parts. As said, Sethi et al. (1992) studied a robotic cell scheduling problem for the first time in the literature. They introduced optimal robot cycles (a sequence of robot moves) that minimize cycle time for 2-machine and 3-machine flow-shop robotic cells which produce identical parts. If $n$ parts are produced in a cycle, this cycle is called an $n$-unit cycle. Sethi et al. (1992) proposed two optimal 1-unit cycles, $S_1$ and $S_2$, which would be described in Chapter 3. Crama and van de Klundert (1997) showed an identical part cyclic robot schedule that has minimum cycle time in an $m$-machine flow-shop robotic cell, where $m > 3$.

Cyclic production schedules can also be used when multiple part types are produced. When multiple part types exist, cyclic schedules are prepared for a minimal part set rather than the whole part set. Given the demand for multiple part types, the minimal part set can be found as described in the following example. If the demand quantities are given as 50 for part A, 40 for part B, and 30 for part C, the corresponding smallest set of parts consists of 5 parts from A, 4 parts from B, and 3 parts from C. These part quantities are the minimum possible values that give the same proportion values as the real demand values. So, for this example, it is sufficient to find an optimal robotic cell schedule for $n$ = 12. For a 2-machine robotic cell, given the robot cycle and customer demand, Sethi et al. (1992) provide a polynomial-time algorithm that finds the optimal part sequence for the minimal part set.

In this thesis, one of the robotic cell scheduling problems that we studied considers a 2-machine flow-shop robotic cell with multiple part types. For such a robotic cell, Hall et al. (1997) proposed an $O(n^4)$ algorithm that finds the optimal robot cycle and part sequence that minimizes cycle time. Agnetis (2000) showed that part sequencing problem in a 2-machine robotic cell can be converted to the classical two machines no-wait flow-shop makespan minimization problem and it is solvable in time

$O(nlogn)$. Aneja and Kamoun (1999) improved the algorithm which is proposed by Hall et al. (1997) by providing an algorithm, which has complexity $O(nlogn)$, that find the optimal part sequence and robot cycles based on the TSP (Travelling salesman problem) with a special cost structure (Gilmore and Gomory, 1964). In this thesis, along with robot cycle and part sequencing decisions, we consider robot speed decisions. Also, we consider the robot energy consumption minimization objective together with the cycle time objective.

There are studies that consider robotic cells with three or more machines and multiple part types. Hall et al. (1997) investigated 1-unit cycles for a given part sequence in a 3-machine cell.For $m$-machine robotic cells, Sriskandarajah et al. (1998) provided a classification of part sequencing problems when the robotic cycle is given. They stated that there are $m!$ possible 1-unit cycles in an $m$-machine cell, and part sequencing problem can be solved in polynomial time for $(m-2)$ of these cycles. Using a particular state graph, Brauner and Finke (1999) studied $k$-unit production cycles as special paths in $m$-machine robotic cells. Later, Brauner and Finke (2001) developed a problem instance where 1-unit production cycles can be dominated by a 2-unit production cycle in 4-machine cells. Crama and de Klundert (1999) presented 1-unit cycles in 3-machine robotic flow-shop cell yield optimal production rate for producing identical parts. Brauner (2008) examined $k$-unit cycles that exactly $k$ parts enter and leave in a cycle for producing identical parts in flow-shop robotic cells.

When the number of machines and number of part types increase, the complexity of a robotic cell scheduling problem also increases. To solve these problems, heuristic and meta-heuristic algorithms were developed in many studies. Under different robot cycles in 3-machine cells, Kamoun et al. (1999) developed a heuristic algorithm for part sequencing problems. They also proposed two heuristics to design the layout of these cells. Hurink and Knust (2002) presented a tabu search (TS) algorithm for a job-shop production environment with a single material handling robot. Soukhal and Martineau (2005) developed an integer programming model for a flow-shop robotic cell problem with multiple parts to minimize makespan. They proposed a genetic algorithm (GA) for large instances. For the same problem, Carlier et al. (2010) proposed a decomposition algorithm that considers machines' blocking time. They devised a method that first determines the part sequence, then the robot move sequence. Kamalabadi et al. (2007) solved a 3-machine robotic cell problem and presented a swarm optimization (PSO) algorithm for large instances of the problem. Zarandi et al. (2013) considered different loading-unloading times for each part and sequence-dependent setup times in a 2-machine robotic cell scheduling problem. To solve large-sized problems, they proposed a simulated annealing (SA) algorithm. For 4-machine robotic cells, Abdulkader et al. (2013) developed a genetic algorithm that finds the part sequence to reduce robot cycle time. Gultekin et al. (2018) developed a mathematical programming formulation to determine the part sequence and robot motion sequence for a robotic cell that produces different types of parts with $m$ machines. This formulation does not reduce the problem to only 1-unit cycles, as is usually done in the literature, and deals with general n-unit cycles. In their study, they also developed a hybrid meta-heuristic method in which GA and TS are integrated to solve the problem.

Depending on the situation, the number and types of robots used may change. There are studies in the literature that deal with the use of multiple robots and dual gripper robots in the cell. Sethi et al. (2001) considered a 2-machine flow-shop with a robot having a dual gripper for producing identical parts. They examined the advantages of the more costly dual gripper robotic cells over single gripper robotic cells. For given cell parameters and the number of machines, they developed a simple heuristic that compares the performance of the dual gripper robot and single gripper robot. Sriskandarajah et al. (2004) investigated 2-machine flow-shop dual gripper robotic cells for producing multiple parts to maximize the long-run throughput rate. Even for the given robot move sequence, they proved finding an optimal part sequence is strongly an NP-hard problem. They proposed a heuristic approach to find robot moves and part sequences. Alcaide et al. (2007) considered an automated production system with several material handling robots to maximize the throughput rate by extending the single robot scheduling approaches. They presented a graph model for solving multi-robot scheduling problems based on the critical path problem. Elmi and Topaloglu (2013) considered multiple robots in hybrid flow-shop cells producing multiple parts with independent parallel machines aiming to minimize the makespan.

Several papers which review robot cell scheduling studies exist in the literature. Lee et al. (1997) review studies on machine scheduling problems published until 1997. They also summarize robotic cell scheduling problems in their survey. Levner et al. (2010) reported computational complexity of fundamental cyclic scheduling problems. The reader can refer to other surveys (Crama et al., 2000; Dawande et al., 2005; Hall and Sriskandarajah, 1996; Levner et al., 2010) for a complete summary of the robotic cell scheduling studies.

While the majority of studies focused on the maximization of the throughput rate, a few studies have addressed different aspects. Gultekin et al. (2007) and Gultekin et al. (2010) considered altering processing times on machines by changing the speeds of the machines in 2-machine and 3-machine robotic cells for producing identical parts. The decrease in processing time causes higher manufacturing costs so they used a bicriteria model to find robots move sequence not only for minimizing the cycle time but also for minimizing the manufacturing cost. Batur et al. (2012) considered processing time controllability on CNC machines in 2-machine cells. They proposed a model as a traveling salesman problem to find robot move sequence and processing times of parts on each machine.

The majority of robotic cell studies assumes that the robot speed is constant and given for all moves in a robotic cycle. Robots may be used at maximum speeds to achieve the highest throughput rate. However, the cells would consume more energy as a result of this. Kobetski and Fabian (2008) developed several strategies for balancing the speeds of moving parts in flexible production systems. They stated that the robot's acceleration and speed have an impact on its energy usage. The speed of robots may be controlled to decrease energy consumption. To minimize the total energy consumption in an industrial robotic cell, Bukata et al. (2017) proposed a mixed-integer linear programming model. They considered the robot speeds, robot positions, operation sequences, and robot's power-saving modes in the model. They devised a heuristic method that can solve real-sized problems with up to 12 robots because the model is insufficient to solve problems with a large number of robots. The proposed algorithms were also tested on a case study of Skoda Auto robotic cell and 20% energy saving is achieved with power-saving modes and optimization of the robot speeds. After this study, Bukata et al. (2019) proposed an algorithm based on the branch and bound method for the problem studied by Bukata et al. (2017). Gürel et al. (2019) studied robot speeds decision in a 2-machine robotic cell scheduling problem. They considered 2 objectives which are cycle time minimization and energy consumption minimization for $S_1$, $S_2$ cycles. The energy consumption is formulated as a nonlinear convex function of robot speed in their study. They showed the trade-off between the two objectives and demonstrated that significant energy savings can be achieved by eliminating robot speed control. Gultekin et al. (2021) studied $m$-machine robotic cell producing identical parts considering cycle time minimization and energy consumption minimization objectives at the same time. Because the energy consumption function is nonlinear, convex and depends on the robot speeds which are decision variables, they constructed an MISOCP model and a heuristic algorithm to find Pareto efficient solutions. Under controllable processing time and robot speed considerations in a 2-machine robotic flow-shop cell producing identical parts, Güzel (2021) proposed a mathematical model to minimize energy consumption and cycle time. $\epsilon$-constraint approach is used to the solution. As said above, in this thesis, we consider a robotic cell scheduling problem on 2-machine robotic cells where robot cycles, part sequencing, and robot speed control decisions are considered at the same time. To the best of our knowledge, there are no studies that consider robot speed control and robot energy consumption objective in a 2-machine flow-shop robotic cell with non-identical parts.

In the next section, we give a review of the literature on scheduling problems in parallel machine environments with servers (or material handling robots).

## 2.2 Studies on Parallel Machine Cells

In a parallel machine scheduling environment, a set of parts is given and each part has to be processed by one of the machines. In some cases, loading a part on a machine or a setup is required before processing a part. This setup is done by a server which can be a human operator, automated guided vehicle, robot, etc. The problem we face in these situations is called parallel machine scheduling with a common server. In this thesis, we study a production cell where two parallel machines and a material handling robot operates. At the beginning, the parts are ready at the input buffer. Each part is first picked from the input buffer by the robot, then loaded on a selected machine and processed and then unloaded and taken to the output buffer. The problem is to find the part sequence, i.e. in which order the parts will be picked from the input buffer, part-machine assignment, i.e. which part will be processed on which machine, robot moves, i.e. robot's travel between the machines and the buffers within the cell and robot speed decisions so that two objectives are to be minimized: makespan and robot energy consumption.

Parallel machine scheduling is a mature area and lots of papers were published. For the review of related studies, we refer the reader to the survey papers (Cheng and Sin, 1990; Pfund et al., 2004; Kaabi and Harrath, 2014).

Parallel machine scheduling with common server problems may have different job characteristics, machine environments, objective functions, etc. Graham et al. (1979) suggested 3-field problem classification $\alpha|\beta|\gamma$ for the scheduling problems. Field $\alpha$ denotes the machine environment. In this field, $P$ means identical parallel machines, $Q$ means uniform parallel machines, and $R$ indicates unrelated parallel machines. Specifically, $Pm$ indicates $m$ parallel machine environment. The second field $\beta$ shows the job characteristics. For instance, $prec$ specifies a precedence relationship exists between the jobs and while $p_j$ shows operation times of the jobs are different, $p_j = 1$ means each operation has a unit processing time. For the server, $s_j = 1$ states unit setup times, $s_j = s$ indicates identical setup times for jobs, $s_j$ means different setup times for the jobs, and $s_{ij}$ denotes the setup times are sequence-dependent. In the field $\gamma$, the objective function of the problem is given. For example, $C_{max}$ means minimization of the time of the last job completed, i.e. makespan. In the study by Hall et al. (2000), the server is denoted in the first field ($\alpha$) as $Sm$ where $m$ represents the number of servers. For example, $P2, S1|p_j, s_{ij}|C_{max}$ defines the problem where we have two parallel machines with a single server. The processing times of parts are different and setup times are sequence-dependent. In this example, we try to minimize makespan as the objective. The problem that we study here can be represented as $R2, S1/controllablerobotspeed/C_{max}, E_S$ which means there are two unrelated machines, i.e. each job has a different processing time on each machine, robot speed is controllable and two objective functions are considered: makespan and robot energy consumption.

Most of the parallel machine scheduling problems without common servers are computationally difficult. There are some studies that performed complexity analysis for parallel machines with common servers. Koulamas (1996) proved that makespan minimization on two parallel machines with a single server is unary NP-hard. S. Kravchenko and Werner (1997) stated that this problem is still unary NP-hard even with constant setup times. Hall et al. (2000) analyzed a number of different scheduling goals for different processing and setup time scenarios for parallel machine scheduling problems with a common server. Glass et al. (2000) proved that optimum scheduling of two dedicated parallel machines on a single server for equal setup and processing times with the objective of makespan minimization is an NP-hard problem. They proposed a heuristic that gives a worst-case ratio of $\frac{3}{2}$. Brucker et al. (2002) continued to discuss the complexity of parallel machine problems with a single server. They also developed a mathematical model and two heuristics for two identical parallel machine environments aiming to minimize completion time. Guirchoun et al. (2005) developed a polynomial reduction between a parallel machine scheduling problem with a single server and a hybrid flow-shop scheduling problem with no wait constraint. Also, an overview of complexity results is presented in that study.

Many studies in the literature have focused on makespan minimization on two identical machines with a single server. Abdekhodaee and Wirth (2002) constructed an integer programming formulation for this environment for short and equal part processing times. They also proposed two heuristic algorithms. Later, Abdekhodaee et al. (2004) proposed two heuristic algorithms with time complexity $O(nlogn)$, for two special cases where processing times are equal and setup times are equal. Abdekhodaee et al. (2006) studied the generalized form of the problem, i.e. $P2, S1|p_i, s_i|C_{max}$ and proposed a genetic algorithm, a greedy heuristic, and also adapted Gilmore-Gomory algorithm for the problem. For the online scheduling problem of $P2, S1|p_j = p, s_j|C_{max}$, Zhang and Wirth (2009) developed two heuristic algorithms. One of these algorithms is for the case where jobs have equal length, and the other one is for the case where equal processing times and equal setup times. Su (2011) considered the dynamic release times of the jobs and proposed the longest processing time algorithm to solve the problem. Gan et al. (2012) extended the study by Abdekhodaee et al. (2006) and developed two MIP (mixed integer programming) formulations and two variants of a branch-and-price scheme for the problem of $P2, S1|p_j, s_j|C_{max}$. Jiang et al. (2013)

allowed preemption of processing and setup. They also developed an algorithm that finds optimal scheduling for non-preemptive with equal processing times and equal setup times. Jiang et al. (2014) took unloading into account and applied two heuristics, namely list scheduling and longest processing time, to deal with the problem. Based on the idea of breaking a schedule into a set of blocks, Hasani et al. (2014a) proposed a mixed-integer linear programming formulation for the $P2, S1|p_j, s_j|Cmax$. Hasani et al. (2015) constructed two algorithms with time complexity $O(n^2)$ and reported the results for the large instances that are up to 10,000 jobs. The problem $P2, S1||C_{max}$ can be very complex to solve for large instances, meta-heuristic algorithms are used widely. Hasani et al. (2014c) presented simulated annealing and genetic algorithm. Later, Arnaout (2016) developed ant colony optimization (ACO) and stated that the proposed ACO outperforms the SA and GA in the study by Hasani et al. (2014c). Alharkan et al. (2020) introduced tabu search and geometric particle swarm optimization algorithms. They noted these algorithms have very good performance for the large instances and also tabu search algorithm outperforms the algorithm proposed by Hasani et al. (2015). For the objective of minimizing the forced idle time, Hasani et al. (2014b) provide a MIP formulation and TS algorithm.

For several parallel machines scheduling problems with a single server, Kim and Lee (2012) developed two MIP formulations which are based on the properties of server waiting to minimize makespan. For the same problem, Elidrissi et al. (2018) presented two alternative MIP formulations. Idrissi et al. (2018) developed two greedy heuristics for the problem of $P, S1|p_j, s_j|C_{max}$. Xie et al. (2012) considered also removal (unloading) times for $m$ parallel machine scheduling problems with a single server and proposed a heuristic algorithm. For the problem of $P, S1|p_j, s_j|C_{max}$, Elidrissi et al. (2020) proposed four MIP formulations: network variables, linear ordering variables, completion time variables, and time-indexed variables. They reported the formulation with time-index variables has better performance than others.

In some studies, setup times can depend on part sequence. Huang et al. (2010) considered multiple parallel dedicated machine scheduling problems with a single server and sequence-dependent setup times to minimize completion time. Hamzadayi and Yildiz (2016a) proposed a simulated annealing algorithm and a dispatching rule for the identical parallel machines scheduling problem with sequence-dependent setup times. Later, Hamzadayi and Yildiz (2016b) considered two objectives: minimization of the number of tardy jobs and minimization of the mean-squared deviation. Again, they proposed a simulated annealing algorithm and a dispatching rule for the problem. For the same problem environment, Hamzadayi and Yildiz (2017) constructed an MIP formulation and presented a genetic algorithm and a simulated annealing algorithm to minimize the makespan. Silva et al. (2019) proposed a MIP formulation based on arc time-indexed formulation and iterated search algorithm for the same version of the problem studied by Hamzadayi and Yildiz (2017). The computational findings revealed that the MIP formulation and iterated local search algorithm performs better than techniques proposed by Hamzadayi and Yildiz (2017). To minimize the total weighted tardiness, Bektur and Saraç (2019) proposed an MIP and two metaheuristic algorithms, which are genetic algorithm and simulated annealing algorithm for unrelated multiple machine scheduling problems with sequence-dependent setup times and a single server.

There are also studies considering multiple servers in the literature. Kravchenko and Wegner (1999) considered $m$ parallel machines scheduling problems with a single server and multiple servers to minimize makespan. They assumed that setup times equal to 1 for all jobs and developed a polynomial-time exact algorithm. Motivating from the limited unloading capacity in vehicle scheduling in logistics, Ou et al. (2009) considered multiple parallel machine scheduling problems with multiple unloading servers. Here, there is no need to set up operations but after the jobs are finished, unloading operations are carried out with servers. Werner and Kravchenko (2010) developed a pseudo-polynomial algorithm for $m$ parallel machines scheduling problems with $m$-1 servers and unit setup times.

Benmansour et al. (2018) investigated single-processor scheduling problems with time restrictions (STR) which are studied by Benmansour et al. (2014) and Braun et al. (2013). In STR, a set of unrelated jobs must be processed on a single processor. The number of jobs being processed during any time must be less than or equal to a particular integer value $B$. Benmansour et al. (2018) showed that the STR with $B = 2$ is equivalent to the problem of $P2, S1|p_j = p, s_j|C_{max}$. According to Bish (2003), the multiple-crane-constrained vehicle scheduling and location problem in a container terminal is similar to $P, S1|p_j, s_j|C_{max}|$, where crane loading and unloading operations represent setup times, cranes represent servers, each container corresponds to a job, and vehicles to machines. Torjai and Kruzslicz (2015) introduced the biomass truck scheduling problem (BTS) and stated that the problem can be formulated as a parallel machine scheduling with a single server problem.

In this thesis, we study two parallel machines served by a material handling robot. Each job is processed on a selected machine and must be carried from the input buffer to its machine first, and then from the machine to the output buffer. Different than the aforementioned studies, we consider that the robot's move time between the buffers/machines can be increased or decreased by changing its speed and speed decisions affect the robot's energy consumption. The problem is to find the part sequence for processing, part-machine assignment, robot moves, and robot speed at each move so that both makespan and robot energy consumption are minimized.

# CHAPTER 3

# CYCLIC SCHEDULING IN A TWO MACHINE FLOW-SHOP ROBOTIC CELL WITH ROBOT SPEED DECISIONS

In this chapter, we consider a cyclic scheduling problem in a two machine flow-shop robotic cell. In this robotic cell, there is an input buffer where the parts are located at the beginning. Each part is first processed on the first machine, then on the second machine, and then delivered to the output buffer. The parts are not identical, i.e. each part has different processing time requirements on the machines. All part handling operations between machines/buffers are done by a robot. This is a cyclic scheduling problem, i.e. we need to schedule the minimum part set. So, this scheduling problem needs to be solved only when the minimum part set changes which usually does not occur every day.

Different than the similar studies in robotic cell scheduling literature, we assume that the robot's speed during its moves between machines/buffers is controllable. Speeding up the robot shortens the cycle time, which is one of the objectives considered in robotic cell scheduling problems. On the other hand, this increases the energy consumption of the robot. Then, the problem is to find efficient solutions for cycle time and energy consumption objectives. Finding a solution to the problem implies sequencing the parts, scheduling robot activities, and finding the best robot speed (or move time) values.

In this chapter, we first give the problem definition. We present the robot cycles given in the literature. We develop a mathematical model for the problem. Finally, we present the computational results.

## 3.1 Problem Definition

In this scheduling environment, each part is carried from the input buffer to the first machine. The part is processed on the first machine then taken to the second machine. After its process is finished on the second machine, it is finally delivered to the output buffer. During the flow of parts, the robot performs all handling and loading-unloading operations. Loading operation is done after the robot with a part arrives at an empty machine. The robot loads the part on the machine. Unloading is done when an empty robot arrives at a loaded machine. If the process on the machine is finished, the robot unloads the part from the machine.

In the literature, it was shown by Aneja and Kamoun (1999) that for the two machine robotic cell scheduling problems, there are two possible robot cycles, i.e. cyclic robot activity sequences. We will give the detailed descriptions of these two cycles in Section 3.1.1. The studies in the literature consider the cycle time minimization problem. In this thesis, we consider the robot speed control decisions in addition to robot scheduling decisions. The energy consumption of a robot depends on its speed, and a nonlinear function is assumed for the energy consumption. In Section 3.1.2, we present the robot energy consumption function used in this study.

In this problem, we are given a set of parts to be produced. In the part set, more than one part of the same type can exist. Parts of the same type have the same characteristics such as processing times on the machines. Therefore, the parts are produced in a repetitive manner from a minimal part set. So, the problem is to find a schedule for a minimal part set. Then, this schedule is used repetitively to produce all parts.

In this problem, since we consider robot speed control decisions, we have two objective functions to minimize: cycle time and robot energy consumption. When the speed of the robot is fixed and given for all moves, the problem is to find the part sequence and robot cycles so that cycle time is minimized. Aneja and Kamoun (1999) provided an exact algorithm based on the traveling salesman problem with a special cost structure and have shown that the algorithm has time complexity $O(nlogn)$. When robot speed control is considered, in addition to part sequencing and robot cycle selection decisions, robot speed decision has to be made for each robot move.

Since we have two conflicting objectives we need to find efficient solutions for the problem.

The basic notation used in this chapter is given below:

**Notation:**

| | |
|---|---|
| $i$: | index for parts |
| $l \in \{1, 2\}$: | index for the cycle type, $S_l$ |
| $\sigma(i)$: | successor part of part $i$ |
| $P_{1,i}$: | processing time of part $i$ in machine 1 |
| $P_{2,i}$: | processing time of part $i$ in machine 2 |
| $\varepsilon$: | loading and unloading time of the machines |
| $h \in \{e, f\}$: | the status of the robot, $f$ : full, $e$ : empty |
| $d_m$: | distance travelled in move $m$ |

In the next section, we describe the possible robot cycles given in the literature.

### 3.1.1 Cycles

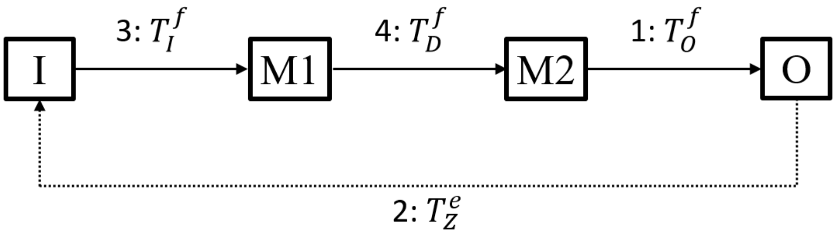
While the parts are processed, the robot follows certain moves that form two cycles named $S_1$ and $S_2$ in the literature. In each cycle, one unit part is produced so they are called 1-unit cycles. In these cycles, the robot performs some activities which are loading-unloading operations, moves between machines/buffers, and waiting in front of machines. The cycles begin and end at the same state. At the beginning and end of a cycle, the first machine is empty, the second machine was just loaded, and the robot is in front of the second machine. In both cycles, one part is processed on machine 2, another part is taken from input buffer and it is loaded on machine 1, processed and then carried to machine 2.

In Figures 3.1 and 3.2, we consider the buffers/machines are in a line, and the robot moves in a linear path. We assume the distances are additive. $I$ indicates the path between the input buffer and machine 1. $D$ represents the path between machines. $O$ is the path between machine 1 and the output buffer. $Z$ in cycle $S_1$, is the path between the input buffer and the output buffer. Because the distances are additive, the distance of path $Z$ is the sum of the distances of paths $I$, $D$, and $O$. In cycle $S_2$, $X$ shows the path between the input buffer and machine 2. $Y$ is the path between machine 1 and the output buffer in cycle $S_2$. In the Figures, next to each arc, we put a label like $a : T_g^h$. In this label $a$ is the order of move in the cycle, $g$ is the path the robot traveled in the move, $h$ is the status of the robot during the move. Continuous lines show full robot moves, whereas dashed lines indicate empty robot moves.

**Cycle** $S_1$

In Figure 3.1, we show the robot move sequence in cycle $S_1$. As mentioned, the cycle begins right after machine 2 is loaded. In cycle $S_1$, the robot performs the following activities. The robot waits for completion of the process at machine 2 ($w_2$: waiting time), then takes the part ($\varepsilon$: load/unload time), moves to the output buffer ($T_0^f$: move time), drops the part ($\varepsilon$). After that, the robot moves to the input buffer ($T_Z^e$), takes the next part ($\varepsilon$), moves to machine 1 ($T_I^f$), load the part to machine 1 ($\varepsilon$), and waits until the process is finished ($w_1$), then unloads the part from ($\varepsilon$), carries the part to machine 2 ($T_D^f$) and loads the part to machine 2.

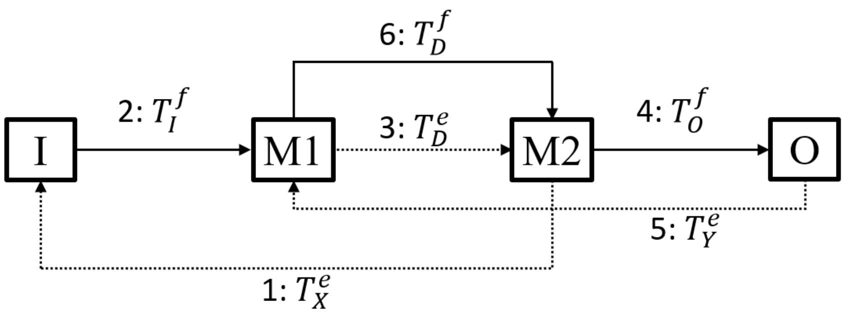
Figure 3.1: The robot move sequence in cycle $S_1$



**Cycle** $S_2$

The robot move sequence in cycle $S_2$ is illustrated in Figure 3.2. At the beginning of the cycle, the robot is in front of machine 2 after loading a part on machine 2. In cycle $S_2$, the robot performs the following activities. The robot moves to the input buffer ($T_X^e$), picks the next part ($\varepsilon$), then carries it to machine 1 ($T_I^f$), and loads to machine 1 ($\varepsilon$). After that, the robot moves to machine 2 ($T_D^e$), if the process of the part in machine 2 is finished, the robot unloads ($\varepsilon$), else the robot waits until the end of the process ($w_2$) and then unloads the part from machine 2 ($\varepsilon$). After that, the robot handles the part to the output buffer ($T_O^f$) and drops it ($\varepsilon$). Then, the robot returns to machine 1 ($T_Y^e$) and waits ($w_1$) if the processing of the part in machine 1 is not finished yet. After the robot unloads the part from machine 1 ($\varepsilon$), it carries the part to machine 2 ($T_D^f$) and loads it to machine 2 ($\varepsilon$).

Figure 3.2: The robot move sequence in cycle $S_2$



For the cycle time minimization in a two machine flow shop robotic cell with non-identical parts, it is shown that the robotic schedule must be a sequence of 1-unit cycles: $S_1$, $S_2$. Since, a part is completed and delivered to the output buffer in each cycle, the solution to the cycle time minimization problem can be represented as a sequence of parts and a sequence of 1-unit cycles. In the next section, we show cycle time calculation for $S_1$ and $S_2$ cycles.

**Cycle Time Calculation for 1-unit Cycles $S_1$ and $S_2$**

In both $S_1$ and $S_2$ cycles, each machine is loaded and unloaded once, the robot unloads one part from the input buffer and loads one part to the output buffer. So, there are 3 loading and 3 unloading activities ($6\times\varepsilon$) during each cycle.

There are two types of waiting: full waiting and partial waiting. Full waiting occurs when the robot loads the machine and waits until the end of the process. In cycle $S_1$, there are two full waiting activities. Partial waiting occurs when the robot loads the machine, then leaves to complete some other operations before returning to the machine. If the process is not finished yet when the robot is back, then the robot waits for the process before unloading the part. In cycle $S_2$, there are two partial waiting activities. The duration of a full waiting is equal to the part's processing time. The duration of a partial waiting is less than the part's processing time. In this chapter, we denote waiting time by $w_k(i)$ where $k$ is machine index and $i$ is the part index, i.e. the waiting time occurred at machine $k$ before unloading part $i$.

Parts are processed in a sequence and during a cycle ($S_1$ or $S_2$), one part is processed on machine 2, and the successor part is processed on machine 1. Let $\sigma(i)$ be the successor of part $i$ in the part sequence. Then, we can define $CT^{l}_{i,\sigma(i)}$ as completion time of 1-unit cycle $l$ where part $i$ is processed on machine 2, and part $\sigma(i)$ is processed on machine 1. Let $T^{h}_{m}$ be time elapsed in robot move $m$ when robot status is $h$ (empty, $e$ or full, $f$, $h \in \{e, f\}$).

The cycle time of cycle $S_1$ for parts $i$ and $\sigma(i)$ is the sum of waiting times, the time elapsed during robot moves, and the time required loading-unloading operations.

$$\begin{aligned} CT^{S_1}_{i,\sigma(i)} &= w_2(i) + \varepsilon + T^f_O + \varepsilon + T^e_Z + \varepsilon + T^f_I + \varepsilon + w_1(\sigma(i)) + \varepsilon + T^f_D + \varepsilon \\ &= 6\varepsilon + w_1(\sigma(i)) + w_2(i) + T^f_O + T^e_Z + T^f_I + T^f_D \end{aligned}$$

In cycle $S_1$, $w_2(i)$ and $w_1(\sigma(i))$ are full waiting activities. $w_2(i)$ equals to the processing time of part $i$ on machine 2 and $w_1(\sigma(i))$ equals to the processing time of part $\sigma(i)$ on machine 1. Cycle time of $S_1$ can be expressed as below:

$$CT^{S_1}_{i,\sigma(i)} = 6\varepsilon + P_{1,\sigma(i)} + P_{2,i} + T^f_O + T^e_Z + T^f_I + T^f_D \tag{3.1}$$

Similarly, one can calculate the cycle time for $S_2$ for parts $i$ and $\sigma(i)$ as below:

$$\begin{aligned} CT^{S_2}_{i,\sigma(i)} &= T^e_X + \varepsilon + T^f_I + \varepsilon + T^e_D + w_2(i) + \varepsilon + T^f_O + \varepsilon + T^e_Y + w_1(\sigma(i)) + \varepsilon + T^f_D \\ &= 6\varepsilon + w_1(\sigma(i)) + w_2(i) + T^e_X + T^f_I + T^e_D + T^f_O + T^e_Y + T^f_D \end{aligned}$$

In $S_2$ cycle, after the robot loads a part to a machine, it performs some tasks and then returns to unload the part. Therefore, as discussed, partial waiting can occur. The waiting time for part $i$ in front of machine 2 ($w_2(i)$) and for part $\sigma(i)$ in front of machine 1 ($w_1(\sigma(i))$) are given below:

$$\begin{aligned} w_2(i) &= max\{0, P_{2,i} - (T^e_X + \varepsilon + T^f_I + \varepsilon + T^e_D)\} \\ &= max\{0, P_{2,i} - (2\varepsilon + T^e_X + T^f_I + T^e_D)\} \qquad (3.2) \\ w_1(\sigma(i)) &= max\{0, P_{1,i} - (T^e_D + w_2(i) + \varepsilon + T^f_O + \varepsilon + T^e_Y)\} \\ &= max\{0, P_{1,\sigma(i)} - (2\varepsilon + T^e_D + w_2(i) + T^f_O + T^e_Y)\} \qquad (3.3) \end{aligned}$$

By adding the waiting times in equations 3.2 and 3.3, we can calculate the total waiting time in cycle $S_2$ as below:

$$w_2(i)+w_1(\sigma(i)) = max\{0, P_{2,i}-(2\varepsilon+T^e_X+T^f_I+T^e_D), P_{1,\sigma(i)}-(2\varepsilon+T^e_D+T^f_O+T^e_Y)\} \tag{3.4}$$

Note that, if we have no waiting for both machines in cycle $S_2$, the total waiting (3.4) will be zero. If we have to wait for only machine 2, then the total waiting is equal to the difference between the processing time of the part loaded to machine 2 and the time elapsed from loading machine 2 to unloading it. If the robot waits for both machines or for only machine 1, the waiting time is equal to the difference between processing time of the part loaded to machine 1 and the time it takes from loading machine 1 to unloading it. In addition, we define $M_l^{S_2}$ to describe moves between loading and unloading machine $l$ in cycle $S_2$. The cycle time for $S_2$ for parts $i$ and $\sigma(i)$ can be expressed as follows:

$$\begin{aligned} CT^{S_2}_{i,\sigma(i)} = {} & 6\varepsilon + T^e_X + T^f_I + T^e_D + T^f_O + T^e_Y + T^f_D \\ & + max\{0, P_{2,i} - (2\varepsilon + T^e_X + T^f_I + T^e_D), P_{1,\sigma(i)} - (2\varepsilon + T^e_D + T^f_O + T^e_Y)\} \end{aligned} \tag{3.5}$$

### 3.1.2 Energy Consumption Function

In this problem, we consider the energy consumed during the moves of the robot between machine/buffer locations. We assume that during a move, the robot moves at a constant speed determined by the decision-maker. The time/energy it takes to accelerate and decelerate the robot is negligible. The amount of energy consumed during a move is determined by the speed of the robot and the distance traveled by the robot. For a robot move, we use the energy consumption function below:

$$F(v) = c \cdot d \cdot v^k \tag{3.6}$$

In $F(v)$, $v$ is the speed of the robot and its value varies between the minimum speed ($v^{min}$) and the maximum speed ($v^{max}$). $d$ is the distance traveled during the move and $c$ is a constant that depends on robot's technical specifications and the load on the robot. For empty moves $c = c_e$ and for full moves $c = c_f$ where $c_f \geq c_e$. The exponent $k > 1$ reflects the natural relationship between speed and energy. $k > 1$ implies the energy consumption is a convex, increasing function of speed. As the speed increases, it requires more energy to speed up the robot. This natural relationship between the speed of a work/process and the resource consumed is used in many different contexts. We use the same energy consumption function as Gürel et al. (2019), Gultekin et al. (2021) and Güzel (2021). Given a schedule, i.e. a sequence of robot routes and parts, the total energy consumption of the robot is the sum of the energy consumed in all moves in the sequence.

For a given robot move, energy consumption can be expressed as a function of its move time. The time elapsed during a move is $\delta = \frac{d}{v}$. By using $\delta$, we can state the energy consumption function of a move as follows:

$$F(\delta) = c \cdot d^{k+1} \cdot \delta^{-k} \tag{3.7}$$

The function 3.7 is a decreasing convex function as we can see in the Figure 3.3.

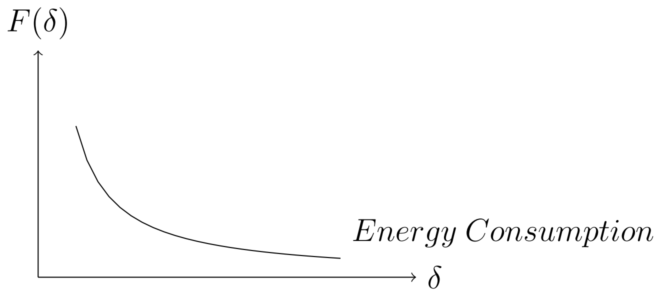
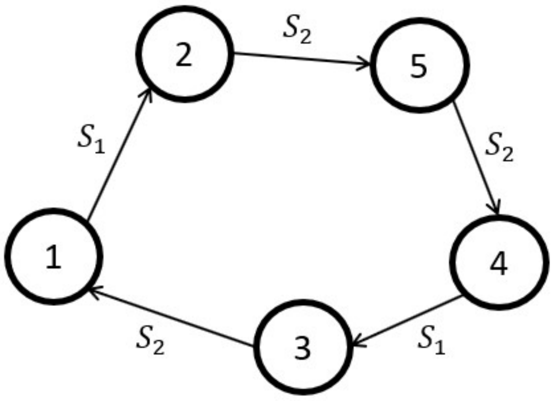


Figure 3.3: Energy consumption function with respect to time elapsed

## 3.2 Mathematical Model

In both types of cycles, a part is completed in machine 2 and placed in the output buffer, the next part is picked up from the input buffer and completed in machine 1, then carried and loaded to machine 2. Therefore, a part is finished and another part is begun to be processed in each cycle. Using this, we can formulate the problem as a specific type of traveling salesman problem (TSP). We can consider the parts as nodes and the cycles can be considered as arcs connecting a part that is to be finished, to a part that will be processed next. Each arc can be one of two types of cycle. In this case, we have two types of cost for an arc, cycle time for the selected 1-unit cycle and energy consumption. We try to find a tour that visits all nodes (parts) that gives the minimum energy consumption and minimum cycle time for the part set. An example solution for 5 parts is given in Figure 3.4. While the nodes represent parts, the arcs denote the cycle types. The part sequence is 1-2-5-4-3 that creates a tour in that example. The arc from part 1 to part 2 indicates that part 1 is processed on machine 2, and part 2 is processed on machine 1 using cycle $S_1$.

Figure 3.4: An example solution



***Sets:***

| | |
|---|---|
| $N$: | set of parts |
| $M$: | set of robot moves |
| $M_l$: | set of robot moves in cycle $S_l$ |
| $M_l^e$, $M_l^f$: | set of full($f$) and empty ($e$) robot moves in cycle $S_l$ |
| $M_l^{S_2}$: | set of moves between loading and unloading machine $l$ in cycle $S_2$ |

***Parameters:***

| | |
|---|---|
| $c_h$: | energy consumption constant when robot is in the state $h \in \{e, f\}$ |
| $\bar{C}$: | cycle time upper bound |

***Decision variables:***

$$x_{i,j,l}: \begin{cases} \text{1, if part } i \text{ is processed on machine 2 and part } j \text{ is processed on} \\ \text{machine 1 using cycle } S_l \\ \text{0, Otherwise} \end{cases}$$

$\delta_{i,j,l,m}$: the time elapsed in move $m$ while part $i$ is processed on machine 2 and part $j$ is processed on machine 1 using cycle $S_l$

$C_{i,j}$: the time elapsed during a cycle in which part $j$ succeeds part $i$ in a robot cycle

$u_i$: auxiliary variable used in subtour eliminating constraints

Note that, during each robot cycle ($S_1$ or $S_2$), one part is processed on machine 1 and another part is processed on machine 2. $x_{i,j,l}$=1 means during a robot cycle $S_l$, part $i$ is processed on machine 2 and delivered to the output buffer and part $j$ is unloaded from the input buffer, processed on machine 1, and delivered to machine 2.

$$\min \sum_{i\in N}\sum_{j\in N}\sum_{l\in\{1,2\}}\left(\sum_{m\in M_l^e} c_e \frac{d_m^{k+1}}{\delta_{i,j,l,m}^k} + \sum_{m\in M_l^f} c_f \frac{d_m^{k+1}}{\delta_{i,j,l,m}^k}\right) \tag{3.8}$$

$$\min \sum_{i\in N}\sum_{j\in N} C_{i,j} \tag{3.9}$$

$$\text{s.t.} \sum_{j\in N}\sum_{l\in\{1,2\}} x_{i,j,l} = 1 \qquad \forall i \in N \tag{3.10}$$

$$\sum_{i\in N}\sum_{l\in\{1,2\}} x_{i,j,l} = 1 \qquad \forall j \in N \tag{3.11}$$

$$\sum_{l\in\{1,2\}} x_{i,j,l} \leq 1 \qquad \forall i,j \in N \tag{3.12}$$

$$u_i - u_j + n\sum_{l\in\{1,2\}} x_{i,j,l} \leq n-1 \qquad \forall i,j \in N\setminus\{1\}, i\neq j \tag{3.13}$$

$$\delta_{i,j,l,m} \leq \frac{d_m}{v_{min}} x_{i,j,l} \qquad \forall i,j\in N, \forall l \in\{1,2\}, \forall m \in M_l \tag{3.14}$$

$$\delta_{i,j,l,m} \geq \frac{d_m}{v_{max}} x_{i,j,l} \qquad \forall i,j\in N, \forall l \in\{1,2\}, \forall m \in M_l \tag{3.15}$$

$$(6\epsilon + P_{2,i} + P_{1,j})\, x_{ij1} + 6\epsilon\, x_{ij2} + \sum_{l\in\{1,2\}}\sum_{m\in M_l} \delta_{i,j,l,m} \leq C_{i,j} \qquad \forall i,j \in N \tag{3.16}$$

$$(6\epsilon + P_{2,i} + P_{1,j})\, x_{ij1} + (4\epsilon + P_{2,i})\, x_{ij2}$$

$$+\sum_{l\in\{1,2\}}\sum_{m\in M_l}\delta_{i,j,l,m}-\sum_{m\in M_2^{S_2}}\delta_{i,j,2,m}\le C_{i,j}\qquad\forall i,j\in N \tag{3.17}$$

$$(6\epsilon+P_{2,i}+P_{1,j})\,x_{ij1}+(4\epsilon+P_{1,j})\,x_{ij2}$$
$$+\sum_{l\in\{1,2\}}\sum_{m\in M_l}\delta_{i,j,l,m}-\sum_{m\in M_1^{S_2}}\delta_{i,j,2,m}\le C_{i,j}\qquad\forall i,j\in N \tag{3.18}$$

$$x_{i,j,l}\in\{0,1\}\qquad\forall i,j\in N,\forall l\in\{1,2\} \tag{3.19}$$

$$\delta_{i,j,l,m}\ge 0\qquad\forall i,j\in N,\forall l\in\{1,2\},\forall m\in M_l \tag{3.20}$$

$$u_i\ge 0\qquad\forall i\in N\setminus\{1\} \tag{3.21}$$

There are two objective functions to minimize. (3.8) gives the total energy consumption of the robot required to process all parts. (3.9) is the total cycle time required to process all parts. Constraints (3.10) and (3.11), ensure that each part precedes a part and succeeds another part in the cycle. Constraints (3.12) guarantee that for two consecutive parts we can choose only one of the two cycles $S_1, S_2$. The order of the parts should form a tour, so we use constraints (3.13) to eliminate subtours. With those constraints, we ensure that every feasible solution should pass through part 1. If we add all the inequalities corresponding to $\sum_{l\in\{1,2\}} x_{i,j,l}=1$ for any subtour with $k$ parts that do not pass through part 1, we get $nk\le(n-1)k$ which is a contradiction. Note that constraints (3.10) - (3.13) ensure there is only one tour that passes through part 1. Thus, we ensure only one tour that covers all parts can be found. Constraints (3.14) - (3.15) guarantee that robot move times satisfy lower and upper bounds determined by the distance taken during that move and the robot's minimum and maximum speed. From cycle time calculations in equations (3.1) and (3.5), Constraints (3.16) - (3.18) determine the cycle times for consecutive two parts in a cycle. Constraints (3.19)-(3.21) define the domains of decision variables.

As discussed, we have two conflicting objectives: cycle time for the given part set and total robot energy consumption. The problem is to find efficient solutions for the problem. We use the $\epsilon$-constraint approach to find efficient solutions. We minimize energy consumption subject to an upper bound on cycle time objective. The resulting problem is given below:

$$\min\sum_{i\in N}\sum_{j\in N}\sum_{l\in\{1,2\}}\left(\sum_{m\in M_l^e}c_e\frac{d_m^{k+1}}{\delta_{i,j,l,m}^k}+\sum_{m\in M_l^f}c_f\frac{d_m^{k+1}}{\delta_{i,j,l,m}^k}\right)+\epsilon\left(\sum_{i\in N}\sum_{j\in N}C_{i,j}\right)$$

$$\text{s.t.} \sum_{i \in N} \sum_{j \in N} C_{i,j} \leq \bar{C} \tag{3.22}$$

constraints (3.10) – (3.21)

where $\bar{C}$ is the upper bound imposed on the total cycle time for the part set. The model above is a mixed-integer nonlinear programming problem. The energy consumption function in the objective includes nonlinear, convex terms. In the next section, we give a second-order conic representation of the model, so that the problem can be solved using branch-and-bound software which can solve conic quadratic subproblems.

### 3.2.1 Second-Order Conic Programming Reformulation

A convex nonlinear objective function is used in the mathematical model proposed for minimizing energy consumption. To solve it, we reformulate it using second-order conic programming (SOCP). Alizadeh and Goldfarb (2001) present methods for solving convex optimization problems. Using a technique they described, we can represent the model as MISOCP. We have $\delta^{-k}$ as a nonlinear and convex term in the energy consumption function (3.7). Different MISOCP reformulations are required for different values of the constant $k$. In this section, we demonstrate two MISOCP formulations with $k$ values of 1.5 and 2.

**MISOCP Formulation for $k = 2$**

When $k = 2$, the objective function is $c \cdot \frac{d^3}{\delta^2}$. Because the variables $x$ are binary in the mathematical model, we can redefine the objective function as $c \cdot \frac{d^3}{\delta^2} \cdot x$. Let $\tau$ be an auxiliary nonnegative continuous variable that presents $\frac{x}{\delta^2}$. Then, we can redefine the objective function as $c \cdot d^3 \cdot \tau$ by adding the constraint (3.24) because at an optimal solution to this minimization problem the constraint is always binding, i.e. $\tau = \frac{x}{\delta^2}$.

$$\min \quad c \cdot \frac{d^3}{\delta^2} \cdot x = c \cdot d^3 \cdot \tau \tag{3.23}$$

$$\frac{x}{\delta^2} \leq \tau \tag{3.24}$$

By multiplying both of the sides of the inequality (3.24) by $\delta^2$, we get $x \leq \tau \cdot \delta^2$. Because $x$ is binary variable we can convert $x \leq \tau \cdot \delta^2$ to the inequality (3.25) by multiplying left hand side by $x^3$, right hand side by $x$.

$$x^4 \leq \tau \cdot \delta^2 \cdot x \tag{3.25}$$

Then, we introduce an auxiliary nonnegative continuous variable $\omega$ satisfying the inequality (3.26).

$$\omega^2 \leq \tau \cdot x \tag{3.26}$$

From the inequalities (3.25) and (3.26), we can get the inequality $x^4 \leq \omega^2 \cdot \delta^2$. If we take the square root of both sides, we get the inequality (3.27).

$$x^2 \leq \omega \cdot \delta \tag{3.27}$$

Here, (3.26) and (3.27) are hyperbolic inequalities which represent inequality (3.24). Inequalities (3.26) and (3.27) can be equivalently expressed as SOCP inequalities (3.28) and (3.29), respectively.

$$4\omega^2 + (\tau - x)^2 \leq (\tau + x)^2 \tag{3.28}$$

$$4x^2 + (\omega - \delta)^2 \leq (\omega + \delta)^2 \tag{3.29}$$

The energy consumption minimization problem can be reformulated as MISOCP below.

$$\min \sum_{i\in K}\sum_{j\in K}\sum_{l\in\{1,2\}}\left(\sum_{m\in M_r^e} c_e \cdot d_m^3 \cdot \tau_{i,j,l,m} + \sum_{m\in M_r^f} c_f \cdot d_m^3 \cdot \tau_{i,j,l,m}\right) + \epsilon\left(\sum_{i\in N}\sum_{j\in N} C_{i,j}\right) \tag{3.30}$$

$$\text{(3.22) and } (3.10) - (3.21) \tag{3.31}$$

$$4(\omega_{i,j,l,m})^2 + (\tau_{i,j,l,m} - x_{i,r,t})^2 \leq (\tau_{i,j,l,m} + x_{i,r,t})^2 \quad \forall i,j \in N, \forall l \in \{1,2\}, \forall m \in M_l \tag{3.32}$$

$$4(x_{i,r,t})^2 + (\omega_{i,j,l,m} - \delta_{i,j,l,m})^2 \leq (\omega_{i,j,l,m} + \delta_{i,j,l,m})^2 \quad \forall i,j \in N, \forall l \in \{1,2\}, \forall m \in M_l \tag{3.33}$$

$$\tau_{i,j,l,m}, \omega_{i,j,l,m} \geq 0 \quad \forall i,j \in N, \forall l \in \{1,2\}, \forall m \in M_l \tag{3.34}$$

**MISOCP Formulation for $k = 1.5$**

When $k = 1.5$, each convex term in the objective function will be of the form $c \cdot \frac{d^{\frac{5}{2}}}{\delta^{\frac{3}{2}}}$. Because the variables $x$ are binary in the mathematical model, we can redefine the objection function as $c \cdot \frac{d^{\frac{5}{2}}}{\delta^{\frac{3}{2}}} \cdot x$. Let $\tau$ be an auxiliary nonnegative continuous variable that presents $\frac{x}{\delta^{\frac{3}{2}}}$. Then, we can redefine the objective function as $c \cdot d^{\frac{5}{2}} \cdot \tau$ by adding the constraint (3.36) because at the optimal solution this constraint is binding satisfying $\tau = \frac{x}{\delta^{\frac{3}{2}}}$.

$$\min \quad c \cdot \frac{d^{\frac{5}{2}}}{\delta^{\frac{3}{2}}} \cdot x = c \cdot d^{\frac{5}{2}} \cdot \tau \tag{3.35}$$

$$\frac{x}{\delta^{\frac{3}{2}}} \leq \tau \tag{3.36}$$

By multiplying both of the sides of the inequality (3.36) by $\delta^{\frac{3}{2}}$, we get $x \leq \tau \cdot \delta^{\frac{3}{2}}$. Because $x$ is binary variable we can convert $x \leq \tau \cdot \delta^{\frac{3}{2}}$ to the inequality (3.37) by squaring both sides, then multiplying left hand side by $x^6$.

$$x^8 \leq \tau^2 \cdot \delta^3 \tag{3.37}$$

Then, we introduce auxiliary nonnegative continuous variables $\omega_1$, $\omega_2$ and $\omega_3$ satisfying the inequalities (3.38)-(3.41).

$$\omega_1^2 \leq \delta \tag{3.38}$$

$$\omega_2^2 \leq \omega_1 \tag{3.39}$$

$$\omega_3^2 \leq \tau \cdot \delta \tag{3.40}$$

$$x^2 \leq \omega_2 \cdot \omega_3 \tag{3.41}$$

Here, (3.38)-(3.41) are conic representable inequalities for the inequality (3.37) and equivalent to the inequalities (3.28) and (3.29).

$$4\omega_1^2 + (\delta - 1)^2 \leq (\delta + 1)^2 \tag{3.42}$$

$$4\omega_2^2 + (\omega_1 - 1)^2 \leq (\omega_1 + 1)^2 \tag{3.43}$$

$$4\omega_3^2 + (\tau - \delta)^2 \leq (\tau + \delta)^2 \tag{3.44}$$

$$4x^2 + (\omega_2 - \omega_3)^2 \leq (\omega_2 + \omega_3)^2 \tag{3.45}$$

The energy consumption minimization problem can be reformulated as MISOCP below.

$$\min \sum_{i \in K} \sum_{r \in R} \sum_{t \in T} \left( \sum_{m \in M_r^e} c_e \cdot d_m^{\frac{5}{2}} \cdot \tau_{i,j,l,m} + \sum_{m \in M_r^f} c_f \cdot d_m^{\frac{5}{2}} \cdot \tau_{i,j,l,m} \right) + \epsilon \left( \sum_{i \in N} \sum_{j \in N} C_{i,j} \right) \tag{3.46}$$

$$(3.22) \text{ and } (3.10) - (3.21) \tag{3.47}$$

$$4(\omega_{1,i,j,l,m})^2 + (\delta_{i,j,l,m} - 1)^2 \leq (\delta_{i,j,l,m} + 1)^2 \quad \forall i, j \in N, \forall l \in \{1,2\}, \forall m \in M_l \tag{3.48}$$

$$4(\omega_{2,i,j,l,m})^2 + (\omega_{1,i,j,l,m} - 1)^2 \leq (\omega_{1,i,j,l,m} + 1)^2 \quad \forall i, j \in N, \forall l \in \{1,2\}, \forall m \in M_l \tag{3.49}$$

$$4(\omega_{3,i,j,l,m})^2 + (\tau_{i,j,l,m} - \delta_{i,j,l,m})^2 \leq (\tau_{i,j,l,m} + \delta_{i,j,l,m})^2 \quad \forall i, j \in N, \forall l \in \{1,2\}, \forall m \in M_l \tag{3.50}$$

$$4(x_{i,r,t})^2 + (\omega_{2,i,j,l,m} - \omega_{3,i,j,l,m})^2 \leq (\omega_{2,i,j,l,m} - \omega_{3,i,j,l,m})^2 \quad \forall i, j \in N, \forall l \in \{1,2\}, \forall m \in M_l \tag{3.51}$$

$$\tau_{i,j,l,m}, \omega_{1,i,j,l,m}, \omega_{2,i,j,l,m}, \omega_{3,i,j,l,m} \geq 0 \quad \forall i, j \in N, \forall l \in \{1,2\}, \forall m \in M_l \tag{3.52}$$

## 3.3 Computational Results

In this section, we present a numerical study on the problem and assess the computational performance of the mathematical model. We first present a set of efficient solutions to observe the behavior of the efficient frontier for energy consumption and cycle time objectives. After that, we examine the effects of problem parameters on the performance of the mathematical model. Finally, we show how robot speed control strategy affects energy consumption levels in the given robotic cell scheduling problem.

### 3.3.1 Experimental Settings

In the two machine flow-shop robotic cell environment, we have an input buffer, two machines, and an output buffer arranged in line. Cycle time and robot energy consumption objectives are affected by the number of parts in the minimum part set, distances between machines/buffers, processing times of jobs on machines ($P_1, P_2$), robot speed upper bound ($v^{max}$) and parameters ($c_e - c_f$ and $k$) of robot energy consumption function. In Table 3.1, we give the alternative values for these experimental parameters.

Table 3.1: Experimental Settings

| | |
|---|---|
| Number of parts: | 10, 20, 50 |
| Distance Scenarios: | Short, Long, Mixed |
| Processing Times: | Equal LV, Equal HV, $P_1 > P_2$, $P_2 > P_1$ |
| $c_f - c_e$: | 2-2, 2.5-2 |
| $v^{max}$: | 2.0,1.5 |
| $k$: | 2.0,1.5 |

We generate instances with 10, 20, and 50 parts. In this problem, the parts are produced in a cyclic schedule. So, a minimal part set (MPS) is used to produce all parts. For example, consider we have 100 units of part A, 200 units of part B, and 300 units of C. In this case, our MPS includes 1 unit of part A, 2 units of part B, and 3 units of part C. We try to find a schedule for the MPS and repeat the schedule 100 times for producing all parts. Therefore, the number of parts in the parameters setting reflects the number of parts in MPS not the total number of parts. A minimum part set of size 10 or 20 is quite realistic. 50 parts are a very high number for the number of real-life parts MPS. Solving the problem for 50 parts may take too long. So, we generate only one instance with 50 parts to show the performance of the model.

As can be seen in the Figures 3.1 and 3.2, we have three distances in the layout: one between the input buffer and machine 1, one between machines, and one between machine 2 and the output buffer. The distances are additive. In experimental settings, we consider three distance scenarios. In scenarios Short and Long, the distances are equal to each other. Distances are 10 m. in scenario Short and 20 m. in scenario Long. In scenario Mixed, the distance between the two machines is 20 m. while other distances are 10 m. In all instances, we assume the time of loading/unloading operations ($\varepsilon$) is equal to one second.

In this problem, the parts are non-identical so their processing times are different from each other. We consider four scenarios for the processing times of the parts. First, in scenario Equal LV and Equal HV, the processing time of a part on machine 1 is equal to the processing time of that part on machine 2. In scenario Equal HV processing times are generated from a larger range, so processing times have higher variance. In Equal LV processing times are generated from a uniform distribution between 80 seconds and 100 seconds ($U(80, 100)$). In scenario Equal HV, processing times are generated from $U(60, 120)$. In scenario $P_1 > P_2$ ($P_2 > P_1$), we assume the processing time of a part on machine 1 (machine 2) is greater than the processing time on machine 2 (machine 1). Processing times are generated using uniform distribution between 80 seconds and 100 seconds for machine 1 (machine 2), between 100 seconds and 120 seconds for machine 2 (machine 1).

$c_f - c_e$, $v^{max}$, and $k$ are parameters related to the energy consumption function which is given in equation 3.6. We consider two parameter settings for $c_f - c_e$. In the first, we assume the energy consumption of the robot is the same for robot status of full and empty ($c_f = c_e = 2$). The other is the energy consumption is higher when the robot is full ($c_f = 2.5, c_e = 2, c_f > c_e$). We assume the minimum speed for the robot is 0.5 m/s. We consider two scenarios for maximum speeds which are 2 m/s (high) and 1.5 m/s (low). We use two values for the exponent k which are 2 (high) and 1.5 (low).

We have a base parameter setting with Short distances, Equal LV, $c_f = c_e = 2$, $v^{max} = 2$, $k = 2$. At each time, we change a parameter while others are the same as in the base setting. For example, if we change processing time scenario from Equal LV to $P_1 > P_2$, other parameters stay the same i.e. Short distances, $c_f = c_e$, $v^{max} = 2$,

$k = 2$. So, we have nine experimental settings in total. For 10 and 20 parts settings, we produced five replications. In replications, we regenerate processing times. We generate only one instance with the base parameter setting for the number of parts 50 to show the performance of the model because the number of 50 parts is not a very common situation in two machine flow-shop manufacturing environments.

For each problem instance, we solve the mathematical model for 10 different cycle time upper bound values, named $\bar{C}^j$ where $1 \leq j \leq 10$. $\bar{C}^1$ is the minimum possible cycle time level achieved by setting robot speed to $v^{max}$ in all moves. The part sequence and robot cycles that minimize cycle time are found by the algorithm developed by Aneja and Kamoun (1999). We found $\bar{C}^{10}$ by setting the robot speed to $v^{min}$ in all moves and by using cycle $S_1$ for all cycles in the schedule. The other cycle time levels ($\bar{C}^2$-$\bar{C}^9$) are equally separated between $\bar{C}^1$ and $\bar{C}^{10}$.

We use IBM ILOG CPLEX 12.10 as the solver to perform the computational tests. We perform the test with a Java compiler on Intel Xeon(R) E-2246G (3.6 GHz and 16 GB RAM). The time limit of the solver is set to 5000 seconds. We get the best solution and gap from the solver if it does not reach optimality in the time limit.

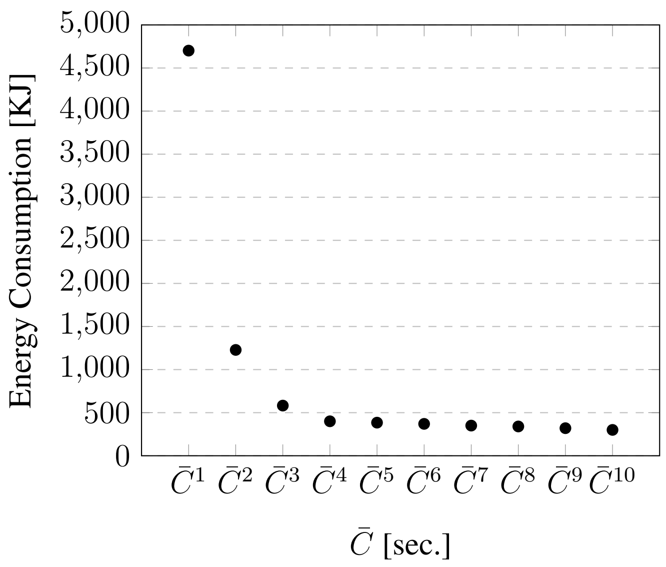
### 3.3.2 Energy Consumption vs. Cycle Time Objectives

In this section, we present a set of efficient solutions for a selected 10 parts instance with the base parameter settings. Figure 3.5 gives efficient points given by the mathematical model.

As can be seen in Figure 3.5, energy consumption decreases as the cycle time increases. When we have more time to complete parts, the robot intends to move slower to decrease energy consumption. In shorter cycles, the robot intends to move faster because we have a shorter time to complete parts. This causes more energy consumption. Because we have a decreasing convex energy consumption function with respect to move time, as the cycle time increases it becomes more costly to decrease energy consumption further.

The trade-off between energy consumption and cycle time can be used to save energy. When a higher throughput rate is needed, the cell would work with lower cycle times,

Figure 3.5: A set of efficient solutions



although it would consume more power. However, when we have a flexible due date for the completion of parts, energy consumption could be more important. Then, the robotic cell can work at higher cycle times and save energy.

For the efficient solutions presented above, in Table 3.2 we give the number of times $S_1$ and $S_2$ cycles are used in each solution. If cycle time is more critical (i.e. shorter cycle times), $S_2$ cycles are dominantly used. As cycle time becomes longer i.e. energy consumption becomes more critical, $S_1$ cycles were used more. In cycle $S_2$, the robot carries out some other activities while a job is being processed on a machine. On the other hand, we have full waiting in cycle $S_1$. So, the time elapsed in $S_2$ cycle is generally lower than the time elapsed when $S_1$ is used. However, the robot travels more distance in cycle $S_2$. Therefore, to achieve lower cycle times, the model prefers more $S_2$ cycles.

### 3.3.3 Computational Performance of Mathematical Model

In this section, we investigate how the parameters affect the performance of the model. We perform tests on instances that are generated as described above. We solved problems using IBM CPLEX. we set the time limit to 5,000 CPU seconds. 92% of the instances with 10 parts and 89% of the instances with 20 parts are solved

Table 3.2: Distribution of robot cycles between $S_1$ and $S_2$

| **Cycle Time** | **# $S_1$** | **# $S_2$** |
|---|---|---|
| $\bar{C}^1$ | 0 | 10 |
| $\bar{C}^2$ | 0 | 10 |
| $\bar{C}^3$ | 0 | 10 |
| $\bar{C}^4$ | 0 | 10 |
| $\bar{C}^5$ | 2 | 8 |
| $\bar{C}^6$ | 3 | 7 |
| $\bar{C}^7$ | 5 | 5 |
| $\bar{C}^8$ | 7 | 3 |
| $\bar{C}^9$ | 8 | 2 |
| $\bar{C}^{10}$ | 10 | 0 |

in less than 5% gap. On average, the model is terminated in 1,755 CPU seconds for the instances with 10 parts while this value is 2946 seconds for the instances with 20 parts.

In Tables 3.3, 3.4, and 3.5, we give the percentage of instances solved with less than 1% gap, percentage of instances that have greater than 1% and less than 5% gap, and average CPU times. The optimality gap levels achieved and the CPU times spent show that it is easier to solve the model for lower and higher cycle time levels. As can be concluded in Table 3.3, while there is almost no difference between instances with different distance scenarios in 10 parts, the instances with long and mixed distances show better performance in 20 parts.

The results for instances with different processing time scenarios are given in Table 3.4. When the variance is high in processing times (Equal HV), we have less CPU times and the performance of the model is better for the instances with 10 parts. For the instances with 20 parts, we get more results that have less than 1% gap. However, the percentage of the results greater than 5% is higher. When the processing times are greater on machine 1 or machine 2, there is no obvious difference between the results.

In Table 3.5, we give the results for different robot parameters. When the energy consumption is greater in full status ($c_f > c_e$), there is no obvious difference in the performance of the model. However, with a lower maximum speed ($v^{max}$), the performance of the model gets worse. When $k = 1.5$, the model has less than 1% gap in only 10% of the instances with 10 parts. There are no results with less than 5% gap for 20 parts. So, with $k = 1.5$, we can say that it is harder to solve the model.

We give the results of the instance with 50 parts using base parameter settings in Table 3.6. For all cycle time upper bounds, the model hits the time limit. However, we can get less than 5% gap in 60% of the cases.

### 3.3.4 How much energy can be saved by the robot speed control strategy?

In practice, usually, the robots are operated at their highest speed to achieve maximum throughput. With the robot speed control approach, we can achieve lower energy consumption while satisfying a required throughput level. We examine how much energy saving is achieved with the idea of controlling robot speed in this section. As mentioned before, to calculate the minimum cycle time ($\bar{C}^1$), we use a cycle time minimization algorithm proposed by Aneja and Kamoun (1999) with the robot moves at their maximum speed. At $\bar{C}^1$, this solution has certain energy consumption. We, then solve our mathematical model with a cycle time level $\bar{C}^1$ and minimize energy consumption. We calculate energy saving by comparing the energy consumption of these two solutions. In Table 3.7, we give the average energy saving rates in percentage with respect to different parameter settings. We have 23.5% energy saving in overall.

Gürel et al. (2019) showed that robot can be slowed down by utilizing the partial waiting times in $S_2$ cycle and save energy. Similarly, in this problem, $S_2$ cycles are used and partial waiting times occur. In Table 3.7, we present the average energy saving (%) achieved. Distance between the machines and buffers affects energy saving. When the distances are shorter higher energy saving is achieved as robot can be slowed down more compared to long distance scenario. However, if we increase only the distance between machines (Mixed), we achieve greater energy saving. This shows the distance between machines is more critical for energy saving.

Table 3.3: Computational Results for Different Distance Scenarios

| **Number of Parts** | **Distance Scenarios** | **Performance Measures** | $\bar{C}^1$ | $\bar{C}^2$ | $\bar{C}^3$ | $\bar{C}^4$ | $\bar{C}^5$ | $\bar{C}^6$ | $\bar{C}^7$ | $\bar{C}^8$ | $\bar{C}^9$ | $\bar{C}^{10}$ | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | gap < 1% | 100% | 100% | 100% | 100% | 100% | 0% | 100% | 40% | 100% | 100% | 84% |
| | **Short** | 1% ≤ gap ≤ 5% | 0% | 0% | 0% | 0% | 0% | 100% | 0% | 60% | 0% | 0% | 16% |
| | | Avg Comp Time | 4.7 | 7.4 | 10.8 | 430.1 | 5000.2 | 5000.4 | 5000.3 | 5000.3 | 595.6 | 4.5 | 2105.4 |
| | | gap < 1% | 100% | 100% | 100% | 100% | 100% | 0% | 20% | 100% | 100% | 100% | 82% |
| **10** | **Long** | 1% ≤ gap ≤ 5% | 0% | 0% | 0% | 0% | 0% | 100% | 80% | 0% | 0% | 0% | 18% |
| | | Avg Comp Time | 5.1 | 6.3 | 71.3 | 28.6 | 108.5 | 5000.4 | 5000.4 | 5000.4 | 2698.9 | 8.7 | 1792.9 |
| | | gap < 1% | 100% | 100% | 100% | 100% | 100% | 0% | 0% | 100% | 100% | 100% | 80% |
| | **Mixed** | 1% ≤ gap ≤ 5% | 0% | 0% | 0% | 0% | 0% | 100% | 100% | 0% | 0% | 0% | 20% |
| | | Avg Comp Time | 4.0 | 45.1 | 24.7 | 28.8 | 84.5 | 5000.3 | 5000.4 | 5000.4 | 663.2 | 3.8 | 1585.5 |
| | | gap < 1% | 100% | 80% | 80% | 100% | 60% | 80% | 100% | 20% | 80% | 100% | 80% |
| | **Short** | 1% ≤ gap ≤ 5% | 0% | 20% | 0% | 0% | 0% | 0% | 0% | 80% | 0% | 0% | 10% |
| | | Avg Comp Time | 190.9 | 2048.3 | 1087.7 | 5001.6 | 5001.1 | 5002.3 | 5002.8 | 5001.0 | 5000.9 | 38.9 | 3337.5 |
| | | gap < 1% | 100% | 80% | 100% | 100% | 100% | 100% | 100% | 80% | 100% | 100% | 96% |
| **20** | **Long** | 1% ≤ gap ≤ 5% | 0% | 20% | 0% | 0% | 0% | 0% | 0% | 20% | 0% | 0% | 4% |
| | | Avg Comp Time | 108.9 | 1067.8 | 435.4 | 223.8 | 5000.8 | 5000.7 | 5000.7 | 5000.7 | 5003.9 | 147.1 | 2699.0 |
| | | gap < 1% | 100% | 80% | 60% | 100% | 40% | 100% | 100% | 100% | 100% | 100% | 88% |
| | **Mixed** | 1% ≤ gap ≤ 5% | 0% | 20% | 20% | 0% | 20% | 0% | 0% | 0% | 0% | 0% | 6% |
| | | Avg Comp Time | 33.9 | 1645.5 | 3099.6 | 1177.2 | 5000.7 | 5000.7 | 5000.7 | 5000.8 | 5001.6 | 49.1 | 3101.0 |

Table 3.4: Computational Results for Different Processing Times

| **Number of Parts** | **Process Times** | **Performance Measures** | $\bar{C}^1$ | $\bar{C}^2$ | $\bar{C}^3$ | $\bar{C}^4$ | $\bar{C}^5$ | $\bar{C}^6$ | $\bar{C}^7$ | $\bar{C}^8$ | $\bar{C}^9$ | $\bar{C}^{10}$ | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **10** | **Equal** | gap < 1% | 100% | 100% | 100% | 100% | 100% | 0% | 100% | 40% | 100% | 100% | 84% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 0% | 0% | 0% | 0% | 100% | 0% | 60% | 0% | 0% | 16% |
| | | Avg Comp Time | 4.7 | 7.4 | 10.8 | 430.1 | 5000.2 | 5000.4 | 5000.3 | 5000.3 | 595.6 | 4.5 | 2105.4 |
| | **High Variance** | gap < 1% | 100% | 100% | 100% | 100% | 100% | 60% | 20% | 100% | 100% | 100% | 88% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 0% | 0% | 0% | 0% | 40% | 80% | 0% | 0% | 0% | 12% |
| | | Avg Comp Time | 6.6 | 5.4 | 37.8 | 1036.9 | 3038.5 | 2247.3 | 5000.4 | 388.0 | 345.4 | 4.4 | 1211.1 |
| | $P_1 > P_2$ | gap < 1% | 100% | 100% | 100% | 100% | 100% | 0% | 100% | 20% | 100% | 100% | 82% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 0% | 0% | 0% | 0% | 100% | 0% | 80% | 0% | 0% | 18% |
| | | Avg Comp Time | 4.9 | 5.1 | 6.0 | 393.1 | 3303.9 | 5000.4 | 5000.3 | 5000.4 | 1054.4 | 5.5 | 1977.4 |
| | $P_2 > P_1$ | gap < 1% | 100% | 100% | 100% | 100% | 100% | 0% | 100% | 20% | 100% | 100% | 82% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 0% | 0% | 0% | 0% | 100% | 0% | 80% | 0% | 0% | 18% |
| | | Avg Comp Time | 4.7 | 4.7 | 9.5 | 130.5 | 4477.8 | 5000.3 | 5000.3 | 5000.4 | 1123.0 | 5.4 | 2075.7 |
| **20** | **Equal** | gap < 1% | 100% | 80% | 80% | 100% | 60% | 80% | 100% | 20% | 80% | 100% | 80% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 20% | 0% | 0% | 0% | 0% | 0% | 80% | 0% | 0% | 10% |
| | | Avg Comp Time | 190.9 | 2048.3 | 1087.7 | 5001.6 | 5001.1 | 5002.3 | 5002.8 | 5001.0 | 5000.9 | 38.9 | 3337.5 |
| | **High Variance** | gap < 1% | 100% | 100% | 100% | 80% | 60% | 100% | 100% | 80% | 80% | 60% | 86% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 20% | 0% | 2% |
| | | Avg Comp Time | 352.2 | 1278.2 | 746.9 | 1359.2 | 5001.3 | 5000.9 | 5001.0 | 5001.1 | 5001.2 | 2069.6 | 3081.2 |
| | $P_1 > P_2$ | gap < 1% | 100% | 100% | 100% | 100% | 100% | 100% | 80% | 0% | 100% | 100% | 88% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 0% | 0% | 0% | 0% | 0% | 20% | 80% | 0% | 0% | 10% |
| | | Avg Comp Time | 1021.4 | 32.0 | 219.4 | 5000.8 | 5002.7 | 5000.9 | 5000.9 | 5000.9 | 4282.3 | 37.9 | 3059.9 |
| | $P_2 > P_1$ | gap < 1% | 100% | 60% | 100% | 100% | 100% | 100% | 80% | 0% | 100% | 100% | 84% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 0% | 0% | 0% | 0% | 0% | 20% | 100% | 0% | 0% | 12% |
| | | Avg Comp Time | 1271.1 | 2019.9 | 1168.1 | 5000.7 | 5000.8 | 5001.0 | 5002.5 | 5001.1 | 5000.8 | 37.5 | 3450.4 |

Table 3.5: Computational Results for Different Robot Parameters

| **Number of Parts** | **Process Times** | **Performance Measures** | $\bar{C}^1$ | $\bar{C}^2$ | $\bar{C}^3$ | $\bar{C}^4$ | $\bar{C}^5$ | $\bar{C}^6$ | $\bar{C}^7$ | $\bar{C}^8$ | $\bar{C}^9$ | $\bar{C}^{10}$ | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **10** | **Base** | gap < 1% | 100% | 100% | 100% | 100% | 100% | 0% | 100% | 40% | 100% | 100% | 84% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 0% | 0% | 0% | 0% | 100% | 0% | 60% | 0% | 0% | 16% |
| | | Avg Comp Time | 4.7 | 7.4 | 10.8 | 430.1 | 5000.2 | 5000.4 | 5000.3 | 5000.3 | 595.6 | 4.5 | 2105.4 |
| | $c_f > c_e$ | gap < 1% | 100% | 100% | 100% | 100% | 100% | 0% | 100% | 40% | 100% | 100% | 84% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 0% | 0% | 0% | 0% | 100% | 0% | 60% | 0% | 0% | 16% |
| | | Avg Comp Time | 4.1 | 13.9 | 5.1 | 619.5 | 5000.3 | 5001.2 | 5000.3 | 5000.4 | 1179.5 | 10.3 | 2183.5 |
| | **low** $v^{up}$ | gap < 1% | 100% | 100% | 100% | 100% | 40% | 100% | 0% | 100% | 80% | 100% | 82% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 0% | 0% | 0% | 60% | 0% | 100% | 0% | 20% | 0% | 18% |
| | | Avg Comp Time | 4.0 | 4.5 | 11.1 | 4888.5 | 5000.3 | 5000.3 | 5000.3 | 5000.4 | 2493.0 | 3.5 | 2740.6 |
| | $k = 1.5$ | gap < 1% | 100% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 10% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| | | Avg Comp Time | 150.1 | 5000.8 | 5001.0 | 5000.9 | 5001.0 | 5001.0 | 5001.1 | 5001.1 | 5001.0 | 5000.9 | 4515.9 |
| **20** | **Base** | gap < 1% | 100% | 80% | 80% | 100% | 60% | 80% | 100% | 20% | 80% | 100% | 80% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 20% | 0% | 0% | 0% | 0% | 0% | 80% | 0% | 0% | 10% |
| | | Avg Comp Time | 190.9 | 2048.3 | 1087.7 | 5001.6 | 5001.1 | 5002.3 | 5002.8 | 5001.0 | 5000.9 | 38.9 | 3337.5 |
| | $c_f > c_e$ | gap < 1% | 100% | 80% | 100% | 80% | 100% | 100% | 100% | 20% | 80% | 100% | 86% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 20% | 0% | 0% | 0% | 0% | 0% | 80% | 0% | 0% | 10% |
| | | Avg Comp Time | 89.7 | 1028.1 | 1036.0 | 5002.3 | 5001.9 | 5000.7 | 5000.9 | 5001.0 | 4047.7 | 31.6 | 3124.0 |
| | **low** $v^{up}$ | gap < 1% | 100% | 80% | 100% | 60% | 20% | 0% | 80% | 80% | 100% | 80% | 70% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 20% | 0% | 40% | 80% | 100% | 20% | 0% | 0% | 0% | 26% |
| | | Avg Comp Time | 158.0 | 1026.0 | 3163.3 | 5000.9 | 5005.4 | 5000.8 | 5000.7 | 5000.9 | 5000.9 | 1030.8 | 3538.8 |
| | $k = 1.5$ | gap < 1% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| | | $1\% \leq$ gap $\leq 5\%$ | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| | | Avg Comp Time | 5001.7 | 5002.1 | 5004.1 | 5003.4 | 5003.7 | 5000.2 | 5003.8 | 5002.7 | 5000.4 | 5001.6 | 5002.5 |

Table 3.6: Results for the instance with 50 parts

| | $C^1_{level}$ | $C^2_{level}$ | $C^3_{level}$ | $C^4_{level}$ | $C^5_{level}$ | $C^6_{level}$ | $C^7_{level}$ | $C^8_{level}$ | $C^9_{level}$ | $C^{10}_{level}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| gap | 8.31% | 29.32% | 1.77% | 0.18% | 2.82% | 0.05% | 0.42% | 0.93% | 17.53% | 100.00% |
| best objective | 26478.0 | 7521.1 | 3044.8 | 1990.0 | 1955.0 | 1816.5 | 1740.0 | 1668.2 | 1910.0 | 1500.0 |

Table 3.7: Energy Saving Rates

| | | **Average Energy Saving (%)** |
|---|---|---|
| **Distance Scenario** | Short | 18.3 |
| | Long | 15.7 |
| | Mixed | 20.6 |
| **Processing Times** | Equal | 18.3 |
| | High Variance | 25.9 |
| | $P_1 > P_2$ | 42.4 |
| | $P_2 > P_1$ | 42.0 |
| **Robot Parameters** | Base | 18.3 |
| | $c_f > c_e$ | 17.6 |
| | low $v^{up}$ | 13.4 |
| | low $k$ | 16.4 |
| | **Overall** | 23.5 |

Processing times have an important impact on energy saving. When the processing times of a part is different on different machines (i.e. $P_1 > P_2$ or $P_2 > P_1$), we could reach up to 42.4% energy saving. This is because different processing times imply longer partial waiting times which lead to slower robot moves. Also, when the processing time of a job is equal on both machines but the processing times of parts have higher variability (Equal HV), average energy saving is higher.

There is a slight reduction in energy saving when we change the coefficient $c_f$ from 2 to 2.5. Like that, decreasing the value $k$ from 2 to 1.5 results in less energy saving rate. This means as the non-linearity of the energy consumption function increases, energy saving is higher. The maximum speed of the robot is also an effective factor in energy saving. With low maximum speed, the energy saving rate decreases. This is again because of the shape of the energy consumption function.

# CHAPTER 4

# SCHEDULING IN A PARALLEL MACHINES ROBOTIC CELL WITH ROBOT SPEED DECISIONS

In this chapter, we consider a robotic cell with two parallel machines and a material handling robot. At the beginning, the parts are ready in the input buffer. Each part is transferred from the input buffer to one of the machines. A machine can process one part at a time. After the process finishes on a machine, the processed part is unloaded and taken to the output buffer. The robot moves between the input buffer, machines, and output buffers carrying the parts. The robot has three types of activity. They are loading-unloading operation, waiting, and move. Loading/unloading operation occurs when the robot is in front of a machine or a buffer. We assume loading-unloading activities take a constant time. The robot may wait in front of a machine. Waiting time depends on the activity sequence of the robot, part sequence, and processing times of the parts.

We will first consider the makespan minimization problem (C$_{max}$ problem) where the robot moves at a constant speed i.e. at its maximum speed to minimize makespan. If the robot speed is infinite, the problem will be a $P2|p_j|C_{max}$ which is NP-hard. So, we can say that the makespan minimization on two parallel machines with input/output buffers and a material handling robot is NP-hard. In order to solve this problem, we propose a mixed integer program (MIP). Also, for large instances where the MIP model may not be solved within reasonable CPU times, we develop a greedy algorithm and two simulated annealing algorithms.

Then, along with makespan objective, we will consider minimization of the energy consumption resulting from robot moves. This second problem with two objectives will be called E&C$_{max}$ problem. During a move between machines/buffers, the energy consumed by the robot depends on the speed of the robot and other parameters. We assume there is a nonlinear relation between energy consumption function and robot speed. Increasing the robot speed decreases the makespan but also increases energy consumption. Having these two conflicting objectives, we will use $\epsilon$-constraint method to find efficient solutions. In particular, we minimize the energy consumption function for a given bound on makespan objective. We give a mathematical model with a nonlinear objective function and linear constraints which will be reformulated via second-order conic inequalities. For large instances, we develop a greedy algorithm and two simulated annealing algorithms.

The decision maker should prepare a schedule when the part set to be processed is determined. Therefore, this scheduling problem is an operational level problem. $C_{max}$ and E&$C_{max}$ problems would be solved for each different part set.

Next, we give the basic notation used throughout this chapter.

***Notation:***

| | |
|---|---|
| $l$: | index for machines |
| $i$: | index for parts |
| $r$: | index for robot routes |
| $m$: | index for robot moves |
| $h \in \{e, f\}$: | index for status of the robot, $e$: empty, $f$: full |
| $L$: | set of machines, $\{1, 2\}$ |
| $R$: | set of robot routes |
| $P_{l,i}$: | processing time of part $i$ on machine $l$ |
| $\varepsilon$: | time required for loading and unloading a part on a machine |

## 4.1 Routes of the Robot

In this scheduling environment, all parts are in the input buffer at the beginning. The robot takes each part from the input buffer and carries it to one of the two machines, and loads it to the machine. The robot picks up each finished part from its machine and carries it to output buffer and loads. As parts flow from the input buffer to the output buffer, the robot carries out activities which are moves between machines/buffers, loading/unloading machines, and waiting at machines.

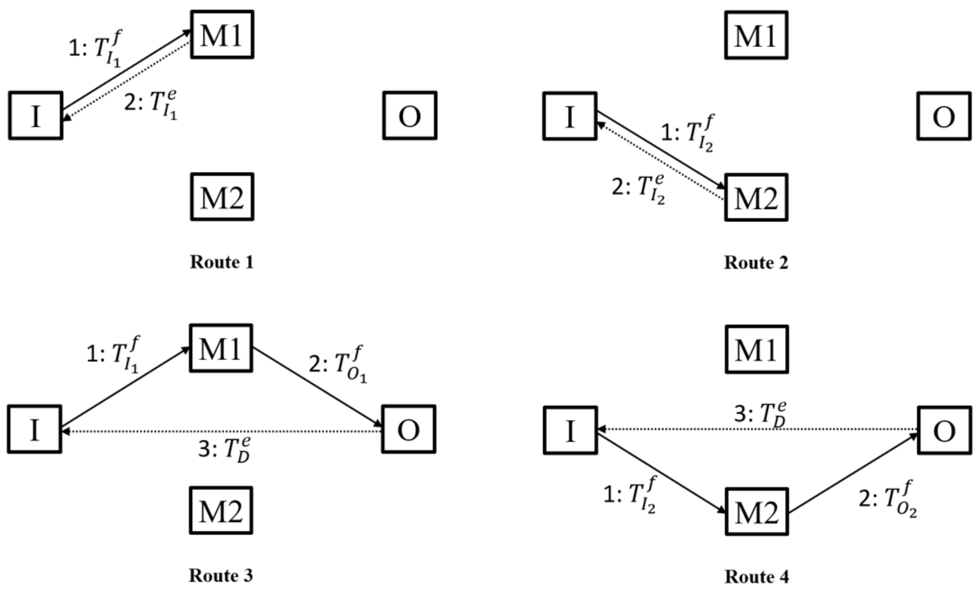
In this study, we first claim that the entire schedule of the robot can be considered as a sequence of robot routes. A robot route consists of a certain sequence of robot activities. At the beginning of each route, the robot is ready in front of the input buffer to unload the next part from the buffer. The robot unloads the part, takes it to a machine, loads the machine then the robot does some other activities and goes back to the input buffer. In each robot route, only one part is taken from the input buffer and loaded on a machine and the route ends when the robot arrives at the input buffer the next time. During a route, the robot may do other activities such as unloading a machine, waiting, moving, loading output buffer. We define 12 possible robot routes which will be described in detail. Any robot schedule for this scheduling environment can be expressed as a sequence of robot routes.

At the beginning of each route, the machines can be in three possible states. In the first state, both machines are empty. In the second state, machine 1 is full and machine 2 is empty. In the third state, machine 1 is empty and machine 2 is full. After executing a robot route, the machines will be in one of these three states again. As will be explained, the set of routes that a robot can execute at a given time depends on the state of the machines at that time. If a machine is already loaded and running, the robot cannot load a part on that machine, so routes which include loading that machine cannot be executed.

We give robot activity sequences in routes in Figures 4.1, 4.2, and 4.3. We consider machines are located parallel and between the buffers. We have paths that robot moves. $I_l$ represents the path between the input buffer and machine $l$, $O_l$ represents the path between machine $l$ and the input buffer, $B$ represents the path between machines, $D$ represents the path between the buffers. In these Figures, the dashed arrows show empty moves during which the robot travels without a part on it. Continuous arrows indicate full moves during which the robot carries a part. The activities in the routes are given following. In the Figures, $a : T_g^h$ point out moves in the routes where $a$ is the order of move, $g$ is the path the robot traveled in the move, $h$ is the status of the robot during the move.

**Route 1:** In Route 1, the robot picks up a part ($\varepsilon$) from the input buffer, then carries

Figure 4.1: The robot moves in routes 1, 2, 3, and 4



it to machine 1 ($T^f_{I_1}$) and loads the part ($\varepsilon$), then returns to the input buffer ($T^e_{I_1}$).

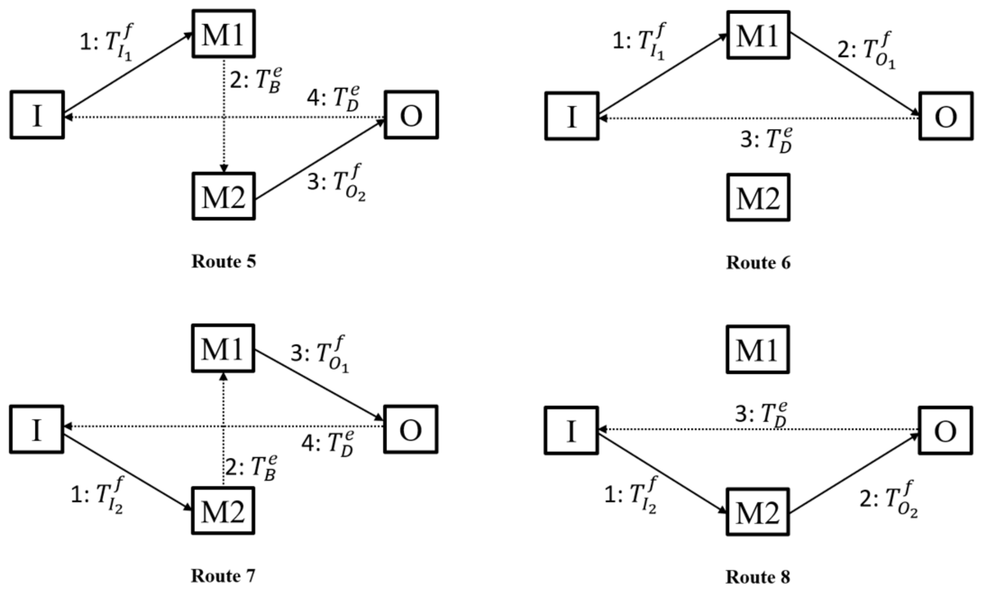
**Route 2:** Route 2 is similar to route 1, but the robot loads machine 2.

**Route 3:** In route 3, the robot picks up a part ($\varepsilon$) from the input buffer, then carries it to machine 1 ($T^f_{I_1}$) and loads the part ($\varepsilon$). The robot waits until the part is processed ($P_{1,i}$), then unloads the part from machine 1 ($\varepsilon$). After that, the robot carries the part to the output buffer ($T^f_{O_1}$) loads the part ($\varepsilon$) to the output buffer. Finally, it returns to the input buffer ($T^e_D$). During the execution of this route, machine 2 is empty.

**Route 4:** Route 4 is similar to Route 3, but this time the robot loads and unloads machine 2.

**Route 5:** In route 5, the robot picks up a part ($\varepsilon$) from the input buffer, then transports to machine 1 ($T^f_{I_1}$) and loads the part ($\varepsilon$), then, moves to machine 2 ($T^e_B$). If the part on machine 2 is finished, the robot unloads the part ($\varepsilon$), else the robot waits until the process is finished ($w^2_5$) and unloads the part ($\varepsilon$). Then, the robot transports the part to the output buffer ($T^f_{O_2}$) and loads the part ($\varepsilon$) to the output buffer. Finally, return to the input buffer ($T^e_D$). This route can be executed only when a part is already loaded

Figure 4.2: The robot moves in routes 5, 6, 7, and 8



on machine 2.

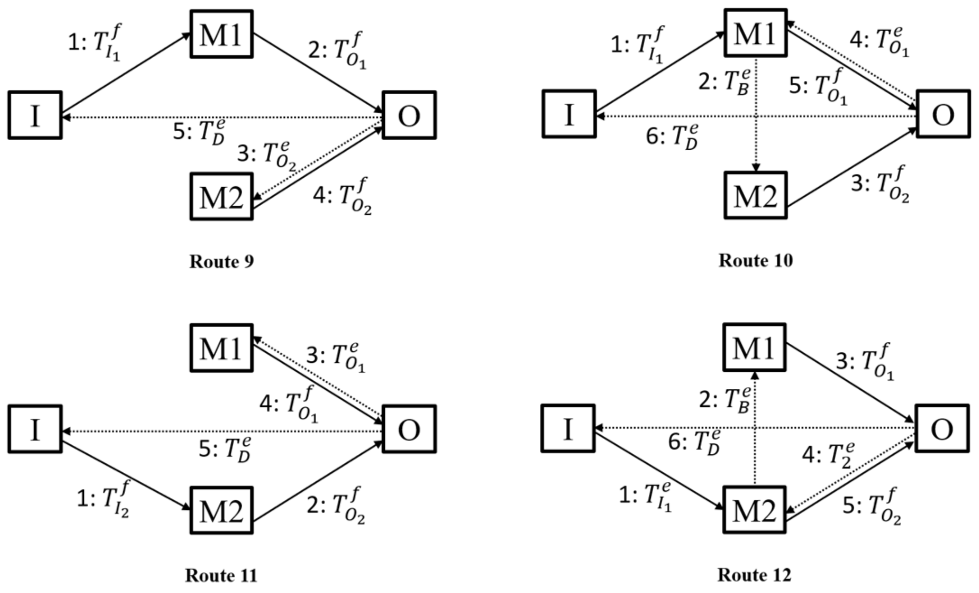
**Route 6:** In route 6, the robot performs the same activities as in route 3 but this time machine 2 is full during the execution of the route.

**Route 7:** Route 7 is similar to Route 5. The robot performs the following activities. The robot picks up a part ($\varepsilon$) from the input buffer, then transports to machine 2 ($T^f_{I_2}$) and loads the part ($\varepsilon$), then, moves to machine 1 ($T^e_B$). If the part is finished on machine 1, the robot unloads the part ($\varepsilon$), else the robot waits until the process is finished ($w^1_7$) and then unloads the part ($\varepsilon$). Then, the robot takes the part to the output buffer ($T^f_{O_1}$) and loads the part ($\varepsilon$) to the output buffer. Finally, the robot returns to the input buffer ($T^e_D$).

**Route 8:** In route 8, the robot performs the same activities in route 4. while machine 1 is empty during route 4, machine 1 is full in this route.

**Route 9:** In route 9, the robot unloads a part ($\varepsilon$) from the input buffer, then transports the part to machine 1 ($T^f_{I_1}$) and loads ($\varepsilon$). The robot waits until the part is finished on machine 1 ($P_{1,i}$) and unloads the part ($\varepsilon$). After that, the robot carries the part to the

Figure 4.3: The robot moves in routes 9, 10, 11, and 12



output buffer ($T^f_{O_1}$) and loads ($\varepsilon$) to the output buffer. The robot moves to machine 2 ($T^e_{O_2}$). If the part on machine 2 is finished, then the robot unloads the part ($\varepsilon$), else it waits until the process is finished ($w^2_9$) and then loads the part ($\varepsilon$). The robot transports the part to the output buffer ($T^f_{O_2}$) and loads the part ($\varepsilon$) to the output buffer. Finally, it returns to the input buffer ($T^e_D$).

**Route 10:** In route 10, the robot first unloads a part ($\varepsilon$) from the input buffer, then transport to the machine 1 ($T^f_{I_1}$) and loads the part ($\varepsilon$), then, moves to machine 2 ($T^e_B$). If the part on machine 2 is finished, then the robot unloads the part ($\varepsilon$), else the robot waits until the process is finished ($w^2_{10}$) and then unloads the part ($\varepsilon$). Then, the robot transports the part to the output buffer ($T^f_{O_2}$) and loads the part ($\varepsilon$) to the output buffer. After that, the robot moves to machine 1 ($T^e_{O_1}$). If the part is finished, the robot unloads the part ($\varepsilon$), else the robot waits until the process is finished ($w^1_{10}$) and unloads the part ($\varepsilon$). Then, the robot transports the part to the output buffer ($T^f_{O_1}$), then loads the part ($\varepsilon$) to the output buffer. Finally, return to the input buffer ($T^e_D$).

**Route 11:** Route 11 is similar to Route 9. The robot unloads a part ($\varepsilon$) from the input layer, then takes the part to machine 2 ($T^f_{I_1}$) and loads ($\varepsilon$). The robot waits until the part is processed ($P_{2,i}$), then unloads the part ($\varepsilon$). After that, the robot carries the part to the output buffer ($T^f_{O_2}$) loads the part ($\varepsilon$). After that, the robot moves to machine 1 ($T^e_{O_1}$). If the part on machine 1 is finished, the unloads the part ($\varepsilon$), else the robot waits until the process is finished ($w^1_{11}$) and then unloads the part ($\varepsilon$). Then, the robot transports the part to the output buffer ($T^f_{O_1}$) and loads ($\varepsilon$). Finally,the robot returns to the input buffer ($T^e_D$).

**Route 12:** Route 12 is similar to Route 10. The robot unloads a part ($\varepsilon$) from the input buffer, then transports to machine 2 ($T^f_{I_2}$) and loads the part ($\varepsilon$), then, moves to the machine 1 ($T^e_B$). If the part on machine 1 is finished, the robot unloads the part ($\varepsilon$), else the robot waits until the process is finished ($w^1_{12}$) and then unloads ($\varepsilon$). Then, the robot transports the part to the output buffer ($T^f_{O_1}$) and loads ($\varepsilon$). After that, the robot moves to machine 2 ($T^e_{O_2}$). If the part on machine 2 is finished, the robot unloads the part ($\varepsilon$), else the robot waits until the process is finished ($w^2_{12}$) and then unloads ($\varepsilon$). Then, the robot transports the part to the output buffer ($T^f_{O_2}$) and load the part ($\varepsilon$) to the output buffer. Finally, it returns to the input buffer ($T^e_D$).

**Proposition 1.** *We can express all possible robot schedules with 12 routes.*

*Proof.* At the beginning of the schedule, the parts are in the input buffer and the robot is in front of the input buffer. At the end of the schedule, the parts are in the output buffer and the robot is in front of the input buffer. At the beginning of a route, the robot is ready to load the next part in front of the input buffer and machines can be three at three states. For each state, there are 4 possible routes which can be executed. If machines are at state 1 (both machines are empty), the robot can load machine 1 or machine 2 (route 1 and route 2). After loading, the robot can immediately return to input buffer or unload the part (route 3 and route 4). If machines are at state 2 (machine 1 is full, machine 2 is empty), the robot can load the part to only machine 2. After loading, the robot can unload only machine 2 (route 8) or unload only machine 1 (route 7) or unload first machine 2 then machine 1 (route 11) or unload first machine 1 then machine 2 (route 12). If machines are at state 3, the robot can load the part to only machine 1. After loading, the robot can unload only machine 1 (route 6) or unload only machine 2 (route 5) or unload first machine 1 then machine 2 (route 9) or unload first machine 2 then machine 1 (route 10). □

Beginning and ending states of the routes, and possible succeeding routes are given in Table 4.1. Here, we can group the routes into subsets. Let $R_1 = \{3, 4, 9, 10, 11, 12\}$, $R_2 = \{1, 2, 3, 4\}$, $R_3 = \{1, 5, 8\}$, $R_4 = \{7, 8, 11, 12\}$, $R_5 = \{2, 6, 7\}$, and $R_6 = \{5, 6, 9, 10\}$ be route sets. The robot should use one of the $R_2$ routes after using the $R_1$ routes, one of the $R_4$ routes after using the $R_3$ routes, and one of the $R_6$ routes after using the $R_5$ routes to compose a feasible sequence of robot routes.

Table 4.1: Beginning and ending states of the routes

| | **Beginnig State** | | **Ending State** | | |
|---|---|---|---|---|---|
| **Route** | **Machine 1** | **Machine 2** | **Machine 1** | **Machine 2** | **Successor Routes** |
| **1** | empty | empty | full | empty | 7,8,11,12 |
| **2** | empty | empty | empty | full | 5,6,9,10 |
| **3** | empty | empty | empty | empty | 1,2,3,4 |
| **4** | empty | empty | empty | empty | 1,2,3,4 |
| **5** | empty | full | full | empty | 7,8,11,12 |
| **6** | empty | full | empty | full | 5,6,9,10 |
| **7** | full | empty | empty | full | 5,6,9,10 |
| **8** | full | empty | full | empty | 7,8,11,12 |
| **9** | empty | full | empty | empty | 1,2,3,4 |
| **10** | empty | full | empty | empty | 1,2,3,4 |
| **11** | full | empty | empty | empty | 1,2,3,4 |
| **12** | full | empty | empty | empty | 1,2,3,4 |

In some routes, the robot waits in front of a machine until the process on the machine is completed. If the robot loads the machine and then waits during the process, this is called full waiting. In the other case, if the robot loads the machine, then leaves to perform some activities before returning to the machine and waits for the part to be unloaded, this waiting is called partial waiting. Waiting times in full waiting are equal to the processing time of the part processed while waiting times in partial waiting are less than the processing time of the part processed.

The length of a full waiting in a route depends on the processing time of the part that is being processed in that route. We have full waiting in front of machine 1 in routes

3,6 and 9, and in front of machine 2 in routes 4,8, and 11. We define $FWR^l$ as the set indicates routes having full waiting in front of machine $l$.

The partial waiting times depend on part sequence and routes used. Since the problem involves part sequence and route sequence decisions, partial waiting times are also decision variables. Table 4.2 shows the routes that load machines and routes unload machines. If a loading route is used to load a machine, one of the subsequent routes must be an unloading route that unloads the same machine. In unloading routes, partial waitings can occur. In routes 7, 11, 12, the robot can have partial waiting at machine 1 if the machine is loaded in the preceding route (1 or 5). In addition, in route 10, a partial waiting can occur at machine 1 for the part that is loaded in the same route. The robot can partially wait for the part which is put in machine 2 in the past routes 2 or 7 in routes 5, 9, 10. The robot can partially wait in route 12 for the part that is loaded to machine 2 in the same route 12. Let $PWR^l$ be the set that indicates routes having partial waiting in front of the machine $l$. Moreover, Let $A_l$ be the set consisting of pairs $(r_1, r_2)$ where the partial waiting in front of machine $l$ can occur in the route $r_2$ for the part loaded in route $r_1$.

Table 4.2: Loading and unloading routes in partial waitings

| **Machine** | **Loading routes** | **Unloading routes** |
|---|---|---|
| 1 | 1, 5 | 7, 11, 12 |
| 2 | 2, 7 | 5, 9, 10 |

We introduce the following move sets, which have an effect on partial waiting times. $AM^l_{r_1}$ contains the robot moves in route $r_1$ after loading a part on machine $l$, $BMr_2{}^l$ which contains the robot moves in route $r_2$ before loading a part on machine $l$ where $(r_1, r_2) \in A_l$. $IM^1_{10}$ consists of robot moves between loading and unloading machine 1 in route 10, and $IM^2_{12}$ consists of robot moves between loading and unloading machine 2 in route 12. We also define the parameter $\xi^l_{r_1,r_2}$ indicating the sum of loading-unloading times after loading machine $l$ in route $r_1$ and before unloading machine $l$ in route $r_2$.

In the next section, we will present C$_{max}$ problem and proposed solution approaches.

## 4.2 Makespan Minimization (C$_{max}$) Problem

We first study the makespan minimization (C$_{max}$) problem. In C$_{max}$ problem, we have parts to be processed. Each part has to be processed on a machine first and then delivered to the output buffer. A solution to this problem is a part sequence and a corresponding route sequence. The part at position $k$ in the part sequence is loaded on a machine by executing the $k^{th}$ route in the route sequence. The total time required to have all parts processed and delivered to the output buffer is the total time of the routes in the route sequence. As discussed before, there may occur a partial waiting time during a robot route, so the time required to execute such a route may depend on the activities in the route, in its processor and the corresponding parts in the sequence.

The time elapsed during a robot move depends on the speed of the robot. In C$_{max}$ problem, we assume that the robot speeds are constant and at the maximum level. The time elapsed in a robot move can be calculated as the distance traveled in the robot move divided by the speed of the robot. We first give a mathematical formulation for the makespan minimization problem. Then, we develop three algorithms to solve large instances.

### 4.2.1 Mathematical Model for C$_{max}$ Problem

In this section, we develop and explain the mathematical model for C$_{max}$ problem. First, we define indexes, sets, parameters, and decision variables used in the model.

***Indexes:***

$t$: position index in part sequence

***Sets:***

| | |
|---|---|
| $K$: | set of parts |
| $T$: | set of positions |
| $M_r$: | set of robot moves in route $r$ |
| $AM_r^l$: | set of robot moves in route $r$ after loading machine $l$ |
| $BM_r^l$: | set of robot moves in route $r$ before loading machine $l$ |

$IM_{10}^{1}$: set of robot moves between loading and unloading machine 1 in route 10

$IM_{12}^{2}$: set of robot moves between loading and unloading machine 2 in route 12

$FWR^{l}$: set of routes having full waiting in front of machine $l$.

$PWR^{l}$: set of routes having partial waiting in front of machine $l$.

$A_l$: set of route pairs $(r_1, r_2)$ where partial waiting in front of machine $l$ can occur in the route $r_2$ for the part loaded in route of $r_1$.

***Parameters:***

$T_m$: time elapsed in robot move $m$

$\xi_{r_1,r_2}^{l}$: the sum of loading-unloading times after loading machine $l$ in route $r_1$ and before unloading machine $l$ in route $r_2$

$d_m$: distance travelled by the robot during move $m$

$\vartheta_r$: the total time required for loading-unloading operations in route $r$

***Decision Variables***

$$x_{i,r,t}: \begin{cases} 1, \text{ if part } i \text{ is started processing in robot route } r \text{ at robot route position } t \\ 0, \text{ otherwise} \end{cases}$$

$c_t$: The time elapsed at position $t$

$w_{r,t}^{l}$: robot's waiting time in front of machine $l$ in route $r$ at position $t$

We assume that the robot travels with its maximum speed level during all moves, the time elapsed in move $m$ is calculated as $T_m = \frac{d_m}{v^{max}}$.

$$\min \sum_{t \in T} c_t \tag{4.1}$$

$$\sum_{i \in K} \sum_{r \in R} x_{i,r,t} = 1 \quad \forall t \in T \tag{4.2}$$

$$\sum_{r \in R} \sum_{t \in T} x_{i,r,t} = 1 \quad \forall i \in K \tag{4.3}$$

$$\sum_{i \in K} \sum_{r \in R_2} x_{i,r,1} = 1 \tag{4.4}$$

$$\sum_{i \in K} \sum_{r \in R_1} x_{i,r,N} = 1 \tag{4.5}$$

$$\sum_{i \in K} \sum_{r \in R_1} x_{i,r,t} = \sum_{i \in K} \sum_{r \in R_2} x_{i,r,t+1} \quad \forall t \in T \setminus \{n\} \tag{4.6}$$

$$\sum_{i\in K}\sum_{r\in R_3} x_{i,r,t} = \sum_{i\in K}\sum_{r\in R_4} x_{i,r,t+1} \quad \forall t \in T\setminus\{n\} \tag{4.7}$$

$$\sum_{i\in K}\sum_{r\in R_5} x_{i,r,t} = \sum_{i\in K}\sum_{r\in R_6} x_{i,r,t+1} \quad \forall t \in T\setminus\{n\} \tag{4.8}$$

$$c_t = \sum_{i\in K}\sum_{r\in R}\left((\vartheta_r + \sum_{m\in M_r} T_m)x_{i,r,t}\right) + \sum_{i\in K}\sum_{l\in L}\sum_{r\in FWR^l} P_{l,i}x_{i,r,t}$$
$$+\sum_{l\in L}\sum_{r\in PWR^l} w^l_{r,t} \quad \forall t\in T \tag{4.9}$$

$$w^l_{r_2,t} \geq \sum_{j\in K} P_{l,j}\left(\sum_{i\in K} x_{i,r_2,t} + x_{j,r_1,z} - 1\right)$$
$$-\sum_{l\in L}\left(\sum_{m\in AM^l_{r_1}} T_m + \sum_{m\in BM^l_{r_2}} T_m\right) - \xi^l_{r_1,r_2}$$
$$-\sum_{k=z+1}^{t-1} c_k \quad \forall l\in L, \forall (r_1,r_2)\in A_l, \forall t\in T\setminus\{1,n\}, z=1,2,..,t-1 \tag{4.10}$$

$$w^1_{10,t} \geq \sum_{i\in K} x_{i,10,t}\left(P_{1,i} - \sum_{m\in IM^1_{10}} T_m - 2\varepsilon\right) - w^2_{10,t} \quad \forall t\in T\setminus\{1\} \tag{4.11}$$

$$w^2_{12,t} \geq \sum_{i\in K} x_{i12t}\left(P_{2,i} - \sum_{m\in IM^2_{12}} T_m - 2\varepsilon\right) - w^1_{12,t} \quad \forall t\in T\setminus\{1\} \tag{4.12}$$

$$w^l_{r,t} \leq P^{max}_l \sum_{i\in K} x_{i,r,t} \quad \forall l\in L, \forall r\in PWR^l, \forall t\in T \tag{4.13}$$

$$x_{i,r,t}\in\{0,1\} \quad \forall i\in K, \forall r\in R, \forall t\in T \tag{4.14}$$

$$w^l_{r,t}\geq 0 \quad \forall l\in L, \forall r\in PWR^l, \forall t\in T \tag{4.15}$$

The objective function (4.1) is the sum of the route times required to finish all parts, i.e. all parts are processed and delivered to the output buffer and the robot is ready at the input buffer. At each position in the schedule, a part will be unloaded from the input buffer and delivered to a machine by one of the routes. This is ensured by constraint (4.2). Constraints (4.3) provide that each part will be taken from the input buffer and loaded on a machine by a selected route at a selected position. At the beginning, we assume both machines are empty, so the route in the first position must be in the set $R_2$ which has routes that can be executed when both two machines are empty. In the end, the machines should be empty so the route at the last position must be selected from the set $R_1$. Therefore, constraints (4.4) and (4.5) guarantee first and last routes are selected appropriately.

As shown in Table (4.1), each route can be succeeded by certain other routes. Constraints (4.6), (4.7), and (4.8) ensure that these precedence restrictions between routes are satisfied. Required time to complete a route at a position is the sum of time elapsed in loading-unloading operations, the times elapsed in robot moves, full and partial waiting times. Constraint (4.9) calculates the execution time of a route at a position.

In constraints (4.10), (4.11), and (4.12) calculate partial waiting times. With constraints (4.10), if part $j$ is loaded to machine $l$ using route $r_1$ at position $z$ and it is unloaded using another route $r_2$ at position $t$, the waiting time for machine $l$ at position $t$ is determined as subtracting the time elapsed until the robot comes to machine $l$ at position $t$ (time elapsed after loading part $j$ at position $z$, time elapsed between positions $z$ and $t$, and time elapsed until unloading the part at position $t$) from the processing time of part $j$ in machine $l$. In route 10 (route 12), we have a waiting time if the processing time of a part loaded to machine 1 (machine 2) is greater than the time elapsed until returning to machine 1 (machine 2). With constraints (4.11) and (4.12), we determine the waiting times if route 10 and route 12 are used at any position. Because the robot performs some activities after loading a part to a machine, the waiting time is less than the processing time of that part in that machine. So, we define upper bounds for partial waiting times for a machine as the maximum processing time in that machine. Constraints (4.13) provide the upper bound for partial waitings. Also, these constraints provide the partial waiting times at a position using a route for a machine is zero if we do not use that route at that position. Constraints (4.14) and (4.15) define the domains of the variables. The makespan minimization problem is formulated above as a mixed-integer linear program.

### 4.2.2 Heuristic Algorithms for C$_{max}$ Problem

The mathematical model proposed in the previous section can be solved by using branch and bound software such as IBM CPLEX and Gurobi. However, for large instances, i.e. as the number of parts increases, the solvers could be insufficient to find the optimal solution in reasonable CPU times. Therefore, to find good solutions quickly, we propose three algorithms, the first one is a greedy neighborhood search (GNS), the other two algorithms are based on the Simulated Annealing metaheuristic. First, we will describe the neighborhood structures used and define some subroutines used in the algorithms. Then, we will give the steps of the algorithms.

**Neighborhood of a Solution**

We define the neighborhood of a solution by introducing four types of operations. The first operation is called 1-PartMove. 1-PartMove changes the position of a part in the part sequence and also changes the robot route of the part. Then, the robot route of the part is marked as fixed. For example, consider a problem with five parts. We have part sequence $\sigma_K = (3, 1, 5, 4, 2)$ and route sequence $\sigma_R = (2, 5, 7, 5, 12)$. One possible 1-PartMove operation is moving part 1 from position 2 to position 4 and changing its route from 5 to 2. After this operation, we achieve the solution $\sigma_K = (3, 5, 4, 1, 2)$ and $\sigma_R = (2, 7, 5, 2, 12)$. In addition, the route of part 1 at position 4 is marked as fixed. We define the set $MR$ as the set of the positions with fixed routes. Notice that the new robot route sequence is not feasible. Therefore, we repair the route sequence without changing the routes at positions in $MR$. We put all possible neighboring solutions of a solution $s$ using 1-PartMove operation in the set of neighborhood 1 ($N_1(s)$).

The second operation is called 2-PartSwap. This operation swaps positions of two parts and then changes their robot routes. Then we mark the routes of these parts as fixed. Consider the solution $s$ in the previous example, one of the possible 2-PartSwap operations is swapping part 3 at position 1 and part 4 at position 4. After swapping, let's say we change the robot route of part 3 to 9 and the robot route for part 4 to 3. After that, the new solution will have $\sigma_K = (4, 1, 5, 3, 2)$ and $\sigma_R = (3, 5, 7, 9, 12)$. We mark the positions of part 3 and part 4 as fixed, so $MR = \{1, 4\}$. The set $N_2(s)$ includes all possible neighboring solutions of a solution $s$ by the second operation type.

The third operation is called 1-RouteChange. This operation selects a part and changes its robot route and then marks that route as fixed. We store all possible neighboring solutions of a solution $s$ generated by 1-RouteChange in set $N_3(s)$. The last operation is called 3-RouteChange. This operation selects three consecutive parts and changes their robot routes. While doing this, it ensures that the new three consecutive routes are compatible with each other. The set $N_4(s)$ stores all possible neighboring solutions of a solution $s$ applying 3-RouteChange.

In all operations, when changing the routes at the first and the last positions, routes in sets $R_2$ and $R_1$ are used, respectively. The solutions we define in the neighborhoods consist of a part sequence ($\sigma_K$), a route sequence ($\sigma_R$), and a set that contains the positions of fixed routes ($MR$). In addition, we combine all neighbor solutions into a single $N(s)$ set. A solution contains a part sequence $\sigma_K$ and a route sequence $\sigma_R$ $s = (\sigma_K, \sigma_R)$.

**Makespan Calculation Algorithm (MCA)**

A solution to C$_{max}$ problem can be represented as a sequence of parts ($\sigma_K$) and a sequence of routes ($\sigma_R$). MCA takes $\sigma_K$ and $\sigma_R$ as input and returns the completion time of the last part (makespan). We give the steps of MCA in Algorithm 1.

Makespan is the sum of loading-unloading times, the times elapsed in the robot moves, times of full waitings, and times of partial waitings. The sequence of these activities is known by the given route sequence. Here, loading-unloading times are parameters, the time elapsed in the robot moves are calculated easily by $\frac{d_m}{v^{max}}$, and time of full waiting in a route is the processing time of the part which is loaded on a machine in that route. Partial waiting times must be carefully calculated. After loading a part on a machine, the robot may leave the machine and carry out some other activities. When the robot is back for unloading the part, if the process has not finished yet, it has to wait. This waiting time is the difference between the processing time of the part and the total duration of robot activities between its departure from the machine and its arrival to the machine. If the difference is greater than or equal to 0, the partial waiting time is the remaining processing time on the machine; otherwise, the partial waiting time is 0 and the robot unloads the part as soon as it arrives at the machine.

Algorithm 1 gives the steps of MCA. MCA first generates robot activity sequence in order of occurrence in line 3. It contains loading-unloading operations and the robot moves. Then, we initialize the variables. $makespan$ is used to calculate makespan. $R_l$ is used to store the remaining time of the process on machine $l$. Then, in for loop (lines 5-14), we update the variables according to loading and unloading activities on the machines. The function $Time(A)$ return the activity times. If the activity is a loading or unloading operation, it returns the parameter $\varepsilon$. If the activity is a robot move, it returns the time elapsed in the move when the speed of the robot is at maximum level.

**Algorithm 1** Makespan Calculation Algorithm (MCA)

```
1: Input: a solution $s = (\sigma_K, \sigma_R)$
2: Output: makespan of the given solution $s$
3: Create the activity sequence ($AS$) of the solution $s$
4: $makespan \leftarrow 0$ , $R_1 \leftarrow 0$, $R_2 \leftarrow 0$
5: for $activity\ A \in AS$ do
6:     $makespan \leftarrow (makespan + Time(A))$
7:     if part $i$ is loaded to machine $l$ at the end of $A$ then
8:         $R_l \leftarrow P_{l,i}$
9:     else if machine $l$ is unloaded at the end of $A$ then
10:        $makespan \leftarrow (makespan + max\{0, R_l\})$, $R_l \leftarrow 0$
11:    else
12:        $R_1 \leftarrow (R_1 - Time(A))$, $R_2 \leftarrow (R_2 - Time(A))$
13:    end if
14: end for
15: return $makespan$
```

As discussed before, each route can succeed certain other routes and can be succeeded by certain routes. When the aforementioned operations are applied to a solution to achieve a neighbor solution, these restrictions may be violated. In such a case, we need to repair the solution as explained below.

**Sequence Repairing Algorithm (SRA)**

SRA is a heuristic algorithm that repairs route precedence constraints in a given solution. SRA outputs a part sequence and a feasible route sequence. We give the steps of SRA in Algorithm 2.

SRA takes a solution consisting of a part sequence ($\sigma_K$) and a route sequence ($\sigma_R$), and a set that contains positions of fixed routes ($MR$). We first initialize an empty set $CR$ and initialize makespan value to infinity. $CR$, stores the positions of the routes that will be changed to achieve a feasible route sequence. Between lines 4-13, SRA determines the routes which will be changed. Here, SRA first determines the routes that are not fixed routes and cannot succeed fixed routes and determines the routes that cannot follow the fixed routes. After that, if $CR$ is empty, it means the route sequence is feasible and the SRA returns the current solution (lines 14-17). If $CR$ is not empty, SRA creates a set of feasible route sequences by changing the routes in $CR$. In line 19, we define the set $PR$ and add the route sequence of the input solution to it.

Note that for any given two routes, we can find one or two suitable routes positioning between these two routes (see Table 4.1). Therefore, if we change a route, possibly one route or two routes can be replaced with that route. For example, consider a route sequence (1,7,$r$,9,4,3,2,$r'$,1,12) that contains 10 routes. Here, $r$ can be only 6 and $r'$ can be 9 and 10. So, at the end, we get two possible feasible route sequences that are (1,7,6,9,4,3,2,9,1,12), (1,7,6,9,4,3,2,10,1,12). In this way, in lines 20-26, we find all possible feasible route sequences by changing the routes in $CR$ and add them to the set $PR$. After determining all possible feasible route sequences, we select one route sequence that has minimum makespan among these feasible route sequences in the lines 27-32. In the end, SRA returns a feasible solution ($\sigma_K$, a feasible route sequence found ($\sigma_R^{final}$)) and makespan of that solution.

**Algorithm 2** Sequence Repairing Algorithm (SRA)

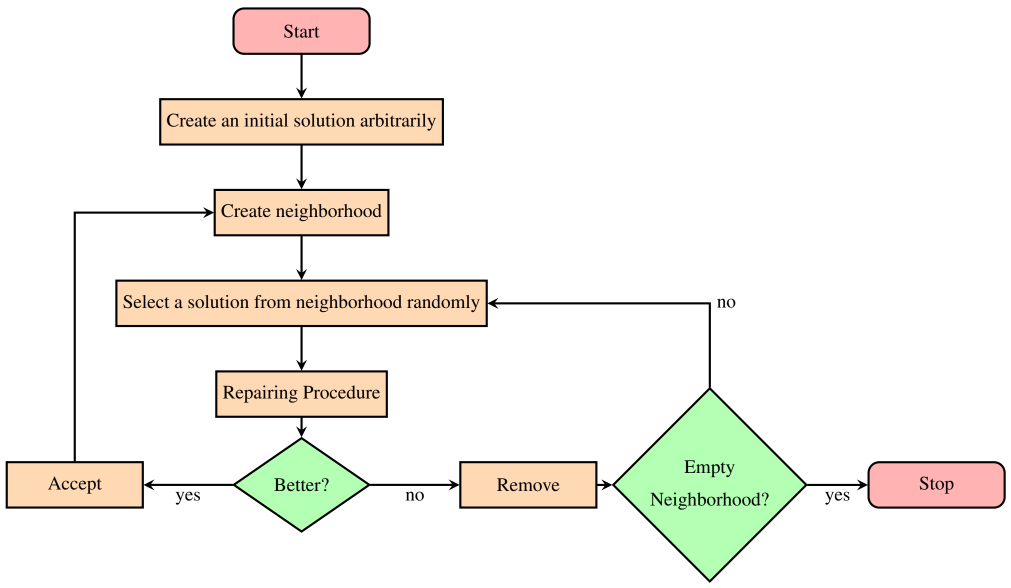
```
1: Input: a solution $s = (\sigma_K, \sigma_R)$, set of fixed routes ($MR$)
2: Output: Repaired solution, makespan of the repaired solution
3: $CR \leftarrow \emptyset, makespan^{final} \leftarrow \infty, n \leftarrow Numberofpart$
4: for $i = 2$ to $n$ do
5:     if $\sigma_K(i-1)$ cannot succeed $\sigma_K(i)$ and $\sigma_K(i) \notin MR$ then
6:         add $i$ to $CR$
7:     end if
8: end for
9: for $j \in MR$ do
10:    if $\sigma_K(j)$ cannot succeed $\sigma_K(j+1)$ then
11:        add $j+1$ to $CR$
12:    end if
```

```
13: end for
14: if $CR = \emptyset$ then
15:     $makespan^{final} \leftarrow$ MCA($\sigma_K, \sigma_R$)
16:     $s = (\sigma_K, \sigma_R)$
17:     return $makespan^{final}$, $s$
18: else
19:     $PR \leftarrow \emptyset$, add $\sigma_R$ to $PR$
20:     for $j \in CR$ do
21:         for $\sigma'_R \in PR$ do
22:             Create two (one) route sequences $\sigma^1_R, \sigma^2_R$ ($\sigma^1_R$) by changing $\sigma'_R(j)$
23:             Add $\sigma^1_R, \sigma^2_R$ ($\sigma^1_R$) to $PR$
24:             Remove $\sigma'_R$ from $PR$
25:         end for
26:     end for
27:     for $\sigma^p_R \in PR$ do
28:         if $makespan^{final}$ > $MCA(\sigma_K, \sigma'_R)$ then
29:             $makespan^{final} \leftarrow MCA(\sigma_K, \sigma^p_R)$
30:             $\sigma^{final}_R \leftarrow \sigma^p_R$
31:         end if
32:     end for
33: end if
34: $s = (\sigma_K, \sigma^{final}_R)$
35: return $s$, $makespan^{final}$
```

#### 4.2.2.1 Greedy Neighborhood Search Algorithm (GNS)

GNS is a neighborhood search approach which attempts to find an improved solution at each iteration. The algorithm generates neighbor solutions at each iteration moves to a better neighbor solution. As can be seen in Figure 4.4, we first generate an arbitrary initial solution and generate the neighborhood of the solution. Then, we select a random solution from the neighborhood and apply repairing procedure. If, the selected solution is better, we accept it and continue with creating the neighborhood of the accepted solution. If the selected solution is worse, we remove it from the neigh-

Figure 4.4: GNS Algorithm



borhood. We continue with selecting another solution from the neighborhood unless the neighborhood is empty. If neighborhood is empty, the algorithm terminates. We give the pseudo-code of GNS in Algorithm 3.

In GNS, $s_0$ indicates the current solution. In lines 2-3, GNS generates an arbitrary initial solution (initial $s_0$). Then, MCA calculates the makespan value of $s_0$. In lines 5-15, GNS selects a random neighbor solution and repairs it with SRA. If the selected solution is better than the current solution, GNS moves to the new solution and creates its neighborhood, else the solution is removed from the neighborhood. The algorithm terminates if the neighborhood is empty, so there is no better solution in the neighborhood. In the end, the algorithm returns the best solution found so far $s_0$.

**Algorithm 3** GNS Algorithm

```
1: Output: The solution found, makespan of the solution found
2: Create an initial solution $s_0 = (\sigma_K, \sigma_R)$ arbitrarily
3: $makespan^{final} \leftarrow MCA(\sigma_K, \sigma_R)$
4: Create the neighborhood of $s_0$ $(N(s_0))$
5: while $N(s_0)$ is not empty do
6:     Select a random solution $s'$ from $N(s_0)$
7:     $(makespan', s) \leftarrow$ SRA$(s', MR)$
```

```
8:     if $makespan^{final} > makespan'$ then
9:         $makespan^{final} \leftarrow makespan'$
10:        $s_0 \leftarrow s$
11:        Create $N(s_0)$
12:    else
13:        Remove the solution $s$ from $N(s_0)$
14:    end if
15: end while
16: return $makespan^{final}, s_0$
```

#### 4.2.2.2 Simulated Annealing based Algorithms

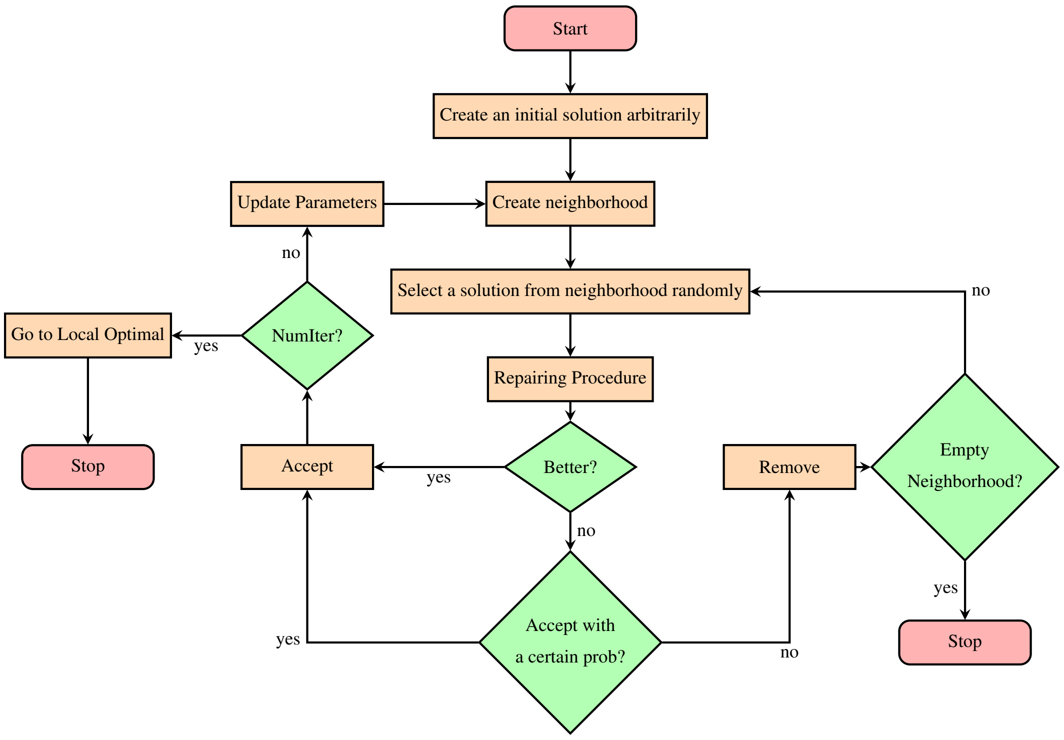
Simulated Annealing is a metaheuristic algorithm inspired by a method of increasing the size of a material's crystals and reducing defects by heating and cooling it in a controlled way. Pincus (1970), Khachaturyan et al. (1981), and Černý (1985) have all independently introduced similar techniques. Kirkpatrick et al. (1983) used this approach to solve the traveling salesman problem in 1983.

Many local optimal points can be found in a solution space of an optimization problem. The GNS algorithm selects a random initial solution and generates the neighborhood. If a randomly selected neighbor solution has a smaller makespan than the current solution, then it is accepted. The disadvantage of this algorithm is that it probably converges to a local optimal point so we do not explore the solution space and the final solution is highly dependent on the initial solution.

Simulated Annealing is an algorithm based on local search, it tries to avoid getting stuck at a local optimal solution as it can move to a worse solution with a certain probability. Algorithmic parameters in Simulated Annealing such as the number of iterations, initial temperature, and cooling rate affect the performance of the algorithm.

Simulated Annealing starts with an initial temperature ($t_0$) and in each iteration the temperature is decreased. Temperature is multiplied by a constant ($\alpha$), which has a value between 0 and 1. We propose two Simulated Annealing based algorithms for

Figure 4.5: SA-I Algorithm



C$_{max}$ problem. The first one, SA-I, creates the neighborhood of a solution by using all operations at the same time. The second algorithm SA-II generates the neighborhood of a solution by using one operation at a time.

In SA-I, different than the GNS algorithm, we can accept an inferior solution with a certain probability as shown in Figure 4.5. That probability value decreases as the iteration number increases. When we reach a certain number of iterations, the algorithm terminates and returns the best solution found so far. Also, when the neighborhood is empty, the algorithm terminates. We give the steps of SA-I in detail in Algorithm 4.

SA-I first creates an initial arbitrary solution ($s_o$). $s^{final}$ stores the best solution found so far. $makespan^{final}$ denotes its makespan level and it is updated as better solutions are found. $s^{final}$ is initialized to $s_o$ in line 2. $s_0$ is the current solution with $makespan$ value. In the next line, the makespan of the initial solution is calculated using $MCA$. In line 4, temperature ($t$) is initialized, iteration number $i$ is initialized to 1, and SA-I generates the neighborhood of $s_o$. Then, in lines 6-18, a random solution ($s'$) is selected from the neighborhood and repaired ($s$). If makespan of $s$ is smaller than the makespan of $s_0$, then $s_0 \leftarrow s$. Iteration number $i$ is incremented and temperature $t$ is updated. If $s$ is not a better solution than $s_0$, then function $GenerateRandom()$ generates a random number between 0 and 1. If the randomly generated number is less than $e^{-\frac{(makespan' - makespan)}{t}}$, $s_0 \leftarrow s$. $i$ and $t$ are updated. Here, $t$ is the current temperature so the probability of accepting a worse solution is decreasing as the iteration number increase. Otherwise, the selected solution is removed from the neighborhood. In lines 19-22, we check whether the current solution is better than the best solution found so far $s^{final}$ or not. If so, we update the $s^{final}$ as the current solution.

Next, in the line 24-41, we go to the local minimum solution if the neighborhood is not empty. SA-I continues this process until reaching the number of iterations limit or the neighborhood has no solution.

**Algorithm 4** SA-I

```
1: Output: Makespan, $s^{final}$
2: Create an initial solution $s_0 = (\sigma_K, \sigma_R)$ randomly, $s^{final} \leftarrow s_0$
3: Create $N(s_0)$, $i \leftarrow 1$, $t \leftarrow t_0$
4: Call MCA to calculate $makespan$, $makespan^{final} \leftarrow makespan$
5: while $i < NumIteration$ and $N(s_0)$ is not empty do
6:     Select a solution $s'$ from $N(s_0)$ randomly
7:     $(makespan', s) \leftarrow$ SRA$(s', MR)$
8:     if $makespan' - makespan < 0$ then
9:         $makespan \leftarrow makespan'$
10:        $s_0 \leftarrow s$
11:        Create $N(s_0)$, $t \leftarrow t\alpha,\ i \leftarrow i + 1$
12:    else if $GenerateRandom() < e^{-\frac{(makespan' - makespan)}{t}}$ then
13:        $makespan \leftarrow makespan'$
14:        $s_0 \leftarrow s$
15:        Create $N(s_0)$, $t \leftarrow t\alpha,\ i \leftarrow i + 1$
16:    else
17:        Remove the solution $s$ from $N(s_0)$
18:    end if
19:    if $makespan < makespan^{final}$ then
20:        $makespan^{final} \leftarrow makespan$
```

```
21:         $s^{final} \leftarrow s_0$
22:     end if
23: end while
24: if $N(s_0)$ is not empty then
25:     Recreate $N(s_0)$
26:     while $N(s_0)$ is not empty do
27:         Select a solution $s'$ from $N(s_0)$ randomly
28:         $(makespan', s) \leftarrow$ SRA$(s', MR)$
29:         if $makespan' - makespan < 0$ then
30:             $makespan \leftarrow makespan'$
31:             $s_0 \leftarrow s$
32:             Create $N(s_0)$
33:         else
34:             Remove the solution $s$ from $N(s_0)$
35:         end if
36:         if $makespan < makespan^{final}$ then
37:             $makespan^{final} \leftarrow makespan$
38:             $s^{final} \leftarrow s_0$
39:         end if
40:     end while
41: end if
42: return $makespan^{final}, s^{final}$
```

In SA-I, when generating the neighborhood of a solution all operations (1-PartMove, 2-PartSwap, 1-RouteChange and 3-RouteChange) are used. In SA-II, different than SA-I, these operations are used sequentially, i.e. neighbor solutions are generated by using only one type of operation. As the algorithm switches from one operation to another, temperature ($t$) and iteration number ($i$) values are restarted. We give the steps of the SA-II in Algorithm 5.

**Algorithm 5** SA-II

```
1: Output: Makespan, $s^{final}$
2: Create an initial solution $s_0 = (\sigma_K, \sigma_R)$ randomly, $s^{final} \leftarrow s_0$
3: Create $N(s_0)$, $i \leftarrow 1$, $t \leftarrow t_0$
```

```
4:  $makespan \leftarrow MCA(\sigma_K, \sigma_R)$, $makespan^{final} \leftarrow makespan$
5:  **for** $j = 1$ to 4 **do**
6:      create $N_j(s_0)$, $i \leftarrow 1$, $t \leftarrow t_0$
7:      **while** $i < NumIteration$ and $N_j(s_0)$ is not empty **do**
8:          Select a solution $s'$ from $N_j(s_0)$ randomly
9:          $(makespan', s) \leftarrow$ SRA$(s', MR)$
10:         **if** $makespan' - makespan < 0$ **then**
11:             $makespan \leftarrow makespan'$
12:             $s_0 \leftarrow s$
13:             Create $N_j(s_0)$, $t \leftarrow t\alpha,\ i \leftarrow i + 1$
14:         **else if** $GenerateRandom() < e^{-\frac{(makespan' - makespan)}{t}}$ **then**
15:             $makespan \leftarrow makespan'$
16:             $s_0 \leftarrow s$
17:             Create $N_j(s_0)$, $t \leftarrow t\alpha,\ i \leftarrow i + 1$
18:         **else**
19:             Remove the solution $s$ from $N_j(s_0)$
20:         **end if**
21:         **if** $makespan < makespan^{final}$ **then**
22:             $makespan^{final} \leftarrow makespan$
23:             $s^{final} \leftarrow s_0$
24:         **end if**
25:     **end while**
26:     **if** $N_j(s_0)$ is not empty **then**
27:         Recreate $N_j(s_0)$
28:         **while** $N_j(s_0)$ is not empty **do**
29:             Select a solution $s'$ from $N_j(s_0)$ randomly
30:             $(makespan', s) \leftarrow$ SRA$(s', MR)$
31:             **if** $makespan' - makespan < 0$ **then**
32:                 $makespan \leftarrow makespan'$
33:                 $s_0 \leftarrow s$
34:                 Create $N_j(s_0)$
35:             **else**
36:                 Remove the solution $s$ from $N_j(s_0)$
```

```
37:         end if
38:         if $makespan < makespan^{final}$ then
39:             $makespan^{final} \leftarrow makespan$
40:             $s^{final} \leftarrow s_0$
41:         end if
42:       end while
43:     end if
44: end for
45: return $makespan^{final}, s_0$
```

In this section, we have presented the C$_{max}$ problem. We define the problem using robot routes and developed a mathematical formulation of the problem. Also, proposed three heuristic approaches: GNS, SA-I, and SA-II. The computational study on these solution methods will be presented Section 4.4. In the next section, we will consider robot speed controllability and robot energy consumption objective along with the makespan objective in the same scheduling environment.

## 4.3 Energy and Makespan Minimization (E&C$_{max}$) Problem

In Section 4.2, we investigated two parallel machines scheduling problems with a single material handling robot to minimize makespan where robot speed is given and fixed. In this section, we consider that the robot speed is controllable, i.e. the robot's speed (move time) during a move between machines/buffers is a decision variable. If the robot moves at a higher speed (i.e. spends less time to move from one location to another), then it consumes more energy. For a feasible part (route) sequence, the makespan objective can be improved by speeding up the robot albeit consuming more energy. Similar to other studies in the literature, for a robot move between two locations we assume a nonlinear relationship between move time (speed) and energy consumption. In Section 3.1.2, we describe the energy consumption function. Makespan minimization and energy minimization are conflicting objectives. Therefore, the problem is to find efficient solutions for makespan and energy consumption objectives.

For this bicriteria scheduling problem we will first give a mathematical formulation in Section 4.3.1. We will use the $\epsilon$-constraint approach to obtain a single objective mathematical model that finds efficient solutions. We will reformulate the mathematical model as a MISOCP (Mixed-integer second-order conic programming). After that, we proposed three heuristic algorithms to find efficient solutions.

In the problem, energy consumption is depending on the robot speed, distance traveled, and robot features as shown in equation 3.6. Therefore, the energy consumption function is the same as in Section 3.1.2.

In the next section, we propose a mathematical formulation of the problem.

### 4.3.1 Mathematical Model for E&C$_{max}$ Problem

There are two objective functions in the problem: Makespan and robot energy consumption. The problem is to find robot route sequence, part sequence, and robot move times such that both the energy consumption of the robot and the makespan of the schedule are minimized. We have additional decision variables $\delta_{i,r,t,m}$ that indicate the time elapsed during the robot move $m$ in route $r$ which loads part $i$ for processing at position $t$ in the part (route) sequence.

$$\min \sum_{i\in K}\sum_{r\in R}\sum_{t\in T}\left(\sum_{m\in M_r^e} c_e \frac{d_m^{k+1}}{\delta_{i,r,t,m}^k} + \sum_{m\in M_r^f} c_f \frac{d_m^{k+1}}{\delta_{i,r,t,m}^k}\right) \tag{4.16}$$

$$\min \sum_{t\in T} c_t \tag{4.17}$$

s.t.constraints (4.2) – (4.8)

$$\begin{aligned} c_t = \sum_{i\in K}\Bigg(&\sum_{r\in R}\sum_{m\in M_r}\delta_{i,r,t,m} + \sum_{r=1}^{2} 2\,\varepsilon\, x_{i,r,t} + \sum_{r=3}^{8} 4\,\varepsilon\, x_{i,r,t} \\ &+ \sum_{r=9}^{12} 6\,\varepsilon\, x_{i,r,t} + \sum_{l\in\{1,2\}}\sum_{r\in FWR^l} P_{l,i}x_{i,r,t}\Bigg) \\ &+ \sum_{l\in\{1,2\}}\sum_{r\in PWR^l} w_{r,t}^l \qquad \forall t\in T \end{aligned} \tag{4.18}$$

$$w^l_{r_2,t} \geq \sum_{j \in K}\left(P_{l,j}\left(\sum_{i \in K}(x_{i,r_2,t} + x_{j,r_1,z} - 1)\right)\right) - \xi^l_{r_1,r_2}$$
$$- \sum_{l \in \{1,2\}}\left(\sum_{m \in AM^l_{r_1}} \delta_{i,r_1,z} + \sum_{m \in BM^l_{r_2}} \delta_{i,r_2,t,m}\right)$$
$$- \sum_{k=z+1}^{t-1} c_k \quad \forall l \in \{1,2\}, \forall (r_1,r_2) \in A_l, \forall t \in T \setminus \{1,n\}, z = 1,2,..,t-1 \tag{4.19}$$

$$w^1_{10,t} \geq \sum_{i \in K} x_{i,10,t}\left(P_{1,i} - 2\varepsilon - \sum_{m \in IM^1_{10}} \delta_{i,10,t}\right) - w^1_{10,t} \quad \forall t \in T \setminus \{1\} \tag{4.20}$$

$$w^2_{12,t} \geq \sum_{i \in K} x_{i,12,t}\left(P_{2,i} - 2\varepsilon - \sum_{m \in IM^2_{12}} \delta_{i,12,t}\right) - w^2_{12,t} \quad \forall t \in T \setminus \{1\} \tag{4.21}$$

$$\delta_{i,r,t,m} \leq \frac{d_m}{v_{min}} x_{i,r,t} \qquad \forall i \in K, r \in R, t \in T, m \in M_r \tag{4.22}$$

$$\delta_{i,r,t,m} \geq \frac{d_m}{v_{max}} x_{i,r,t} \qquad \forall i \in K, r \in R, t \in T, m \in M_r \tag{4.23}$$

$$x_{i,r,t} \in \{0,1\} \qquad \forall i \in K, \forall r \in R, \forall t \in T \tag{4.24}$$

$$w^l_{r,t} \geq 0 \qquad \forall l \in \{1,2\}, \forall r \in PWR^l, \forall t \in T \tag{4.25}$$

$$\delta_{i,r,t,m} \geq 0 \qquad \forall i \in K, \forall r \in R, \forall m \in M_r, \forall t \in T \tag{4.26}$$

The objective function (4.16) is the total energy consumption of the robot moves in a solution. The second objective function (4.17) to minimize is the makespan of the schedule. The assignment and sequencing constraints (4.2) - (4.8) are borrowed from the makespan minimization model given in Section 4.2.1. The time required to complete a robot route in a position is the sum of the robot moves times, loading-unloading times, full and partial waiting times in the route. This time is calculated by constraint (4.18). Constraints (4.19)-(4.21) give the waiting time of the robot before unloading a part from a machine. Constraints (4.22) and (4.23) ensure that the robot move time is between its lower and upper bounds. Constraints (4.24),(4.25) and (4.26) define the domains of the decision variables.

Decreasing the move time of the robot may decrease the makespan of the schedule but increases energy consumption. Therefore, makespan and energy consumption are conflicting objectives. The problem is to find efficient solutions for the problem.

We use the $\epsilon$-constraint approach and find efficient solutions by solving energy consumption minimization problem subject to an upper bound ($\bar{C}_{max}$) on makespan. The resulting single objective problem is given below:

$$\min \sum_{i\in K}\sum_{r\in R}\sum_{t\in T}\left(\sum_{m\in M_r^e} c_e \frac{d_m^{k+1}}{\delta_{i,r,t,m}^k} + \sum_{m\in M_r^f} c_f \frac{d_m^{k+1}}{\delta_{i,r,t,m}^k}\right) + \epsilon\left(\sum_{t\in T} c_t\right)$$

$$\text{s.t.} \sum_{t\in T} c_t \le \bar{C}_{max} \tag{4.27}$$

constraints (4.2) – (4.8) and (4.18) – (4.26)

The model above is a mixed-integer nonlinear programming problem. The energy consumption function in the objective includes nonlinear, convex terms. In the next section, we give a second-order conic representation of the model, so that the problem can be solved using branch-and-bound software that can solve conic quadratic subproblems.

### 4.3.2 Second-Order Conic Programming Reformulation

In Section 3.2.1, we give the SOCP reformulations for $k = 2$ and $k = 1.5$ for the problem of two machine flow shop environments. Because we have a similar objective function in E&C$_{max}$ problem, the same steps are used to reformulate E&C$_{max}$ problem as MISOCP.

**MISOCP Formulation for $k = 2$**

$$\min \sum_{i\in K}\sum_{r\in R}\sum_{t\in T}\left(\sum_{m\in M_r^e} c_e \cdot d_m^3 \cdot \tau_{i,r,t,m} + \sum_{m\in M_r^f} c_f \cdot d_m^3 \cdot \tau_{i,r,t,m}\right) + \epsilon\left(\sum_{t\in T} c_t\right) \tag{4.28}$$

s.t. constraints (4.2) – (4.8) and (4.18) – (4.27)

$$4(\omega_{i,r,t,m})^2 + (\tau_{i,r,t,m} - x_{i,r,t})^2 \le (\tau_{i,r,t,m} + x_{i,r,t})^2$$
$$\forall i \in K, \forall r \in R, \forall m \in M_r, \forall t \in T \tag{4.29}$$

$$4(x_{i,r,t})^2 + (\omega_{i,r,t,m} - \delta_{i,r,t,m})^2 \le (\omega_{i,r,t,m} + \delta_{i,r,t,m})^2$$

$$\forall i \in K, \forall r \in R, \forall m \in M_r, \forall t \in T \tag{4.30}$$

$$\tau_{i,r,t,m}, \omega_{i,r,t,m} \geq 0$$
$$\forall i \in K, \forall r \in R, \forall m \in M_r, \forall t \in T \tag{4.31}$$

**MISOCP Formulation for $k = 1.5$**

$$\min \sum_{i\in K}\sum_{r\in R}\sum_{t\in T}\left(\sum_{m\in M_r^e} c_e \cdot d_m^3 \cdot \tau_{i,r,t,m} + \sum_{m\in M_r^f} c_f \cdot d_m^3 \cdot \tau_{i,r,t,m}\right)$$
$$+ \epsilon\left(\sum_{t\in T} c_t\right) \tag{4.32}$$

s.t. constraints (4.2) – (4.8) and (4.18) – (4.27)

$$4(\omega_{1,i,r,t,m})^2 + (\delta_{i,r,t,m} - 1)^2 \leq (\delta_{i,r,t,m} + 1)^2$$
$$\forall i \in K, \forall r \in R, \forall m \in M_r, \forall t \in T \tag{4.33}$$

$$4(\omega_{2,i,r,t,m})^2 + (\omega_{1,i,r,t,m} - 1)^2 \leq (\omega_{1,i,r,t,m} + 1)^2$$
$$\forall i \in K, \forall r \in R, \forall m \in M_r, \forall t \in T \tag{4.34}$$

$$4(\omega_{3,i,r,t,m})^2 + (\tau_{i,r,t,m} - \delta_{i,r,t,m})^2 \leq (\tau_{i,r,t,m} + \delta_{i,r,t,m})^2$$
$$\forall i \in K, \forall r \in R, \forall m \in M_r, \forall t \in T \tag{4.35}$$

$$4(x_{i,r,t})^2 + (\omega_{2,i,r,t,m} - \omega_{3,i,r,t,m})^2 \leq (\omega_{2,i,r,t,m} - \omega_{3,i,r,t,m})^2$$
$$\forall i \in K, \forall r \in R, \forall m \in M_r, \forall t \in T \tag{4.36}$$

$$\tau_{i,r,t,m}, \omega_{i,r,t,m} \geq 0 \quad \forall i \in K, \forall r \in R, \forall m \in M_r, \forall t \in T \tag{4.37}$$

CPLEX, Gurobi, and other solvers can be used to solve the developed MISOCPs in the previous section. However, as the number of parts increase, solving MISOCP models may require excessively long CPU times. In the next section, we propose heuristic solution methods for the problem.

### 4.3.3 Heuristic Algorithms for E&C$_{max}$ Problem

In Section 4.2.2 we have proposed three heuristic search algorithms GNS, SA-I, and SA-II for the C$_{max}$ problem. In E&C$_{max}$ problem, we have two objective functions to minimize and robot move time decisions to be made in addition to part and robot route sequencing decisions in C$_{max}$ problem. In this section, we will use the same solution neighborhood definition as described in Section 4.2.2 and adapt GNS, SA-I, and SA-II algorithms to E&C$_{max}$ problem.

The aim of the heuristic algorithms for E&C$_{max}$ problem will be finding an efficient solution for the problem. Each algorithm will try to find a solution with minimum energy consumption for a given makespan upper bound. In E&C$_{max}$ problem, we have three groups of decisions: part sequencing, route sequencing, robot move time. In the algorithms, alternative part and route sequences will be explored by using the solution neighborhoods generated by 1-PartMove, 2-PartSwap, 1-RouteChange, and 3-RouteChange operations which were defined before. When a new part and route sequence is achieved by these operations, we have to make the optimal robot move time decisions so that the energy consumption of the schedule is minimized and the makespan of the schedule is not higher than a given upper bound. For making speed optimization decisions we propose an algorithm called Speed Optimization Algorithm (SOA). The details of the algorithm will be discussed later in this section.

In Section 4.2.2, we have given SRA algorithm which repairs infeasible robot route sequences achieved by the neighborhood search operations. For the E&C$_{max}$ problem, we have revised SRA such that it also makes the robot move time decisions by calling SOA.

As discussed in Section 4.2.2, the makespan value of a schedule has to be calculated carefully for which the MCA was used. The MCA was using fixed robot move time values. In the heuristic algorithms developed for E&C$_{max}$ problem, we will use the MCA but the MCA will use the robot move times determined by the SOA. In the next section, we will describe SOA.

**Speed Optimization Algorithm (SOA)**

A solution to E&C$_{max}$ problem can be represented as a sequence of parts, a sequence of robot routes , and move times for all robot moves. A sequence of robot routes can be considered as a sequence of robot activities. Some of those activities are robot moves between machines/buffers. Indeed, these robot moves can be considered as a sequence of moves. Let $m$ denote the position of a move in the sequence of moves. In this section, we call a move in position $m$ as move $m$ briefly. $T_m$ be the time elapsed during the robot move $m$. $T_m^{min}$ is the shortest possible time for move $m$ i.e. when the robot moves at its maximum speed. $T_m^{max}$ is the longest possible time for move $m$ i.e. when the robot moves at its minimum speed. $T_m^{min}$ and $T_m^{max}$ depend on the type of the move. $h_m$ is the status of the robot (empty or full) during the move and $d_m$ is the distance traveled by the robot in move $m$. $Moves$ denotes the set of $m$ in a given robot route sequence. Given a schedule set $Moves$ can be easily found.

$SOA$ determines the robot move times for all robot moves in a given schedule. The inputs of $SOA$ are a part sequence ($\sigma_K$), a route sequence ($\sigma_R$) and an upper bound on makespan ($\bar{C}_{max}$). Given $\sigma_R$, SOA first generates all robot moves in set $Moves$. Then, $T_m$ for each robot move $m$ in $Moves$ is initialized to $T_m^{min}$. Makespan value ($C_{max}$) of the initial schedule is calculated (line 5). In each iteration of $SOA$, the algorithm chooses a robot move $m$ and increases its move time by a constant value ($\Delta$). The robot move is chosen using a greedy approach. When the move time of a robot move is increased, the energy consumption of the move will decrease and possibly the makespan of the schedule will increase. Therefore, we want to choose the robot move which will give the highest energy saving per unit time increase in the makespan objective. However, trying all robot moves one by one and finding the changes in makespan and energy consumption takes too much time. Instead, for each robot move, we calculate the derivative of the energy consumption function (3.7) with respect to time (4.38) at its current move time, and then we choose the robot move which has the smallest derivative value.

$$\frac{\partial EC(m)}{\partial T_m} = -k \cdot c_{h_m} \cdot d_m^{k+1} \cdot T_m^{-k-1} \tag{4.38}$$

After increasing the move time for the selected move (line 14), $SOA$ calculates the makespan of the new schedule. If the makespan does not exceed the given bound, $SOA$ moves to the next iteration. If the makespan exceeds the upper bound (line 17), then the last $T_m$ value is adjusted, so that the makespan bound is not violated. The steps of $SOA$ are given in Algorithm 6.

**Algorithm 6** Speed Optimization Algorithm (SOA)

```
1: Input: $\sigma_K$, $\sigma_R$, $\bar{C}_{max}$
2: Output: $T_m$ values and total energy consumption
```

```
3: Initialize: $Moves \leftarrow$ {all robot moves in the robot route sequence}.
4: Initialize: $T_m \leftarrow T_m^{min}$ for all $m \in Moves$
5: Call MCA to calculate $C_{max}$
6: if $C_{max} > \bar{C}_{max}$ then
7:     return No solution exist
8: else
9:     while $Moves$ is not empty do
10:        Choose $m \in Moves$ which has the minimum derivative value (4.38).
11:        if $T_m = T_m^{max}$ then
12:            remove $m$ from $Moves$ and go to line 9
13:        else
14:            $T_m \leftarrow T_m + \min\{\Delta, T_m^{max} - T_m\}$
15:            Call MCA to calculate $C_{max}$
16:        end if
17:        if $C_{max} > \bar{C}_{max}$ then
18:            $T_m \leftarrow T_m - (\bar{C}_{max} - C_{max})$
19:            remove move $m$ from $Moves$
20:        end if
21:    end while
22:    return $T_m$ for all moves and total energy consumption of the solution.
23: end if
```

**Sequence Repairing Algorithm (SRA$^{\mathbf{E}}$)**

As mentioned in Section 4.2.2, the robot route sequence generated by neighborhood operators can be infeasible. SRA was developed to repair such route sequences for $C_{max}$ problem. A similar route sequence repair algorithm is required within the heuristic algorithms developed for E&C$_{max}$ problem. So, we develop SRA$^E$ and give its detail in algorithm 7. Like in the SRA, we first determine the routes that will be changed ($CR$). By changing these routes, we create a set $PR$ consisting of feasible route sequences. If $PR$ is empty, that means the given solution is feasible. In this case, we call SOA to make move time decisions for a given makespan ($\bar{C}_{max}$) and get the total energy consumption in the line 15. In SRA$^E$, different from SRA, we try to find a feasible route sequence that gives minimum energy consumption for a given makespan ($\bar{C}_{max}$). Therefore, in SRA$^E$, we call SOA for all feasible route sequences in $PR$ and keep the route sequence with minimum energy consumption between lines 27-33. SRA$^E$ returns the solution kept and the value of energy consumption for this solution.

**Algorithm 7** Sequence Repairing Algorithm (SRA$^{\text{E}}$)

```
1: Input: A solution $s = (\sigma_K, \sigma_R)$, set of fixed routes ($MR$)
2: Output: Repaired solution, energy consumption of the repaired solution
3: $CR \leftarrow \emptyset, energy^{final} \leftarrow \infty, n \leftarrow Numberofpart$
4: for $i = 2$ to $n$ do
5:     if $\sigma_K(i-1)$ cannot succeed $\sigma_K(i)$ and $\sigma_K(i) \notin MR$ then
6:         add $i$ to $CR$
7:     end if
8: end for
9: for $j \in MR$ do
10:     if $\sigma_K(j)$ cannot succeed $\sigma_K(j+1)$ then
11:         add $j+1$ to $CR$
12:     end if
13: end for
14: if $CR = \emptyset$ then
15:     $energy^{final} \leftarrow$ SOA($\sigma_K, \sigma_R, \bar{C}_{max}$)
16:     $s = (\sigma_K, \sigma_R)$
17:     return $energy^{final}$, $s$
18: else
19:     $PR \leftarrow \emptyset$, add $\sigma_R$ to $PR$
20:     for $j \in CR$ do
21:         for $\sigma'_R \in PR$ do
22:             Create two (one) route sequences $\sigma^1_R$, $\sigma^2_R$ ($\sigma^1_R$) by changing $\sigma'_R(j)$
23:             Add $\sigma^1_R$, $\sigma^2_R$ ($\sigma^1_R$) to $PR$
24:             Remove $\sigma'_R$ from $PR$
25:         end for
26:     end for
27:     for $\sigma^p_R \in PR$ do
```

```
28:         $energy' \leftarrow SOA(\sigma_K, \sigma'_R, \bar{C}_{max})$
29:         **if** $energy^{final} > energy'$ **then**
30:             $energy^{final} \leftarrow energy'$
31:             $\sigma_R^{final} \leftarrow \sigma_R^{p}$
32:         **end if**
33:     **end for**
34: **end if**
35: $s = (\sigma_K, \sigma_R^{final})$
36: **return** $energy^{final}$, $s$
```

#### 4.3.3.1 Greedy Neighborhood Search Algorithm (GNS$^{\text{E}}$)

GNS$^{\text{E}}$ uses similar logic as GNS. Its steps are given in Algorithm 8. We start with an initial solution. The makespan of the initial solution when robot speeds are at the maximum level should be less than or equal to the given makespan ($\bar{C}_{max}$). In line 3, we call SOA to determine move times for $\bar{C}_{max}$ and get the energy consumption of the initial solution. Then, GNS$^{\text{E}}$ searches the neighborhood of the current solution and if finds a better solution, then that solution becomes the current solution. The selected neighbor solutions are repaired by calling SRA$^{E}$ in line 7. In the next line, we check whether the repaired solution is better than the current solution or not. If better, we replace the current solution with it between lines 9-11. Else, we remove it from the neighborhood and continue with another randomly selected solution. The algorithm terminates when no better solution is found in the neighborhood of the current solution. GNS$^{\text{E}}$ aims to find a solution with minimum energy consumption and with a makespan value not exceeding a given bound. So, when evaluating a neighbor solution, GNS$^{\text{E}}$ uses SOA to optimize robot move times when calling SRA$^{E}$.

**Algorithm 8** The Greedy Neighborhood Search Algorithm GNS$^{\text{E}}$

```
1: Input: An initial solution ($s_0$)
2: Output: The solution found, energy consumption of the solution found
3: $energy^{final} \leftarrow SOA(\sigma_K, \sigma_R, \bar{C}_{max})$
4: Create the neighborhood of $s_0$ ($N(s_0)$)
5: **while** $N(s_0)$ is not empty **do**
```

```
 6:     Select a random solution $s'$ from $N(s_0)$
 7:     $(energy', s) \leftarrow$ SRA$^E(s', MR)$
 8:     **if** $energy^{final} > energy'$ **then**
 9:         $energy^{final} \leftarrow energy'$
10:         $s_0 \leftarrow s$
11:         Create $N(s_0)$
12:     **else**
13:         Remove the solution $s$ from $N(s_0)$
14:     **end if**
15: **end while**
16: **return** $energy^{final}, s_0$
```

**Simulated Annealing Based Algorithms**

In this section, we propose two SA algorithms with combined and sequential neighborhoods, as we did in C$_{max}$ problem. The first SA (SA-I$^E$) algorithm uses the combined neighborhood. SA parameters (initial temperature, number of iterations, the value of $\alpha$) and an initial solution are inputs of SA-I$^E$. Like in GNS$^E$, when robot speeds are at the maximum level in all moves, the makespan of that initial solution should be less than or equal to the given makespan ($\bar{C}_{max}$). The logic behind SA-I$^E$ is the same as SA-I. While we consider the makespan minimization objective in SA-I, the objective of SA-I$^E$ is the minimization of energy consumption. So, we use SOA to make robot speed decisions in the moves while calling SRA$^E$. The steps of SA-I$^E$ are given in algorithm 9. Between the line 4 and line 22, we search the solutions allowing acceptance of a worse solution than the current solution. After that, we reach a local minimum solution.

**Algorithm 9** SA-I$^E$

```
1: Input: An initial solution $(s_0)$
2: Output: Energy consumption value, best found solution
3: Create $N(s_0)$, $i \leftarrow 1$, $t \leftarrow t_0$, $energy^{final} \leftarrow SOA(\sigma_K, \sigma_R, \bar{C}_{max})$
4: **while** $i < NumIteration$ and $N(s_0)$ is not empty **do**
5:     Select a solution $s'$ from $N(s_0)$ randomly
6:     $(energy', s) \leftarrow$ SRA$^E(s', MR)$
7:     **if** $energy' - energy < 0$ **then**
```

```
8:        $energy \leftarrow energy'$
9:        $s_0 \leftarrow s$
10:       Create $N(s_0), t \leftarrow t\alpha,\ i \leftarrow i+1$
11:     else if $GenerateRandom() < e^{-\frac{\delta}{t}}$ then
12:       $energy \leftarrow energy'$
13:       $s_0 \leftarrow s$
14:       Create $N(s_0), t \leftarrow t\alpha,\ i \leftarrow i+1$
15:     else
16:       Remove the solution $s$ from $N(s_0)$
17:     end if
18:     if $energy < energy^{final}$ then
19:       $energy^{final} \leftarrow energy$
20:       $s^{final} \leftarrow s_0$
21:     end if
22:   end while
23:   if $N(s_0)$ is not empty then
24:     Recreate $N(s_0)$
25:     while $N(s_0)$ is not empty do
26:       Select a solution $s'$ from $N(s_0)$ randomly
27:       $(energy', s) \leftarrow$ SRA$^E(s', MR)$
28:       if $energy' - energy < 0$ then
29:         $energy \leftarrow energy'$
30:         $s_0 \leftarrow s$
31:         Create $N(s_0)$
32:       else
33:         Remove the solution $s$ from $N(s_0)$
34:       end if
35:       if $energy < energy^{final}$ then
36:         $energy^{final} \leftarrow energy$
37:         $s^{final} \leftarrow s_0$
38:       end if
39:     end while
40:   end if
```

```
41: return $energy^{final}, s_0$
```

The second SA (SAII$^E$) algorithm sequentially uses four different types of neighborhood structures. We begin with an initial solution that has greater makespan than the given makespan ($\bar{C}_{max}$) when speeds of the robot are at the maximum in all moves. In each neighborhood type, we search the neighborhood accepting worse solutions with a probability that reduces as the iteration number increases. When the limit on the number of iterations is reached, we go to the local minimum solution at the current neighborhood type. Algorithm 10 has the steps of the SA algorithm with sequential neighborhood structure for the energy minimization problem.

**Algorithm 10** SA-II$^E$

```
1: Input: An initial solution ($s_0$)
2: Output: Energy consumption value, best found solution
3: $energy^{final} \leftarrow SOA(\sigma_K, \sigma_R, \bar{C}_{max})$
4: for $j = 1$ to 4 do
5:     create $N_j(s_0)$, $i \leftarrow 1$, $t \leftarrow t_0$
6:     while $i < NumIteration$ and $N_j(s_0)$ is not empty do
7:         Select a solution $s$ from $N_j(s_0)$ randomly
8:         $(energy', s) \leftarrow$ SRA$^E(s', MR)$
9:         if $energy' - energy < 0$ then
10:            $energy \leftarrow energy'$
11:            $s_0 \leftarrow s$
12:            Create $N_j(s_0)$, $t \leftarrow t\alpha,\ i \leftarrow i + 1$
13:        else if $GenerateRandom() < e^{-\frac{\delta}{t}}$ then
14:            $energy \leftarrow energy'$
15:            $s_0 \leftarrow s$
16:            Create $N_j(s_0)$, $t \leftarrow t\alpha,\ i \leftarrow i + 1$
17:        else
18:            Remove the solution $s$ from $N_j(s_0)$
19:        end if
20:        if $energy < energy^{final}$ then
21:            $energy^{final} \leftarrow energy$
22:            $s^{final} \leftarrow s_0$
```

```
23:         end if
24:     end while
25:     if $N_j(s_0)$ is not empty then
26:         Recreate $N_j(s_0)$
27:         while $N_j(s_0)$ is not empty do
28:             Select a solution $s$ from $N_j(s_0)$ randomly
29:             $(energy', s) \leftarrow$ SRA$^E(s', MR)$
30:             if $energy' - energy < 0$ then
31:                 $energy \leftarrow energy'$
32:                 $s_0 \leftarrow s$
33:                 Create $N_j(s_0)$
34:             else
35:                 Remove solution $s$ from $N_j(s_0)$
36:             end if
37:             if $energy < energy^{final}$ then
38:                 $energy^{final} \leftarrow energy$
39:                 $s^{final} \leftarrow s_0$
40:             end if
41:         end while
42:     end if
43: end for
44: return $energy^{final}, s_0$
```

## 4.4 Computational Results

In this section, we present computational results for C$_{max}$ and E&C$_{max}$ problems. We test the performance of mathematical models and the algorithms proposed in this chapter. We give a numerical study on the efficient frontier for energy consumption and cycle time objectives. We test the computational performance of mathematical models and algorithms under different parameter settings. Finally, we show the benefit of the robot speed control for the parallel machine robotic cell environment.

### 4.4.1 Experimental Settings

In C$_{max}$ and E&C$_{max}$ problems, we consider a robotic cell that has two machines located between input and output buffers as shown in Figure 4.1. Each part is first taken from the input buffer, then processed on one of the machines, and then delivered to the output buffer. The processing time of a part may be different on different machines. The speed of the robot is controllable and has a lower and an upper bound. In addition, we have the constants $c_f - c_e$ and $k$ which define the energy consumption function. In Table 4.3, we give the experimental settings used to generate the instances.

Table 4.3: Experimental Settings

| | |
|---|---|
| Number of Parts | 10, 30, 50 |
| Distance Scenarios: | Short, Long, Mixed |
| Processing Times: | Equal LV, Equal HV, $P_1 < P_2$ - LV, $P_1 < P_2$ - HV |
| $c_f - c_e$: | 2-2, 2.5-2 |
| $v^{max}$: | 1.5, 2.0 |
| $k$: | 2.0, 1.5 |

We generate instances for the number of parts 10, 30, and 50. We created mathematical models and heuristic algorithms to solve C$_{max}$ and E&C$_{max}$ problems. When the number of parts increases, the mathematical model cannot be solved in reasonable CPU times. For this reason, we compare the performance of heuristic algorithms with the mathematical models on the instances with 10 parts . We run algorithms to solve the instances with 30 and 50 parts.

In the cell, there are four locations (machines/buffers) that the robot travels between and hence the robot travels on six line segments two of them are between the input buffer and machines, two of them are the output buffer and machines, one of them is between machines, and one of them is between buffers. Three distance scenarios are generated. In scenario Short, the distance between machines is 5 m., the distance between buffers is 15 m., other distances are 10 m. In scenario Long, all distances in scenario Short are multiplied by two. In scenario Mixed, the distance between buffers is 15 m and other distances are 10 m.

We assume that the parts processed are non-identical so their processing times are different from each other. Also a part's processing time can be different on different machines. We consider four scenarios for the processing times of the parts. In the first two scenarios (Equal LV and Equal HV), the processing time of a part on machine 1 is equal to the processing time of that part on machine 2. In Equal HV scenario, processing times are generated from a larger interval compared to Equal LV scenario. In the last two scenarios, we assume that the processing time of a part on machine 2 is longer than the processing time of that part on machine 1. Processing times are generated for scenario Equal LV using uniform distribution between 80 seconds and 100 seconds (U(80; 100)). In scenario Equal HV, processing times are generated using uniform distribution between 40 seconds and 140 seconds (U(40; 140)). In scenario $P_2 > P_1$ - LV ($P2 > P1$ - HV), processing times are generated using uniform distribution between 80 seconds and 100 seconds (20 seconds and 100 seconds) for machine 1, between 100 seconds and 120 seconds (100 seconds and 180 seconds) for machine 2.

The parameters related to the energy consumption function are $c_f - c_e$, $v^{max}$, and $k$. We consider two parameter settings for $c_f - c_e$. In the first setting, we assume the energy consumption of the robot is the same for robot status of full and empty ($c_f = c_e = 2$). In the second setting, the energy consumption is higher when the robot is full ($c_f = 2.5, c_e = 2, c_f > c_e$). We assume the minimum speed of the robot is 0.5 m/s. We consider two scenarios for maximum speeds which are 1.5 m/s (low) and 2 m/s (high). We use two values for the exponent $k$ which are 2 and $1.5$. In all instances, we assume the time of loading/unloading operations ($\varepsilon$) is equal to one second.

We have a base parameter setting with scenario Short, Equal LV, $c_f = c_e = 2$, $v^{max} = 1.5$, $k = 2$. At each time, we change a parameter value while the others are the same as in the base setting. So, we generate 9 parameter settings in total. For each parameter setting, we randomly generate five problem instances. For each problem instance, we will find efficient solutions for ten different levels of $C_{max}$.

We use IBM ILOG CPLEX 12.10 as the solver to solve mathematical models. We perform the test with a Java compiler on Intel Xeon(R) E-2246G (3.6 GHz and 16 GB RAM). The time limit of the solver is set to 5000 seconds. We get the best solution and gap from the solver if it does not reach optimality in the time limit. In addition, algorithms GNS, SA-I, and SA-II are used to solve C$_{max}$ problem, GNS$^E$, SA-I$^E$, and SA-II$^E$ are used to solve E&C$_{max}$ problem.

SA-I, SA-II, SA-I$^E$, and SA-II$^E$ are simulated annealing based algorithms. They have parameters that are initial temperature ($t_0$), cooling schedule ($\alpha$), number of iterations ($NumIteration$). We set $t_0 = 1000$, $\alpha = 0.99$ and $NumIteration = 2000$. In addition, we call SOA in heuristic algorithms to determine robot speeds in the moves. In this algorithm, we have parameter $\Delta$ that is related to the increment of the time of moves. We set the value of $\Delta$ as two seconds.

First, we solve C$_{max}$ problem to determine the minimum makespan. Then, giving a makespan upper bound ($\sum_{t \in T} c_t$), we solve E&C$_{max}$ problem. We solve the instances for ten different makespan upper bounds, named $\bar{C}^j_{max}$ where $1 \leq j \leq 10$. $\bar{C}^1_{max}$ is the minimum makespan value found by the algorithms in C$_{max}$ problem. We found $\bar{C}^{10}_{max}$ by adjusting the robot speed as the minimum speed ($v^{min}$) in all moves by using route 3 (or route 4) for all parts in the schedule. Other makespan levels ($\bar{C}^2_{max}$-$\bar{C}^9_{max}$) are equally separated between $\bar{C}^1_{max}$ and $\bar{C}^{10}_{max}$.

We solve E&C$_{max}$ problem for makespan upper bounds in ascending order ($\bar{C}^1_{max}$, $\bar{C}^2_{max}$,...,$\bar{C}^{10}_{max}$). For makespan upper bound $\bar{C}^1_{max}$, GNS$^E$ (SA-I$^E$, SA-II$^E$) takes the output solution of GNS (SA-I, SA-II) as the initial solution. For makespan upper bound $\bar{C}^j_{max}$, GNS$^E$ (SA-I$^E$, SA-II$^E$) takes the output solution of GNS$^E$ (SA-I$^E$, SA-II$^E$) solved for makespan upper bound $\bar{C}^{j-1}_{max}$ as the initial solution where $2 \leq j \leq 10$.

### 4.4.2 Results for the C$_{max}$ Problem

We first compare IBM CPLEX with the heuristic algorithms for the instances with 10 parts. Then, for 30 and 50 parts, we use heuristic algorithms to solve C$_{max}$ problem. We compare heuristic algorithms with each other.

In Table 4.4, we summarize the results for the instances with 10 parts. For all instances, IBM CPLEX finds optimal solutions. GNS finds optimal solutions in 96% of the instances. SA-I and SA-II find optimal solutions in 87% and 89% of the instances. For all heuristics, the average optimality gap is less than 0.1%. While IBM CPLEX spends 123.5 seconds to find the optimal solution, heuristic algorithms terminate in 1.3 seconds or less on average.

Table 4.4: Results of the instances with 10 parts for C$_{max}$ Problem

| | **IBM CPLEX** | **GNS** | **SA-I** | **SA-II** |
|---|---:|---:|---:|---:|
| **Optimal** | 100% | 96% | 87% | 89% |
| **Avg. Gap** | 0.000% | 0.019% | 0.075% | 0.036% |
| **Avg. Comp. Time (sec)** | 123.5 | 0.3 | 0.3 | 1.3 |

For the instances with 30 and 50 parts, we run heuristic algorithms. In Table 4.5, we summarize the results. We give % of times an algorithm finds the best solution (Best) among the three alternative algorithms, the average gap from the best objective value achieved (Avg. Deviation) and average CPU times (Avg. Comp. Time) required to converge to a solution. CPU times increase as the number of parts increases. In overall, SA-II requires shortest CPU time. SA-I performs better in finding best solutions overall. However, Avg. Deviation levels show that all three heuristics find very close $C_{max}$ values.

In the next section, we will give the results on E&C$_{max}$ problem.

### 4.4.3 Energy Consumption vs Makespan

In this section, we first present a set of efficient solutions for a selected problem instance with the base parameter setting and with 10 parts. Figure 4.6 gives the efficient points which represent the behavior of the efficient frontier. The amount of energy consumed by the robot falls as the makespan level increases. The robot intends to move slower to decrease energy consumption if we have more time for completing the parts. The robot intends to move faster to achieve a shorter makespan. This increases energy consumption. Due to the convex energy consumption function, energy

Table 4.5: Results of the instances with 30 and 50 parts for C$_{max}$ Problem

| | | **Number of Parts** | | |
|---|---|---|---|---|
| | | **30** | **50** | **Overall** |
| **GNS** | Best | 64.4% | 53.3% | 58.9% |
| | Avg. Deviation | 0.05% | 0.07% | 0.06% |
| | Avg. Comp. Time (sec) | 5.8 | 57.8 | 31.8 |
| **SA-I** | Best | 62.2% | 62.2% | 62.2% |
| | Avg. Deviation | 0.06% | 0.05% | 0.06% |
| | Avg. Comp. Time (sec) | 6.3 | 59.8 | 33.1 |
| **SA-II** | Best | 51.1% | 31.1% | 41.1% |
| | Avg. Deviation | 0.12% | 0.23% | 0.17% |
| | Avg. Comp. Time (sec) | 5.1 | 29.7 | 17.4 |

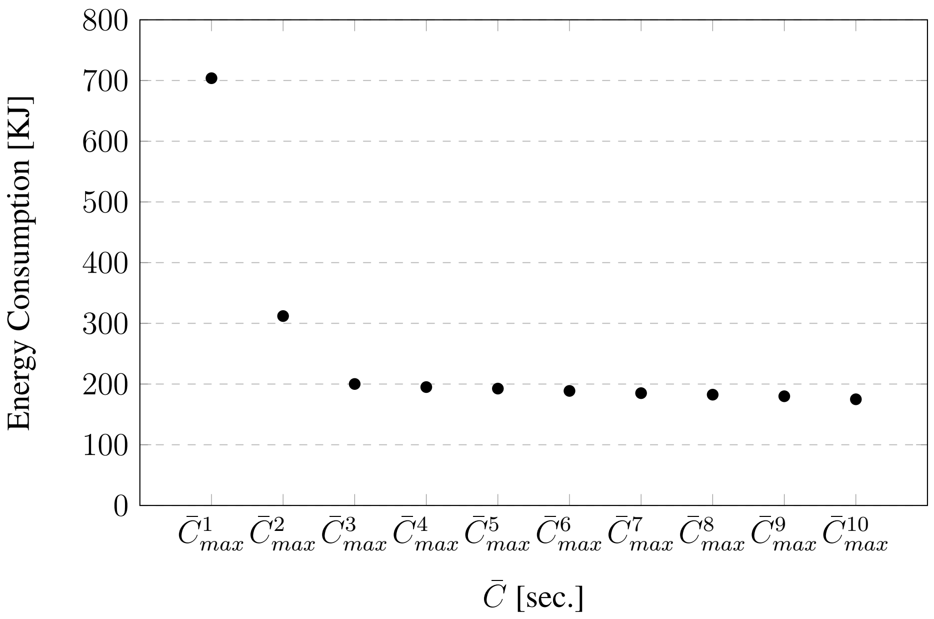
consumption increases sharply as the makespan decreases.

The trade-off between energy consumption and cycle time can be used to save energy. When a high rate of output is needed, for example, the cell would work in lower makespan times, although it would consume more power. However, when we have a flexible due date for the completion of parts, energy consumption could be more important. Then, the robotic cell can work at higher makespans and reduces energy consumption. Therefore, energy saving can be achieved according to the time available at hand.

### 4.4.4 Results of E&C$_{max}$ Problem

In this section, we first compare the performance of the mathematical model and the algorithms for the instances with 10 parts. Then, for 30 and 50 parts, we present the effects of the parameters on the performance of the algorithms.

Figure 4.6: A set of efficient solutions



#### 4.4.4.1 Comparison of Heuristic Algorithms with the Mathematical Model

We solve the mathematical model for the instances with 10 parts and the results are given in Table 4.6. For ten different values of makespan upper bounds ($\bar{C}^1_{max} - \bar{C}^{10}_{max}$), we give the percentage of instances for which IBM CPLEX achieved less than $1\%$ optimality gap, percentage of instances with an optimality gap between $1\%$ and $5\%$, and the percentage of instances no solution is found within the given time limit. We also give average CPU times. Except when we look for a solution at minimum makespan upper bound ($\bar{C}^1_{max}$), in almost all cases IBM CPLEX could find a feasible solution for the problem within the given time limit. At $\bar{C}^1_{max}$, which is the minimum $C_{max}$ case, finding a feasible solution itself becomes a difficult problem and the performance of IBM CPLEX deteriorates. We observe that as $C_{max}$ increases, the solution quality achieved by IBM CPLEX gets better. In general, in more than $70\%$ of the cases IBM CPLEX could find a solution within $5\%$ optimality gap.

IBM CPLEX found optimal solution in $56.9\%$ of the instances with 10 parts. For these instances, the average CPU time of IBM CPLEX is 3163.2 seconds. In Table 4.7, we give a comparison of the performances of the heuristic algorithms with the optimal solutions found by IBM CPLEX. We can conclude that heuristic algorithms

Table 4.6: The Mathematical Model Results of the instances with 10 parts for E&C$_{max}$ Problem

| | $\bar{C}^1_{max}$ | $\bar{C}^2_{max}$ | $\bar{C}^3_{max}$ | $\bar{C}^4_{max}$ | $\bar{C}^5_{max}$ | $\bar{C}^6_{max}$ | $\bar{C}^7_{max}$ | $\bar{C}^8_{max}$ | $\bar{C}^9_{max}$ | $\bar{C}^{10}_{max}$ | Overall |
|---|---|---|---|---|---|---|---|---|---|---|---|
| gap < 1% | 4.4% | 11.1% | 35.6% | 62.2% | 73.3% | 75.6% | 75.6% | 80.0% | 66.7% | 86.7% | 57.1% |
| 1% $\leq$ gap $\leq$ 5% | 6.7% | 0.0% | 33.3% | 22.2% | 15.6% | 11.1% | 13.3% | 8.9% | 22.2% | 0.0% | 13.3% |
| No Solution Found | 48.9% | 4.4% | 6.7% | 2.2% | 2.2% | 2.2% | 0.0% | 0.0% | 0.0% | 4.4% | 7.1% |
| Avg. Comp. Time | 4962.3 | 4918.9 | 4261.5 | 4361.1 | 4340.5 | 4107.3 | 4129.4 | 4158.4 | 3571.2 | 704.3 | 3951.5 |

converge to a solution within few seconds and they find energy consumption values within 1% of the optimal solution. Among the algorithms, SA-II$^E$ is better to find optimal solutions and it has the minimum average gap.

Table 4.7: Comparison of heuristic algorithms with optimal solutions

| | **GNS$^E$** | **SA-I$^E$** | **SA-II$^E$** |
|---|---|---|---|
| **Optimal** | 64.45% | 66.80% | 71.09% |
| **Avg. Gap** | 1.09% | 1.03% | 0.98% |
| **Avg. Comp. Time (sec)** | 3.0 | 3.9 | 5.2 |

#### 4.4.4.2 Effects of problem parameters on the performance of heuristic algorithms

We run heuristic algorithms on the instances with 30 and 50 parts. The effect of parameters on the performances of the algorithms is analyzed. For each algorithm, we give the percentage of instances the best solution is found (Best), average deviation from the best solution found (Avg. Deviation), and average computation time.

Table 4.8 compares the performance of heuristic algorithms for different distance scenarios. In all distance scenarios, SA-II$^E$ performs better in terms of finding best solution and deviation from the best solution. When all distances are long (scenario Long), deviation from the best solution found is lower for all algorithms. However, deviation from the best solution found is higher and algorithms' percentages of best solutions found are close to each other in scenario Mixed. Also, there is an increase in computational times in scenario Mixed. We should also note that, Avg. Deviation values are less than 0.3% for all algorithms and all scenarios. This means that the energy consumption levels achieved by the algorithms are close to each other.

Table 4.9 summarizes the effects of processing time scenarios on the performance of heuristic algorithms. SA-II$^E$ outperforms the other algorithms in finding the best solution for the problem. The maximum Avg. Deviation is 0.53%, so the algorithms achieve close energy consumption levels. We observe that deviation from best solu-

Table 4.8: Effects of Distance Scenarios

| **Distance Scenarios** | | **GNS$^E$** | **SA-I$^E$** | **SA-II$^E$** |
|---|---|---|---|---|
| **Short** | Best | 60.0% | 70.0% | 82.7% |
| | Avg. Deviation | 0.21% | 0.12% | 0.08% |
| | Avg. Comp. Time (sec) | 251.9 | 247.9 | 238.6 |
| **Long** | Best | 52.7% | 75.3% | 84.7% |
| | Avg. Deviation | 0.06% | 0.01% | 0.00% |
| | Avg. Comp. Time (sec) | 296.7 | 266.6 | 265.6 |
| **Mixed** | Best | 63.3% | 68.0% | 73.3% |
| | Avg. Deviation | 0.27% | 0.18% | 0.16% |
| | Avg. Comp. Time (sec) | 340.7 | 282.5 | 301.4 |

tions increases when the variance between the processing times is high and processing times of the parts on machines are different. With high variance processing times, the computational times and deviations significantly increase for all algorithms. In addition, different processing times on machines raise computational times and deviation from the best solutions found.

Table 4.10 gives results for different levels of parameters related to robot specifications, i.e. $c_f - c_e$ , $v^{max}$ and $k$. Similar to previous analysis, SA-II$^E$ performs better than the other algorithms for all scenarios.

All algorithms converged to a solution in 500 CPU seconds at most. They find solutions which have close energy consumption levels, but SA-II$^E$ finds the best solution in more cases than the others. SA-II$^E$ finds the best solution in 70% of the cases in the worst case.

Table 4.9: Effects of Processing Times

| **Processing Time** | | **GNS$^E$** | **SA-I$^E$** | **SA-II$^E$** |
|---|---|---|---|---|
| **Equal - LV** | Best | 60.0% | 70.0% | 82.7% |
| | Avg. Deviation | 0.21% | 0.12% | 0.08% |
| | Avg. Comp. Time (sec) | 251.9 | 247.9 | 238.6 |
| **Equal - HV** | Best | 48.0% | 67.3% | 70.7% |
| | Avg. Deviation | 0.52% | 0.27% | 0.18% |
| | Avg. Comp. Time (sec) | 493.5 | 321.7 | 379.6 |
| $P_1 < P_2$ **- LV** | Best | 57.3% | 66.7% | 75.3% |
| | Avg. Deviation | 0.30% | 0.24% | 0.25% |
| | Avg. Comp. Time (sec) | 357.0 | 324.4 | 303.1 |
| $P_1 < P_2$ **- HV** | Best | 54.0% | 72.7% | 72.7% |
| | Avg. Deviation | 0.53% | 0.31% | 0.35% |
| | Avg. Comp. Time (sec) | 267.2 | 234.7 | 277.4 |

Table 4.10: Effects of Robot Parameters

| **Robot Parameters** | | **GNS$^E$** | **SA-I$^E$** | **SA-II$^E$** |
|---|---|---|---|---|
| **Base** | Best | 60.0% | 70.0% | 82.7% |
| | Avg. Deviation | 0.21% | 0.12% | 0.08% |
| | Avg. Comp. Time (sec) | 251.9 | 247.9 | 238.6 |
| $c_f > c_e$ | Best | 64.7% | 75.3% | 81.3% |
| | Avg. Deviation | 0.17% | 0.09% | 0.06% |
| | Avg. Comp. Time (sec) | 266.6 | 269.8 | 229.5 |
| **high** $v^{max}$ | Best | 60.0% | 66.7% | 76.7% |
| | Avg. Deviation | 0.34% | 0.18% | 0.18% |
| | Avg. Comp. Time (sec) | 276.2 | 277.6 | 200.2 |
| $k = 1.5$ | Best | 30.0% | 48.0% | 76.0% |
| | Avg. Deviation | 0.17% | 0.09% | 0.07% |
| | Avg. Comp. Time (sec) | 264.4 | 251.3 | 286.8 |

### 4.4.5 The Benefit of Robot Speed Control

In general, robots are operated at the highest speed in practice to obtain the maximum throughput at the lowest makespan. By controlling the speeds of the robot, we can reduce energy consumption whıle achieving a target makespan. We analyze how much energy saving is possible by the robot speed control in this section.

As discussed the performance of SA-II$^E$ is the best among the three heuristic algorithms, therefore we use the solutions of SA-II$^E$ to calculate energy saving. SA-II$^E$ uses the solution of SA-II as input to solve E&C$_{max}$ problem. Using SA-II, we first find the minimum makespan assuming the robot moves at its maximum speed in all moves. Then, for the minimum makespan found by SA-II, SA-II$^E$ finds a solution with minimum energy consumption. We calculate the energy saving rate by dividing the difference between the energy consumption of the solution SA-II and energy consumption found in SA-II$^E$ by the energy consumption of the solution SA-II. In Table 4.11, we give the average energy saving rates in percentage with respect to different parameter settings. We have 10.0% energy saving overall.

We see the energy saving rate falls when all distances are long. However, we can save more energy in scenario Mixed. Note that the distance between machines in scenario Mixed is longer than in scenario Short. So, this implies the distance between machines is important in energy saving.

The high variance between the processing times of the parts causes less energy saving as can be concluded from Table 4.11. When the processing time on a machine is longer than the processing time on the other machine ($P_1 < P_2$ - LV, $P_1 < P_2$ - HV), higher energy saving is possible.

If the robot consumes more energy when carrying a part i.e., an increase in the value of $c_f$, we have less energy saving rate. The maximum speed of the robot has also an effect on the energy saving rate. With high maximum speed, we can reduce energy consumption by controlling the robot's speeds. Changing the value of $k$ from 2 to 1.5 brings a reduction in energy saving. So, we can conclude that with higher $k$, we can achieve more energy saving where $k > 1$.

Table 4.11: Energy Saving Rates

| | | **Average Energy Saving (%)** |
|---|---|---|
| **Distance Scenario** | Short | 10.4 |
| | Long | 4.5 |
| | Mixed | 16.3 |
| **Processing Times** | Equal - LV | 10.4 |
| | Equal - HV | 7.6 |
| | $P_1 < P_2$ - LV | 13.2 |
| | $P_1 < P_2$ - HV | 9.2 |
| **Robot Parameters** | Base | 10.4 |
| | $c_f > c_e$ | 9.5 |
| | high $v^{max}$ | 10.9 |
| | $k = 1.5$ | 8.8 |
| | **Overall** | 10.0 |

In this section, we generated instances with different numbers of parts, distance scenarios, processing times, and robot parameters for C$_{max}$ and E&C$_{max}$ problems. We constructed two mathematical models and three heuristic algorithms as solution approaches. We use IBM CPLEX to solve mathematical models. Because of the complexities of the problems, IBM CPLEX can solve the instances with 10 parts. We use the optimal solutions from IBM CPLEX to show the performances of the algorithms. We showed the effects of distance scenarios, processing times, and robot parameters on the performance of the algorithms. We gave a set of efficient solutions for E&C$_{max}$ problem. At the end, we showed the benefit of robot speed control by giving energy savings.

# CHAPTER 5

# CONCLUSIONS

In this thesis, we studied two robotic cell scheduling problems with robot energy consumption considerations. We consider cells with two machines, an input buffer and an output buffer. Machines process the parts and the robot performs material handling activities such as loading-unloading operations, moving parts between machines/buffers. We consider two different scheduling environments, a flow-shop manufacturing environment and a parallel machine environment, separately.

For the flow-shop robotic cell, we find a cyclic schedule for a minimum part set assuming that robot speed can be controlled. There are two possible robot cycles $S_1$ and $S_2$, which are certain move sequences that the robot follows. The problem is to find a part sequence, choose a robotic cycle for each position and robot speeds so that the two conflicting objectives cycle time and energy consumption are both minimized. For that, a mathematical model is developed and we apply $\epsilon$-constraint approach to find efficient solutions.

We conduct experimental analysis by generating instances with different parameter settings. We present a set of efficient solutions to see the relation of energy consumption with the cycle time. We observe the energy consumption is decreasing when cycle time increases. Then, we test the performance of the robot under different levels of experimental factors, i.e. distances, processing times, the maximum speed of the robot, and coefficients affecting the energy consumption. We find the possible minimum cycle time using the algorithm proposed by Aneja and Kamoun (1999) assuming the robot speed is at the maximum level in all moves. Then, we give the model that minimum cycle time as upper bound to find the minimum energy. We show the energy saving comparing these two solutions. The effects of factors on energy saving rates are also presented.

For the parallel machine robotic cell, we consider a set of parts to be completed on the given machines. The problem is to find the part sequence, choose a machine for each part, find a robot activity sequence and robot speed at each move so that the makespan and energy consumption objectives are minimized. We first showed that a robot schedule can be represented by a sequence of robot routes. Each route includes a set of robot activities. Defining the robot routes enabled us to develop a mathematical model for the scheduling problem. We presented two mathematical models. The first one minimizes the makespan when robot speed values are given and fixed. The second model, achieved by using $\epsilon$-constraint method, finds an efficient solution by minimizing energy consumption for a given level of makespan. We have also developed neighborhood search algorithms for both the makespan and energy objectives for a given makespan level. For each case, we have three algorithms. One of them uses greedy logic to converge to a local optimal solution. The other two algorithms are based on Simulated Annealing metaheuristic.

Using different parameter settings, we generate instances to perform mathematical models and algorithms. For the makespan minimization problem, we compare the results of the mathematical model with the algorithms using small instances. Then, we compare the algorithms with each other for large instances. We present a set of efficient solutions with respect to makespan upper bounds using the results of the makespan and energy minimization problem. We show that as the cycle time increases, the energy consumption decreases. Then, for the makespan and energy minimization problem, we compare the results of the mathematical model with the algorithms using small instances. Using large instances, we compare the algorithms with each other under different parameter settings. We show the effect of parameters on the performance of algorithms. The energy saving rates are presented using the energy consumption of the solution from the makespan minimization problem and energy consumption of the solution of makespan and energy minimization problem where makespan upper bound is the minimum makespan found in makespan minimization problem. In addition, the effects of parameters on energy saving are analyzed.

Energy efficient robotic scheduling becomes more important with the widespread use of robotic cells. With this study, we show that significant energy saving could be achieved by carefully scheduling parts and robot activities/speeds. In future research, other and maybe more realistic energy consumption functions can be considered such as the ones which model acceleration/deceleration of the robot. Besides, when the robot is not working, i.e., waiting in front of a machine, standby energy consumption can be taken into account for a more realistic energy consumption calculation.

In this thesis, we consider flow-shop and parallel machine manufacturing environments consisting of only two machines. Robotic cell scheduling environments that have more than two machines can be considered as another possible research direction. In addition, the objectives are minimization of energy consumption and the completion time of the last part. Different scheduling objectives like completion time or tardiness can be considered with the minimization of energy consumption.

# REFERENCES

Abdekhodaee, A. H., & Wirth, A. (2002). Scheduling parallel machines with a single server: Some solvable cases and heuristics. *Computers & Operations Research*, *29*(3), 295–315.

Abdekhodaee, A. H., Wirth, A., & Gan, H. S. (2004). Equal processing and equal setup time cases of scheduling parallel machines with a single server. *Computers & Operations Research*, *31*(11), 1867–1889.

Abdekhodaee, A. H., Wirth, A., & Gan, H.-S. (2006). Scheduling two parallel machines with a single server: The general case. *Computers & Operations Research*, *33*(4), 994–1009.

Abdulkader, M., ElBeheiry, M., Afia, N., & El-Kharbotly, A. (2013). Scheduling and sequencing in four machines robotic cell: Application of genetic algorithm and enumeration techniques. *Ain Shams Engineering Journal*, *4*(3), 465–474.

Agnetis, A. (2000). Scheduling no-wait robotic cells with two and three machines. *European Journal of Operational Research*, *123*(2), 303–314.

Alcaide, D., Chu, C., Kats, V., Levner, E., & Sierksma, G. (2007). Cyclic multiple-robot scheduling with time-window constraints using a critical path approach. *European Journal of Operational Research*, *177*(1), 147–162.

Alharkan, I., Saleh, M., Ghaleb, M. A., Kaid, H., Farhan, A., & Almarfadi, A. (2020). Tabu search and particle swarm optimization algorithms for two identical parallel machines scheduling problem with a single server. *Journal of King Saud University - Engineering Sciences*, *32*(5), 330–338.

Alizadeh, F., & Goldfarb, D. (2001). Second-order cone programming. *Mathematical Programming*, *95*, 3–51.

Aneja, Y., & Kamoun, H. (1999). Scheduling of parts and robot activities in a two machine robotic cell. *Computers & Operations Research*, *26*(4), 297–312.

Arnaout, J. P. (2016). Heuristics for the two-machine scheduling problem with a single server. *International Transactions in Operational Research*, *24*(6), 1347–1355.

Batur, G. D., Karasan, O. E., & Akturk, M. S. (2012). Multiple part-type scheduling in flexible robotic cells. *International Journal of Production Economics*, *135*(2), 726–740.

Bektur, G., & Saraç, T. (2019). A mathematical model and heuristic algorithms for an unrelated parallel machine scheduling problem with sequence-dependent setup times, machine eligibility restrictions and a common server. *Computers & Operations Research*, *103*, 46–63.

Benmansour, R., Braun, O., & Artiba, A. (2014). On the single-processor scheduling problem with time restrictions. *2014 International Conference on Control, Decision and Information Technologies (CoDIT)*.

Benmansour, R., Braun, O., & Hanafi, S. (2018). The single-processor scheduling problem with time restrictions: Complexity and related problems. *Journal of Scheduling*, *22*(4), 465–471.

Bish, E. K. (2003). A multiple-crane-constrained scheduling problem in a container terminal. *European Journal of Operational Research*, *144*(1), 83–107.

Braun, O., Chung, F., & Graham, R. (2013). Single-processor scheduling with time restrictions. *Journal of Scheduling*, *17*(4), 399–403.

Brauner, N. (2008). Identical part production in cyclic robotic cells: Concepts, overview and open questions. *Discrete Applied Mathematics*, *156*(13), 2480–2492.

Brauner, N., & Finke, G. (1999). On a conjecture about robotic cells: New simplified proof for the three-machine case. *Infor-Information Systems and Operational Research*, *37*(1), 20–36.

Brauner, N., & Finke, G. (2001). Cycles and permutations in robotic cells. *Mathematical and Computer Modelling*, *34*(5), 565–591.

Brucker, P., Dhaenens-Flipo, C., Knust, S., Kravchenko, S. A., & Werner, F. (2002). Complexity results for parallel machine problems with a single server. *Journal of Scheduling*, *5*(6), 429–457.

Bukata, L., Sucha, P., Hanzalek, Z., & Burget, P. (2017). Energy optimization of robotic cells. *IEEE Transactions on Industrial Informatics*, *13*(1), 92–102.

Bukata, L., Šůcha, P., & Hanzálek, Z. (2019). Optimizing energy consumption of robotic cells by a branch & bound algorithm. *Computers & Operations Research*, *102*, 52–66.

Bunse, K., Vodicka, M., Schönsleben, P., Brülhart, M., & Ernst, F. O. (2011). Integrating energy efficiency performance in production management – gap analysis between industrial needs and scientific literature. *Journal of Cleaner Production*, *19*(6-7), 667–679.

Carlier, J., Haouari, M., Kharbeche, M., & Moukrim, A. (2010). An optimization-based heuristic for the robotic cell problem. *European Journal of Operational Research*, *202*(3), 636–645.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, *45*(1), 41–51.

Cheng, T., & Sin, C. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, *47*(3), 271–292.

Crama, Y., & de Klundert, J. V. (1999). Cyclic scheduling in 3-machine robotic flow shops. *Journal of Scheduling*, *2*(1), 35–54.

Crama, Y., Kats, V., Van de Klundert, J., & Levner, E. (2000). Cyclic scheduling in robotic flowshops. *Annals of operations Research*, *96*(1-4), 97–124.

Crama, Y., & van de Klundert, J. (1997). Cyclic scheduling of identical parts in a robotic cell. *Operations Research*, *45*(6), 952–965.

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

Elidrissi, A., Benmansour, R., Benbrahim, M., & Duvivier, D. (2018). *MIP formulations for identical parallel machine scheduling problem with single server*.

Elidrissi, A., Benmansour, R., Benbrahim, M., & Duvivier, D. (2020). Mathematical formulations for the parallel machine scheduling problem with a single server. *International Journal of Production Research*, 1–19.

Elmi, A., & Topaloglu, S. (2013). A scheduling problem in blocking hybrid flow shop robotic cells with multiple robots. *Computers & Operations Research*, *40*(10), 2543–2555.

Engelmann, J. (2009). Methoden undwerkzeuge zur planung und gestaltung energieeffizienter fabriken. *Wissenschaftliche Schriftenreihe des Institutes für Betriebswissenschaften und Fabriksysteme: Institut für Betriebswissenschaften und Fabriksysteme.*, *19*(6-7), 667–679.

Gan, H.-S., Wirth, A., & Abdekhodaee, A. (2012). A branch-and-price algorithm for the general case of scheduling parallel machines with a single server. *Computers & Operations Research*, *39*(9), 2242–2247.

Gilmore, P. C., & Gomory, R. E. (1964). Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, *12*(5), 655–679.

Glass, C. A., Shafransky, Y. M., & Strusevich, V. A. (2000). Scheduling for parallel dedicated machines with a single server. *Naval Research Logistics*, *47*(4), 304–328.

Graham, R., Lawler, E., Lenstra, J., & Kan, A. (1979). *Optimization and approximation in deterministic sequencing and scheduling: A survey.*

Guirchoun, S., Souhkal, A., & Martineau, P. (2005). *Complexity results for parallel machine scheduling problems with a server in computer systems.*

Gultekin, H., Akturk, M. S., & Karasan, O. E. (2007). Bicriteria robotic cell scheduling. *Journal of Scheduling*, *11*(6), 457–473.

Gultekin, H., Akturk, M. S., & Karasan, O. E. (2010). Bicriteria robotic operation allocation in a flexible manufacturing cell. *Computers & Operations Research*, *37*(4), 779–789.

Gultekin, H., Coban, B., & Akhlaghi, V. E. (2018). Cyclic scheduling of parts and robot moves in m-machine robotic cells. *Computers & Operations Research*, *90*, 161–172.

Gultekin, H., Gürel, S., & Taspinar, R. (2021). Bicriteria scheduling of a material handling robot in an m-machine cell to minimize the energy consumption of the robot and the cycle time. *Robotics and Computer-Integrated Manufacturing*, *72*, 102207.

Gürel, S., Gultekin, H., & Akhlaghi, V. E. (2019). Energy conscious scheduling of a material handling robot in a manufacturing cell. *Robotics and Computer-Integrated Manufacturing*, *58*, 97–108.

Güzel, B. N. (2021). *Energy conscious machine and robot speed decisions in a two machine robotic cell* (Master's thesis). Middle East Technical University, Industrial Engineering Department. Turkey.

Hall, N. G., Kamoun, H., & Sriskandarajah, C. (1997). Scheduling in robotic cells: Classification, two and three machine cells. *Operations Research*, *45*(3), 421–439.

Hall, N. G., Potts, C. N., & Sriskandarajah, C. (2000). Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, *102*(3), 223–243.

Hall, N. G., & Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, *44*(3), 510–525.

Hamzadayi, A., & Yildiz, G. (2016a). Event driven strategy based complete rescheduling approaches for dynamic m identical parallel machines scheduling problem with a common server. *Computers & Industrial Engineering*, *91*, 66–84.

Hamzadayi, A., & Yildiz, G. (2016b). Hybrid strategy based complete rescheduling approaches for dynamic m identical parallel machines scheduling problem with a common server. *Simulation Modelling Practice and Theory*, *63*, 104–132.

Hamzadayi, A., & Yildiz, G. (2017). Modeling and solving static m identical parallel machines scheduling problem with a common server and sequence dependent setup times. *Computers & Industrial Engineering*, *106*, 287–298.

Hasani, K., Kravchenko, S. A., & Werner, F. (2014a). Block models for scheduling jobs on two parallel machines with a single server. *Computers & Operations Research*, *41*, 94–97.

Hasani, K., Kravchenko, S. A., & Werner, F. (2014b). Minimising interference for scheduling two parallel machines with a single server. *International Journal of Production Research*, *52*(24), 7148–7158.

Hasani, K., Kravchenko, S. A., & Werner, F. (2014c). Simulated annealing and genetic algorithms for the two-machine scheduling problem with a single server. *International Journal of Production Research*, *52*(13), 3778–3792.

Hasani, K., Kravchenko, S. A., & Werner, F. (2015). Minimizing the makespan for the two-machine scheduling problem with a single server: Two algorithms for very large instances. *Engineering Optimization*, *48*(1), 173–183.

Huang, S., Cai, L., & Zhang, X. (2010). Parallel dedicated machine scheduling problem with sequence-dependent setups and a single server. *Computers & Industrial Engineering*, *58*(1), 165–174.

Hurink, J., & Knust, S. (2002). A tabu search algorithm for scheduling a single robot in a job-shop environment. *Discrete Applied Mathematics*, *119*(1-2), 181–203.

Idrissi, A. E., Benbrahim, M., Benmansour, R., & Duvivier, D. (2018). Greedy heuristics for identical parallel machine scheduling problem with single server to minimize the makespan (B. A. E. Majd & H. E. Ghazi, Eds.). *MATEC Web of Conferences*, *200*, 00001.

Jiang, Y., Dong, J., & Ji, M. (2013). Preemptive scheduling on two parallel machines with a single server. *Computers & Industrial Engineering*, *66*(2), 514–518.

Jiang, Y., Zhang, Q., Hu, J., Dong, J., & Ji, M. (2014). Single-server parallel-machine scheduling with loading and unloading times. *Journal of Combinatorial Optimization*, *30*(2), 201–213.

Kaabi, J., & Harrath, Y. (2014). A survey of parallel machine scheduling under availability constraints. *International Journal of Computer and Information Technology*, *3*, 238–245.

Kamalabadi, I. N., Gholami, S., & Mirzaei, A. H. (2007). *Considering a cyclic multiple-part type three-machine robotic cell problem.*

Kamoun, H., Hall, N. G., & Sriskandarajah, C. (1999). Scheduling in robotic cells: Heuristics and cell design. *Operations Research*, *47*(6), 821–835.

Khachaturyan, A., Semenovsovskaya, S., & Vainshtein, B. (1981). The thermodynamic approach to the structure analysis of crystals. *Acta Crystallographica Section A*, *37*(5), 742–754.

Kim, M.-Y., & Lee, Y. H. (2012). MIP models and hybrid algorithm for minimizing the makespan of parallel machines scheduling problem with a single server. *Computers & Operations Research*, *39*(11), 2457–2468.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Kobetski, A., & Fabian, M. (2008). Velocity balancing in flexible manufacturing systems. *2008 9th International Workshop on Discrete Event Systems*.

Koulamas, C. P. (1996). Scheduling two parallel semiautomatic machines to minimize machine interference. *Computers & Operations Research*, *23*(10), 945–956.

Kravchenko, S., & Werner, F. (1997). Parallel machine scheduling problems with a single server. *Mathematical and Computer Modelling*, *26*(12), 1–11.

Kravchenko, & Wegner. (1999). *Scheduling on parallel machines with single and multiple servers*.

Lee, C. Y., Lei, L., & Pinedo, M. (1997). *Annals of Operations Research*, *70*, 1–41.

Levner, E., Kats, V., de Pablo, D. A. L., & Cheng, T. (2010). Complexity of cyclic scheduling problems: A state-of-the-art survey. *Computers & Industrial Engineering*, *59*(2), 352–361.

Ou, J., Qi, X., & Lee, C.-Y. (2009). Parallel machine scheduling with multiple unloading servers. *Journal of Scheduling*, *13*(3), 213–226.

Pfund, M., Fowler, J. W., & Gupta, J. N. D. (2004). A survey of algorithms for single and multi-objective unrelated parallel machine deterministic scheduling problems. *Journal of the Chinese Institute of Industrial Engineers*, *21*(3), 230–241.

Pincus, M. (1970). Letter to the editor—a monte carlo method for the approximate solution of certain types of constrained optimization problems. *Operations Research*, *18*(6), 1225–1228.

Sethi, S., Sidney, J., & Sriskandarajah, C. (2001). Scheduling in dual gripper robotic cells for productivity gains. *IEEE Transactions on Robotics and Automation*, *17*(3), 324–341.

Sethi, S., Sriskandarajah, C., Sorger, G., Blazewicz, J., & Kubiak, W. (1992). Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems*, *4*(3 – 4), 331–358.

Silva, J. M. P., Teixeira, E., & Subramanian, A. (2019). Exact and metaheuristic approaches for identical parallel machine scheduling with a common server and sequence-dependent setup times. *Journal of the Operational Research Society*, *72*(2), 444–457.

Soukhal, A., & Martineau, P. (2005). Resolution of a scheduling problem in a flowshop robotic cell. *European Journal of Operational Research*, *161*(1), 62–72.

Sriskandarajah, C., Drobouchevitch, I., Sethi, S., & Chandrasekaran, R. (2004). Scheduling multiple parts in a robotic cell served by a dual-gripper robot. *Operations Research*, *52*(1), 65–82.

Sriskandarajah, C., Hall, N. G., & Kamoun, H. (1998). Scheduling large robotic cells without buffers. *Annals of Operations Research*, *76*, 287–321.

*Statistical review of world energy, 70th edition* (tech. rep.). (2021). BP.

Su, C. (2011). Online LPT algorithms for parallel machines scheduling with a single server. *Journal of Combinatorial Optimization*, *26*(3), 480–488.

Torjai, L., & Kruzslicz, F. (2015). Mixed integer programming formulations for the biomass truck scheduling problem. *Central European Journal of Operations Research*, *24*(3), 731–745.

Werner, F., & Kravchenko, S. A. (2010). Scheduling with multiple servers. *Automation and Remote Control*, *71*(10), 2109–2121.

*World robotics report international federation of robotics* (tech. rep.). (2020). IFR International Federation of Robotics.

Xie, X., Li, Y., Zhou, H., & Zheng, Y. (2012). Scheduling parallel machines with a single server. *Proceedings of 2012 International Conference on Measurement, Information and Control*.

Zarandi, M. F., Mosadegh, H., & Fattahi, M. (2013). Two-machine robotic cell scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, *40*(5), 1420–1434.

Zhang, L., & Wirth, A. (2009). On-line scheduling of two parallel machines with a single server. *Computers & Operations Research*, *36*(5), 1529–1553.