Maecker, Söhnke

Novel Solution Algorithms for Machine Scheduling Problems Emerging in Complex Systems

Dissertation
for obtaining the degree of Doctor of Business and Economics
(Doctor rerum politicarum – Dr. rer. pol.)

at WHU – Otto Beisheim School of Management

June 2022

First advisor: Prof. Dr. Liji Shen
Second advisor: Prof. Dr. Bahram Alidaee

Acknowledgments

The writing of this dissertation would not have been possible without the great support and assistance of many people whom I would like to thank for this.

First of all, I would like to express my sincerest gratitude to my supervisor Professor Liji Shen who gave me the great opportunity to write this dissertation in the first place. Your consistent guidance and the sharing of your expertise are invaluable to this work.

Furthermore, I am particularly grateful to Professor Bahram Alidaee who acted as my second supervisor and at all times supported me in this endeavor through his advice.

I would also like to thank Professor Dauzère-Pérès, Professor Lars Mönch, and Professor Haibo Wang for their co-authorship in the research papers that are the basis for this cumulative dissertation. It was a great honor to work with all of you and I am very grateful for your contributions.

Finally, I would like to thank my family and friends. Your support and encouragement never let me doubt that I could reach this goal.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Within the broad field of combinatorial optimization, sequencing and scheduling problems play a crucial role in manufacturing and service industries in order to maintain operational efficiency and competitiveness in the marketplace. Effective sequencing and scheduling can help to meet shipping dates and therefore prevent loss of goodwill or ensure an efficient use of the available resources (Pinedo, 2016). Nowadays, the significant increase in computing power over the last decades has made it possible to consider more complex problems and to develop sophisticated techniques to tackle those in practice.

Therefore, as a response to recent developments in industry, the integration of new aspects into classical scheduling problems shall be the subject of this dissertation. To be precise, the first major contribution is the consideration of new scheduling problems including the job delivery aspect in parallel machine environments to address the trend of more service-oriented production systems. Another important development in industry is to include the energy consumption and cost into machine scheduling decisions due to a rising environmental awareness. In response, the investigation of energy costs in the context of a flexible job shop environment constitutes the second major contribution of this thesis. The combinatorial optimization problems considered in this work belong to the complexity class of NP-hard problems, i. e. it is very unlikely that there exists an exact algorithm with polynomially bounded runtime. Consequently, the objective is not only to formalize these problems and to devise mathematical programming formulations, which can determine optimal solutions for small-sized problem instances, but also to conduct theoretical problem analyses based on which sophisticated (meta-)heuristic approaches can be developed in order to obtain good solutions for problem instances of practical size within reasonable time.

The dissertation is organized as follows: In Chapter 2, an identical parallel machine scheduling

problem with machine-dependent delivery times is considered, where the objective is to minimize the total weighted tardiness. A mixed integer liner programming (MILP) formulation is proposed for this problem as well as a variable neighborhood search (VNS) and a memetic algorithm (MA). Furthermore, structural properties of the problem are used to reduce the computational cost of local search and to improve the VNS. In a computational study, it is shown that the VNS outperforms the other solution approaches.

The delivery aspect considered in Chapter 2 is extended in Chapter 3, where job-machine-dependent delivery times in an unrelated parallel machine environment are studied with the objective of total weighted completion time minimization. The problem is formulated as a binary quadratic assignment program and a tabu search (TS) algorithm is proposed, that employs an efficient data structure. In a computational study, the TS is compared with a modified version of the VNS approach from Chapter 2.

In Chapter 4, the problem studied in Chapter 2 is generalized as an unrelated parallel machine total weighted tardiness problem with eligibility constraints and job-machine-dependent delivery times. An MILP formulation is developed and theoretical analyses, that incorporate previous findings from Chapter 2, are conducted to derive properties, that are used to improve the performance of a VNS scheme. In a computational study, the integration of the theoretical results into the VNS is investigated and the VNS scheme is compared with the application of commercial solver software to the MILP as well as a constructive heuristic.

Finally, Chapter 5 deals with the total energy cost (TEC) minimization of flexible job shop schedules when time-of-use electricity tariffs (TOU) are present, subject to an upper bound on the makespan. Additionally, the premise of a given machine allocation and sequence of operations, that may not be changed, is imposed. The corresponding optimization problem is formulated as an MILP and analyzed from a theoretical perspective. Several TOU constellations are examined for which optimal solutions can be determined in polynomial time. For the general TOU case, optimality properties are formulated, based on which heuristic procedures are devised. The proposed solution approaches are compared in a computational study and the potential of TEC savings through a relaxation of the makespan is investigated.

Chapter 2

Solving Parallel Machine Problems with Delivery Times and Tardiness Objectives*

2.1 Introduction

In most manufacturing and distribution systems, semi-finished jobs are transferred from one facility to another for further processing or finished jobs are delivered to the customer. In the latter case, the job completion time is defined as the time by which the job arrives at the customer. Therefore, job processing and delivery must be carefully coordinated to achieve ideal overall system performance (Lee and Chen, 2001). This becomes especially important for large companies, which operate on a global scale and whose production resources are geographically distributed. In global supply chains it often occurs that jobs are shipped from these production facilities to a centralized warehouse or distribution center for further use. This study is also inspired by similar structures in the context of *cloud manufacturing* (CM). CM arises as a new manufacturing paradigm through the shift from production-oriented to service-oriented manufacturing. In CM, manufacturing resources are managed in a centralized way encapsulated in cloud services and made available to clients, who can use these resources according to their requirements (Xu, 2012). For further reading on CM, please refer to Wu et al. (2013), Zhang et al. (2014b), or Wu et al. (2015). To be precise, we are motivated by the example of a cloud garment manufacturing company which utilizes cloud manufacturing resources to produce customized suits. Customers can design the details of their suit exactly as they want and place their orders via a mobile application. After batching and assigning the orders to its nationwide factories, the company must guarantee a delivery within 10 days. In

*This chapter is based on the paper *Solving Parallel Machine Problems with Delivery Times and Tardiness Objectives*, which is published in the journal *Annals of Operations Research* and cited as Maecker and Shen (2020).

terms of scheduling, these facilities can be regarded as parallel machines where the delivery of a job is affected by the geographical location of the respective machine. Therefore, the delivery aspect has to be taken into account when making decisions regarding the assignment of jobs to machines. However, traditional scheduling models often ignore this important factor. Particularly, the case of delivery times, which depend on distributed production resources, requires more investigation.

To address this problem, in this research, we formulate a parallel machine problem where a machine-dependent delivery time (MDT) occurs after processing a job on a machine and the objective is to minimize the total weighted tardiness (TWT). Since the parallel machine total tardiness problem (PMTT) is NP-hard in the strong sense (Pfund et al., 2004), this problem is NP-hard in the strong sense, too. Thus, finding an algorithm which provides an optimal solution in polynomially bounded time is very unlikely.

This chapter is organized as following: First, we present a review on related machine scheduling literature involving delivery times as well as total (weighted) tardiness minimization. Thereafter, a mixed integer linear programming formulation (MILP), which is based on a problem transformation and does not require machine-indexed decision variables, is proposed for the problem. To develop a solution approach for large-sized instances, we design a variable neighborhood search (VNS) algorithm, that uses fast evaluation techniques (FET) during local search (LS). These FETs calculate only the objective value difference based on neighborhood-specific rules and therefore lead to significantly reduced requirement on computing power. To compare the single-solution-based VNS against other approaches, we propose a memetic algorithm (MA) as well as an Apparent Tardiness Cost (ATC) heuristic and test these (meta-)heuristics on a large set of randomly generated test instances. Additionally, the CPLEX solver is run on a set of small instances and results regarding the computational benefit of our FETs are presented.

2.2 Related Literature

In this section, first, we present a comprehensive review on related literature considering machine scheduling problems involving delivery times. Thereafter, we summarize relevant studies concerned with total (weighted) tardiness minimization.

Maggu and Das (1980) first consider job transportation in a two-machine flow shop makespan problem where job-dependent transportation times occur between the processing stages and transportation capacity is unlimited. Both Potts (1980) and Hall and Shmoys (1992) study the problem of scheduling jobs on a single machine with release dates and job-dependent delivery times to min-

imize the time by which all jobs are delivered and provide heuristics with worst case analysis. The parallel machine case of this problem is considered by Carlier (1987). A worst case analysis for Jackson's rule (Jackson, 1955) for this problem is provided as well as a branch and bound (B&B) algorithm. A similar problem with identical release dates is studied by Woeginger (1994), who proposes several list scheduling heuristics with worst case analysis. For the problem considered by Carlier (1987), Gharbi and Haouari (2002) develop another B&B procedure and Gharbi and Haouari (2007) improve the performance of Jackson's rule through a preprocessing method. Furthermore, Mateo et al. (2018) study a bi-objective parallel machine problem with job release dates, delivery times, and eligibility constraints, where a penalty is induced depending on the quality of the machine processing a job. An algorithm is proposed to find an approximate Pareto Front for minimization of makespan and the total penalty.

An on-line scheduling approximation algorithm (AA) for the single machine case with release dates to minimize the time by which all jobs are delivered is proposed by Hoogeveen and Vestjens (2000) and later improved by Tian et al. (2008). Tian et al. (2007) consider a similar problem on a single batch processing machine with release dates and delivery times and develop on-line AAs for the bounded and unbounded case. Variations of this problem are also studied and solved through on-line AAs by Yuan et al. (2009), Fang et al. (2011a), Tian et al. (2011), and Tian et al. (2012). Fang et al. (2011b) consider this problem on parallel batch machines and solve it using AAs. Another online AA is proposed by Liu and Lu (2015) for the case with two identical machines.

Lee and Chen (2001) define two types of scheduling problems with job delivery where the transportation capacity is limited in terms of both available number of vehicles and vehicle capacity. *Type-1* transportation considers job transportation inside a manufacturing facility between processing stages and *type-2* transportation takes place between the facility and a customer area. In both cases, jobs share a common delivery time and the objective is to minimize the makespan. A complexity analysis is presented for single machine, parallel machine, and flow shop environments. Chang and Lee (2004) study *type-2* transportation for the one and two machine case with jobs that require different amounts of space on the transportation vehicle and develop AAs. Li et al. (2005) investigate an extension of the problem by Chang and Lee (2004) for the single machine case, where multiple customer areas exist to which jobs are delivered by a single vehicle with limited capacity and the objective is to minimize the sum of arrival times. Li et al. (2005) discuss the complexity of the general case as well as special cases and propose a dynamic programming algorithm. For the problem raised by Chang and Lee (2004), AAs are also proposed by He et al. (2006), Zhong et al. (2007), Lu and Yuan (2008a), and Liu and Lu (2011) for the single machine case and by Su et al. (2009) for the two machine case. Cakici et al. (2014) investigate a similar problem, where jobs are

first processed on parallel machines and then delivered in batches by a single capacitated vehicle with a limited number of trips. Heuristics are proposed for minimization of total weighted completion time. Wang and Cheng (2007) incorporate machine unavailability into the model by Chang and Lee (2004) and propose AAs for the single and parallel machine case. Lu and Yuan (2008b) propose an AA for the unbounded parallel batch machine case. Koulamas and Kyparisis (2010) and Liu et al. (2012) study a single machine problem where jobs have *past-sequence-dependent* delivery times proportional to their waiting time before processing and present polynomial-time algorithms for multiple optimization criteria. For TWT minimization, Liu et al. (2012) develop a polynomial time algorithm for a special case of the problem. Dong et al. (2013) present AAs for *type-2* problem on two machines and in open-shop production with one customer. Chen et al. (2015) show, that the preemptive case of *type-2* problem on identical parallel machines with one customer is strongly NP-hard and propose an AA. A batch delivery scheduling problem with job release dates on a single machine is considered by Ahmadizar and Farhadi (2015), where all jobs of a batch are instantly delivered upon finishing the last job in the batch and the objective is to minimize the sum of multiple cost components including delivery costs. To solve the problem, an imperialist competitive algorithm is proposed. Pei et al. (2015) study a serial bounded batching machine problem, where the job processing time is a linear function of its starting time. After processing, batches are delivered on a single vehicle with limited capacity to a customer. Another definition of job delivery times is presented by Chen et al. (2016) who investigate a parallel machine problem where a set of delivery times are given and each delivery time needs to be assigned to an individual job.

Most of the studies on machine scheduling with delivery times focus on makespan minimization, while due-date-related criteria are almost left out of consideration. However, in order to measure customer satisfaction and delivery efficiency, TWT is a more relevant objective. A vast amount of literature covering parallel machine problems has been published over the past decades. However, the proportion of studies considering TWT minimization is rather limited in comparison to completion- or flow time related objectives (Shim and Kim, 2007). Following, we present relevant literature on the parallel machine problem with total (weighted) tardiness minimization.

With regards to the parallel machine TWT problem (PMTWT), multiple solution approaches have been proposed. Alidaee and Rosa (1997) extend the Modified Due Date (MDD) rule by Baker and Bertrand (1982) for the PMTWT. Another heuristic called PSK is developed by Panwalkar et al. (1993) to minimize the mean tardiness on a single machine. PSK combines principles of the Shortest-Processing-Time and Earliest-Due-Date rules. Koulamas (1997) develops the KPM-heuristic which is an extension of PSK for the same problem on parallel machines. Furthermore,

Vepsalainen and Morton (1987) introduce the ATC rule for the single machine TWT problem, which has been applied to multiple other tardiness-related scheduling problems. For the PMTT problem, Biskup et al. (2008) develop several efficient heuristics. Furthermore, Lin et al. (2011) apply modifications of MDD, KPM and ATC as well as a genetic algorithm to the unrelated PMTWT. Azizoglu and Kirca (1998), Yalaoui and Chu (2002), and Shim and Kim (2007) investigate the PMTT and develop B&B procedures. Another B&B algorithm is proposed by Liaw et al. (2003) for the unrelated PMTWT. Srinivasa Raghavan and Venkataramana (2009) use ant colony optimization (ACO) to solve the PMTWT. ACO algorithms are also implemented for the unrelated PMTWT problem by Zhou et al. (2007), Mönch (2008), and Lin et al. (2013).

To conclude, among literature considering delivery times in machine scheduling, due-date-related criteria are hardly considered even though due dates play a major role in supply chain environments. Additionally, the case of MDT has not yet been discussed. To the best of our knowledge, this is the first study to address the problem of TWT minimization on parallel machines with MDT.

2.3 Problem Formulation

The problem can be described as follows: Given are a set of n jobs $j = 1, \ldots, n$ and m identical machines $h = 1, \ldots, m$. Each job needs to be processed by one and only one machine without interruption while each machine can handle exactly one job at a time. All jobs are available at time zero. Each job j has a specific processing time p_j, due date d_j, and weight w_j. A machine-dependent delivery time q_h occurs immediately upon completing a job on the respective machine. While a job is being transferred, the machine may already start processing the next job in line. The transportation capacity is assumed to be unlimited in terms of both vehicle availability and vehicle capacity. The problem is to determine a schedule π, that minimizes the TWT ($\sum w_j T_j$). The tardiness of a job T_j is defined as $T_j = \max\{C_j - d_j, 0\}$ with C_j being the time job j reaches the customer. Following the conventional three-field-notation by Graham et al. (1979), the problem can be expressed as $Pm|q_h|\sum w_j T_j$.

2.3.1 A Mixed Integer Linear Programming Formulation

We formulate the $Pm|q_h|\sum w_j T_j$ problem as an MILP model. We are able to omit machine-dependent variables through a transformation of the problem into a regular PMTWT problem:

For each machine, a dummy job is defined with completion time equal to the delivery time of the respective machine, resulting in a total of $n + m$ jobs, where $j_{n+1}, \ldots, j_{n+m}$ are the dummy jobs with $C_{n+h} = q_h$ $(h = 1, \ldots, m)$. Each dummy job is scheduled at the first position of the machine and therefore effectively increases the completion times of all following jobs by the machine delivery time as depicted in Figure 2.1. This way, delivery times do not need to be considered for the completion time calculation of the remaining jobs. This structural property as well offers opportunities for the design of FETs, which will be discussed in Section 2.4.3. With regards to the mathematical formulation, this transformation allows the problem to be modeled similar to regular PMTWT problems by using a binary precedence variable x_{ij} with

$$x_{ij} = \begin{cases} 1, & \text{if job } j \text{ is sequenced immediately after job } i, \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Note that x_{ij} only needs to be defined for $i = 1, \ldots, n + m$ and $j = 1, \ldots, n + 1$, since one job $(n + 1)$ is sufficient to mark the end of the schedule on each machine. The problem can now be formulated as following:

$$\min \sum_{i=1}^{n} w_j T_j \tag{2.2}$$

subject to

$$\sum_{j=1;j\neq i}^{n+1} x_{ij} = 1 \quad i = 1, \ldots, n + m; \tag{2.3}$$

$$\sum_{i=1;i\neq j}^{n+m} x_{ij} = 1 \quad j = 1, \ldots, n; \tag{2.4}$$

$$C_{n+h} \geq q_h \quad h = 1, \ldots, m; \tag{2.5}$$

$$C_j \geq C_i + p_j - H(1 - x_{ij}) \quad i = 1, \ldots, n + m; j = 1, \ldots, n; i \neq j; \tag{2.6}$$

$$T_j \geq C_j - d_j \quad j = 1, \ldots, n; \tag{2.7}$$

$$T_j \geq 0 \quad j = 1, \ldots, n; \tag{2.8}$$

$$x_{ij} \in \{0, 1\} \quad i = 1, \ldots, n + m; j = 1, \ldots, n + 1. \tag{2.9}$$

The objective (2.2) is to minimize the TWT. Constraint (2.3) states, that all jobs and dummy jobs must have exactly one following job. According to (2.4), all non-dummy jobs must be preceded by exactly one job which is either a dummy job or a regular job. Note that combining (2.3) and (2.4) ensures, that dummy jobs are scheduled first. Inequality (2.5) defines the completion times of the dummy jobs. In constraint (2.6), the completion time of all remaining jobs is calculated based

on the completion time of the preceding job in the sequence, where H is a sufficient large number. Constraints (2.7) through (2.9) define the tardiness and precedence variable. An upper bound for H can be calculated by

$$H = \sum_{j=1}^{n} p_j + \max_{h=1,\dots,m} \{q_h\}.$$

Note that our MILP is also valid if $n < m$. In this case, jobs are assigned to the machines with n smallest delivery times. On the remaining machines, dummy jobs at the start of the schedule are immediately succeeded by dummy job $n + 1$ so that equality constraints (3) hold. An alternative formulation utilizing a machine-indexed variable, that marks the first job on each machine, can be found in the appendix.

2.4 Variable Neighborhood Search

VNS is an LS-based metaheuristic that searches changing neighborhood structures for improving solutions to escape from local optima (Mladenović and Hansen, 1997). For a recent overview of basic VNS schemes and variants, the reader is referred to Hansen et al. (2017). VNS has been successfully applied to multiple combinatorial optimization problems (Hansen et al., 2010), including various parallel machine scheduling problems ((De Paula et al., 2007), (Driessel and Mönch, 2011), (Cheng et al., 2012)). Furthermore, VNS is used in multiple hybrid approaches for parallel machine problems (see e. g. (Anghinolfi and Paolucci, 2007), (Chen and Chen, 2009), (Behnamian et al., 2009), (Chen et al., 2013)). The basic structure of our VNS is given in Algorithm 1. Note that our VNS procedure differs from the conventional basic VNS structure such that after each LS, the resulting locally optimal solution π'' is kept even if it is worse than the incumbent solution π. This design facilitates diversification during the search process so that unpromising regions of the search space may be escaped more quickly. Furthermore, we use neighborhood-

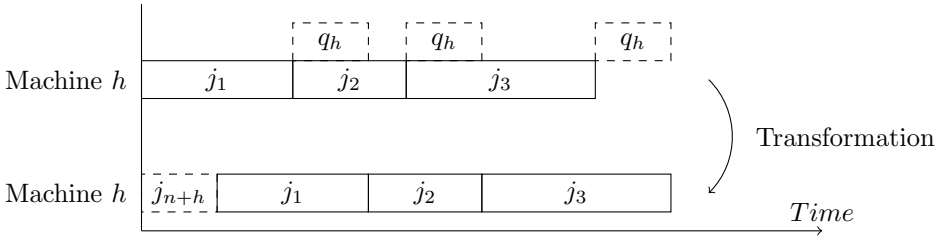

Figure 2.1: The problem can be transformed by defining a dummy job which is sequenced at the first position of the machine with completion time equal to the machine delivery time. For all remaining jobs, delivery times are now omitted.

specific FETs in our LS to speed up the search process. In the following, we will describe the components of our VNS.

2.4.1 Neighborhood Structures

In our VNS, a solution π is represented by m job sequences π_h $(h = 1, \ldots, m)$. Let S denote the set of all feasible solutions for the problem at hand. We use a total of $l_{max} = 4$ different neighborhoods $N_l(\pi) \subseteq S$ $(l = 1, \ldots, l_{max})$. A neighborhood is defined by how an incumbent solution is altered to generate a new feasible solution. These neighborhoods are commonly used in VNS algorithms for parallel machine scheduling problems and consist of job swaps and job insertion moves:

- N_1 : Swap two jobs on the same machine
- N_2 : Swap two jobs across two different machines
- N_3 : Reinsertion of a job on the same machine
- N_4 : Reinsertion of a job on a different machine

The sequence in which different neighborhoods are visited is important for the success of VNS. The proposed sequence was determined based on preliminary experiments.

Algorithm 1 Basic VNS structure (I)

Input Neighborhood structures N_l $(l = 1, \ldots, l_{max})$, random initial solution π_0;
$\pi = \pi_0$;
$f_{best} = f(\pi)$;
while Time limit not reached **do**
    $l = 1$;
    repeat
        Shaking: Pick random solution π' from $N_l(\pi)$;
        Local Search: Find local optimum π'' in $N_l(\pi')$;
        if $f(\pi'') < f_{best}$ **then**
            $f_{best} = f(\pi'')$;
            $l = 1$;
        else
            $l = l + 1$;
        end if
        $\pi = \pi''$;
    until $l > l_{max}$
end while
Return f_{best};

2.4.2 Shaking and Local Search

At the start of each iteration a random solution π' is picked from the current neighborhood $N_l(\pi)$ of the incumbent solution π. This step is called *Shaking* and allows the search to escape from local optima. Afterwards, an LS is performed on π' until the local optimum π'' is reached. The LS systematically examines all possible moves within the current neighborhood $N_l(\pi)$ and uses FETs, which will be presented in the next section, to evaluate neighborhood solutions. Furthermore, a *first improvement* strategy is employed to reduce computational effort. In this strategy, the first improving solution found is accepted instead of evaluating a complete neighborhood. The LS returns the local optimum π'', which then becomes the incumbent solution π of the next iteration. f_{best} is used to record the best found objective value. Each time, a new best solution is found, the search continues with neighborhood N_1 ($l = 1$) in the next iteration. Otherwise, the search switches to the next neighborhood structure in line. If all neighborhoods have been visited, the search continues with the first neighborhood. This procedure is repeated until a prescribed time limit is reached.

2.4.3 Fast Evaluation Techniques for Local Search

LS is a computationally costly procedure due to a high number of evaluations of neighboring solutions. To reduce this computational effort, we design neighborhood-specific FETs, that exploit the circumstance, that only a small subset of jobs is affected by neighborhood moves. Therefore, it is important to define dedicated neighborhood structures. For example, instead of defining a neighborhood which consists of swapping any two jobs, it is advantageous to distinguish the cases, where these two jobs are located on the same machine and on different machines, since for each situation an individual FET can be designed. Hence, the exploitation of problem-specific properties is crucial to the efficient implementation of our VNS. Other FETs can be found in Tasgetiren et al. (2009) and Xu et al. (2014) for the single machine TWT problem with sequence-dependent setup times. Note that our FETs are also applicable to the PMTWT problem without delivery times.

The basic FET procedure is presented in Algorithm 2. Based on the parameters of the move in a certain neighborhood structure, e. g., the positions of two jobs to be swapped in a sequence, the set of affected jobs K is determined. For these jobs, the new completion times C'_j induced by the move can be calculated based on neighborhood-specific rules. After calculating the new tardiness values T'_j for the affected jobs, the total objective value difference Δ induced by the move can be calculated as the weighted sum of differences between new and old tardiness values. If $\Delta < 0$, the

move improves the current solution and is executed. The objective value is incremented by Δ.

In the following, we describe how to determine set K and calculate C'_j for the different neighborhood structures used in our VNS. We use $[i]_h$ to represent the job scheduled at the i-th position on machine h and $[l]_h$ to represent the job at the last position on machine h.

- *Swapping jobs on one machine (N_1)*: When swapping jobs on one machine, the positions of two jobs $[i]_h$ and $[j]_h$ in the sequence of a machine h are interchanged. For $i < j$, $K = \{[i]_h, [i+1]_h, \ldots, [j]_h\}$. Let $\delta = p_{[i]_h} - p_{[j]_h}$, then

$$C'_j = C_j - \delta, \quad j \in K \setminus \{[i]_h, [j]_h\}, \tag{2.10}$$

$$C'_{[i]_h} = C_{[j]_h}, \tag{2.11}$$

and

$$C'_{[j]_h} = C_{[i]_h} - \delta. \tag{2.12}$$

- *Swapping jobs on two machines (N_2)*: This move exchanges the positions of two jobs $[i]_h$ on machine h and $[j]_{h'}$ on another machine h'. In this case, $K = K_1 \cup K_2$, where $K_1 = \{[i]_h, [i+1]_h, \ldots, [l]_h\}$ and $K_2 = \{[j]_{h'}, [j+1]_{h'}, \ldots, [l]_{h'}\}$. Let $\delta = p_{[i]_h} - p_{[j]_{h'}}$, then

$$C'_j = C_j - \delta, \quad j \in K_1 \setminus [i]_h, \tag{2.13}$$

$$C'_j = C_j + \delta, \quad j \in K_2 \setminus [j]_{h'}, \tag{2.14}$$

$$C'_{[i]_h} = C_{[j]_{h'}} + \delta, \tag{2.15}$$

Algorithm 2 Basic FET structure

```
1: Input: Solution π with objective value f, parameters for move to solution π' ∈ N_l(π)
2: Determine set K of affected jobs;
3: for j ∈ K do
4:     Calculate new completion time C'_j;
5:     Calculate new tardiness T'_j = max{C'_j − d_j, 0};
6: end for
7: Δ = Σ_{j∈K} w_j (T'_j − T_j);
8: if Δ < 0 then
9:     π = π';
10:    f = f + Δ;
11: end if
```

and

$$C'_{[j]_{h'}} = C_{[i]_h} - \delta. \tag{2.16}$$

- *Job insertion on the same machine (N_3)*: Job insertion on the same machine consists of selecting a job $[i]_h$ on some machine h and reinserting it at another position j in the sequence. Let $\delta = p_{[i]_h}$. If $j < i$ holds, indicating that job $[i]_h$ is moved to the left, then $K = \{[j]_h, [j+1]_h, \ldots, [i]_h\}$ with

$$C'_j = C_j + \delta, \quad j \in K \setminus [i]_h, \tag{2.17}$$

and

$$C'_{[i]_h} = C_{[j]_h} + \left(p_{[i]_h} - p_{[j]_h}\right). \tag{2.18}$$

In the case of $j > i$, $K = \{[i]_h, [i+1]_h, \ldots, [j]_h\}$ with

$$C'_j = C_j - \delta, \quad j \in K \setminus [i]_h, \tag{2.19}$$

and

$$C'_{[i]_h} = C_{[j]_h}. \tag{2.20}$$

- *Job insertion on other machine (N_4)*: Job insertion on another machine consists of selecting a job $[i]_h$ from the sequence on a machine h and reinserting it at some position j in the sequence on another machine h'. In this case, $K = K_1 \cup K_2$, where $K_1 = \{[i]_h, [i+1]_h, \ldots, [l]_h\}$ and $K_2 = \{[j]_{h'}, [j+1]_{h'}, \ldots, [l]_{h'}\}$. Let $\delta = p_{[i]_h}$, then

$$C'_j = C_j - \delta, \quad j \in K_1 \setminus [i]_h, \tag{2.21}$$

$$C'_j = C_j + \delta, \quad j \in K_2, \tag{2.22}$$

and

$$C'_{[i]_h} = C_{[j]_{h'}} + \left(p_{[i]_h} - p_{[j]_{h'}}\right). \tag{2.23}$$

There are two special cases to consider: If no jobs are scheduled to machine h', then

$$C'_{[i]_h} = p_{[i]_h} + q_{h'}. \tag{2.24}$$

Furthermore, if $[i]_h$ is inserted at the end of the schedule on h', then

$$C'_{[i]_h} = C_{[j-1]_{h'}} + p_{[i]_h}. \tag{2.25}$$

2.5 Memetic Algorithm

Next, we design an MA as a second approach to solve the problem at hand. As one of the most promising population-based metaheuristics, MA is used here to compare with the trajectory method VNS. The term *memetic algorithm* was introduced by Moscato and Norman (1992) and refers to evolutionary algorithms incorporating LS procedures to locally improve individuals of the population. A formal description of MAs is given in Radcliffe and Surry (1994). In our MA, the local improvement phase, often referred to as *hill-climbing*, improves each offspring generated in the recombination phase through a standard LS procedure.

2.5.1 Representation and Initialization

The population consists of P_n individuals with a permutation-like representation of solutions, where each individual i consists of m job sequences π_h^i $(h = 1, \ldots, m)$ corresponding to the machines. The initial population is randomly generated.

2.5.2 Evaluation and Selection

A fitness value f_i is associated with each individual. Since we deal with a minimization problem, the fitness of an individual is given by the reciprocal of its objective value:

$$f_i = \frac{1}{\sum_{j=1}^{n} w_j T_j}. \tag{2.26}$$

The population size is comparably small in our experiments and therefore prone to sampling errors, which can lead to premature convergence. Therefore, we use the *stochastic universal sampling* algorithm by Baker (1987) for selection of parent individuals, since the minimum and maximum number of times an individual may be selected is bounded in this approach. Additionally, we employ an *elitist survival* strategy, where the best found individual is always kept.

2.5.3 Recombination

Recombination consists of crossover and mutation. The crossover operator combines two parent individuals to generate an offspring solution and the mutation operator alters single individuals. In total, $P_c \times P_n$ offspring individuals are generated by crossover, where P_c is the crossover probability, and $P_m \times P_n$ offspring solutions are generated by mutation, where P_m is the mutation probability, respectively.

For crossover, we use the *subschedule-preservation crossover operator* (SPC) proposed by Cheng et al. (1995). SPC randomly selects two parents from the pool of parents and copies the entire job sequence of a random machine from the first parent to the offspring solution as well as the overall job-machine-partitioning. According to this partitioning, the remaining jobs are then copied from the second parent by a left-to-right scan. Furthermore, single parent individuals are randomly selected and altered by the mutation operator through the reinsertion of a random job at another random position in the schedule.

2.5.4 Hill-Climbing

In the hill-climbing phase, a standard LS procedure with *first improvement* strategy is applied to each offspring solution, that systematically examines all possible job swaps and insertion moves on one or two machines. After hill-climbing, each offspring solution constitutes a locally optimal solution. Finally, the new generation is randomly selected from the offspring and the former generation while the best found solution is always kept.

2.6 Computational Results

Metaheuristic experiments were conducted on an Intel Xeon CPU E5-2697 v3 computer with 2.60 GHz and 128 GB RAM. Algorithms were coded in C++ and compiled by the g++-compiler. Furthermore, we used no multi-threading. Results for metaheuristic approaches were generated as the best result out of 5 independent runs. We conducted experiments for three levels of maximum computing time: 5, 10, and 30 seconds. The parameter setting for the MA was determined in preliminary tests with $P_n = 10$, $P_c = 0.2$, and $P_m = 0.8$. For our ATC heuristic, reported results represent the best solution out of 10 runs with different k-values: $k \in \{0.5, 1, \ldots, 5\}$. The mathematical model from Section 2.3 was implemented in IBM ILOG CPLEX v12.8 using the CPLEX C++ API. The CPLEX tests were executed on the same hardware as metaheuristics with

number of cores limited to 12 and a prescribed time limit of 30 minutes per run, since the intention was to use CPLEX to generate benchmark results.

2.6.1 Problem Instance Data

We generated problem instances with different sizes $m \in \{5, 10, 20\}$ and $n = \alpha \times m$, where the factor α represents the average number of jobs per machine with $\alpha \in \{4, 8, 16\}$. Processing times were chosen from the uniform distribution $p_j \sim U(1, 100)$ and job weights from $w_j \sim U(1, 10)$. Furthermore, we used a modified version of the method proposed by Potts and Van Wassenhove (1982) to determine due dates:

$$d_j \sim U\left(\bar{p}\left(1 - T - \frac{R}{2}\right) + \bar{q}, \bar{p}\left(1 - T + \frac{R}{2}\right) + \bar{q}\right), \tag{2.27}$$

where T is the *tardiness factor*, R the *relative range of due dates*,

$$\bar{p} = \frac{1}{m}\sum_{j=1}^{n} p_j, \quad \text{and} \quad \bar{q} = \frac{1}{m}\sum_{h=1}^{m} q_h. \tag{2.28}$$

We chose $R \in \{0.4, 0.8\}$ and $T \in \{0.4, 0.8\}$. Moreover, we use three different machine delivery time settings: $q_h \sim U(1, 50)$, $q_h \sim U(1, 100)$, and $q_h \sim U(101, 200)$ to represent delivery times which are smaller, similar to, and larger than processing times. In total, we have 108 different configurations, for each one we generated 10 instances. Table 2.1 summarizes our instance data settings.

2.6.2 Comparison of VNS, MA, and ATC

In this experiment, we compare the performance of our single-solution-based metaheuristic VNS and the population-based MA. We ran the tests with three different levels of maximum computing time to examine whether the maximum computing time affects the relative performance of the approaches. Additionally, we report results of a heuristic approach based on the well-known ATC rule. A detailed description of our ATC heuristic is given in Appendix 2.A.2. The relative percentage deviation (RPD) is used for performance evaluation with

$$RPD = \frac{Z - Z^*}{Z^*} \times 100, \tag{2.29}$$

where Z is the best objective value found by an individual solution approach and Z^* is the best result among all approaches for each instance. Tables 2.2-2.4 report the results as average RPD

(ARPD) depending on instance size, due date settings, and delivery times.

The results for the complete instance set in Table 2.2 show that VNS outperforms the MA for all levels of computing time and the ATC heuristic. It can be observed that the ARPD increases with problem size for all approaches, except for $m = 5$, where the results for $\alpha = 16$ are better than for $\alpha = 8$ in the case of VNS and ATC. For $m = 20$, ATC results are better for $\alpha = 16$ than for $\alpha = 8$ as well. In general, the ARPD for MA increases significantly stronger than for VNS and ATC. The results of MA for large instances with $n = 160$ and $n = 320$ are non-competitive to VNS.

Table 2.1: Test instance configurations (I)

Factor	Level	Count
Machines m	$m \in \{5, 10, 20\}$	3
Jobs n	$n = \alpha \times m$, $\alpha \in \{4, 8, 16\}$	3
Processing times p_j	$p_j \sim U(1, 100)$	1
Weights w_j	$w_j \sim U(1, 10)$	1
Due date settings	$T \in \{0.4, 0.8\}$	2
	$R \in \{0.4, 0.8\}$	2
Delivery times q_h	$q_h \sim U(1, 50)$	
	$q_h \sim U(1, 100)$	3
	$q_h \sim U(101, 200)$	
Total number of configurations		108

Table 2.2: ARPD results of VNS, MA, and ATC depending on problem size

Problem size			5 *sec*		10 *sec*		30 *sec*	
m	α	ATC	VNS	MA	VNS	MA	VNS	MA
5	4	17.66	0.00	0.04	0.00	0.03	0.00	0.00
	8	24.69	0.75	1.16	0.68	0.66	0.21	0.42
	16	24.19	0.38	1.93	0.29	1.16	0.05	0.50
10	4	20.17	0.43	1.27	0.35	0.87	0.09	0.59
	8	24.31	0.60	3.08	0.37	2.13	0.15	0.99
	16	24.72	0.98	8.14	0.71	7.62	0.17	6.33
20	4	25.39	0.56	3.06	0.36	2.48	0.08	1.42
	8	29.37	0.78	6.28	0.53	5.19	0.12	3.90
	16	26.33	2.39	20.40	0.84	14.83	0.15	11.64
	Mean	24.09	0.76	5.04	0.46	3.88	0.11	2.87

With regard to the due date settings, Table 2.3 shows that instances with loose due dates ($T = 0.4$) appear harder to solve than instances with tight due dates. Furthermore, for both tight and loose due dates, the results are worse if a wide range of due dates ($R = 0.8$) is considered compared to a small range. Moreover, the ARPD in the case of loose due dates with a wide range is remarkably greater than that for all other due date settings.

Considering the results depending on the range of delivery times which are given in Table 2.4, no definitive influence of the delivery times setting on the results of the three approaches can be observed.

Table 2.3: ARPD results of VNS, MA, and ATC depending on due dates

Due dates			5 *sec*		10 *sec*		30 *sec*	
T	R	ATC	VNS	MA	VNS	MA	VNS	MA
0.4	0.4	20.61	0.28	1.68	0.18	1.26	0.05	0.78
	0.8	70.55	2.62	17.72	1.55	13.70	0.38	10.27
0.8	0.4	2.41	0.07	0.29	0.04	0.22	0.01	0.16
	0.8	2.79	0.09	0.46	0.06	0.36	0.02	0.26

Table 2.4: ARPD results of VNS, MA, and ATC depending on delivery times

Delivery times		5 *sec*		10 *sec*		30 *sec*	
q_h	ATC	VNS	MA	VNS	MA	VNS	MA
$U(1, 50)$	24.46	0.75	4.80	0.47	3.79	0.12	2.83
$U(1, 100)$	24.27	0.64	5.07	0.43	3.94	0.12	2.95
$U(101, 200)$	23.55	0.90	5.25	0.47	3.93	0.10	2.82

2.6.3 CPLEX Results for Small Instances

In addition, we solve small instances ($m = 5$, $\alpha \in \{4, 8\}$) using the CPLEX solver, as reported in Table 2.5. The ARPD values are calculated based on the metaheuristic results from Section 2.6.2 and the column *Best* reports the percentage of all instances per setting for which the CPLEX solution was identical to the best found solution. Note that for no instance CPLEX found a better objective value than the other approaches.

While for $\alpha = 4$, CPLEX in some cases is able to match the best found solutions by the metaheuristic approaches, it is consistently outperformed for the larger instances with $\alpha = 8$. Furthermore, CPLEX solutions confirm the observation from Section 2.6.2 that especially instances

with loose and wide-ranged due dates are hard. Interestingly, in contrast to the metaheuristic approaches, the performance of the model seems to be affected by the delivery times setting as in most cases the ARPD increases for larger delivery times.

2.6.4 Computational Benefits of FET

In order to investigate the computational advantage of our FETs, we conducted experiments on a subset of our test instances to compare FET with a normal recalculation of the objective value. For each of the four neighborhood structures presented in Section 2.4.3, we designed straightforward deterministic LS procedures, that start with a random initial solution and then systematically examine neighborhood solutions. Each time an improving move is found, the current solution is replaced by the corresponding neighborhood solution until a local optimum is reached. To ensure comparability, we used predefined seed values to generate the initial solutions. Ten initial solutions

Table 2.5: CPLEX results for small instances: $m = 5$, $\alpha \in \{4, 8\}$

Problem setting				CPLEX	
α	T	R	q_h	Best	ARPD
4	0.4	0.4	$U(1, 50)$	10%	2.12
			$U(1, 100)$	10%	2.08
			$U(101, 200)$	10%	4.35
		0.8	$U(1, 50)$	70%	1.50
			$U(1, 100)$	20%	4.54
			$U(101, 200)$	0%	7.08
	0.8	0.4	$U(1, 50)$	50%	0.05
			$U(1, 100)$	70%	0.04
			$U(101, 200)$	40%	0.43
		0.8	$U(1, 50)$	80%	0.04
			$U(1, 100)$	60%	0.15
			$U(101, 200)$	0%	0.58
8	0.4	0.4	$U(1, 50)$	0%	17.65
			$U(1, 100)$	0%	19.02
			$U(101, 200)$	0%	38.90
		0.8	$U(1, 50)$	0%	150.24
			$U(1, 100)$	0%	82.99
			$U(101, 200)$	0%	129.17
	0.8	0.4	$U(1, 50)$	0%	2.46
			$U(1, 100)$	0%	3.54
			$U(101, 200)$	0%	5.55
		0.8	$U(1, 50)$	0%	4.38
			$U(1, 100)$	0%	5.26
			$U(101, 200)$	0%	6.13

were generated, for which an LS utilizing normal recalculation is first performed and afterwards an LS using FET to evaluate moves. Since in both cases the LS is applied to the same initial solutions, the search trajectory is identical for both methods. For each LS, we recorded the times t_{norm} and t_{FET} necessary to reach local optima for these ten different initial solutions.

Since the computational benefit of FET depends on the problem size, i. e., the number of jobs and machines, we tested problems with 9 different sizes and 10 instances per size. We measure the benefit of FET by the average benefit factor (ABF)

$$ABF = \frac{t_{norm}}{t_{FET}}. \tag{2.30}$$

Test results are reported in Table 2.6 for each problem size and neighborhood structure. Our tests show significant benefits when integrating FETs. Specifically, the number of machines m affects the ABF for all neighborhoods, as ABF increases for instances with more machines. This can be explained by the fact that the number of machines, on which tardiness values of jobs need to be recalculated, remains constant: Tardiness values of jobs change only on one machine for moves in N_1 or N_3 and on two machines in N_2 and N_4. Assume that jobs are equally distributed among the machines. In this case, the computational effort of normal recalculation increases with the number of machines, whereas the effort in our FETs remains unchanged. Hence, the ABF increases with m. Note that the effort necessary to identify all relevant jobs is negligibly small. As for the effect of α on the ABF, similar results can be observed for N_1 and N_3 neighborhoods as well as for N_2 and N_4 neighborhoods. For N_1 and N_3, a strong and consistent increase of ABF can be observed depending on the average number of jobs per machine. This suggests that the portion of jobs, which are irrelevant for recalculation, becomes larger if the average number of jobs per machine

Table 2.6: Computational benefit of FET (ABF)

Problem size		Neighborhood structure			
m	α	N_1	N_3	N_2	N_4
5	4	5.26	5.35	7.40	10.17
	8	13.08	14.44	8.40	10.37
	16	21.33	23.02	9.30	10.44
10	4	12.37	12.91	15.26	19.62
	8	24.49	27.01	16.85	19.55
	16	37.86	41.39	18.25	20.12
20	4	22.56	24.51	30.72	37.78
	8	39.80	48.15	33.10	37.84
	16	61.56	73.29	36.01	39.09

increases. In the case of N_2 and N_4 neighborhoods, however, there is only a slight effect of α on the ABF.

2.7 Summary and Research Perspectives

In this research, we considered a parallel machine scheduling problem with machine-dependent delivery times to minimize total weighted tardiness. An MILP without machine-indexed variables was formulated. Furthermore, we proposed a VNS algorithm and designed FETs, that reduce computational effort during the LS. In our experiments we compared multiple solution approaches and showed that the proposed VNS outperforms an MA utilizing a standard LS and ATC heuristic as well as the CPLEX solver for small instances. Moreover, it was observed that the test problem size and due date setting affected the performance of all approaches, while the delivery time setting only affected the performance of CPLEX. Additionally, we examined the benefit of integrating problem-specific properties through our FETs in LS, which significantly reduces computing time.

The limitation of this work lies mainly in the delivery definition. While this offers opportunities for exploiting structural properties in solution approaches, we suggest for future work to consider general delivery conditions to reflect practical relevance. Additionally, even though our tests show promising results, the VNS with FETs should be further tested against other methods and be implemented for related problem settings to verify quality of this approach. In this context, it would also be interesting to show whether mechanisms similar to the FET can be integrated in other existing algorithms to further improve performance.

2.A Appendix

2.A.1 Alternative MILP Formulation for $Pm|q_h|\sum w_jT_j$

The following MILP is an alternative to the formulation presented in Section 2.3 and explicitly incorporates machine-dependent delivery times. It uses two binary decision variables x_{ij} with

$$x_{ij} = \begin{cases} 1, & \text{if job } j \text{ is sequenced immediately after job } i, \\ 0, & \text{otherwise,} \end{cases} \tag{2.31}$$

and y_{jh} with

$$y_{jh} = \begin{cases} 1, & \text{if job } j \text{ is the first sequenced job on machine } h, \\ 0, & \text{otherwise.} \end{cases} \tag{2.32}$$

Furthermore, a dummy job $j = n + 1$ is introduced with $p_{n+1} = 0$ that marks the end of the schedule on each machine. The problem can now be formulated as follows:

Notation:

p_j	Processing time of job j
w_j	Weight of job j
d_j	Due date of job j
q_h	Delivery time of machine h
C_j	Completion time of job j
T_j	Tardiness of jobs j
H	Sufficient large number
n	Number of jobs
m	Number of machines

$$\min \sum_{i=1}^{n} w_jT_j \tag{2.33}$$

subject to

$$\sum_{j=1}^{n} y_{jh} \leq 1 \quad h = 1, \ldots, m; \tag{2.34}$$

$$\sum_{h=1}^{m} y_{jh} \leq 1 \quad j = 1, \ldots, n; \tag{2.35}$$

$$y_{jh} + \sum_{i=1, i\neq j}^{n} x_{ij} = 1 \quad j = 1, \ldots, n; \quad h = 1, \ldots, m; \tag{2.36}$$

$$\sum_{i=1,i\neq j}^{n+1} x_{ji} = 1 \quad j = 1,\ldots,n; \tag{2.37}$$

$$C_j \geq (p_j + q_h)y_{jh} \quad j = 1,\ldots,n; \quad h = 1,\ldots m; \tag{2.38}$$

$$C_j \geq C_i + p_j - H(1 - x_{ij}) \quad i,j = 1,\ldots,n; i \neq j; \tag{2.39}$$

$$T_j \geq C_j - d_j \quad j = 1,\ldots,n; \tag{2.40}$$

$$x_{ij} \in \{0,1\} \quad i,j = 1,\ldots,n; \tag{2.41}$$

$$y_{jh} \in \{0,1\} \quad j = 1,\ldots,n; \quad h = 1,\ldots,m; \tag{2.42}$$

$$C_j, T_j \geq 0 \quad i = 1,\ldots,n \tag{2.43}$$

The objective (2.33) is to minimize the TWT. Constraints (2.34) and (2.35) ensure that at most one job is assigned to the first position on each machine. Constraints (2.36) and (2.37) determine job sequences. The completion time of the first jobs on machines is calculated by inequality (2.38) while (2.39) defines the completion time for the remaining jobs, where H is a sufficient large number. The tardiness of each job is calculated by inequality (2.40). Constraints (2.41) through (2.43) define the variables.

The MILP was tested on the set of small instances described in section 6.3 as well. In a direct comparison, a superiority of MILP performances could not be confirmed with statistical significance.

2.A.2 ATC Heuristic for $Pm|q_h|\sum w_jT_j$

The ATC rule was originally proposed for the single machine PMTWT problem but can be applied to other tardiness-related scheduling problems as well. The general idea of ATC is to calculate a priority value I_j for each job, which increases as its slack, i. e., the remaining time until its due date, reduces:

$$I_j = \frac{w_j}{p_j} \exp\left(-\frac{\max\{d_j - q_h - p_j - t_h, 0\}}{k\bar{p}}\right), \tag{2.44}$$

where k represents a *look-ahead parameter*, $\bar{p}$ the average processing time of jobs, and t_h the current workload of machine h. If jobs have no slack left, priority values equal shortest weighted processing time. Our implementation of ATC for $Pm|q_h|\sum w_jT_j$ is given in Algorithm 3.

In the first phase, an initial schedule is constructed using ATC. In each iteration, first, the fastest machine is determined under consideration of machine delivery times. For each unscheduled job, the priority value is calculated based on the current machine workload. Among all unscheduled

Algorithm 3 ATC heuristic for $Pm|q_h|\sum w_jT_j$

Initialize: Workloads $t_h = 0 \quad (h = 1, \ldots, m)$, set of unscheduled jobs $U = \{1, \ldots, n\}$
repeat
    Determine fastest machine $h^* = \arg\min_{h=1,\ldots,m}\{t_h + q_h\}$;
    for all $j \in U$ **do**
        Calculate priority value: $I_j = \frac{w_j}{p_j}\exp\left(-\frac{\max\{d_j - p_j - q_h - t_h, 0\}}{k\bar{p}}\right)$;
    end for
    Determine job j^* from U such that $I_{j^*} = \max_{j\in U}\{I_j\}$;
    (If ties exist, choose job with lowest index);
    Schedule job j^* at the first available position on machine h^*;
    Update machine workload $t_{h^*} = t_{h^*} + p_{j^*}$ and remove job j^* from U;
until $U = \emptyset$
Apply adjacent pairwise exchange procedure to improve given schedule;

jobs, the one with largest priority value is scheduled to the fist available position on the machine and the machine workload is incremented by its processing time. This procedure is repeated until all jobs are scheduled. In the second phase, an adjacent pairwise exchange procedure is applied to each machine to improve the given schedule.

Chapter 3

Unrelated Parallel Machine Scheduling in Distributed Manufacturing Systems*

3.1 Introduction

Moving from production-oriented to service-oriented manufacturing, *cloud manufacturing* (CM) arises as a new paradigm, where manufacturing resources are managed in a centralized way encapsulated in cloud services and made available to clients, who can use these resources according to their requirements (Xu, 2012). A detailed description of the CM concept, characteristics and key enabling technologies can be found e. g. in Zhang et al. (2014b). In our study, we consider CM in the context of supply chains, where jobs generated by customers can be assigned to geographically distributed production resources and need to be shipped to the customer upon completion. More precisely, we are motivated by the example of a cloud garment manufacturing company, whose customers can customize and order their suit via a mobile application. Orders are then batched and assigned to nationwide factories, which, in terms of scheduling, can be regarded as parallel machines. The delivery of a job is affected by the location of the respective factory and the location of the customer. According to the definition of scheduling problems with job delivery by Lee and Chen (2001), this situation is referred to as *type-2* transportation, which has received considerable attention. However, the existing literature almost exclusively considers the objective of makespan minimization. Recently, Maecker and Shen (2020) studied a problem with machine-dependent delivery times on identical parallel machines and pointed out, that a more general formulation of

*This chapter is based on the conference paper *Unrelated Parallel Machine Scheduling in Distributed Manufacturing Systems*, which was presented at the *International Conference on Optimization and Learning 2020* in Cádiz, Spain, and is cited as Maecker et al. (2020).

delivery times is beneficial.

In our study, we formulate the emerging problem of distributing jobs among multiple factories as an unrelated parallel machine scheduling problem, where the machines represent manufacturing plants and job-machine-dependent delivery times are considered. The objective is to minimize the total weighted completion time (TWC) of the jobs. We show that the problem reduces to an assignment problem and present a binary quadratic programming formulation. Furthermore, we propose an efficient data structure applicable in local search (LS) for large-scaled instances which is used in Tabu Search (TS) and Variable Neighborhood Search (VNS). In a computational study, we compare the two approaches.

3.2 Problem Formulation

Given are a set of n jobs $j = 1, \ldots, n$ and m unrelated machines $i = 1, \ldots, m$. Each job needs to be processed by one and only one machine without interruption while each machine can handle at most one job at a time. All jobs are available at time zero. Each job j has a weight w_j, a processing time p_{ij} associated with machine i, and a delivery time q_{ij}, that occurs immediately after completing j on the respective machine. While a job is being transferred, the machine may already process the next job in line. Transportation capacity is assumed to be unlimited in terms of both vehicle availability and capacity. The objective is to minimize the TWC ($\sum w_j C_j$) of the schedule. The completion time of a job, C_j, is defined as the time by which job j reaches the customer. Following the conventional three-field-notation by Graham et al. (1979), the problem can be expressed as $Rm|q_{ij}|\sum w_j C_j$.

For any given assignment of jobs to machines, sequencing jobs on each machine according to Smith's rule (Smith et al., 1956), i e. in non-decreasing order of p_{ij}/w_j-values, is optimal, since delivery times are decision-irrelevant. Given a schedule $\pi = \{S_1, \ldots, S_i, \ldots, S_m\}$, let $f(\pi)$ denote its objective value, S_i the job sequence on machine i and $\hat{C}_j$ the conventional completion time before delivery of job j. Jobs in S_i are sorted according to Smith's rule and by index, if for any two jobs j and j' we have $p_{ij}/w_j = p_{ij'}/w_{j'}$. Then, $f(\pi)$ is calculated by

$$f(\pi) = \sum_{i=1}^{m} \left[\sum_{j \in S_i} w_j \left(\hat{C}_j + q_{ij} \right) \right] = \sum_{i=1}^{m} \left[\sum_{j \in S_i} w_j \hat{C}_j + \sum_{j \in S_i} w_j q_{ij} \right]. \tag{3.1}$$

Since the sum of weighted delivery times is constant for a given assignment, the problem reduces to m $1||\sum w_j C_j$ problems which can be solved optimally by Smith's rule. Therefore, $Rm|q_{ij}|\sum w_j C_j$

reduces to an assignment problem which we formulate as a binary quadratic program using the binary assignment variable x_{ij} with

$$x_{ij} = \begin{cases} 1, & \text{if job } j \text{ is assigned to machine } i, \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

The problem can now be formulated as follows:

$$\min \sum_{j=1}^{n} w_j C_j \tag{3.3}$$

subject to

$$\sum_{i=1}^{m} x_{ij} = 1 \quad j = 1, \ldots, n; \tag{3.4}$$

$$C_j = \sum_{i=1}^{m} x_{ij} \left(p_{ij} + q_{ij} + \sum_{k \in S_i; k<j} p_{ik} x_{ik} \right) \quad j = 1, \ldots, n. \tag{3.5}$$

The objective (3.3) is to minimize TWC. According to constraints (3.4), each job must be assigned to exactly one machine and (3.5) calculate the job completion times.

3.3 Tabu Search Metaheuristic

Wang and Alidaee (2019) recently proposed a TS algorithm using an efficient data structure for problem $Rm||\sum w_j C_j$. This technique can be adapted to solve the problem with job-machine-dependent delivery times. Given a schedule π, the contribution of job $j \in S_i$ to the objective is

$$f_j(S_i) = w_j \left(\sum_{k \in S_i; k \leq j} p_{ik} \right) + p_{ij} \left(\sum_{k \in S_i; k>j} w_k \right) + w_j q_{ij}. \tag{3.6}$$

The contribution value can be calculated for each job on all machines. Hence, moving job j from machine i to l, which shall henceforth be called *jump*, induces an objective value change of $f_j(S_l) - f_j(S_i)$. Given a feasible solution, we can use (3.6) to calculate a contribution matrix of $n \times m$. To perform an LS, the matrix simply needs to be scanned for improving moves. Suppose i is the machine to which a job j is currently assigned and an improving jump to machine l is found such that $f_j(S_l) < f_j(S_i)$. If the move is executed, only the two columns $f_k(S_i)$ and $f_k(S_l)$ $(k = 1, \ldots, n; k \neq j)$ of the contribution matrix need to be updated, which is done in $O(2(n-1))$

time.

In our TS algorithm, starting from a random initial solution, an LS is conducted examining exclusively jump moves until a local optimum is reached. Every time a jump is carried out, a short term memory freezes the respective job for a predefined number of search steps. The aspiration criterion consists of allowing jump moves, that generate a new globally best solution. After a local optimum is found, a random number of *jump* moves are performed to facilitate diversification of the search. The random number is drawn from a predefined interval. This procedure is repeated until a maximum amount of runtime is reached.

3.4 Computational Results

In a preliminary computational study, we compare the proposed TS with a VNS (Mladenović and Hansen, 1997), that uses the same data structure but also considers *swap* moves, where two jobs across different machines are exchanged. Examining *swap* leads to a higher computational cost and is therefore only examined if *jump* has reached a local optimum. TS and VNS were coded in C++ and tested on a set of randomly generated instances with up to $m = 50$ and $n = 3200$. Tests were run on an Intel Xeon CPU E5-2697 v3 computer with 2.60 GHz. Multi-threading was not used. Processing times were generated from uniform distribution $U(1, 100)$. For delivery times, we examined six different settings. Five instances were generated for each configuration. Maximum time per run for each approach was set at $n/10$ seconds. Per instance, results were generated as the average best found objective value out of 30 independent runs. The relative percentage deviation (RPD) is used for evaluation with $RPD = (Z_{avg} - Z^*_{avg})/(Z^*_{avg}) \times 100$, where Z_{avg} is the average best objective value found by an individual algorithm and Z^*_{avg} is the best result of both algorithms for an instance. Table 3.1 reports the average RPD (ARPD) depending on instance size and delivery times.

Table 3.1: Results of TS and VNS $\left[ARPD \cdot 10^3\right]$

m	n	TS	VNS	m	n	TS	VNS	q_{ij}	TS	VNS
5	80	0.00	0.00	20	320	0.00	2.95	$U(1, 30)$	0.00	6.68
	160	0.00	0.00		640	0.00	5.88	$U(1, 100)$	0.00	6.42
	320	0.00	0.04		1280	0.00	4.33	$U(1, 200)$	0.00	5.33
10	160	0.00	0.16	50	800	0.00	16.11	$U(100, 200)$	0.00	2.30
	320	0.00	0.74		1600	0.00	11.42	$U(170, 200)$	0.00	1.44
	640	0.00	1.42		3200	0.00	6.80	$U(100, 500)$	0.00	2.75

Overall, the TS outperforms the VNS in terms of ARPD. For smaller instances with $m = 5$ and $n \leq 160$, results of TS and VNS are identical, while for the larger instances TS consistently yields better results. With respect to delivery times, it can be observed that ARPD values of VNS are higher for the first three settings. Interestingly, ARPD decreases for instances, where delivery times are larger than the processing times. This effect deserves further investigation.

The preliminary experiment shows that our TS using a highly efficient data structure for LS is promising. Nonetheless, more extensive testing is required. In addition, we point out that the data structure can also be applied to other related problems.

Chapter 4

Unrelated Parallel Machine Scheduling with Eligibility Constraints and Delivery Times to Minimize Total Weighted Tardiness*

4.1 Introduction

In terms of Pinedo (2016), parallel machine scheduling is a problem of both theoretical and practical importance. This study considers parallel machine scheduling in the context of the new manufacturing paradigm *cloud manufacturing* (CM), which arises through the shift from production-oriented to service-oriented manufacturing. In CM, distributed manufacturing resources are managed in a centralized way encapsulated in cloud services. These resources are made available to clients, who can use them according to their requirements (Xu, 2012). The goal of CM is the full sharing and circulation, high utilization, and on-demand usage of manufacturing resources and capabilities. Therefore, one of its core services is matching, searching, and scheduling for the provider of manufacturing services and consumers (Tao et al., 2011). For globally operating companies with geographically distributed production resources this is especially challenging. In such settings, if semi-finished jobs are transferred from one facility to another for further processing or finished jobs are delivered to the customer, it is important that job processing and delivery are carefully coordinated to achieve ideal overall system performance (Lee and Chen, 2001). In the context of CM, where manufacturing jobs can be scheduled centrally to a network of geographi-

*This chapter is based on the unpublished working paper *Unrelated Parallel Machine Scheduling with Eligibility Constraints and Delivery Times to Minimize Total Weighted Tardiness* cited as Maecker et al. (2021), which has been submitted to the *Computers and Operations Research* journal and is currently under second review after the first revision.

cally distributed production facilities, these may be regarded as parallel machines. In this setting, delivery and transportation aspects should be formulated and integrated in an appropriate way (Mönch and Shen, 2020). Furthermore, heterogeneity of the available production resources should be considered.

The problem at hand is inspired by the example of a provider for cloud-based manufacturing services for printed circuit boards (PCB). Customers can upload and edit their PCB designs via an online tool and place production orders. The provider has access to a large network of PCB factories with different capabilities. It can thus schedule the jobs to the resources according to their requirements. To ensure reliable lead times, the delivery aspect has to be taken into consideration for this scheduling decision.

The aforementioned problem of assigning jobs to distributed, heterogeneous manufacturing resources is addressed in this research as an unrelated parallel machine scheduling problem (PMSP), where a delivery time occurs after processing a job, which is dependent on both the selected machine and the respective job. Furthermore, there exist eligibility constraints such that the processing of a job may be restricted to a subset of the available machines. The objective considered is to minimize the total weighted tardiness (TWT) of all jobs. Since the parallel machine total tardiness problem (PMTT), which constitutes a special case of this problem, is NP-hard in the strong sense (Pfund et al., 2004), this problem is NP-hard in the strong sense, too. Hence, it is unlikely that there exists an exact algorithm for this problem with polynomially bounded time.

The remainder of this chapter is organized as follows. In Section 4.2, we present a review on related machine scheduling literature involving delivery times as well as PMSPs with machine eligibility constraints or total (weighted) tardiness minimization. Section 4.3 presents an MILP-formulation to determine optimal schedules for small-sized problem instances. In Section 4.4, we examine the problem from a theoretical perspective to derive precedence properties that can be used to improve local search (LS) procedures. Additionally, we propose a computational scheme to accelerate the evaluation of schedules in different neighborhood structures. In Section 4.5, we propose a heuristic, which is based on the Apparent Tardiness Cost (ATC) rule, as well as a VNS-algorithm. Section 4.6 conducts a computational study on a large set of randomly generated instances to investigate the effect on the VNS performance when the previous theoretical findings are integrated into the search. Furthermore, we compare the VNS with the heuristic and the Gurobi solver when applied to the MILP. The chapter concludes with Section 4.7.

4.2 Related Work

In this section, first, a comprehensive review covering related literature of machine scheduling problems involving delivery times is provided. Afterwards, noteworthy studies covering total (weighted) tardiness minimization on parallel machines are presented. Finally, relevant PMSP literature including machine eligibility constraints is summarized.

Maggu and Das (1980) first consider job transportation in a two-machine flow shop makespan problem where job-dependent transportation times occur between the processing stages and transportation capacity is unlimited. Both Potts (1980) and Hall and Shmoys (1992) study the problem of scheduling jobs on a single-machine with release dates and job-dependent delivery times to minimize the time by which all jobs are delivered and provide heuristics with worst-case analysis. The parallel machine case of this problem is considered by Carlier (1987). A worst case analysis for Jackson's rule (Jackson, 1955) for this problem is provided as well as a branch and bound (B&B) algorithm. A similar problem with identical release dates is studied by Woeginger (1994), who proposes several list scheduling heuristics with worst case analysis. For the problem considered by Carlier (1987), Gharbi and Haouari (2002) develop another B&B procedure and Gharbi and Haouari (2007) improve the performance of Jackson's rule through a preprocessing method. Furthermore, Mateo et al. (2018) study a bi-objective PMSP with unequal job release dates, delivery times, and eligibility constraints, where a penalty is induced depending on the quality of the machine processing a job. An algorithm is proposed to find an approximate Pareto front for minimization of makespan and the total penalty.

On-line scheduling approximation algorithms (AA) for several single-machine problems involving delivery times are developed by Hoogeveen and Vestjens (2000), Tian et al. (2007), Tian et al. (2008), and Yuan et al. (2009). Furthermore, on-line scheduling on parallel machines with delivery times is studied by Fang et al. (2011a), Fang et al. (2011b), Tian et al. (2011), Tian et al. (2012), and Liu and Lu (2015).

Lee and Chen (2001) define two types of scheduling problems with job delivery where the transportation capacity is limited in terms of both available number of vehicles and vehicle capacity. *Type-1* transportation considers job transportation inside a manufacturing facility between processing stages and *type-2* transportation takes place between the facility and a customer area. In both cases, jobs share a common delivery time and the objective is to minimize the makespan. A complexity analysis is presented for single-machine, parallel-machine, and flow-shop environments. Chang and Lee (2004) study *type-2* transportation for the one and two machine case with jobs

that require different amounts of space on the transportation vehicle and develop AAs. Li et al. (2005) investigate an extension of the problem by Chang and Lee (2004) for the single-machine case, where multiple customer areas exist to which jobs are delivered by a single vehicle with limited capacity and the objective is to minimize the sum of arrival times. Li et al. (2005) discuss the complexity of the general case as well as special cases and propose a dynamic programming algorithm. For the problem raised by Chang and Lee (2004), AAs are also proposed by He et al. (2006), Zhong et al. (2007), Lu and Yuan (2008a), and Liu and Lu (2011) for the single-machine case and by Su et al. (2009) for the two-machine case. Cakici et al. (2014) investigate a similar problem, where jobs are first processed on parallel machines and then delivered in batches by a single capacitated vehicle with a limited number of trips. Heuristics are proposed for minimizing the total weighted completion time. Wang and Cheng (2007) incorporate machine unavailability into the model by Chang and Lee (2004) and propose AAs for the single and parallel machine case. Lu and Yuan (2008b) propose an AA for the unbounded parallel batch machine case. Koulamas and Kyparisis (2010) and Liu et al. (2012) study a single-machine problem where jobs have *past-sequence-dependent* delivery times proportional to their waiting time before processing and present polynomial-time algorithms for multiple optimization criteria. For TWT minimization, Liu et al. (2012) develop a polynomial-time algorithm for a special case of the problem. Dong et al. (2013) present AAs for *type-2* problem on two machines and in open-shop production with one customer. Chen et al. (2015) show that the preemptive case of *type-2* problem on identical parallel machines with one customer is strongly NP-hard and propose an AA. A batch delivery scheduling problem with job release dates on a single machine is considered by Ahmadizar and Farhadi (2015), where all jobs of a batch are instantly delivered upon finishing the last job in the batch and the objective is to minimize the sum of multiple cost components including delivery costs. To solve the problem, an imperialist competitive algorithm is proposed. Pei et al. (2015) study a serial bounded batching machine problem, where the job processing time is a linear function of its starting time. After processing, batches are delivered on a single vehicle with limited capacity to a customer. Another definition of job delivery times is presented by Chen et al. (2016) who investigate a parallel machine problem where a set of delivery times are given and each delivery time needs to be assigned to an individual job.

Recently, Maecker and Shen (2020) studied an identical parallel machine total weighted tardiness problem (PMTWT) with machine-dependent delivery times. They propose two MILP formulations as well as an ATC-based heuristic, a memetic algorithm, and a VNS. Furthermore, Mönch and Shen (2020) address an identical PMSP with job-machine-dependent delivery times to minimize total weighted completion time, given that the completion time is defined as the time by which

a job reaches the customer. Several special cases are exmained that can be solved optimally in polynomial time and a greedy randomized adaptive search framework as well as a genetic algorithm are proposed for the general problem.

Most of the studies on machine scheduling problems with delivery times focus on makespan minimization, while due-date-related criteria are less frequently considered. Furthermore, the definition of delivery times is restricted to job-dependent delivery times (see e. g. Potts (1980), Carlier (1987), Woeginger (1994), Gharbi and Haouari (2002)) or common delivery times with limitations on transportation capacity (see e. g. Lee and Chen (2001), Chang and Lee (2004)). However, to measure customer satisfaction and delivery efficiency in a service-oriented environment, a due-date-related criterion such as the TWT is a more relevant performance measure.

With regards to the PMTWT, multiple solution approaches have been proposed. Alidaee and Rosa (1997) extend the Modified Due Date (MDD) rule by Baker and Bertrand (1982) for the PMTWT. Another heuristic called P-S-K is developed by Panwalkar et al. (1993) to minimize the mean tardiness on a single machine. P-S-K combines principles of the Shortest-Processing-Time and Earliest-Due-Date rules. Koulamas (1997) develops the heuristic KPM which is an extension of P-S-K for the same problem on parallel machines. Furthermore, Vepsalainen and Morton (1987) introduce the ATC rule for the single machine TWT problem, which has been applied to various other tardiness-related scheduling problems. For the PMTT problem, Biskup et al. (2008) develop several efficient heuristics. Furthermore, Lin et al. (2011) apply modifications of MDD, KPM, and ATC as well as a genetic algorithm to the unrelated PMTWT. Azizoglu and Kirca (1998), Yalaoui and Chu (2002), and Shim and Kim (2007) investigate the PMTT and develop B&B procedures. Another B&B algorithm is proposed by Liaw et al. (2003) for the unrelated PMTWT. Srinivasa Raghavan and Venkataramana (2009) use ant colony optimization (ACO) to solve the PMTWT. ACO algorithms are also implemented for the unrelated PMTWT problem by Zhou et al. (2007), Mönch (2008), and Lin et al. (2013).

If jobs may not be processed on all of the available machines but only on a subset, this poses as an operational constraint which is often referred to as machine eligibility or processing set restrictions. Centeno and Armacost (1997) study a problem with identical machine, release dates, and eligibility constraints to minimize maximum lateness. The same problem with makespan objective is investigated by Centeno and Armacost (2004). An identical PMSP with machine availability is studied by Liao and Sheen (2008) for makespan minimization and by Sheen et al. (2008) for minimization of maximum lateness. An unrelated PMSP with machine eligibility constraints is studied by Gao (2010) with the two objectives minimization of makespan and total earliness/tardiness. Identical

PMSP with release dates, machine eligibility, and sequence-dependent-setup times to minimize total weighted completion time is considered by Gokhale and Mathirajan (2012). Wang et al. (2013) study an unrelated PMSP with release dates, machine eligibility, and sequence-dependent-setup times, where the objective is the weighted sum of makespan and tardy jobs. Furthermore, various unrelated, resource-constrained PMSPs with machine eligibility are studied by Afzalirad and Rezaeian (2016), Afzalirad and Rezaeian (2017), and Afzalirad and Shafipour (2018). Su et al. (2017) address the identical PMTWT with machine eligibility constraints and propose an ATC-based heuristic with flexibility considerations.

To conclude, due-date-related optimization criteria are hardly considered in machine scheduling problems with delivery times. Yet, due dates play a major role in supply chain environments. To the best of our knowledge, the problem of TWT minimization on unrelated parallel machines with job-machine-dependent delivery times and eligibility constraints has not been addressed so far.

4.3 Problem Formulation

Given are a set of n jobs $j = 1, \ldots, n$ and m unrelated parallel machines $i = 1, \ldots, m$. Each job needs to be processed by one and only one machine from its set of eligible machines M_j without interruption while each machine can handle at most one job at a time. All jobs are available at time zero. Each job j has a weight w_j, a processing time p_{ij} associated with machine i, a delivery time q_{ij} ($p_{ij} = q_{ij} = \infty, i \notin M_j$) that occurs immediately after completing j on the respective machine, and a due date d_j. While a job is being transferred, the machine may already process the next job in line. Transportation capacity is assumed to be unlimited in terms of both vehicle availability and capacity. The objective is to minimize the TWT ($\sum w_j T_j$) of the schedule. The tardiness of a job T_j is calculated by $T_j = \max\{C_j - d_j, 0\}$, where the completion time C_j is defined as the time by which a job reaches the customer. Following the conventional three-field-notation by Graham et al. (1979), the problem can be expressed as $Rm|M_j, q_{ij}|\sum w_j T_j$, where Rm indicates unrelated parallel machines.

Next, we present an MILP, that is based on disjunctive constraints and uses a binary assignment variable y_{ij} with

$$y_{ij} = \begin{cases} 1, & \text{if job } j \text{ is processed by machine } i, \\ 0, & \text{otherwise,} \end{cases} \tag{4.1}$$

as well as a binary precedence variable z_{jk} with

$$z_{jk} = \begin{cases} 1, & \text{if job } j \text{ precedes job } k \text{ in the sequence,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

Furthermore, the variable s_j denotes the start time of job j, H is a sufficient large number, and J_i denotes the set of jobs for which machine i is eligible. The model is formulated as follows:

$$\min \sum_{j=1}^{n} w_j T_j \tag{4.3}$$

subject to

$$\sum_{i \in M_j} y_{ij} = 1, \quad j = 1, \ldots, n, \tag{4.4}$$

$$s_k \geq s_j + p_{ij} - H(3 - y_{ik} - y_{ij} - z_{jk}), \quad i = 1, \ldots, m, \quad j, k \in J_i, \quad j \neq k, \tag{4.5}$$

$$s_j \geq s_k + p_{ik} - H(2 - y_{ik} - y_{ij} + z_{jk}), \quad i = 1, \ldots, m, \quad j, k \in J_i, \quad j \neq k, \tag{4.6}$$

$$C_j \geq s_j + y_{ij}(p_{ij} + q_{ij}), \quad i = 1, \ldots, m, \quad j = 1, \ldots, n, \tag{4.7}$$

$$T_j \geq C_j - d_j, \quad j = 1, \ldots, n, \tag{4.8}$$

$$T_j, C_j \geq 0, \quad j = 1, \ldots, n, \tag{4.9}$$

$$x_{ij}, z_{jk} \in \{0, 1\}, \quad i = 1, \ldots, m, \quad j, k \in J_i. \tag{4.10}$$

The objective (4.3) is to minimize the TWT. Constraint (4.4) states that each job must be assigned to exactly one of its eligible machines. Disjunctive constraints (4.5) and (4.6) handle the precedence relationship among jobs on the machines. The job completion times and tardiness values are calculated according to inequalities (4.7) and (4.8). Constraints (4.9) and (4.10) define the variables. H is an upper bound on the job starting time and can be calculated by

$$H = \max_{i=1,\ldots,m} \left\{ \sum_{j \in J_i} p_{ij} - \min_{j \in J_i} \{p_{ij}\} \right\}. \tag{4.11}$$

4.4 Structural Properties for Local Search

4.4.1 Lemmata for Improving Moves in Local Search

For the $1||\sum w_jT_j$ problem, Kanet (2007) introduces general precedence relationships among jobs. Assume an assignment of jobs to machines for problem $Rm|M_j, q_{ij}|\sum w_jT_j$ is given. Now, sequencing jobs on each machine can be denoted as problem $1|q_j|\sum w_jT_j$, where q_j is the delivery time of job j on the respective machine. Also, for simplification, let p_j denote the processing time of j on this machine. In this subsection, we present lemmata with sufficient conditions for this problem to quickly identify improving moves in swap- and insertion-neighborhoods when delivery times are present. The lemmata are derived by extending the theorems provided by Kanet (2007) for problem $1||\sum w_jT_j$. Moreover, we provide a lemma for improving swap moves across different machines in the $Rm|M_j, q_{ij}|\sum w_jT_j$ problem.

Swap on the Same Machine

In this subsection, we present lemmata with sufficient conditions for swaps of two jobs on the same machine, that improve the TWT value of the schedule. Assume that there exists a schedule π with any two jobs j and k, such that $k \prec j$. Let W denote the start time of k and C the time by which j is completed on the machine ($C = C_j - q_j$). Furthermore, let π' denote the schedule generated by swapping j and k in the sequence. The swap of j and k is illustrated in Figure 4.1, where W is the start time of job k in the initial schedule π and C is the time by which job j is completed on the machine. Using this notation, we can now formulate the following lemmata.

Lemma 4.1. *Given a feasible schedule π for problem $1|q_j|\sum w_jT_j$ with any two jobs j and k such that $k \prec j$, and a schedule π' generated by swapping j and k with $p_j \leq p_k$, $w_j > w_k$, and*

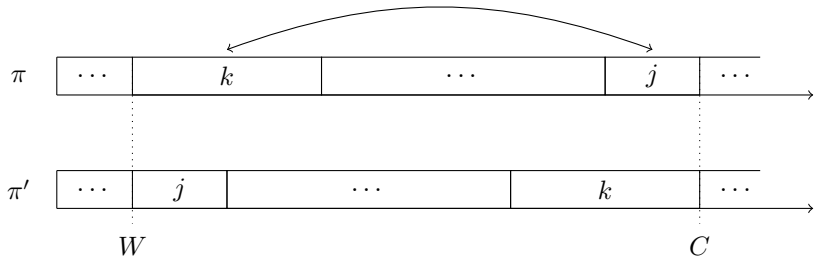

Figure 4.1: Swapping two jobs j and k on the same machine.

$d_j < C + q_j$, *if*

$$q_k - d_k \leq q_j - d_j \tag{4.12}$$

or

$$w_j(C + q_j - d_j) > w_k(C + q_k - d_k) \tag{4.13}$$

is valid, then $f(\pi') < f(\pi)$ *holds, where* $f(\cdot)$ *denotes the objective function.*

Proof: If $p_j \leq p_k$ holds, the tardiness of all jobs scheduled between j and k may not increase. Inequality $d_j < C + q_j$ implies that j is tardy in π, which is a premise for an improving move. Therefore, swapping j and k leads to an improvement of j's weighted tardiness. To ensure that the improvement of j's weighted tardiness induced by the swap is larger than the degradation of k's weighted tardiness, the following condition must hold:

$$\begin{aligned} &\min\{w_j(C + q_j - d_j), w_j(C - W - p_j)\} \\ >&\min\{w_k(C + q_k - d_k), w_k(C - W - p_k)\}. \end{aligned} \tag{4.14}$$

Conditions $p_j \leq p_k$ and $w_j > w_k$ indicate

$$w_j(C - W - p_j) > w_k(C - W - p_k). \tag{4.15}$$

Therefore, if (4.13) holds, inequality (4.14) is satisfied. Note that subject to (4.12) and $w_j > w_k$, (4.13) is valid. □

Lemma 4.2. *Given a feasible schedule* π *for problem* $1|q_j|\sum w_jT_j$ *with any two jobs* j *and* k *such that* $k \prec j$*, and a schedule* π' *generated by swapping* j *and* k *with* $p_j \leq p_k$*, and* $w_j > w_k$*, if*

$$w_j(C + q_j - d_j) > w_k(C - W - p_k) \tag{4.16}$$

is valid, then $f(\pi') < f(\pi)$ *holds.*

Proof: If $p_j \leq p_k$ holds, the tardiness of all jobs scheduled between j and k may not increase. Since $w_k(C - W - p_k) > 0$, inequality (4.16) implies that j is tardy in π. Also, $p_j \leq p_k$ and $w_j > w_k$ lead to inequality (4.15). Therefore, if (4.16) holds, (4.14) is satisfied. □

Lemma 4.3. *Given a feasible schedule* π *for problem* $1|q_j|\sum w_jT_j$ *with any two jobs* j *and* k *such that* $k \prec j$*, and a schedule* π' *generated by swapping* j *and* k *with* $p_j \leq p_k$*,* $w_j \leq w_k$*, and* $d_j < C + q_j$*, if*

$$w_j(C - W - p_j) > w_k(C + q_k - d_k) \tag{4.17}$$

and

$$w_j(C + q_j - d_j) > w_k(C + q_k - d_k) \tag{4.18}$$

are valid, then $f(\pi') < f(\pi)$ *holds.*

Proof: If $p_j \leq p_k$ holds, the tardiness of all jobs scheduled between j and k may not increase. If inequalities (4.17) and (4.18) hold, (4.14) is satisfied. Again, $d_j < C + q_j$ ensures that j is tardy in π so that swapping j and k must lead to an improvement of j's weighted tardiness. □

Lemma 4.4. *Given a feasible schedule* π *for problem* $1|q_j|\sum w_jT_j$ *with any two jobs* j *and* k *such that* $k \prec j$*, and a schedule* π' *generated by swapping* j *and* k *with* $p_j \leq p_k$*, and* $w_j \leq w_k$*, if*

$$w_j(C - W - p_j) > w_k(C - W - p_k) \tag{4.19}$$

and

$$w_j(C + q_j - d_j) > w_k(C - W - p_k) \tag{4.20}$$

are valid, then $f(\pi') < f(\pi)$ *holds.*

Proof: If $p_j \leq p_k$ holds, the tardiness of all jobs scheduled between j and k may not increase. Since $w_k(C-W-p_k) > 0$, inequality (4.20) implies that j is tardy in π. Furthermore, if inequalities (4.19) and (4.20) hold, (4.14) is satisfied. □

Insertion on the Same Machine

The following lemmata contain sufficient conditions for job insertions on the same machine that improve the TWT of the schedule. For these lemmata, we assume there exists a schedule π with any two jobs j and k, such that $k \prec j$, and a schedule π' generated by reinserting k immediately

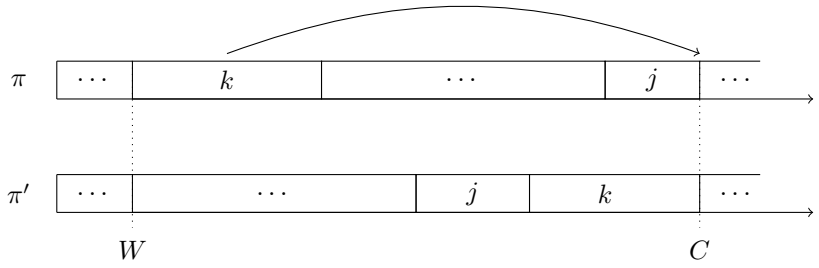

Figure 4.2: Insertion of job k immediately after job j on the same machine.

after j in the schedule. Again, W denotes the start time of k in π and C the time by which j is completed on the machine. The reinsertion of k after j is illustrated in Figure 4.2.

Lemma 4.5. *Given a feasible schedule π for problem $1|q_j|\sum w_jT_j$ with any two jobs j and k such that $k \prec j$, and a schedule π' generated by inserting k directly after j with $d_j < C + q_j$, if*

$$\min\{w_j p_k, w_j(C + q_j - d_j)\} > w_k(C + q_k - d_k) \tag{4.21}$$

is valid, then $f(\pi') < f(\pi)$ holds.

Proof: If k is inserted immediately after j, the tardiness of all jobs scheduled between k and j in π may not increase. Inequality $d_j < C + q_j$ implies that j is tardy in π. The improvement of j's weighted tardiness induced by the reinsertion of k is given by the left-handed term of (4.21), while the right-handed term must be larger than or equal to the weighted tardiness degradation of job k. □

Lemma 4.6. *Given a feasible schedule π for problem $1|q_j|\sum w_jT_j$ with any two jobs j and k such that $k \prec j$, and a schedule π' generated by inserting k directly after j, if*

$$\min\{w_j p_k, w_j(C + q_j - d_j)\} > w_k(C - W - p_k) \tag{4.22}$$

is valid, then $f(\pi') < f(\pi)$ holds.

Proof: If k is inserted immediately after j, the tardiness of all jobs scheduled between k and j in π may not increase. Since $w_k(C - W - p_k) > 0$, inequality (4.22) implies that j is tardy in π. Furthermore, the improvement of j's weighted tardiness induced by the reinsertion of k is given by the left-handed term of (4.22), while the right-handed term constitutes an upper bound on the weighted tardiness degradation of job k. □

Swap Across Two Machines

Next, we present an additional lemma for improving swaps of two jobs across different machines for problem $Rm|M_j, q_{ij}|\sum w_jT_j$. Assume there exists a schedule π with any two jobs j scheduled on machine h and job k scheduled on machine i. Let W_j denote the start time of j in π and W_k the start time of k, respectively. Furthermore, schedule π' is generated by swapping the positions of j and k in the schedule. Figure 4.3 illustrates the respective move, where W_k denotes the start time of job k in the initial schedule on machine i and W_j the start time job k on machine h, respectively. We now propose the following lemma:

Lemma 4.7. *Given a feasible schedule π for problem $Rm|M_j, q_{ij}|\sum w_jT_j$ with any two jobs j on machine h and k on machine i, and a schedule π' generated by swapping j and k with $p_{ij} \leq p_{ik}$ and $p_{hk} \leq p_{ij}$, if*

$$\begin{aligned} & w_j \max\{W_k + p_{ij} + q_{ij} - d_j, 0\} + w_k \max\{W_j + p_{hk} + q_{hk} - d_k, 0\} \\ <& w_j \max\{W_j + p_{hj} + q_{hj} - d_j, 0\} + w_k \max\{W_k + p_{ik} + q_{ik} - d_k, 0\} \end{aligned} \tag{4.23}$$

is valid, then $f(\pi') < f(\pi)$ holds.

Proof: If inequality (4.23) is true, the combined weighted tardiness of jobs j and k in π' is less than in π. The premise $p_{ij} \leq p_{ik}$ and $p_{hk} \leq p_{ij}$ ensures, that the tardiness of following jobs on h and i in π' must be less than or equal to their tardiness in π. □

We have now formulated lemmata for problem $1|q_j|\sum w_jT_j$ to identify improving moves when swapping or inserting jobs on the same machine as well as another lemma for improving moves when swapping jobs across machines for problem $Rm|M_j, q_{ij}|\sum w_jT_j$. These lemmata can be integrated in LS procedures to avoid a computationally costly evaluation of neighboring solutions.

4.4.2 Neighborhood Evaluation Acceleration

The lemmata presented in Subsection 4.4.1 can be used to quickly identify improving moves in multiple neighborhoods. However, if these lemmata do not apply, examining whether a move is beneficial or not is computationally costly. To reduce the computational burden, we design a

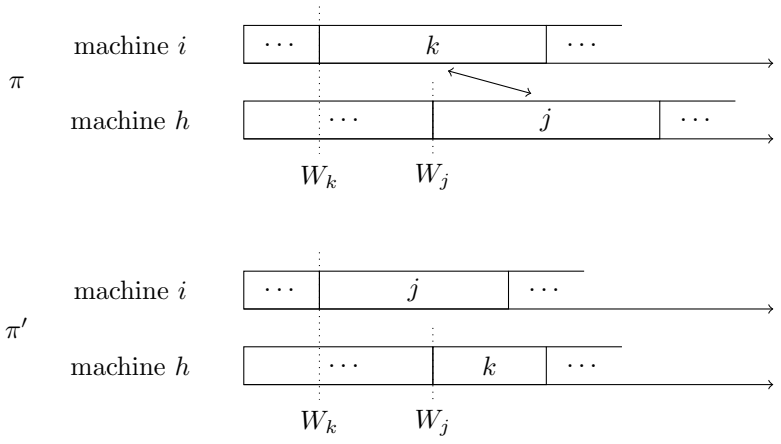

Figure 4.3: Swapping two jobs j and k across different machines i and h.

neighborhood evaluation acceleration (NEA) that exploits the fact that only a small subset of jobs is affected by neighborhood moves. In addition, we use the problem structure to efficiently recalculate the objective function value contribution of these jobs. Therefore, it is important to design dedicated neighborhood structures. For example, it is advantageous to distinguish between the swap of two jobs located on the same machine and the swap of two jobs which are located on different machines, instead of defining a neighborhood which consists of swapping any two jobs, since for each situation an individual NEA can be designed. Other examples of NEA techniques can be found in Tasgetiren et al. (2009), Xu et al. (2014) and Maecker and Shen (2020).

The basic NEA procedure is presented in Algorithm 4. First, the set K of affected jobs is determined based on the parameters of the move in a certain neighborhood structure, e. g., the positions of two jobs which are to be swapped. For all jobs in K, their new completion times C'_j induced by the move can be calculated according to neighborhood-specific rules and thus also their new tardiness values T'_j. Afterwards, the total objective function value difference Δ induced by the move is calculated as the weighted sum of differences between new and old tardiness values. The move improves the current schedule, if $\Delta < 0$. The objective value is incremented by Δ.

Next, we describe how to determine the set K and to calculate C'_j for neighborhood structures used later in the VNS scheme proposed in this research. We use $[j]$ to indicate the position of any job j. Furthermore, let $[j]_i$ denote the job in the $[j]$-th position in the sequence on machine i. Accordingly, $([j]+1)_i$ represents the job in the immediate succeeding position. Furthermore, we define n_i as the total number of jobs scheduled on machine i so that $(n_i)_i$ is the last job in the sequence on machine i. Using that notation, we formulate the following NEA rules:

- *Swapping jobs on a single machine (N_1)*: When swapping jobs on a single machine, the positions of two jobs j and k in the sequence on i are interchanged. If j is the job sequenced

Algorithm 4 Basic NEA procedure

Input: Schedule π with objective function value f, parameters for move to schedule $\pi' \in N_l(\pi)$
Determine the set K of affected jobs;
for $j \in K$ **do**
    Calculate new completion time C'_j;
    Calculate new tardiness $T'_j \leftarrow \max\{C'_j - d_j, 0\}$;
end for
$\Delta \leftarrow \sum_{j \in K} w_j \left(T'_j - T_j\right)$;
if $\Delta < 0$ **then**
    $\pi \leftarrow \pi'$;
    $f' \leftarrow f + \Delta$;
end if

first among j and k, i. e. $[j] < [k]$, we have $K = \{[j]_i, ([j]+1)_i, \ldots, [k]_i\}$. Let $\delta = p_{ij} - p_{ik}$, then

$$C'_{j'} = C_{j'} - \delta, \quad j' \in K \setminus \{j, k\}, \tag{4.24}$$

$$C'_j = C_k - q_{ik} + q_{ij}, \tag{4.25}$$

and

$$C'_k = C_j - q_{ij} - \delta + q_{ik}. \tag{4.26}$$

- *Swapping jobs on two machines (N_2)*: This move exchanges the positions in the schedule of two jobs j on machine i and k on another machine i'. In this case, $K = K_1 \cup K_2$, where $K_1 = \{[j]_i, ([j]+1)_i, \ldots, (n_i)_i\}$ and $K_2 = \{[k]_{i'}, ([k]+1)_{i'}, \ldots, (n_{i'})_{i'}\}$. Let $\delta_i = p_{ij} - p_{ik}$ and $\delta_{i'} = p_{i'k} - p_{i'j}$, then

$$C'_{j'} = C_{j'} - \delta_i, \quad j' \in K_1 \setminus j, \tag{4.27}$$

$$C'_{j'} = C_{j'} - \delta_{i'}, \quad j' \in K_2 \setminus k, \tag{4.28}$$

$$C'_j = C_k - q_{i'k} - \delta_{i'} + q_{i'j}, \tag{4.29}$$

and

$$C'_k = C_j - q_{ij} - \delta_i + q_{ik}. \tag{4.30}$$

- *Job insertion on the same machine (N_3)*: Job insertion on the same machine consists of selecting a job j on some machine i and reinserting it at another position $[k]$ in the sequence, where k is the job currently scheduled on that position. Let $\delta = p_{ij}$. If $[k] < [j]$ holds, indicating that job j is moved to the left, then $K = \{[k]_i, ([k]+1)_i, \ldots, [j]_i\}$ with

$$C'_{j'} = C_{j'} + \delta, \quad j' \in K \setminus j, \tag{4.31}$$

and

$$C'_j = C_k - q_{ik} - p_{ik} + p_{ij} + q_{ij}. \tag{4.32}$$

In the case of $[k] > [j]$, $K = \{[j]_i, ([j]+1)_i, \ldots, [k]_i\}$ with

$$C'_{j'} = C_{j'} - \delta, \quad j' \in K \setminus j, \tag{4.33}$$

and

$$C'_j = C_k - q_{ik} + q_{ij}. \tag{4.34}$$

- *Job insertion on other machine (N_4)*: Job insertion on another machine consists of selecting a job j from the sequence on machine i and reinserting it at some position $[k]$ in the sequence on another machine i', where k is the job currently scheduled on that position on i'. In this situation, $K = K_1 \cup K_2$, where $K_1 = \{[j]_i, ([j]+1)_i, \ldots, (n_i)_i\}$ and $K_2 = \{[k]_{i'}, ([k]+1)_{i'}, \ldots, (n_{i'})_{i'}\}$. Let $\delta_i = p_{ij}$ and $\delta_{i'} = p_{i'j}$, then

$$C'_{j'} = C_{j'} - \delta, \quad j' \in K_1 \setminus j, \tag{4.35}$$

$$C'_{j'} = C_{j'} + \delta, \quad j' \in K_2, \tag{4.36}$$

and

$$C'_j = C_k - q_{i'k} - p_{i'k} + p_{i'j} + q_{i'j}. \tag{4.37}$$

There are two special cases to consider. If no jobs are scheduled on machine i', then

$$C'_j = p_{i'j} + q_{i'j}. \tag{4.38}$$

Furthermore, if j is inserted at the end of the schedule on i' and $l = (n_{i'})_{i'}$, then

$$C'_j = C_l - q_{i'l} + p_{i'j} + q_{i'j}. \tag{4.39}$$

4.5 Heuristic and Metaheuristic Approaches

4.5.1 ATC-D Heuristic

An ATC implementation for problem $Rm||\sum w_jT_j$ is proposed by Lin et al. (2011) and by Maecker and Shen (2020) for problem $Pm|q_i|\sum w_jT_j$. For the problem at hand, machine eligibility and job-machine-dependent delivery times need to be incorporated. We present our implementation called ATC-D for problem $Rm|M_j, q_{ij}|\sum w_jT_j$ in Algorithm 5, where D refers to delivery.

In each iteration, first, the fastest machine is determined among those eligible for the remaining unscheduled jobs. Afterwards, for all unscheduled jobs assignable to this machine, the priority value is calculated based on its current workload. Among these jobs, the one with largest priority value is selected to be scheduled next. If there exist machines eligible for this job, on which it can be processed without being tardy, we choose the one with smallest processing time, since, in contrast to the delivery time, the processing time of this job also affects the completion times of successive

jobs. If no such machine exists, a greedy rule is applied that schedules the job to the eligible machine with earliest completion time. Finally, the machine workload is updated accordingly. The described procedure is repeated until all jobs are scheduled. In the second phase, an adjacent pairwise exchange procedure is applied to each machine to improve the given schedule. In this step, all possible swaps of adjacent jobs in the sequences of the machines are systematically examined until no further improvement by an adjacent swap is possible. We refer to Algorithm 7 in the appendix for a detailed description of the adjacent pairwise exchange. The ATC-D heuristic serves as a benchmark scheme for the VNS procedure proposed in the next subsection.

4.5.2 Variable Neighborhood Search

VNS is an LS-based metaheuristic that searches changing neighborhood structures for improving schedules to escape from local optima (Mladenović and Hansen, 1997). For a recent overview of basic VNS schemes and variants, the reader is referred to Hansen et al. (2017). VNS has been successfully applied to multiple combinatorial optimization problems (Hansen et al., 2010), including various PMSPs (see e. g. De Paula et al. (2007), Driessel and Mönch (2011), or Cheng et al. (2012)). Furthermore, VNS is used in multiple hybrid approaches for parallel machine problems (see e. g. Anghinolfi and Paolucci (2007), Chen and Chen (2009), Behnamian et al. (2009), or Chen et al. (2013)). In this subsection, we first present our VNS scheme for problem $Rm|M_j, q_{ij}|\sum w_jT_j$.

Algorithm 5 ATC-D heuristic for $Rm|M_j, q_{ij}|\sum w_jT_j$

Initialize: Workloads $t_i = 0 \quad (i = 1, \ldots, m)$, set of unscheduled jobs $U = \{1, \ldots, n\}$;
Calculate $\bar{p}_i = \sum_{j=1}^{n} p_{ij}/n; i = 1, \ldots, m;$
repeat
  Determine fastest eligible machine $i^* = \arg \min_{\substack{i=1,\ldots,m;\\ J_i \cap U \neq \emptyset}} \{t_i\}$;
  for all $j \in U \cap J_{i^*}$ **do**
    Calculate priority value: $I_{i^*j} = \frac{w_j}{p_{i^*j}} \exp\left(-\frac{\max\{d_j - p_{i^*j} - q_{i^*j} - t_{i^*}, 0\}}{k\bar{p}_{i^*}}\right)$;
  end for
  Determine next job $j^* = \arg \max_{j \in U \cap J_{i^*}} \{I_{i^*j}\}$;
  Schedule j^* on machine $i^{**} = \arg \min_{i \in M_j} \{p_{ij^*} | t_i + p_{ij^*} + q_{ij^*} \leq d_{j^*}\}$;
  if i^{**} does not exist **then**
    Schedule j^* on machine $i^{**} = \arg \min_{i \in M_j} \{t_i + p_{ij^*} + q_{ij^*}\}$;
  end if
  Update $t_{i^{**}} = t_{i^{**}} + p_{i^{**}j^*}$ and $U = U \setminus i^**$;
until $U = \emptyset$
Apply adjacent pairwise exchange procedure to improve given schedule;

The basic structure of the VNS scheme is given in Algorithm 6. Note that the proposed VNS procedure differs from the conventional basic VNS structure. After each LS phase, the resulting locally optimal schedule π'' is kept, even if it does not improve the incumbent schedule π. This design facilitates diversification during the search process in order to escape unpromising regions of the search space more quickly. In the following, we will describe the components of our VNS. In our VNS scheme, a schedule π is encoded by m job sequences π_i $(i = 1, \ldots, m)$. The initial schedule π_0 is generated randomly. Note that in preliminary experiments we compared multiple initialization approaches such as ATC-D and randomized workload balancing, but could not find statistically significant differences between the individual approaches. To evaluate statistical significance, the Wilcoxon signed-rank test (Wilcoxon, 1945) with a significance level of 0.05 was used for all experiments reported in this research.

Let S denote the set of all feasible schedules for the problem at hand and l_{max} the maximum number of neighborhood structures $N_l(\pi) \subseteq S$ $(l = 1, \ldots, l_{max})$. A neighborhood is defined by how an incumbent schedule is altered to generate a new feasible schedule. In our VNS, we use neighborhood structures according to the scheme below:

- N_1: Reinsertion of a job on the same machine
- N_2: Reinsertion of a job on a different machine
- N_3: Swap two jobs on the same machine

Algorithm 6 Basic VNS structure (II)

Input Neighborhood structures N_l $(l = 1, \ldots, l_{max})$, initial schedule π_0;
$\pi = \pi_0$;
$f_{best} = f(\pi)$;
repeat
    $l = 1$;
    repeat
        Shaking: Choose randomly a schedule π' from $N_l(\pi)$;
        Local Search: Find local minimum π'' in $N_l(\pi')$;
        if $f(\pi'') < f_{best}$ **then**
            $f_{best} = f(\pi'')$;
            $l = 1$;
        else
            $l = l + 1$;
        end if
        $\pi := \pi''$;
    until $l > l_{max}$
until Termination criterion
Return f_{best};

- N_4: Swap two jobs across two different machines
- N_5: Perform k consecutive reinsertions of jobs at other positions in the schedule
- N_6: Perform k consectuive swaps of job pairs in the schedule

Note that N_5 and N_6 include reinsertion or swap both on the same machine as well as across different machines. Also, the total number of neighborhood structures can be increased arbitrarily if N_5 and N_6 are repeatedly revisited with an incremented value of k in each iteration. The sequence in which different neighborhood structures are visited is important for the success of VNS. The proposed sequence was determined based on preliminary experiments.

At the start of each iteration, a random schedule π' is chosen from the current neighborhood $N_l(\pi)$ of the incumbent schedule π. This step is called *Shaking* and allows the search to escape from local optima. Afterwards, an LS is performed on π' until the local optimum π'' is reached. In the LS phase, all possible moves within the current neighborhood $N_l(\pi)$ are systematically examined. Note that for N_5 and N_6, the number of moves to be examined is limited, since the size of these neighborhood structures grows considerably based on k. Furthermore, a *first improvement* strategy is employed to reduce computational effort. In this strategy, the first improving solution found is accepted as new incumbent solution. We choose this strategy instead of the *best improvement* approach, where the entire neighborhood is evaluated before selecting the best improving neighbor, since the neighborhood sizes grow exponentially with the problem size.

The LS returns the local optimum π'', which then becomes the incumbent solution π of the next iteration. f_{best} is used to record the best found objective function value. Each time a new best solution is found, the search continues with neighborhood structure N_1 ($l = 1$) in the next iteration. Otherwise, the search switches to the next neighborhood structure in line. If all neighborhood structures have been visited, the search returns to the first neighborhood structure. This procedure is repeated until a predetermined termination criterion is satisfied, such as a computing time limit or a maximum number of iterations.

Next, the NEA and lemmata proposed in Section 4.4 shall be integrated into the VNS scheme to improve its performance. The respective computational experiments are presented in the following section.

4.6 Computational Results

4.6.1 Design of Experiments

All experiments in this chapter are conducted on an Intel Xeon CPU E5-2697 v3 computer with 2.60 GHz and 128 GB RAM. Algorithms are coded in the C++ programming language and compiled by the g++-compiler. Furthermore, we use no multi-threading, except for the MILP experiments. In our experiments, we use three different instance sets, whose parameter settings are shown in Table 4.1. The different instance configurations are randomly generated. To generate due dates, a modified version of the scheme proposed by Potts and Van Wassenhove (1982) is used:

$$d_j \sim U\left(\bar{p}\left(1 - T - \frac{R}{2}\right) + q^{min}, \bar{p}\left(1 - T + \frac{R}{2}\right) + q^{min}\right), \tag{4.40}$$

where T is the *tardiness factor*, R the *relative range of due dates*,

$$\bar{p} = \frac{1}{m}\sum_{j=1}^{n} \min_{i \in M_j}\{p_{ij}\}, \quad \text{and} \quad q^{min} = \min_{\substack{j=1,\ldots,n \\ i \in M_j}}\{q_{ij}\}. \tag{4.41}$$

Furthermore, in our experiments, we use the relative percentage deviation RPD as a measure to compare different solution approaches. We calculate the RPD by

$$RPD = \frac{Z(A) - Z^*}{Z^*} \times 100, \tag{4.42}$$

where $Z(A)$ is the best objective value found by an individual approach A, and Z^* is the best result among all approaches tested for an instance.

4.6.2 Integration of NEA and Lemmata

Next, we present results of several experiments to examine the effect on the VNS performance, when the NEA as well as the lemmata from Section 4.4 are integrated into the search. In these experiments, we use four different variants of the VNS scheme:

1. VNS^0: The standard variant of the VNS scheme that uses neither NEA nor lemmata. Neighborhood solutions are evaluated by recalculating the objective value contribution of jobs scheduled to machines affected by the respective move.
2. VNS^L: In this variant, the lemmata from Section 4.4.1 are used exclusively to evaluate

neighborhood solutions.

3. VNS^N: In this variant, the NEA proposed in Section 4.4.2 is used exclusively to evaluate neighborhood solutions.

4. VNS^{NL}: In this variant, the lemmata and NEA are combined to evaluate neighborhood solutions. For a certain portion of all iterations, only the lemmata are used to find improving schedules. In all remaining iterations, the algorithm switches to NEA. The parameter $\gamma \in [0, 1]$ controls the fraction of iterations, where lemmata are used.

Note that these variants consider only the neighborhood structures N_1-N_4 during the search phase. The effect of including N_5 and N_6 shall be examined separately. Initially, we conducted preliminary experiments with a computing time limit termination criterion and observed that VNS^N yields significantly better RPD results than VNS^0 due to a larger number of schedules being evaluated within the same time. VNS^L, on the other hand, did not match the solution quality of the other approaches, since the lemmata can identify only a fraction of improving moves. Yet, in the VNS^L variant, neighborhood solutions were evaluated considerably faster by the lemmata. These observations motivated us to develop the integrated VNS strategy denoted by VNS^{NL} with partial use of the lemmata. Next, we show the detailed results of our experiments for this variant

Table 4.1: Test instance configurations (II)

	Instance set		
Factor	I	II	III
Machines m	$\{2, 4\}$	$\{5, 10, 20\}$	$\{2, 4, 8\}$
Job rate α ($n = \alpha \times m$)	$\{5, 7.5\}$	$\{4, 8, 16\}$	$\{10, 20, 40\}$
Eligibility probability P_{elig}	$\{0.5, 0.7, 0.9\}$		
Processing times	$p_j \sim U(1, 100)$		
Weights	$w_j \sim U(1, 10)$		
Due date setting	$T \in \{0.4, 0.8\}$ $R \in \{0.4, 0.8\}$		
Delivery times	$q_{ij} \sim U(1, 30)$ $q_h \sim U(1, 100)$ $q_h \sim U(101, 200)$ $q_h \sim U(170, 200)$		
Independent replications	10		
Total # of instances	1920	4320	4320

of the VNS scheme.

First, we present an experiment to determine an appropriate value for the γ parameter that controls the fraction of iterations in which the lemmata are used for evaluation instead of NEA. We test $\gamma \in \{0, 0.01, 0.1, 0.2, 0.33\}$ and use as termination criterion a maximum number of iterations $iter_{max} = 1000$, after which the algorithm is terminated. We record not just the RPD, but also the computing time required to complete the specified number of iterations. For the experiments, we use instance Sets II and III as described in Table 4.1. VNS^{NL} is performed five times for each value of γ on each of the 8640 instances. Out of the five solutions generated, we use the one with best objective function value to calculate the RPD. The computing time is calculated as the average over all runs. The RPD and computing time results based on γ are summarized in Figure 4.4. A more detailed view of the results based on the instance size can be found in Tables 4.6 and 4.7 in the appendix.

Overall, the RPD increases if γ is increased. On the other hand, the computing time decreases. More specifically, we observe that for instances with less machines, the computing time reduction is more apparent if γ increases. Note that the difference of RPD between $\gamma = 0$ and $\gamma = 0.01$ is not statistically significant, while the difference in computing time is. This result suggests integrating the lemmata into the VNS since it reduces computational effort. However, it needs to be carefully balanced with NEA.

Next, we compare VNS^{NL} with the original VNS^0, when both variants have the time limit termination criterion $t_{max} = \beta n$ seconds with $\beta \in \{0.01, 0.05, 0.1, 0.2\}$. Figure 4.5 displays the RPD based on β and Figure 4.6 the number of iterations executed within the specified time limit.

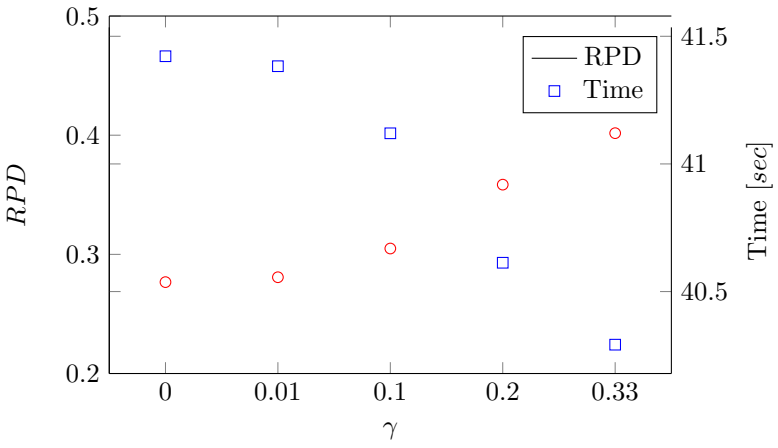

Figure 4.4: RPD and mean time [sec] of VNS^{NL} based on γ.

With respect to the results depicted in Figure 4.5, we observe that the RPD of VNS^{NL} and VNS^0 are very similar, if the time limit is increased. Furthermore, as can be seen in Figure 4.6, VNS^{NL} is able to execute a considerably larger amount of iterations within the time limit due to the speed-up of NEA and the lemmata.

4.6.3 MILP Experiments

We test the MILP on instance Set I, which contains small-sized problems with up to $n = 30$ jobs. The model was solved using the Gurobi v8.1 solver (Gurobi Optimization, 2021). The time limit per instance was set to 1800 seconds with a maximum of four threads. Results of the tests are shown based on the different instance parameters in Tables 4.2 and 4.3. Column *Opt* shows the

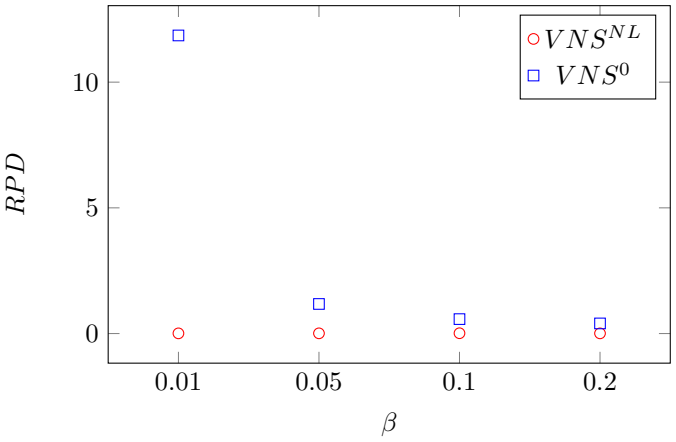

Figure 4.5: RPD of VNS^{NL} and VNS^0 based on β

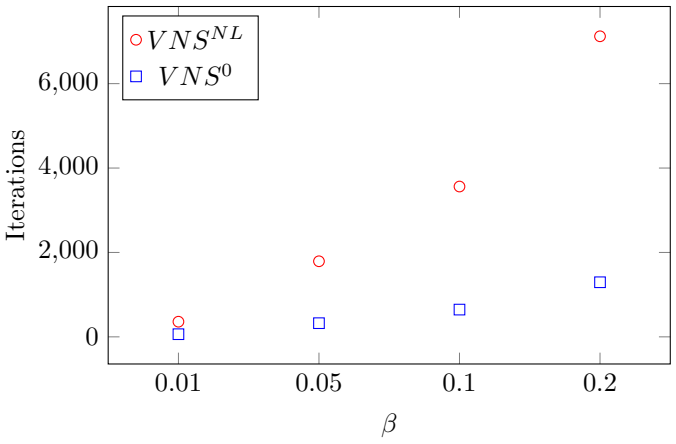

Figure 4.6: Mean no. of iterations of VNS^{NL} and VNS^0 based on β

portion of instances, for which an optimal solution was found and verified within the time limit, *Time* shows the average computing time, *Gap* shows the average gap of the best found solution, *#Var* and *#Constr* show the number of decision-variables and constraints of the MILP for the respective problem size.

Given the results in Table 4.2, we observe that even for small problems not all instances can be solved optimally. While for $n = 20$ we are able to determine an optimal solution for 55% of the instances with an average MIP gap of 7%, the larger instances with $n = 30$ already are not optimally solvable in our tests. Note that the MIP gap is an upper bound on the deviation of the best found objective function value from the optimal value.

With respect to the due dates, as shown in Table 4.3, both tardiness factor resp. tightness and range of due dates appear to affect the portion of instances solved optimally. The fraction of instances solved optimally is larger for the setting with tight due dates than for loose due dates. Also, the instances where due dates are drawn from a large range have a higher fraction of optimally solved instances compared to the instances with a small due date range.

With regard to the eligibility probability, we observe that a smaller eligibility rate leads to a larger fraction of instances solved optimally. Furthermore, the results show no considerable influence of the delivery time setting.

Table 4.2: Gurobi results based on problem size

m	n	Opt	Time	Gap	#Var	#Constr
2	10	1.00	0.78	0.00	150	400
	15	0.88	333.66	0.03	300	900
4	20	0.55	1024.92	0.07	540	3160
	30	0.00	1800.00	0.51	1110	7140

Table 4.3: Gurobi results based on due date setting, eligibility probability, and delivery time setting

T	R	Opt	Time	Gap	P_{elig}	Opt	Time	Gap	q_{ij}	Opt	Time	Gap
0.4	0.4	0.64	725.5	0.21	0.5	0.71	589.7	0.11	1-30	0.61	768.1	0.18
	0.8	0.68	644.3	0.14	0.7	0.60	814.1	0.16	1-100	0.60	800.5	0.14
0.8	0.4	0.54	918.3	0.14	0.9	0.51	965.7	0.18	101-200	0.59	820.2	0.13
	0.8	0.58	871.2	0.13					170-200	0.63	770.6	0.16

4.6.4 Extension of VNS Neighborhood Structures

We proceed with an experiment to investigate, whether the performance of VNS^{NL} can be improved by increasing the number of neighborhood structures visited during the search. To be precise, we compare VNS^{NL} with the variant VNS_2^{NL}, where also neighborhood structures N_5 and N_6 with $k = 2$ are visited so that $l_{max} = 6$, and the variant VNS_4^{NL}, where N_5 and N_6 are included with $k \in \{2, 3, 4\}$ so that $l_{max} = 10$, respectively. The neighborhoods are visited in an order such that the number of consecutive moves k is step-wise increased and insertion neighborhoods are examined before swap neighborhoods at each level of k. We test the three variants with fixed time limits $t_{max} \in \{1, 2, 5, 10, 30, 60\}$ seconds to reflect a practical setting. These tests are run on instance Set II. The RPD is calculated based on the best result out of five independent runs for each instance. Note that the lemmata and NEA cannot be applied directly to evaluate solutions in neighborhood structures N_5 and N_6. Hence, the motivation of this experiment is to test, whether an intensified, efficient search in the basic neighborhood structures N_1-N_4 is preferable to a diversified, computationally more costly search or vice versa. Figure 4.7 shows the RPD results of the three variants for the different time limits.

As the results show, VNS^{NL} with no extended neighborhoods yields the smallest RPD for all tested time limits. Furthermore, we observe that overall the RPD increases, if the maximum number of consecutive moves becomes larger. We therefore conjecture that the continuous efficient search in the relatively small neighborhoods is preferable to the extended neighborhood functions.

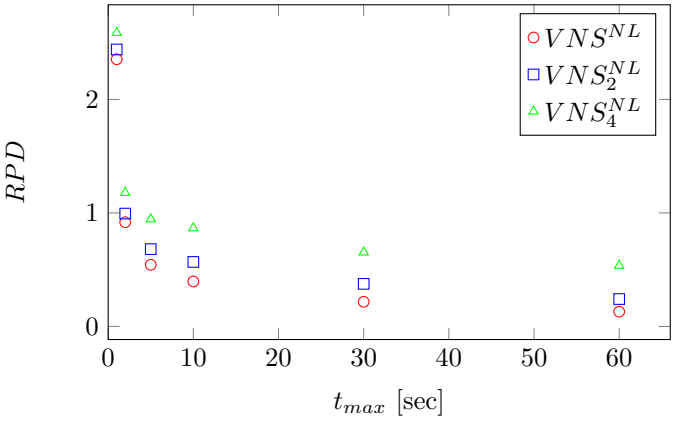

Figure 4.7: RPD of VNS^{NL}, VNS_2^{NL}, and VNS_4^{NL} based on t_{max} [sec]

4.6.5 Comparison of Gurobi, VNS, and ATC-D

Next, we compare the solution approaches VNS^{NL}, ATC-D, and the Gurobi solver when applied to the MILP. As previous results show, the MILP is restricted to small instances. Therefore, we first compare the three approaches on Set I with respect to RPD and computing time. The termination criterion of VNS^{NL} is set to $t_{max} = 60$ seconds and $\gamma = 0.1$. ATC-D is performed ten times with look-ahead parameter $k \in \{0.5, 1.0, \ldots, 5.0\}$ and the best solution is kept. The computing time reported for ATC-D is the cumulated time of all ten runs. The results are shown in Table 4.4.

For the small-sized instances, VNS^{NL} yields overall best RPD. The fact that the solutions determined by VNS^{NL} consistently match all optimal solutions provided by the solver, indicates that the VNS scheme is implemented correctly. Furthermore, VNS^{NL} found also for all other instances the best solution among all approaches. For $m = 4$ and $n = 30$, the solver could not find and verify optimal solutions. Nonetheless, the objective value of the solutions found deviates on average about 1.62 % from the solutions generated by VNS^{NL}, yet, at the expense of large computing times. On the one hand, the RPD of ATC-D is considerably larger than the RPD of Gurobi and VNS^{NL}. On the other hand, ATC-D requires significantly less computing time.

Finally, we compare the performance of ATC-D with VNS^{NL} on instance Set II. The time limit for these tests was set to $t_{max} = 60$ seconds. Since VNS^{NL} consistently found better or equal solutions compared to ATC-D, in Table 4.5, we only report RPD and computing time of the ATC-D.

It is noteworthy that ATC-D requires the modest computing times. However, the RPD is not competitive. Interestingly, the α-parameter appears to strongly affect the RPD, while the overall numbers of jobs and machines do not.

Table 4.4: Results of Gurobi, VNS^{NL}, and ATC-D based on instance size

		RPD			Time [sec]	
m	α	Gurobi	VNS^{NL}	ATC-D	Gurobi	ATC-D
2	5	0.00	0.00	14.07	0.7765	0.0004
	7.5	0.00	0.00	26.23	333.6603	0.0041
4	5	0.02	0.00	19.82	1024.9155	0.0011
	7.5	1.62	0.00	41.46	1800.0000	0.0022
	Mean	0.41	0.00	25.40	789.8421	0.0019

4.7 Summary and Research Perspectives

In this research, we considered an NP-hard unrelated parallel machine scheduling problem with job-machine-dependent delivery times, eligibility constraints, and the objective of TWT minimization. For this problem, an MILP was proposed. Furthermore, we formulated several properties to quickly evaluate neighborhood solutions in LS procedures. The properties were incorporated in a VNS algorithm and a computational study was conducted to investigate the effect of these properties on the algorithm performance. The study showed, that the computational effort can be reduced significantly through the use of structural properties. Furthermore, the ATC-D heuristic was proposed. In our experiments, we compared the three approaches. While the applicability of the MILP is strongly limited by the instance size, it can be used to determine optimal schedules for small-sized instances. Large-sized instances can be solved efficiently by the VNS and the ATC-D. Even though the ATC-D is outperformed by the VNS in terms of solution quality, it can be used to find acceptable solutions with significantly smaller computational effort.

For future research, we suggest that the incorporation of the theoretical aspects into solution approaches is also tested for other problems, since we expect the properties to be easily transferable to special cases of the problem such as $Pm||\sum w_jT_j$. Also, we believe it is worthy to investigate whether they can be adapted for more general problem settings, e. g. if sequence-dependent-setup-times are included into the problem formulation. With respect to the proposed VNS scheme, it is necessary to compare it with other solution approaches that can solve large-sized instances to validate its performance. Furthermore, the practical relevance of the problem could be further increased by imposing additional constraints on the delivery aspect such as limited availability of

Table 4.5: Results of ATC-D based on instance size

m	n	RPD	Time $[sec]$
5	20	13.255	0.001
	40	42.773	0.003
	80	112.108	0.014
10	40	12.883	0.004
	80	40.025	0.016
	160	123.906	0.072
20	80	13.438	0.021
	160	33.951	0.089
	320	102.396	0.409
Mean		54.971	0.070

transportation vehicles and vehicle capacity.

4.A Appendix

Table 4.6 shows a detailed view of the RPD results summarized in Figure 4.4 based on the problem size and γ.

Table 4.6: RPD of VNS^{NL} based on instance size and γ

		γ				
m	n	0	0.01	0.1	0.2	0.33
2	20	0.001	0.000	0.000	0.000	0.000
	40	0.065	0.010	0.002	0.129	0.029
	80	0.175	0.084	0.131	0.153	0.227
4	40	0.041	0.038	0.056	0.042	0.045
	80	0.357	0.306	0.375	0.351	0.497
	160	0.975	1.120	1.035	1.444	1.632
8	80	0.178	0.153	0.184	0.207	0.199
	160	0.615	0.598	0.644	0.766	0.811
	320	0.925	1.083	1.220	1.446	1.640
5	20	0.015	0.010	0.010	0.005	0.012
	40	0.024	0.025	0.041	0.031	0.036
	80	0.273	0.321	0.281	0.357	0.377
10	40	0.071	0.087	0.095	0.104	0.103
	80	0.137	0.119	0.148	0.150	0.188
	160	0.460	0.475	0.570	0.523	0.617
20	80	0.225	0.212	0.232	0.266	0.283
	160	0.138	0.120	0.150	0.163	0.179
	320	0.305	0.292	0.316	0.317	0.353
	Mean	0.277	0.281	0.305	0.358	0.402

Table 4.7 shows a detailed view of the computing time results summarized in Figure 4.4 based on the problem size and γ.

Table 4.7: Computing time [*sec*] of VNS^{NL} based on instance size and γ

		γ				
m	n	0	0.01	0.1	0.2	0.33
2	20	0.184	0.182	0.178	0.179	0.173
	40	1.101	1.101	1.082	1.061	1.032
	80	8.731	8.708	8.585	8.427	8.249
4	40	0.986	0.985	0.975	0.963	0.947
	80	7.153	7.138	7.073	6.996	6.906
	160	59.786	59.697	59.292	58.749	58.376
8	80	5.475	5.468	5.419	5.365	5.303
	160	41.520	41.490	41.238	40.864	40.457
	320	354.687	354.672	352.497	350.040	348.364
5	20	0.172	0.172	0.169	0.175	0.170
	40	0.943	0.940	0.929	0.914	0.901
	80	6.517	6.505	6.456	6.318	6.235
10	40	0.797	0.795	0.790	0.777	0.763
	80	5.010	5.002	4.959	4.861	4.796
	160	36.590	36.515	36.270	35.538	35.111
20	80	3.884	3.880	3.847	3.774	3.724
	160	25.074	25.028	24.821	24.338	24.052
	320	186.975	186.613	185.579	181.696	179.698
	Mean	41.421	41.383	41.120	40.613	40.292

Algorithm 7 shows the pseudo-code of the adjacent pairwise exchange procedure which is used to improve the solution constructed in the first phase of the ATC-D heuristic.

Algorithm 7 Adjacent pairwise exchange procedure

Input: Feasible schedule π for problem $Rm|M_j, q_{ij}|\sum w_j T_j$.
repeat
  $found\_improvement \leftarrow false$;
  for $j \in \pi$ **do**
    if j is not the last job on its machine **then**
      $\pi' \leftarrow$ swap j and its immediate successor in π;
      if $f(\pi') < f(\pi)$ **then**
        $\pi \leftarrow \pi'$;
        $found\_improvement \leftarrow true$;
      end if
    end if
  end for
until $found\_improvement == false$
Return: π;

Chapter 5

Energy Efficient Scheduling for a Fixed Sequence in Flexible Job-Shop Manufacturing Systems*

5.1 Introduction

With sustainability becoming a key objective for manufacturing systems and the increase of energy prices, energy costs can no longer be ignored in production planning and scheduling problems. This is particularly important for companies using time-of-use electricity tariffs (TOU), which can exploit the flexibility in their schedules to reduce their total energy cost. In this research, we are considering, within the context of a flexible job-shop scheduling system and for a fixed sequence of operations on the machines, the flexibility offered by allowing the makespan to be completed before a predefined time limit. By allowing operations to be started later, and not always as early as possible as it is most often considered in the scheduling literature, it is possible to benefit from lower energy prices while maintaining a high level of productivity. More precisely, as shown in our numerical results, a maximum allowed makespan that is only slightly larger than or even equal to the minimum makespan may lead to beneficial reductions of the total energy cost.

It should be pointed out that the minimization of the total energy cost is a non-regular criterion. As for instance studied in Mati et al. (2011), a regular criterion is an increasing function of the completion times of the jobs, i.e., for a regular criterion, there is always an optimal schedule in which jobs start (and are completed) as early as possible. Hence, when minimizing the total energy cost, an optimal schedule is not straightforward to determine for a given sequence of operations on

*This chapter is based on the unpublished working paper *Energy Efficient Scheduling for a Fixed Sequence in Flexible Job-Shop Manufacturing Systems* cited as Shen et al. (2021), which has been submitted to the *European Journal of Operational Research* and is currently in second revision.

the machines. This is why we need to propose new solution approaches.

Section 5.2 provides a literature review on the minimization of energy costs in scheduling problems, that shows the originality of our problem. The latter is formalized in Section 5.3 as a Mixed Integer Linear Program (MILP). Several properties are derived in Section 5.4 to support the analysis of the complexity of the problem. Some of the properties are used to propose two heuristics, a shifting procedure and a dynamic-programming-based procedure, in Sections 5.5. Computational experiments are conducted in Section 5.6 to compare the results of the IBM ILOG CPLEX solver when applied to the MILP and the two heuristic approaches, which show the interest of the shifting procedure. The computational experiments also illustrate that interesting cost savings can be obtained for relatively small values of the maximum allowed makespan and for different sequences of the same flexible job-shop problem. Finally, some conclusions are drawn together with some perspectives in Section 5.8.

5.2 Literature Review

The integration of energy aspects into classical machine scheduling problems gains increasing attention over the past years. Energy-aware scheduling (EAS) has been studied in multiple production environments including single machines, parallel machines, flow-shops, and job-shops. Comprehensive surveys of the EAS literature can be found in Gahm et al. (2016), Biel and Glock (2016) and Akbar and Irohara (2018). Gahm et al. (2016) develop a research framework for energy-efficient scheduling, according to which the existing literature is classified. Biel and Glock (2016) provide a systematic literature review of the state-of-the-literature for short- and mid-term energy-efficient production planning. Akbar and Irohara (2018) present a framework to conduct literature review for sustainable manufacturing literature.

Next, we review the relevant machine scheduling literature covering the minimization of the total energy cost (TEC) under consideration of time-of-use electricity tariffs (TOU). For a single machine, Shrouf et al. (2014) investigate the minimization of TEC in the presence of TOU prices, where the operator can choose between different machine states affecting the energy consumption rate of the machine. A mathematical formulation and a GA approach are presented. Che et al. (2016) consider a problem with job-dependent energy consumption rates to minimize TEC, and propose an MILP as well as a heuristic. Another MILP and a heuristic are proposed by Zhang et al. (2018) for the same problem. Rubaiee et al. (2018) investigate a multi-objective problem with constant energy consumption rates to minimize the total tardiness and TEC. An MILP, three

different GA approaches and a heuristic algorithm are proposed to solve the problem.

TEC minimization under TOU has also been considered for parallel machines. Moon et al. (2013) investigate the problem of minimizing the sum of the makespan and TEC on unrelated parallel machines with machine-dependent energy consumption rates (MDCR) and propose an MILP and a GA. A similar problem is studied by Ding et al. (2016) in the presence of job-machine-dependent energy consumption rates (JMDCR), and the objective is to minimize TEC, while there is a bound imposed on the makespan. They propose an MILP and a column generation heuristic. Che et al. (2017) develop another MILP and a two-stage heuristic for this problem. Wang et al. (2018) consider a multi-objective problem to minimize TEC and the makespan on identical parallel machines with MDCR, and implement the augmented ϵ-constraint method, a two-stage heuristic, and a non-dominated sorting algorithm II (NSGA-II). Zhou et al. (2018) study the same problem with uniform batch machines, and Zeng et al. (2018) with uniform machines to minimize TEC and the number of used machines.

TEC minimization can also be found in the context of (hybrid) flow shop scheduling in the literature. Luo et al. (2013) study a multi-objective hybrid flow shop scheduling problem with MDCR and uniform parallel machines at each stage, where the objectives are TEC and the makespan, and compare different metaheuristic approaches. Zhang et al. (2014a) propose an MIP and a hybrid Ant Colony Optimization (ACO) algorithm for a multi-objective flow shop scheduling problem with JMDCR, where the objectives are minimization of TEC and the CO2 emissions. Zheng et al. (2020) propose an MIP and a multi-objective ACO algorithm to minimize TEC and the makespan in a two-stage blocking permutation flow shop scheduling problem with variable-speed machines. Recently, Schulz et al. (2020) study a multi-objective hybrid flow shop scheduling problem with variable discrete production speed levels to minimize both the total tardiness and TEC.

The literature on TEC minimization in (flexible) job shop environments remains very limited and is rather case specific. Moon and Park (2014) develop models for two scenarios of the Flexible Job Shop Scheduling Problem (FJSP) with time-machine-dependent electricity costs, where the objective is to minimize the sum of the production overtime cost and electricity cost. The second model additionally incorporates distributed energy resources and energy storage. Liu et al. (2015) study a classical job shop problem inspired by a real-world production setting, where regular stops of energy supply require the company to use generators resulting in higher CO2 emissions and electricity costs. A mathematical model is developed to minimize the three objectives: Total weighted tardiness, total energy consumption, and TEC. Zhang et al. (2017) study an FJSP with machine speed selection to minimize the makespan and TEC under TOU tariffs, and propose a

modified biogeography-based optimization algorithm combined with variable neighborhood search.

To our knowledge, the problem settings in this chapter have so far never been addressed in the literature.

5.3 Problem Formulation

We consider the flexible job shop scheduling (FJS) problem where the total energy cost is minimized. A set J of n jobs and a set M of m machines are given. Each job i consists of a sequence of n_i consecutive operations, where the jth operation of job i, denoted by $o_{ij} \in O$, can be processed on any machine in a subset $M_{ij} \subseteq M$ of compatible (also called eligible) machines.

For each operation o_{ij}, let P_{ij}^k be its processing time on machine $k \in M_{ij}$. In addition to the classic FJS settings, we adopt the Time-of-Use pricing scheme TOU. Different unit electricity power costs, denoted by $TOU = \{E_1, \ldots, E_b, \ldots, E_B\}$, are present for each individual pricing period b, with b specifying the starting point of the associated time interval.

Compared to traditional scheduling problems, our primary objective is to ensure a desired throughput rate by giving a maximum makespan $C_{max} \leq \bar{C}$. The planning horizon is thus given by $T = \{1, \ldots, \bar{C}\}$. The focus is then on minimizing the total energy cost TEC with a constraint on the makespan.

We first use an existing algorithm to generate solutions for FJS problem instances. This algorithm is based on the tabu search procedure proposed in Shen et al. (2018) by excluding setup-related elements. According to the time-based measure makespan C_{max}, operations are divided into a set O_t^c of critical operations and a set O_t^n of non-critical operations. The algorithm starts at a random solution π^0. In each iteration, the current solution is improved either by resequencing critical operations $o_{ij} \in O_t^c$ on the same machine or reassigning $o_{ij} \in O_t^c$ to a different eligible machine. In this context, structural properties on the FJS ensure move feasibility and accelerate the evaluation of neighbours. As a result, a set Π of schedules with makespan satisfying $C_{max} \leq \bar{C}$ are accepted.

The algorithm thus delivers different feasible schedules. Let $\pi \in \Pi$ be the assignment and sequence of a resulting schedule with $C_{max} \leq \bar{C}$. Each operation o_{ij} is now assigned to a machine $k \in M_{ij}$ with a corresponding start time s_{ij}. Let $\mathcal{P}_{ij}$, respectively $\mathcal{S}_{ij}$, be the set with the direct predecessors, respectively successors, of o_{ij}. By using the well-known definitions of head r_{ij} and

tail q_{ij} in job shop scheduling, we have $s_{ij} = r_{ij}$ with

$$r_{ij} = \max_{o_{i'j'} \in \mathcal{P}_{ij}} \left\{ r_{i'j'} + P^{k'}_{i'j'} \right\} \tag{5.1}$$

$$q_{ij} = \max_{o_{i'j'} \in \mathcal{S}_{ij}} \left\{ P^{k'}_{i'j'} + q_{i'j'} \right\}. \tag{5.2}$$

For operations $o_{ij} \in O^c_t$, the equation $r_{ij} + P^k_{ij} + q_{ij} = C_{max}$ holds. Now, let us consider the case where the makespan is allowed to be extended to $\bar{C}$. The earliest start time s^{min}_{ij} and latest start time s^{max}_{ij} for each operation o_{ij} on the assigned machine can be expressed as follows:

$$s^{min}_{ij} = r_{ij} \tag{5.3}$$

$$s^{max}_{ij} = \bar{C} - q_{ij} - P^k_{ij} \tag{5.4}$$

Accordingly, we define the feasible processing range for all operations.

Definition 5.1 (Feasible processing range)**.** *The periods* $V_{ij} \in \left[s^{min}_{ij}, s^{max}_{ij} + P^k_{ij}\right]$ *define the feasible processing range for operation* o_{ij}*.*

We assume that the feasible processing range of each operation can cross at most one price interval in TOU. The difference in pricing of adjacent intervals for o_{ij} is thus given by ΔE_{ij}. If an operation has to span over multiple intervals, the system flexibility is very limited. Also, subject to $C_{max} \leq \bar{C}$, it is common that each operation does not have much idle time.

The problem now consists in determining optimal start times s_{ij} for a given sequence π. We first introduce the following time-indexed decision variables:

$$x^{kt}_{ij} = \begin{cases} 1, & \text{if } o_{ij} \text{ is processed during time period t on machine } k, \\ 0, & \text{otherwise,} \end{cases} \tag{5.5}$$

and

$$y^t_{ij} = \begin{cases} 1, & \text{if the processing of } o_{ij} \text{ starts in time period t,} \\ 0, & \text{otherwise.} \end{cases} \tag{5.6}$$

Furthermore, assume that k' is the machine to which o_{ij} is assigned in sequence π. The problem can be formulated as the following mixed linear integer programming model (MIP):

$$\min \sum_{o_{ij} \in O^n_t} \sum_{t \in V_{ij}} E_t x^{k't}_{ij}. \tag{5.7}$$

subject to

$$\sum_{t \in V_{ij}} x_{ij}^{k't} = P_{ij}^{k'} \quad \forall O_{ij} \in O_t^n; \tag{5.8}$$

$$\sum_{o_{ij} \in O_t^n} x_{ij}^{kt} \leq 1 \quad \forall t \in V_{ij}; k \in M; \tag{5.9}$$

$$\sum_{t \in V_{ij}} y_{ij}^{t} = 1 \quad \forall o_{ij} \in O_t^n; \tag{5.10}$$

$$y_{ij}^{1} \geq x_{ij}^{k'1} \quad \forall o_{ij} \in O_t^n; \tag{5.11}$$

$$y_{ij}^{t} \geq x_{ij}^{k't} - x_{ij}^{k't-1} \quad \forall o_{ij} \in O_t^n; \quad t \in V_{ij} \setminus 1; \tag{5.12}$$

$$s_{ij} \geq t y_{ij}^{t} \quad \forall o_{ij} \in O_t^n; t \in V_{ij}; \tag{5.13}$$

$$s_{ij} \leq t y_{ij}^{t} + \left(C_{max} - P_{ij}^{k'}\right)\left(1 - y_{ij}^{t}\right) \quad \forall o_{ij} \in O_t^n; t \in V_{ij}; \tag{5.14}$$

$$s_{ij} \geq s_{ij-1} + P_{ij-1}^{k'} \quad \forall o_{ij} \in O_t^n; \tag{5.15}$$

$$s_{ij} \geq s_{i'j'} + P_{i'j'}^{k'} \quad \forall o_{ij} \in O_t^n; (o_{i'j'}, o_{ij}) \in \pi; \tag{5.16}$$

$$s_{ij} = r_{ij} \quad \forall o_{ij} \in O_t^c; \tag{5.17}$$

$$x_{ij}^{kt}, y_{ij}^{t} \in \{0, 1\} \quad \forall o_{ij} \in O; k \in M; t \in V_{ij}; \tag{5.18}$$

According to (5.7), the objective is to minimize the total energy cost in the planning horizon. Constraint (5.8) requires that the sum of the processing periods of an operation is equal to its processing time. Constraint (5.9) ensures that at most one operation is being processed in each period on each machine. According to Constraint (5.10), each operation has exactly one start period. Constraints (5.11) and (5.12) define the start period and avoid preemption, while constraints (5.13) and (5.14) connect the operation start period and start time variables. Constraint (5.15) follows the precedence constraints for operations that belong to the same job. Constraint (5.16) prevents the overlapping of operations on the same machine. Constraint (5.17) fixes the start times for time-critical operations, and constraint (5.18) defines the binary decision variables.

5.4 Problem Analysis

Note that, in the presence of *TOU* prices, conventional theories do not apply in general. In this section, we first introduce relevant definitions. Then, several properties are derived for the TEC to be minimized while satisfying the maximum makespan $\bar{C}$.

5.4.1 Definitions

So far, we have used the term critical operations $o_{ij} \in O_t^c$ defined on the classical criterion C_{max}. While some properties for the makespan criterion also hold here, additional definitions are required when minimizing TEC. To consider the case where operations become critical if they have the potential to improve TEC by being moved, we next present the definition of critical operations in terms of C_{max} and TEC.

Definition 5.2 (Time-critical operation). *For a given sequence with a makespan C_{max}, an operation o_{ij} is time-critical if $r_{ij} + P_{ij}^k + q_{ij} = C_{max}$ or, equivalently, if $s_{ij}^{max} - s_{ij}^{min} = \bar{C} - C_{max}$.*

Definition 5.3 (Energy-critical operation). *For a given sequence with a given TEC, an operation o_{ij} is energy-critical if $s_{ij}^{min} < b < s_{ij}^{max} + P_{ij}^k$, where b is specified by TOU.*

The set of energy-critical operations is denoted by O_e^c compared to the conventional set O_t^c of time-critical operations. In Figure 5.1, the operations in blue $O_t^c = \{o_{21}, o_{32}, o_{22}, o_{33}\}$ are the time-critical operations for the makespan. The remaining operations are not time-critical. On the other hand, o_{12} can move around b_1 while the feasible processing ranges of o_{23} and o_{13} cover the last two intervals. Using Definition 5.3, the energy-critical operations in red are $O_e^c = \{o_{12}, o_{23}, o_{13}\}$. Relating to different TOU cases, it is necessary to introduce some additional definitions.

Definition 5.4 (Left-shifted schedule). *A schedule is left-shifted if all operations $o_{ij} \in O$ start at s_{ij}^{min}.*

Definition 5.5 (Right-shifted schedule). *A schedule is right-shifted if all operations $o_{ij} \in O$ start at s_{ij}^{max}.*

Definition 5.6 (Compact schedule). *A schedule is compact if it is either left- or right-shifted.*

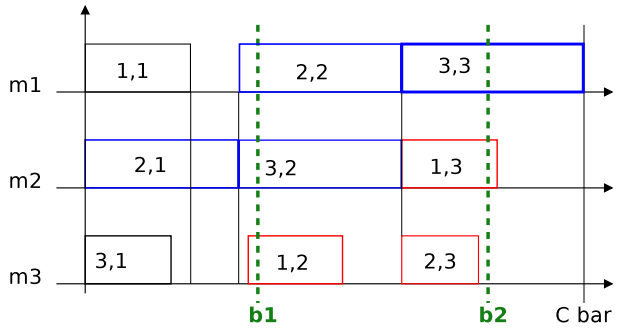

Figure 5.1: Illustration of critical operations

Definition 5.7 (Partial compactness)**.** *Given a schedule and a set $O^f \subset O$ of operations with fixed start times, the successors of operations in O^f ensure partial compactness if they are left-shifted.*

Alternatively, partial compactness can also be defined on operations with fixed completion times and their predecessors.

As illustrated in Figure 5.2, starting from a non-compact schedule in the left Gantt chart, all operations in the second and third intervals achieve partial compactness, whereas the start times of operations in $O^f = \{(1,2),(2,2),(3,2)\}$ remain unchanged in the right Gantt chart.

5.4.2 Properties with Varying TOU Settings

Considering the fluctuating prices of TOU, energy costs can benefit from varying the start times of operations as formalized below. Using the new definitions, we have the following lemmata.

Lemma 5.1. *For a given sequence, shifting non-energy-critical operations $o_{ij} \notin O_e^c$ does not change TEC.*

Lemma 5.2. *If TEC is changed for a fixed sequence π, then the start time of at least one operation o_{ij} is altered with*

$$s_{ij}^{min} \leq b \leq s_{ij}^{max} + P_{ij}^{k}, \tag{5.19}$$

where b is the start time of an interval according to TOU.

Given a fixed sequence π, moving operations in the same price interval leads to a constant TEC. This contradicts condition (5.19). Therefore, the only possibility to change TEC in this case is to change the start times of operations whose feasible processing ranges cross price intervals.

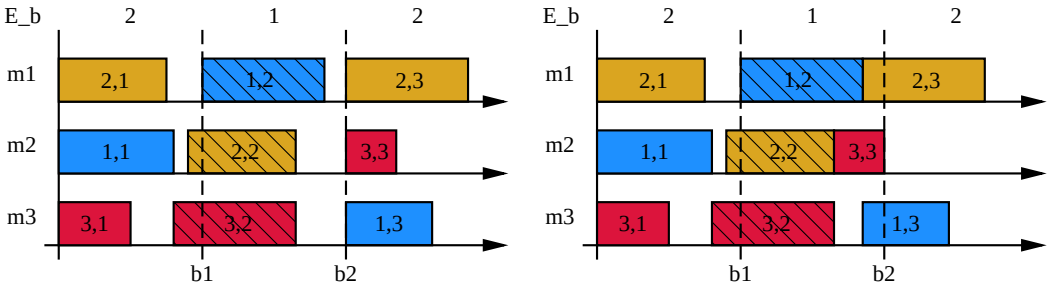

Figure 5.2: Illustration of a compact schedule

Therefore, we should concentrate on critical operations in terms of TEC, i.e., $o_{ij} \in O_e^c$. In an optimal solution, the remaining operations shift accordingly but with no effect on TEC.

Next, we investigate the structure of TOU. Starting with two-interval TOUs with two cases. It is trivial to either left or right shift all operations to reach an optimal solution. Cases with three-interval TOUs, denoted by $TOU(3)$, are depicted in Figure 5.3. After examining the increasing and decreasing tendencies of adjacent price intervals, we present the following properties.

Lemma 5.3. *For case 1 of $TOU(3)$ as depicted in Figure 5.3, a left-shifted schedule is optimal.*

Proof: In case 1 of $TOU(3)$, the prices are strictly increasing. Assume that an operation o_{ij} is not left-shifted, i.e. $s_{ij} > s_{ij}^{min}$. The completion time of this operation is greater than the left-shifted position $C_{ij} > C_{ij}^l = s_{ij}^{min} + P_{ij}^k$. Therefore, the energy cost for this operation cannot be smaller, since $E(s_{ij}) \geq E\left(s_{ij}^{min}\right)$ and $E(C_{ij}) \geq E\left(C_{ij}^l\right)$ hold. □

Similarly, we have the next lemmata for cases 2 and 3 of $TOU(3)$. Prices are decreasing in case 2 while case 3 reaches the highest price in the middle interval.

Lemma 5.4. *For case 2 of $TOU(3)$ as depicted in Figure 5.3, a right-shifted schedule is optimal.*

Lemma 5.5. *For case 3 of $TOU(3)$ as depicted in Figure 5.3, a schedule is optimal if all operations $o_{ij} \in O_e^c$ with $s_{ij}^{max} + P_{ij}^k \leq b_2$ are left-shifted and the remaining ones are right-shifted.*

So far, the first three cases of $TOU(3)$ can be solved to optimality by directly applying lemmata 5.3–5.5. We next discuss the case 4 of $TOU(3)$.

Lemma 5.6. *For case 4 of $TOU(3)$ as depicted in Figure 5.3 with only two prices, the problem is equivalent to maximizing the total processing time in the second interval.*

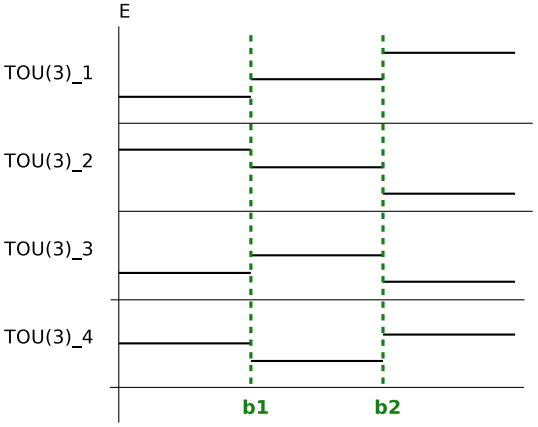

Figure 5.3: TOU combination

Lemma 5.7. *For case 4 of TOU (3) as depicted in Figure 5.3 with three different prices, maximizing the total processing time in the second interval is not always optimal.*

According to Lemma 5.6, the first and the last intervals have identical prices. Solving the problem indicates processing most possible operations in the second interval. It is not true for case 4 with three different prices where prices for the first and the last intervals are different. Figure 5.4 shows a counter-example, where the prices of $TOU(3)$ are 3, 1, and 2. Compared to the left Gantt chart, the total idle time in the second interval in the right Gantt chart is larger, as indicated by the hatched area on the third machine. As a result, TEC is further reduced.

Proposition 5.1. *For case 4 of TOU (3) as depicted in Figure 5.3, there exists an optimal schedule where only operations $o_{ij} \in O_e^c$ such that $s_{ij}^{min} < b_1$ are not left-shifted.*

Proof: In an optimal schedule, operations o_{ij} such that $s_{ij}^{min} < b_1$ and $s_{ij}^{max} + P_{ij}^k \leq b_1$ can be left-shifted, right-shifted or in between since they can only be scheduled in the first interval. Also, operations o_{ij} such that $s_{ij}^{min} \geq b_1$ can always be left-shifted in an optimal schedule, since the energy costs for these operations can only be reduced by left-shifting them as the energy price in the third interval is larger than the energy price in the second interval. Hence, only operations such that $s_{ij}^{min} < b_1$ and $s_{ij}^{max} + P_{ij}^k > b_1$, i.e. operations $o_{ij} \in O_e^c$ such that $s_{ij}^{min} < b_1$ (see Definition 5.3), might be delayed to save energy costs by moving processing times between the first and third intervals. □

Note that, in an optimal schedule and following Proposition 5.1, if we note $O_e^{c'}$ the subset of operations O_e^c such that $s_{ij} < b_1$ and $s_{ij} + P_{ij}^k \geq b_1$, then the operations in $O_e^{c'}$ ensure partial compactness, i.e. all the successors of operations in $O_e^{c'}$ are left-shifted (see Definition 5.7).

According to Proposition 5.1, there is at least one optimal schedule that ensures partial compactness. Conversely, a schedule of partial compactness is not necessarily optimal. Ideally, operations

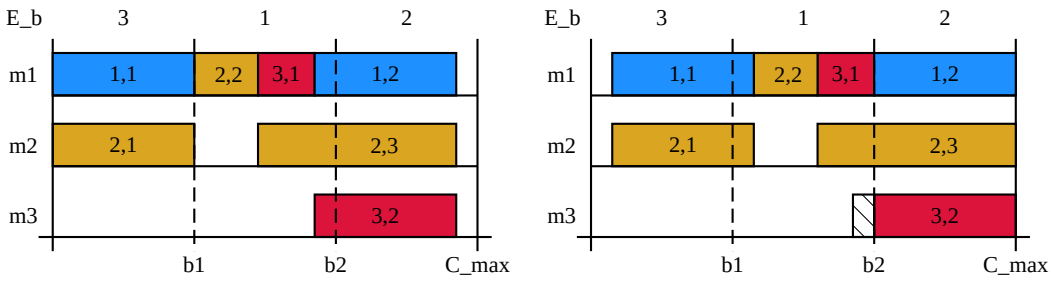

Figure 5.4: Illustration of Lemma 5.7

are concentrated in cost-efficient intervals without deteriorating the makespan. A considerable number of remaining operations are attached to their predecessors/successors. The crucial part is to identify which operations to delay.

If we expand TOU and consider the schedule on a daily basis, TOU is typically divided into four intervals, consisting of off-, mid-, and on-peak levels. In general, depending on the sequence of peak levels, different cases of TOU can be formulated. As shown in Figure 5.5, there are two sets of symmetrical combinations, and thus eight combinations in total. Without loss of generality, we investigate the four cases on the left side. Similar to $TOU(3)$, we can derive the two following lemmata immediately.

Lemma 5.8. *Lemma 5.3 applies to case 1 of TOU (4) as depicted in Figure 5.5.*

Lemma 5.9. *For case 2 of TOU (4) as depicted in Figure 5.5, a schedule is optimal if all operations $o_{ij} \in O_e^c$ with $s_{ij}^{max} + P_{ij}^k \leq b_3$ are left-shifted and the remaining ones are right-shifted.*

With respect to cases 3 and 4 of $TOU(4)$, it is again essential to determine which operations should be delayed and how to delay their processing. The start times of the remaining operations then follow automatically as left-shifted to ensure partial compactness.

Proposition 5.2. *For case 3 of TOU (4) as depicted in Figure 5.5, there exists an optimal schedule where only operations $o_{ij} \in O_e^c$ such that $s_{ij}^{min} < b_1$ are neither left-shifted nor right-shifted.*

Proposition 5.3. *For case 4 of TOU (4) as depicted in Figure 5.5, there exists an optimal schedule where only operations $o_{ij} \in O_e^c$ such that $s_{ij}^{min} < b_2$ are not left-shifted.*

For these cases, we can further narrow the relevant operations down to the energy-critical operations crossing a specific interval in TOU. Following the analysis, we can see that, for $TOU(3)$

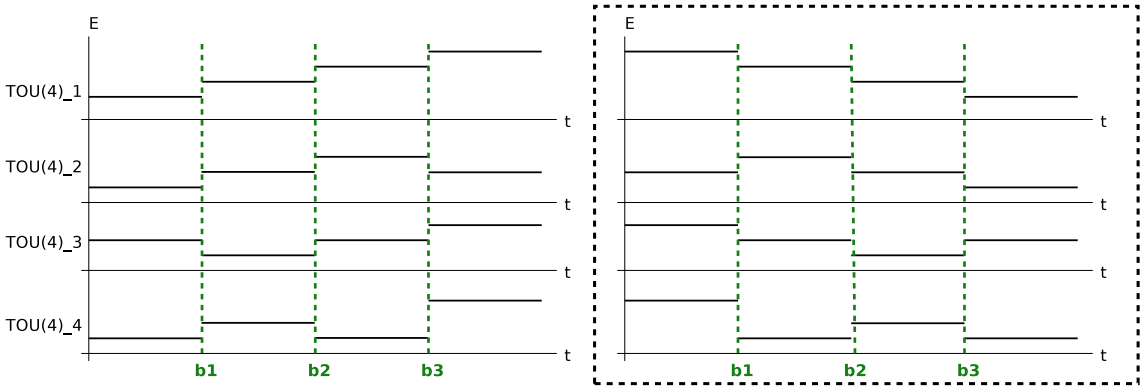

Figure 5.5: Combinations of TOU with four intervals

and $TOU(4)$, a majority of cases can be solved to optimality by applying the properties. For the remaining cases, we focus on operations $o_{ij} \in O_e^c$ and stress compactness.

It is also noteworthy that there are optimal but not compact schedules. Especially when TOU intervals are further increasing, ensuring compactness can become time-consuming. Alternatively, we can specify the possible positions for energy-critical operations as four cases.

Proposition 5.4. *For a given sequence π, the start point of an operation $o_{ij} \in O_e^c$ satisfies one of the following cases of its feasible processing range in an optimal solution to minimize TEC:*

Case 1 $s_{ij} = r_{ij}$*;*

Case 2 $s_{ij} = C_{max} - q_{ij} - P_{ij}^k$*;*

Case 3 $s_{ij} = b$ *where* o_{ij} *starts at* b *according to* TOU*;*

Case 4 $s_{ij} = b - P_{ij}^k$ *where* o_{ij} *ends at* b *according to* TOU*.*

Proof: The four cases are depicted in Figure 5.6. Case 1 indicates that o_{ij} is left-shifted and directly follows its predecessor(s). In case 2, o_{ij} is right-shifted and attached to its successor to reach its maximum delay. Following the previous analysis in Lemmata 5.3–5.9, we know that they are the optimal positions for increasing and decreasing TOU prices. Note that the feasible processing range V_{ij} of an operation o_{ij} can cross at most one interval. If $E(b-1) > E(b)$ and $s_{ij}^{max} \geq b + P_{ij}^k$ hold, start time $s_{ij} < b$ leads to an increased TEC while $s_{ij} > b$ does not improve TEC. Therefore, $s_{ij} = b$ is optimal (Case 3). Similarly, it can be proved that $s_{ij} = b - P_{ij}^k$ (Case 4) is optimal if $E(b) < E(b+1)$ and $s_{ij}^{min} \leq b - P_{ij}^k$ are true. □

We were not able to formally prove the complexity of case 4 of $TOU(3)$ and cases 3 and 4 of $TOU(4)$, that we leave for future research. Our belief is that case 4 of $TOU(3)$ is NP-hard. If true, this would mean that cases 3 and 4 of $TOU(4)$, that include case 4 of $TOU(3)$, are also NP-hard.

5.5 Heuristic Approaches

Based on the previous analysis and observations, we now develop heuristic approaches to complement the MIP model to determine the start times of operations for a given sequence π. The first heuristic, the shifting procedure (SP) presented in Section 5.5.1, mainly focuses on the unsolvable cases as discussed in Section 5.4, while the second heuristic, the dynamic-programming-based procedure (DP) presented in Section 5.5.2, solves the FJSP with general TOU settings.

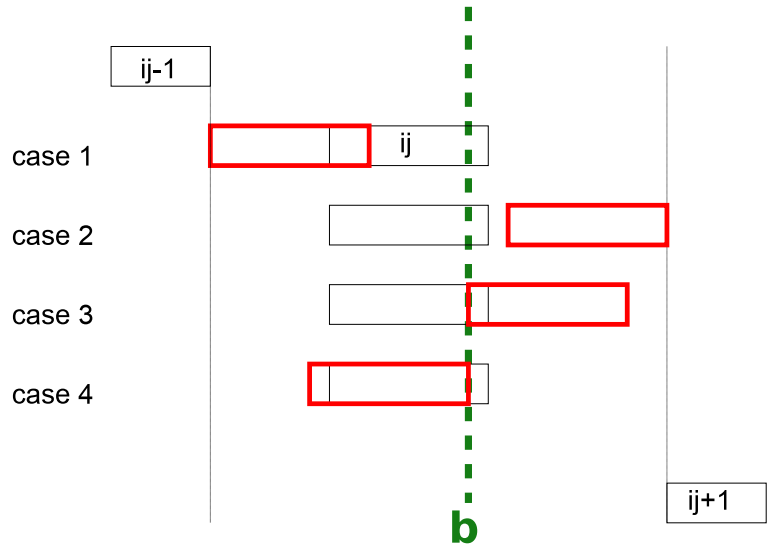

Figure 5.6: Illustration of Propositions 5.4

5.5.1 Shifting Procedure

Before presenting the shifting procedure (SP), we first introduce the concept of critical interval.

Definition 5.8 (Critical interval)**.** *Given a TOU pricing, an interval, defined by the left and right price breaks* $[b_l^*, b_r^*]$*, is critical if the prices of its two adjacent intervals are higher.*

By definition, the interval with $b_l^* = b_1$ and $b_r^* = b_2$ in case 3 of $TOU(4)$ is critical. In an optimization procedure, we can thus focus solely on the operations crossing b_1 or b_2. Furthermore, delaying an operation o_{ij} by t units causes multiple operations (o') to shift simultaneously as a result of partial compactness. These operations are on diverse paths passing through o_{ij}, i.e. $o' \in PA(o_{ij})$. The TEC variation can thus be expressed as follows:

$$\Delta TEC = \sum_{o' \in PA(o_{ij})} c(o') \cdot t, \quad c(o') \in \{0, \Delta E\}, \tag{5.20}$$

where ΔE is the price difference when o' crosses intervals. According to equation (5.20), TEC can be constant or temporarily increasing before it is improved. Therefore, we can set $t = 1$ and stepwise increase the start time of o_{ij} within the feasible processing range while calculating the TEC variation, until the best combination is determined. Next, we propose SP as follows.

SP is described in Algorithm 8 and relies primarily on the systematic shifting of operations crossing the price breaks adjacent to the critical interval in order to reduce TEC. SP starts with

a left-shifted initial solution π^0 and first identifies the relevant operations in O^r. In the forward phase, the start times of operations o_{ij} in O^r are stepwise increased ($s_{ij} = s_{ij}+1$). After each step, succeeding operations, whose start times become infeasible due to the shift of o_{ij}, are updated. The resulting schedule is also adjusted to ensure partial compactness of all preceding and succeeding operations of o_{ij}. As long as the resulting TEC does not deteriorate, the schedule is updated. This step is repeated until the start time reaches its maximum value ($s_{ij} = s_{ij}^{max}$). The procedure then turns to the next operation in O^r. The process is equivalent in the backward phase, where the start times of operations in O^r are stepwise decreased ($s_{ij} = s_{ij} - 1$). Both forward and backward phases ensure that energy-critical operations are being pushed into the most cost-efficient interval. An advantage of SP is that, for each shifting step, the energy cost variation of the entire schedule and not of the individual operation is evaluated.

For illustration, Figures 5.7, 5.8 and 5.9 show an initial solution, the forward phase, and the backward phase of SP, respectively. For simplification, all operations $o_{ij} \in O^r$ in the initial solution

Algorithm 8 SP – Shifting Procedure

Input:
Initial left-shifted solution π^0 with problem $s_{ij} = r_{ij}$; $\pi_b = \pi^0$
Price breaks adjacent to the critical interval b_l^* and b_r^*;
Set of relevant operations
$O^r = \{o_{ij} : s_{ij}^{min} < b_l^*, s_{ij}^{max} + P_{ij}^k \geq b_l^*\} \cup \{o_{ij} : s_{ij}^{min} < b_r^*, s_{ij}^{max} + P_{ij}^k \geq b_r^*\}$
repeat
    for all operations $o_{ij} \in O^r$ **do** ▷ Forward phase
        $\pi_c \leftarrow \pi_b$
        while $s_{ij} < s_{ij}^{max}$ **do**
            $\pi_c \leftarrow Increase\_Start\_Time(\pi_c, s_{ij})$
            $\pi_c \leftarrow Ensure\_Compactness(\pi_c)$
            if $TEC(\pi_c) \leq TEC(\pi_b)$ **then**
                $\pi_b \leftarrow \pi_c$
            end if
        end while
    end for
    for all operations $o_{ij} \in O^r$ **do** ▷ Backward phase
        $\pi_c \leftarrow \pi_b$
        while $s_{ij} > s_{ij}^{min}$ **do**
            $\pi_c \leftarrow Decrease\_Start\_Time(\pi_c, s_{ij})$
            $\pi_c \leftarrow Ensure\_Compactness(\pi_c)$
            if $TEC(\pi_c) \leq TEC(\pi_b)$ **then**
                $\pi_b \leftarrow \pi_c$
            end if
        end while
    end for
until No further improvement found
Return π_b

are already shifted towards the critical interval as far as possible without affecting the start times of other operations in order to ensure partial compactness. Red-framed operations are time-critical. In the forward phase, operations (4,1), (9,2), and (2,3) are relevant operations, i.e. in O^r. Each Gantt chart in Figure 5.8 illustrates the shifting of an operation in O^r. The shifting of other energy-critical operations in this example does increase TEC and is therefore not displayed. Thick black frames highlight the block of operations that are being moved accordingly to ensure compactness. Figure 5.9 then shows a step in the backward phase, where operation (11,4) in O^r along with the block (7,2,...,8,5) are being shifted into the second interval. As a result, TEC is then further reduced by 15 units.

SP does not always reach optimal solutions. For the small example in Figure 5.10, initial and optimal solutions are shown in the top-left and top-right Gantt charts. Two illustrative moves of SP are depicted in the bottom-left and bottom-right Gantt charts. Due to the increased TEC, such intermediate moves are not realized. In order to find the optimum, SP would have to either perform inferior moves or move operations simultaneously. Computationally, however, it would be very expensive. On the other hand, as shown in our extensive computational experiments, excluding these cases requires considerably less computational time and leads to small deviation from an optimal solution.

5.5.2 Dynamic-Programming-Based Procedure

As discussed in Section 5.4.2, several TOU combinations can be solved to optimality by integrating the corresponding properties. Furthermore, we develop SP to address the remaining cases, where a large number of operations are excluded. In return, we can afford to stepwise increase/decrease start times.

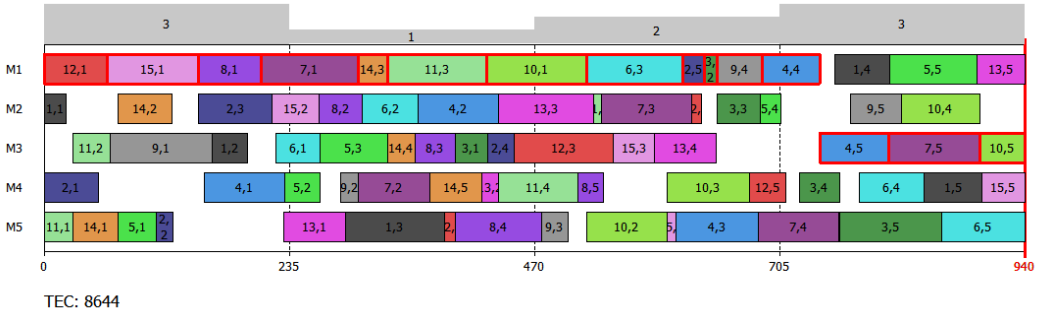

Figure 5.7: Illustration of SP – initial solution

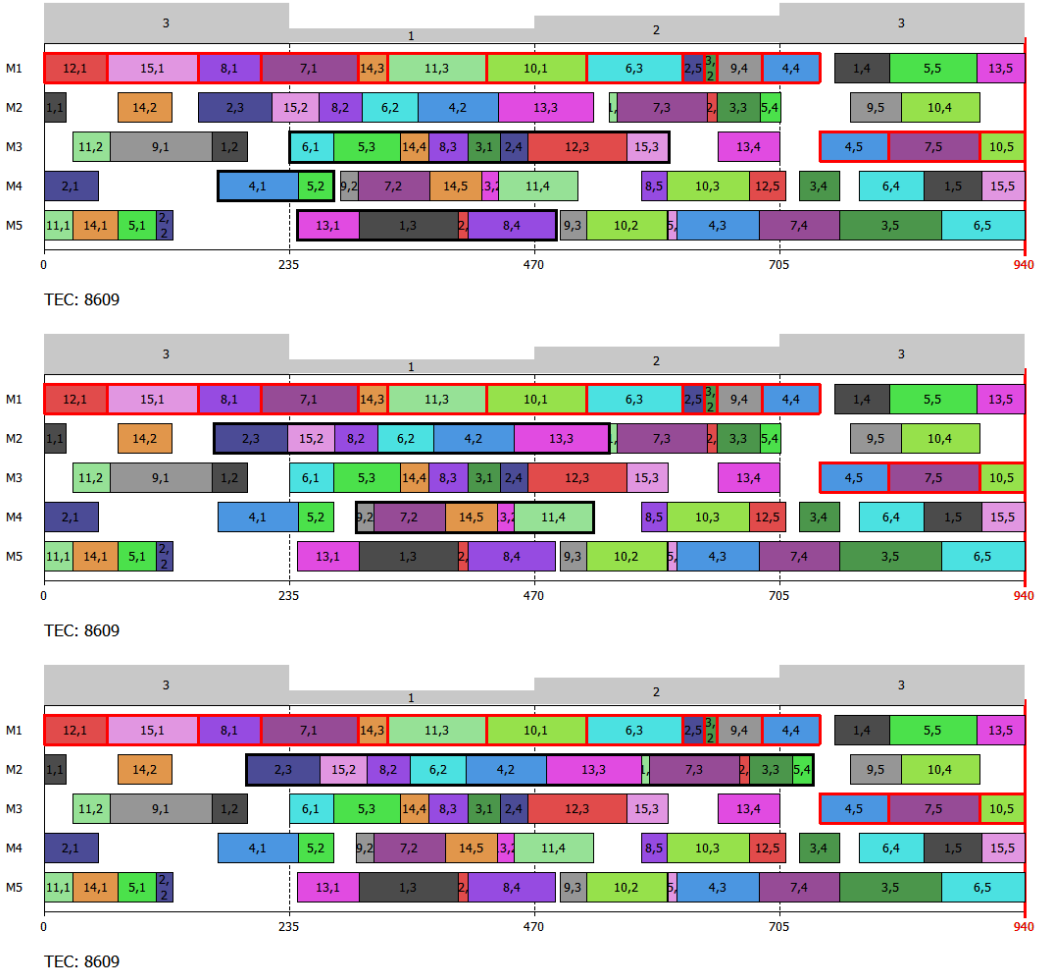

Figure 5.8: Illustration of SP – forward phase

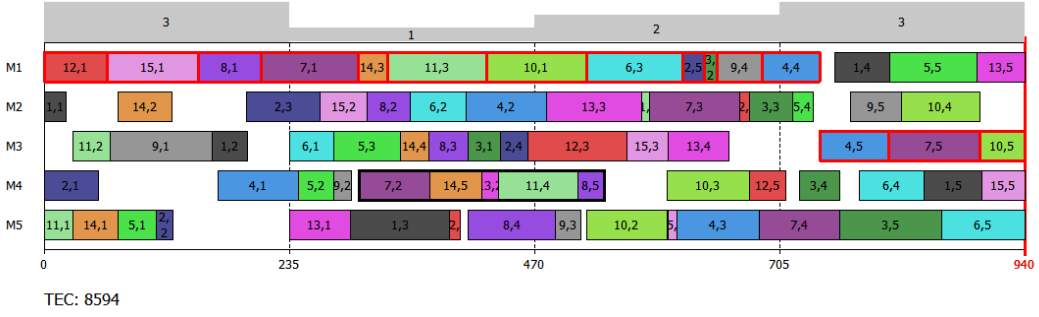

Figure 5.9: Illustration of SP – backward phase

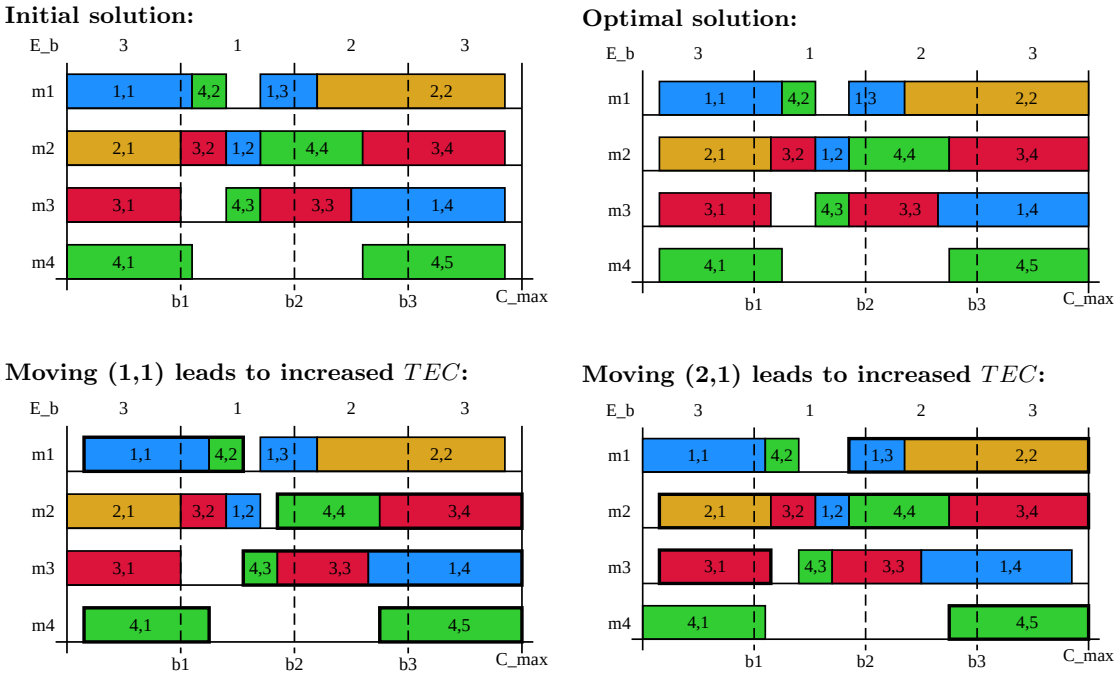

Figure 5.10: Illustration of case not solved to optimality

In general, the number of operations relevant to the variation of TEC increases when TOU includes more intervals. Recall that Lemma 5.1 identifies promising operations, while Proposition 5.4 indicates how to move these operations. Combining these properties, we also develop a dynamic-programming-based procedure (DP) to improve the start times of operations for a given sequence.

According to the DP Algorithm 9, the values of b_{ij}, r_{ij}, q_{ij} for operations $o_{ij} \in O_e^c$ are calculated based on a given sequence π^0. In the forward DP procedure, operations whose predecessors are non-existent or have been labelled are considered. By building partial solutions, we calculate all four cases given in Proposition 5.4. Note that the backward version is only performed on a subset of partial solutions Π_v, where the start times of the operations are not fixed yet. In the end, the algorithm returns a solution π^* with improved start times for all operations and with the best-known TEC.

For better understanding, Figure 5.11 shows a small example of the DP procedure which starts with an initial sequence (I) and $TOU = \{3, 1, 2\}$. Note that operations $o_{1,1}$ and $o_{2,1}$ are not energy-critical, since their feasible processing range does not cross b_1. Hence, we set $s_{1,1} = r_{1,1}$, $s_{2,1} = r_{2,1}$, and update $R = \{o_{2,2}; o_{1,2}\}$. In step (II), $o_{2,2}$ is selected and $R = \{o_{1,2}\}$. For $o_{2,2}$, one new partial solution is built according to Case 1 ($s_{2,2} = r_{2,2}$) of Proposition 5.4. Without loss of generality, we show only the part of the search tree corresponding to the forward phase of the

Algorithm 9 DP – Dynamic-programming-based Procedure

Input: Initial solution π^0 with r_{ij}, q_{ij}, O_e^c
$\Pi_c \leftarrow \{\pi^0\}$; $\Pi_v \leftarrow \emptyset$ ▷ Start forward DP
Determine set $R = \{o_{ij} : o_{i'j'} \in \mathcal{P}_{ij}$ is non-existent or labelled$\}$
while R is not empty **do**
  Select an operation $o_{ij} \in R$ and $R \leftarrow R \setminus o_{ij}$;
  if $o_{ij} \in O_e^c$ **then**
    for $\pi \in \Pi_c \cup \Pi_v$ **do**
      Determine s_{ij}^{min} according to (5.3)
      Determine $s_{ij}^{max} = \bar{C} - q_{ij}$;
      Determine $b_{ij} \in [s_{ij}^{min}, s_{ij}^{max}]$
      $\Pi_c \leftarrow \Pi_c \cup \pi \oplus o_{ij}^{min}$, where $s_{ij} = s_{ij}^{min}$ ▷ Corresponding to case 1
      $\Pi_c \leftarrow \Pi_c \cup \pi \oplus o_{ij}^{b}$, where $s_{ij} = b_{ij}$ ▷ Corresponding to case 3
      $\Pi_v \leftarrow \Pi_c \cup \pi \oplus o_{ij}^{v}$, where s_{ij} is variable
    end for
  end if
  Label o_{ij} and update R;
end while
Determine set $Q = \{o_{ij} : o_{i'j'} \in \mathcal{S}_{ij}$ is non-existent or labelled$\}$ ▷ Perform backward DP
while Q is not empty **do**
  Select an operation $o_{ij} \in Q$ and $Q \leftarrow Q \setminus o_{ij}$
  if $o_{ij} \in O_e^c$ **then**
    for $\pi \in \Pi_v$ **do**
      Determine s_{ij}^{max} according to (5.4)
      $\Pi_v \leftarrow \Pi_v \cup \pi \oplus o_{ij}^{max}$, where $s_{ij} = s_{ij}^{max}$ ▷ Corresponding to case 2
      $\Pi_v \leftarrow \Pi_v \cup \pi \oplus o_{ij}^{b^-}$, where $s_{ij} = b_{ij} - P_{ij}^k$ ▷ Corresponding to case 4
    end for
  end if
  Label o_{ij} and update Q;
end while
return $\pi^* \leftarrow \arg\min\{TEC(\pi), \pi \in \Pi_c \cup \Pi_v\}$

algorithm and do not examine set Π_v of partial solutions, where s_{ij} remains variable. Based on solution (II), two extended partial solutions are generated corresponding to Case 1 ($s_{1,2} = r_{1,2}$) and Case 3 ($s_{1,2} = b$). Labelling $o_{1,2}$ leads to $R = \{o_{1,3}, o_{2,3}\}$. According to Proposition 5.1, $o_{1,3}$ and $o_{2,3}$ must ensure partial compactness. Therefore, it is not necessary to consider multiple start times besides Case 1. Note that integrating the structural properties of Section 5.4 is vital, since a significant number of partial solutions can be ruled out. The final solution with $s_{1,2} = b$ corresponds to the optimal TEC.

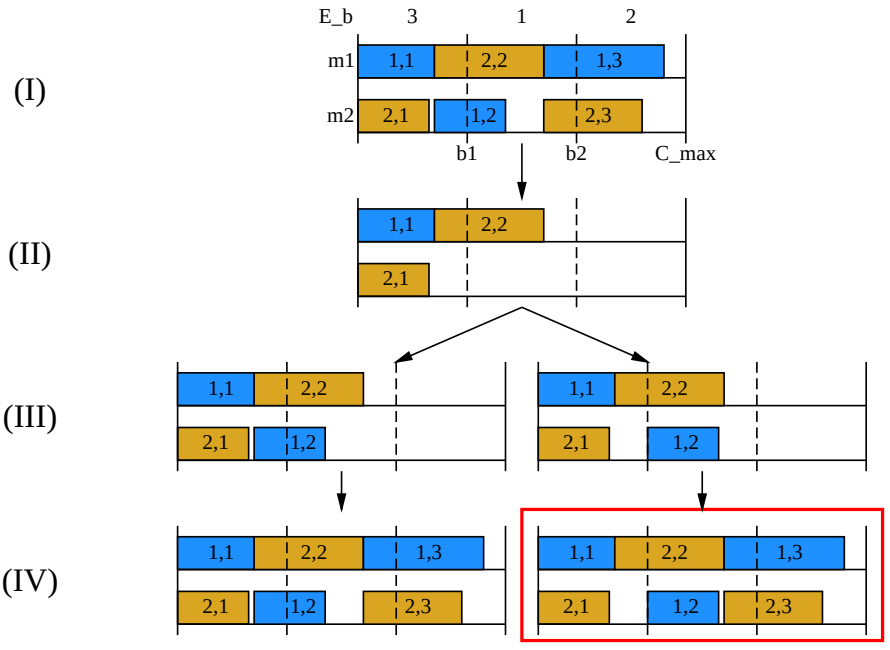

Figure 5.11: Example for DP heuristic with search tree containing partial and final solutions. Optimal solution is marked by a red frame.

5.6 Computational Experiments

Section 5.6.1 first presents how our extensive experiments are designed. Then, in Section 5.6.2, we first optimally solve the benchmark instances with small values of $\bar{C}$ by using IBM ILOG CPLEX, and the results are compared with those given by SP and DP. Since IBM ILOG CPLEX cannot reach optimal solutions with increasing values of $\bar{C}$, we then only focus in Section 5.6.3 on the comparison of SP and DP.

5.6.1 Experimental Design

Our experiments are performed on existing benchmark instances for the flexible job shop scheduling problem. We first summarize the benchmark sets in previous studies:

- Demirkol/Mehta/Uzsoy instances from Demirkol et al. (1998): Dmu01–80 ranging from 20 jobs on 15 machines to 50 jobs on 20 machines;
- Hurink/Jurisch/Thole instances from Hurink et al. (1994): car1–8, abz5–9, la1–40, orb1–10, mt06, 10, 20 including edata, rdata, and vdata, ranging from 6 jobs on 6 machines to 30 jobs

on 10 machines;

- Dauzère-Pérès/Paulli instances from Dauzère-Pérès and Paulli (1997): 01–18a ranging from 10 jobs on 5 machines to 20 jobs on 10 machines;
- Brandimarte instances from Brandimarte (1993): Mk01–10 ranging from 10 jobs on 6 machines to 20 jobs on 15 machines;
- Barnes/Chambers instances from Barnes and Chambers (1996): mt10∗, setb4∗, and seti5∗ sets ranging from 10 jobs on 11 machines and 15 jobs on 18 machines.

With respect to *TOU* pricing schemes, we are motivated by practical applications. Table 5.1 simulates the electricity prices in Germany (Schulz et al., 2020), where daily prices are divided into four intervals of three levels. This setting is equivalent to Case 2 in Figure 5.5, which, following Lemma 5.9, can be solved to optimality.

Furthermore, we are also interested in solving other cases, since they are of both theoretical and practical relevance. Without loss of generality, we specify the *TOU* price levels as 1, 2, 3 and 4, and examine 10 combinations as given in Table 5.2. Combination 3 corresponds to the German electricity market setting. According to the analysis in Section 5.4, optimal solutions can be obtained for the first four combinations while the remaining combinations are challenging. Therefore, besides the $TOU(4)$ cases, we add two additional $TOU(3)$ cases.

As a starting point, we generate a sequence π^0 for the best known makespan by using the Tabu

Table 5.1: *TOU* prices

Time (hours)	0-7	8-15	16-20	21-23
Level	off-peak	mid-peak	on-peak	mid-peak
Price $\left(\frac{€}{MWh}\right)$	80	160	240	160

Table 5.2: *TOU* settings

Main combination	*TOU*	Symmetrical combination	*TOU*
1	{1,2,3,4}	2	{4,3,2,1}
3	**{1,2,3,2}**	4	{2,3,2,1}
5	{1,2,1,2}	6	{2,1,2,1}
7	{2,1,2,3}	8	{3,2,1,2}
9	{2,1,3}	10	{3,1,2}

Search algorithm of Shen et al. (2018), i.e. $C^*_{max} = C_{max}(\pi^0)$. An upper bound of the makespan is given by

$$\bar{C} = (1 + a) \cdot C^*_{max}, \tag{5.21}$$

where a is selected from the range $[0\%, 20\%]$. For comparison purposes, we also use a set of alternative sequences satisfying $C_{max} \leq \bar{C}$ obtained by the same Tabu Search algorithm.

Both SP and DP are coded using the C++ programming language. The MILP is implemented and solved using IBM ILOG CPLEX 12.8. All the computational experiments are carried out on a PC with Intel Xeon CPU E5-2697 v3 with 2.60GHz and 128 GB RAM.

5.6.2 Comparison of Heuristics SP and DP with IBM ILOG CPLEX

Considering the limited capacity of commercial solvers, we first use IBM ILOG CPLEX, SP, and DP to determine the schedule without deteriorating the best known makespan, i.e., $\bar{C} = C^*_{max}$. Note that we implement two models in CPLEX. The first one, denoted by MIP_0 is presented in Section 5.3. The second model MIP_1 integrates Lemmata 5.3–5.9 to reduce the number of operations to consider.

The models are able to solve the benchmark sets Hurink et al. (1994); Dauzère-Pérès and Paulli (1997); Brandimarte (1993), and Barnes and Chambers (1996) of 115 instances in total, except for one with $TOU = \{2, 1, 2, 3\}$, one with $TOU = \{2, 1, 2, 1\}$, and one with $TOU = \{2, 1, 3\}$. Tables 5.3 and 5.4 compare the solution quality and computational time of MIP_0, MIP_1, SP, and DP variants. The relative percentage deviation (RPD)is determined using:

$$RPD = \left(\frac{TEC(A)}{TEC^*} - 1\right) 100, \quad A \in \{\text{MIP\_0, MIP\_1, SP, DP variants}\}, \tag{5.22}$$

where TEC^* denotes the minimal cost and $TEC(A)$ the resulting cost after applying approach A. We can see from Table 5.3 that, for the first four TOU combinations, the procedure relying on the optimization properties determines optimal solutions instantaneously. It is also worth mentioning that, compared to the original model MIP_0, the improved version MIP_1 is significantly faster.

For the remaining cases, Table 5.4 provides the RPD and CPU time for each procedure. The columns in "Optima" indicate the number ("No.") and the ratio ("Ratio") of problem instances for which the solutions are optimal. Note that SP obtains all optimal solutions, while DP obtains optimal solutions for 85% of the instances on average. The deviation of the solutions obtained by DP from the optimal solutions is lower than 0.01%. Moreover, SP and DP require considerably

less computational time than IBM ILOG CPLEX.

Note that the total number of potential start times significantly increase with the maximum makespan $\bar{C}$. Tables 5.5 and 5.6 report the computational time of MIP_0, MIP_1, SP and DP when the makespan is increased by 5%. The time limit is set to one hour. We can observe that the computational time for MIP_0 and MIP_1 to determine an optimal solution increases greatly, while the instances in Table 5.6 are not solvable for MIP_0 in one hour. In comparison, the computational time of the procedure with the optimization properties (Columns "Properties") is negligible (see Table 5.5), and the computational times of SP and DP are small and very close, about 5 seconds

Table 5.3: Comparison of IBM ILOG CPLEX and procedure with optimization properties

TOU	MIP_0	MIP_1	Procedure with properties	
	Time	Time	RPD	Time
{1,2,3,4}	10.44	2.19	0.00	0.00081
{4,3,2,1}	6.87	2.12	0.00	0.00080
{1,2,3,2}	9.99	2.34	0.00	0.00080
{2,3,2,1}	11.33	2.25	0.00	0.00080
Mean	9.66	2.23	0.00	0.00080

Table 5.4: Comparison of SP and DP based on all instances solved optimally by IBM ILOG CPLEX

TOU	CPLEX		SP				DP			
	MIP_0	MIP_1	Optima				Optima			
	Time	Time	RPD	No.	Ratio	Time	RPD	No.	Ratio	Time
{2,1,2,3}	43.95	36.91	0.000	114	1.00	0.65	0.008	90	0.79	1.72
{3,2,1,2}	50.29	11.74	0.000	115	1.00	0.78	0.006	94	0.82	1.85
{1,2,1,2}	99.39	7.18	0.000	115	1.00	0.49	0.006	93	0.81	0.12
{2,1,2,1}	50.71	34.71	0.000	114	1.00	0.40	0.006	92	0.81	0.12
{2,1,3}	42.32	34.06	0.000	114	1.00	0.49	0.001	107	0.94	0.12
{3,1,2}	19.34	3.19	0.000	115	1.00	0.60	0.003	105	0.91	0.15
Mean	51.00	21.30	0.000		1.00	0.58	0.005		0.85	0.68

Table 5.5: Computational times when increasing $\bar{C}$ by 5% (I)

TOU	MIP_0	MIP_1	Properties	*TOU*	MIP_0	MIP_1	Properties
{1,2,3,4}	10.11	4.72	0.0008	{4,3,2,1}	8.60	4.11	0.0008
{1,2,3,2}	16.13	5.53	0.0008	{2,3,2,1}	31.79	5.43	0.0008

(see Table 5.6).

5.6.3 Performance of SP and DP

For the problem sizes that cannot be solved using IBM ILOG CPLEX, only SP and DP are compared. The relative percentage of TEC improvement (RPI) is expressed by

$$RPI = \left(1 - \frac{TEC(A)}{TEC(\pi^0)}\right) \cdot 100, \quad A \in \{\text{SP, DP variants}\}, \tag{5.23}$$

where $TEC(\pi^0)$ and $TEC(A)$ are the initial costs of π^0 and the costs after applying SP or DP. In our experiments, we first set $a \in \{0\%, 1\%, 5\%, 10\%, 15\%, 20\%\}$ to relax $\bar{C}$. As reported in Table 5.7, TEC is further improved for all TOUs and values of a. SP obtains slightly better results. Figure 5.12 shows that the computational time of SP is significantly larger than the computational time of DP as a and $\bar{C}$ increase.

In addition, we have tested problem instances with extended TOUs of up to 8 intervals and two levels for a. The results, summarized in Table 5.8, confirm that DP and SP determine solutions that are very close in terms of quality, and that the computational times of SP quickly increase

Table 5.6: Computational times when increasing $\bar{C}$ by 5% (II)

TOU	MIP_0	MIP_1	SP	DP	TOU	MIP_0	MIP_1	SP	DP
{1,2,1,2}	–	2523.47	3.33	2.68	{2,1,2,1}	–	2636.18	3.65	2.61
{2,1,2,3}	–	2329.03	6.90	4.42	{3,2,1,2}	–	2248.60	7.55	5.75
{2,1,3}	–	1889.36	5.46	3.09	{3,1,2}	–	1584.23	5.55	3.02

Table 5.7: Comparison of percentage of TEC improvement (RPI) of SP and DP for individual TOUs

TOU	SP						DP					
a	0%	1%	5%	10%	15%	20%	0%	1%	5%	10%	15%	20%
{1,2,1,2}	0.61	0.68	0.85	1.05	1.21	1.30	0.60	0.66	0.77	0.94	1.09	1.15
{2,1,2,1}	1.32	1.92	4.53	7.74	10.96	14.18	1.32	1.90	4.47	7.65	10.86	14.04
{2,1,2,3}	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.47	0.47	0.46	0.46	0.46
{3,2,1,2}	1.08	1.56	3.66	6.31	8.86	11.33	1.07	1.55	3.66	6.30	8.86	11.33
{2,1,3}	0.49	0.50	0.50	0.50	0.50	0.50	0.49	0.49	0.49	0.49	0.49	0.49
{3,1,2}	1.11	1.59	3.73	6.34	8.87	11.38	1.11	1.59	3.72	6.33	8.86	11.37
Mean	0.85	1.12	2.29	3.74	5.15	6.53	0.84	1.11	2.26	3.70	5.10	6.47

with the maximum makespan.

5.7 Benefits of Minimizing Electricity Costs

In Section 5.7.1, the benefits and potential of cost reduction are closely examined with different *TOU*s and values of the maximum makespan. Then, further aspects are analyzed in Section 5.7.2, including the flexibility of the resources in the instances, alternative sequences, and their impact on the algorithm performance.

Table 5.8: Comparison of DP and SP for more than 4 *TOU* intervals (RPI)

TOU	$a = 0\%$				$a = 10\%$			
	SP		DP		SP		DP	
	RPI	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time
{2,1,3,1,2}	2.44	0.39	2.41	2.49	4.67	21.29	5.81	8.12
{2,1,2,3,1,2}	2.27	0.43	2.22	3.88	4.62	20.86	5.54	8.78
{2,1,2,1,2,1,2}	4.02	0.43	3.97	4.12	1.80	36.68	1.36	9.06
{2,1,2,3,2,1,2,3}	5.87	0.53	5.83	5.39	1.47	59.45	1.06	9.80
Mean	3.65	0.45	3.61	3.97	3.14	34.57	3.45	8.94

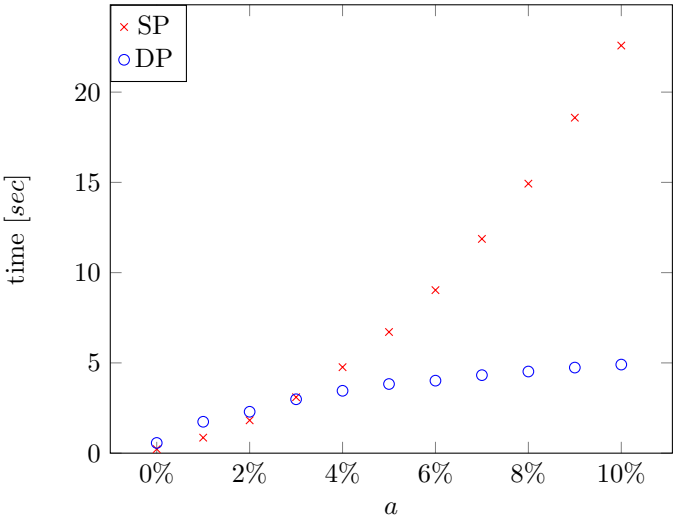

Figure 5.12: Comparison of computational times of SP and DP depending on a.

5.7.1 Potential of TEC Reduction

We can observe from Table 5.7 that the TEC improvement varies widely for different TOU settings. More specifically, Tables 5.9 and 5.10 provide the min, max and mean percentage improvement according to TOUs. Also, the tables are roughly divided into three categories which show strong, distinct and slight correlation between the TEC improvement and TOUs.

When the price is increasing in the final phase of TOU, relaxing $\bar{C}$ only leads to a minor TEC improvement. On the other hand, if the prices are fluctuating, the cost reduction becomes promising, around 1% without degrading the minimum makespan and over 10% with a 20% increase of the minimum makespan. Finally, with small prices in the final phase of TOU, major TEC improvements up to 20% are achieved.

In general, even when TOU settings have more and irregular intervals as in Table 5.11, a significant TEC improvement is possible.

In addition, we investigated the selection of a in smaller steps: $a \in \{1\%, 2\%, \ldots, 10\%\}$. Figure 5.13 shows similar patterns with three major groups according to TOUs.

For illustration purposes, we next use several Gantt charts to visualize the modified schedules and TEC improvements. Figure 5.14 shows the substantial TEC improvement for problem instance la01 after increasing the makespan by 20%. For the same problem instance, Figure 5.15 shows that TEC can still be improved when the best known makespan is the maximum allowed makespan. On the other hand, Figure 5.16 shows that, despite increasing prices in the extended period, TEC

Table 5.9: Min, max, and mean TEC improvement (RPI) (I)

TOU	a								
	0%			1%			5%		
	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
{1,2,3,4}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
{2,1,2,3}	0.00	2.38	0.48	0.00	2.38	0.48	0.00	2.38	0.48
{2,1,3}	0.00	2.83	0.49	0.00	2.83	0.50	0.00	2.83	0.50
{1,2,1,2}	0.00	3.12	0.61	0.00	3.18	0.68	0.00	3.32	0.85
{1,2,3,2}	0.00	3.06	0.61	0.00	3.06	1.05	1.90	5.61	3.06
{3,2,1,2}	0.06	4.02	1.08	0.43	4.41	1.56	2.47	6.57	3.66
{3,1,2}	0.06	5.06	1.11	0.32	5.47	1.59	1.89	8.64	3.73
{2,1,2,1}	0.00	5.50	1.32	0.00	5.50	1.92	2.79	8.56	4.53
{2,3,2,1}	0.10	0.10	1.14	0.30	0.30	1.96	4.06	4.06	5.65
{4,3,2,1}	0.13	5.41	1.37	0.46	5.69	2.35	4.56	10.12	6.70

Table 5.10: Min, max, and mean TEC improvement (RPI) (II)

TOU	a = 10%			a = 15%			a = 20%		
	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
{1,2,3,4}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
{2,1,2,3}	0.00	2.38	0.48	0.00	2.38	0.48	0.00	2.38	0.48
{2,1,3}	0.00	2.83	0.50	0.00	2.83	0.50	0.00	2.83	0.50
{1,2,1,2}	0.00	4.95	1.05	0.00	6.52	1.21	0.00	7.05	1.30
{1,2,3,2}	3.81	8.16	5.57	6.98	9.93	8.07	8.57	12.61	10.46
{3,2,1,2}	4.42	10.22	6.31	6.98	13.80	8.86	9.41	16.45	11.33
{3,1,2}	4.10	11.57	6.34	4.88	13.96	8.87	7.05	15.79	11.38
{2,1,2,1}	5.18	11.62	7.74	9.45	15.60	10.96	12.35	18.65	14.18
{2,3,2,1}	7.54	7.54	10.26	13.65	13.65	14.88	15.64	15.64	19.30
{4,3,2,1}	8.66	15.56	12.14	16.11	21.64	17.58	20.50	26.39	22.96

Table 5.11: Minimum and Maximum values of RPI for more than 4 TOU intervals

TOU	$a = 0\%$		$a = 10\%$	
	Min	Max	Min	Max
{2,1,3,1,2}	0.47	6.89	0.95	10.80
{2,1,2,3,1,2}	0.11	7.21	0.69	10.36
{2,1,2,1,2,1,2}	1.21	8.89	0.00	6.21
{2,1,2,3,2,1,2,3}	0.60	11.60	0.00	5.56

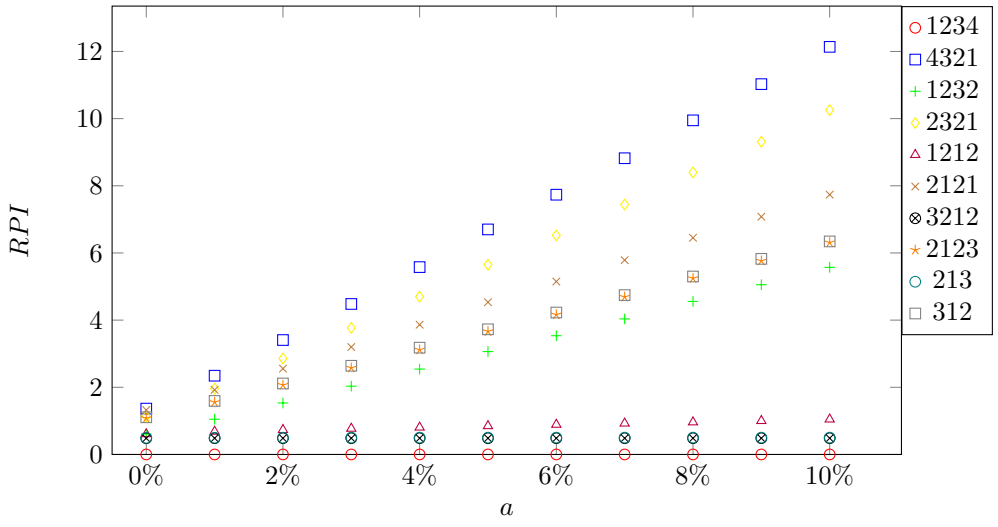

Figure 5.13: Percentage TEC improvement (RPI) with additional values of a

can be reduced.

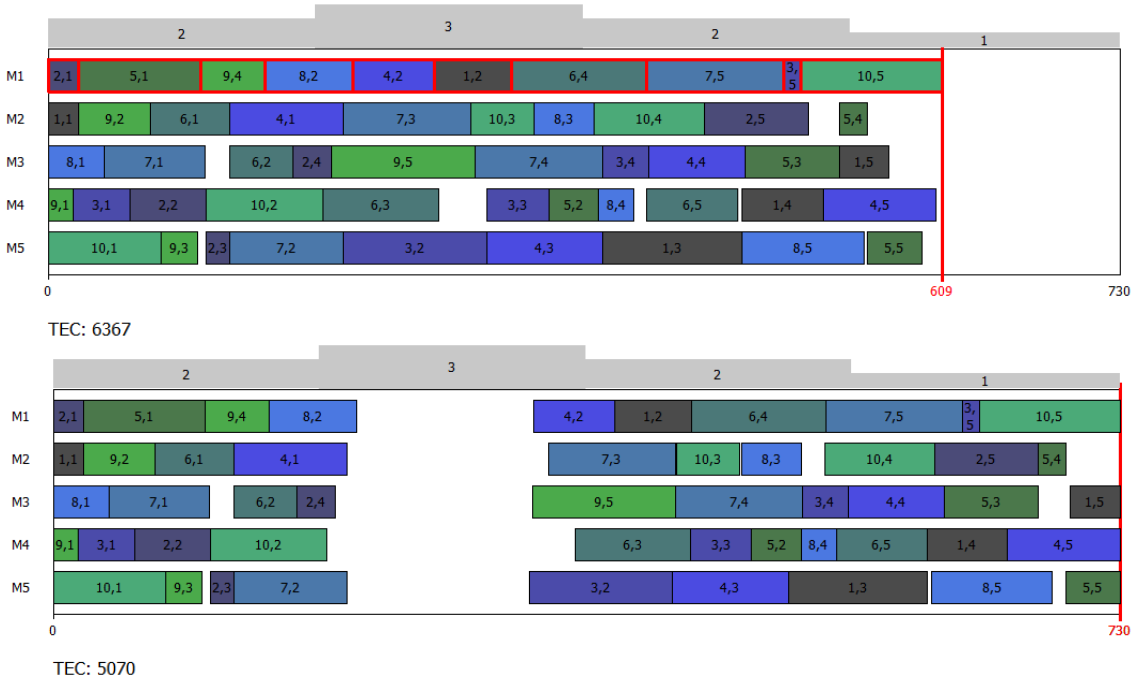

Figure 5.14: TEC improvement for instance la01 ($\bar{C} > C^*_{max}$)

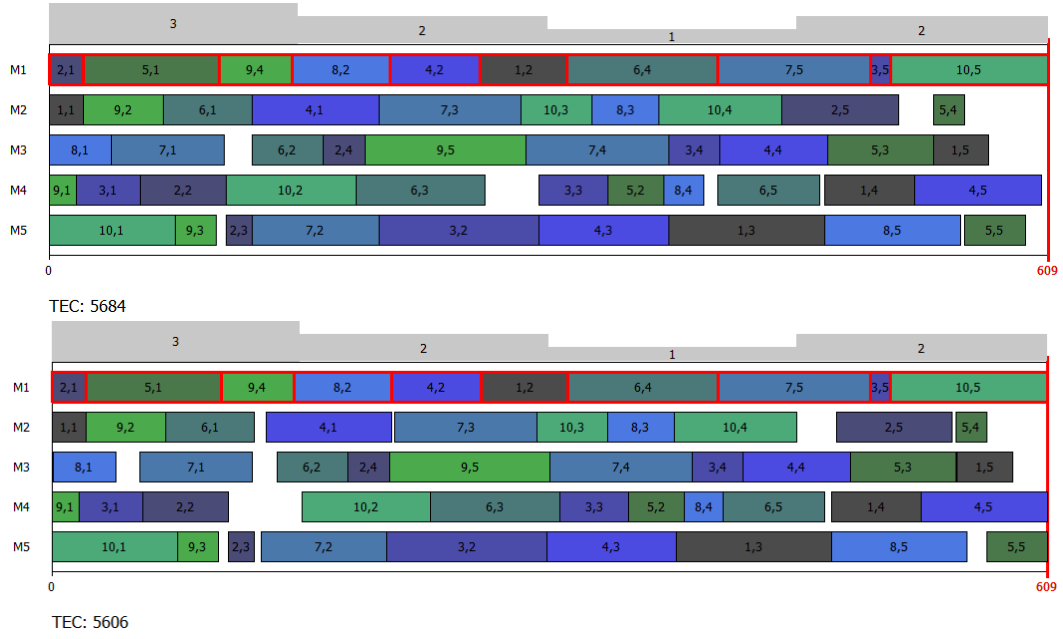

Figure 5.15: TEC improvement for instance la01 ($\bar{C} = C^*_{max}$)

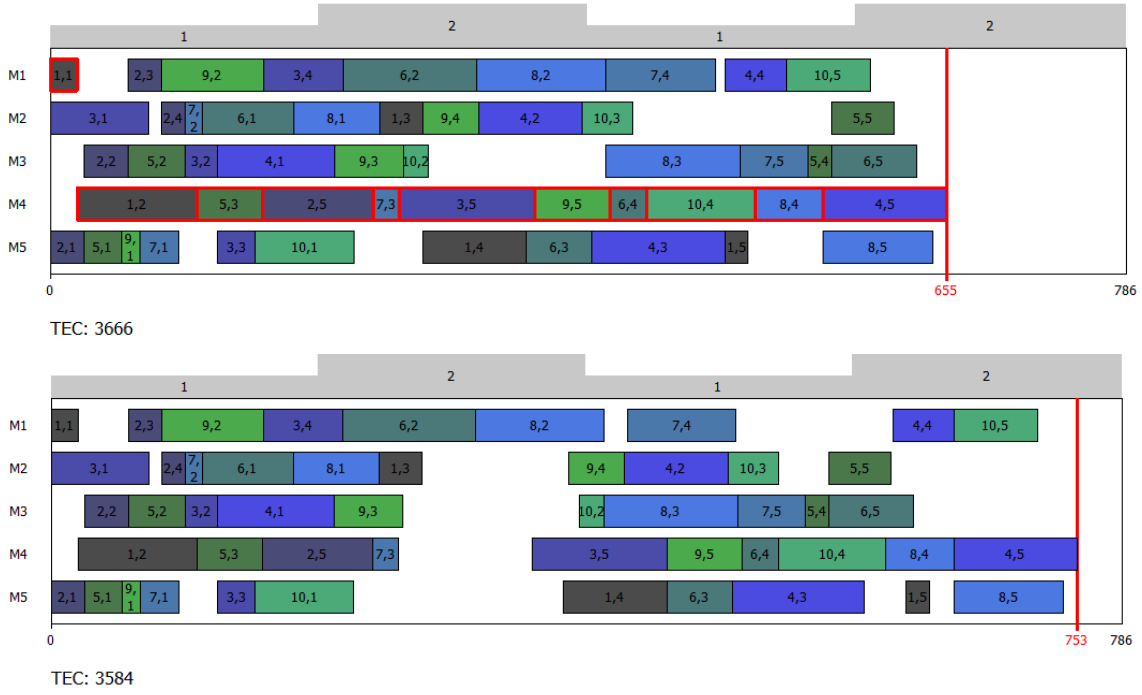

Figure 5.16: TEC improvement for instance la02 despite increasing prices

5.7.2 Further Analysis

In this section, we analyze some additional aspects of our problem. Figure 5.17 summarizes the relative TEC improvement according to the problem sizes. The consistent pattern suggests that the TEC improvement is relatively stable with the increase of the maximum makespan.

In addition, we sorted out the Hurink instances (Hurink et al., 1994) to analyze the TEC improvement potential in relation to the flexibility of the resources in the instances. All TOU combinations are selected and $a \in \{5\%, 15\%\}$. All instances in sdata, edata, rdata, and vdata are tested, and the average value is used for comparison. It can be seen from Table 5.12 that the RPI slightly varies with the flexibility of the resources. No major differences are observed.

For comparison, we next use alternative sequences of the flexible job-shop scheduling problem such that $C_{max} \leq \bar{C}$. In Table 5.13, the improvement is calculated by using the initial TEC of the alternative sequence with a makespan close to $\bar{C}$. It is interesting to compare the results with the TEC improvement for a sequence with the best known makespan C^*_{max}. In this respect, results are clearly divided into two parts where negative values are present for TOUs with increasing prices in the last intervals. This is due to the fact that alternative sequences with makespan close to $\bar{C}$ have more operations processed in the final periods. If these operations are time-critical, then it

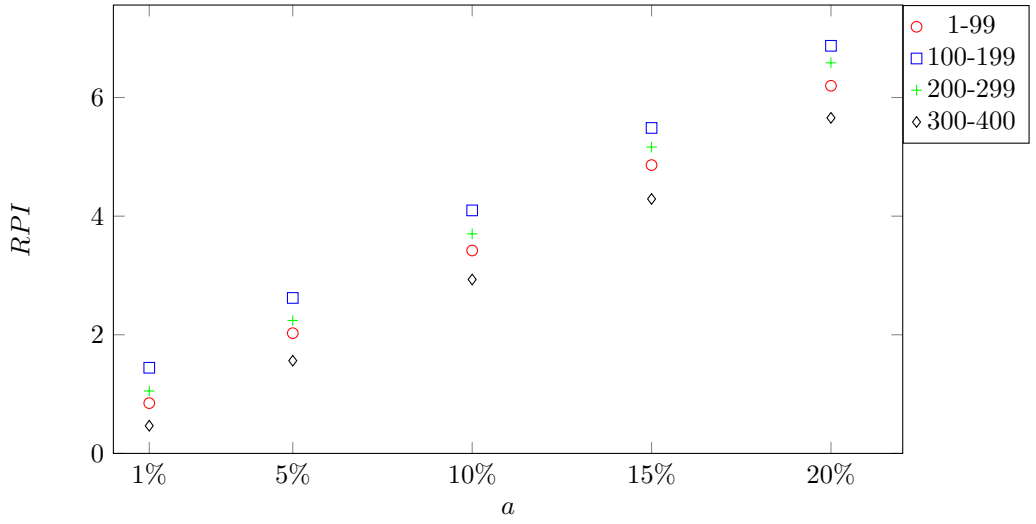

Figure 5.17: Percentage TEC improvement (RPI) based on a and number of operations

Table 5.12: Percentage TEC improvement (RPI) for Hurink instances with diverse flexibility

TOU		sdata		edata		rdata		vdata	
	a	5%	15%	5%	15%	5%	15%	5%	15%
{1,2,3,4}		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
{4,3,2,1}		7.24	18.28	6.69	17.52	6.71	17.51	7.76	18.70
{1,2,3,2}		3.08	8.01	3.04	8.06	3.14	8.17	3.31	8.21
{2,3,2,1}		5.97	15.25	5.63	14.76	5.69	14.80	6.39	15.67
{1,2,1,2}		1.59	2.46	0.81	1.03	0.64	0.74	2.12	3.17
{2,1,2,1}		4.89	11.32	4.40	10.83	4.47	10.87	5.56	12.08
{2,1,2,3}		0.71	0.71	0.42	0.42	0.37	0.37	0.96	0.96
{3,2,1,2}		4.62	10.38	3.67	8.87	3.48	8.49	5.19	10.78
{2,1,3}		0.82	0.84	0.50	0.50	0.37	0.37	0.99	0.99
{3,1,2}		4.66	10.02	3.84	8.92	3.45	8.57	4.99	10.67

is impossible to move them. For the other combinations, the TEC improvement remains close for each a. This suggests that, by using a sequence with the best known makespan C^*_{max}, the result strongly depends on the initial sequence and TOU settings. On the other hand, good results are obtained with all TOU settings when using alternative sequences which do not violate $\bar{C}$.

5.8 Summary and Research Perspectives

In this research, we investigated the potential of energy cost savings in the flexible job shop environment, where the sequence of operations on the machines is fixed and ending the schedule after its minimum makespan is allowed. By considering diverse tariffs-of-use, we derived propositions and lemmata to solve special cases to optimality. These properties also help to formalize the problem as a mixed integer linear program. We also proposed two heuristic approaches which are able to obtain high-quality solutions in small computational times for the most complex cases. Numerical results also confirm that substantial TEC savings can be achieved with a relatively small increase, or even no increase, of the minimum makespan.

Our observation and experiments suggest that the most general case of the problem is not polynomially solvable. Its complexity status deserves further investigation. More importantly, and as illustrated in our numerical experiments, changing the assignment and the sequence of operations lead to substantial additional TEC savings. Also, the non-regularity of the TEC criterion makes it a challenging problem to explore, since a left-shifted schedule is not always optimal for a given sequence of jobs. Indeed, not only the jobs must be assigned and sequenced on the machines, but

Table 5.13: Percentage TEC improvement based on alternative sequences

TOU		$C_{max} \leq \bar{C}$			$C_{max} = C^*_{max}$		
	a	1%	5%	10%	1%	5%	10%
{1,2,3,4}		0.00	0.00	0.00	-0.47	-3.19	-6.28
{1,2,3,2}		0.55	0.70	0.81	0.51	0.37	0.22
{1,2,1,2}		0.59	0.70	0.89	0.38	-0.83	-2.18
{2,1,2,3}		0.49	0.53	0.62	0.10	-1.98	-4.33
{2,1,3}		0.50	0.59	0.70	0.22	-1.79	-4.04
{4,3,2,1}		1.25	1.55	1.89	1.59	3.80	6.29
{2,3,2,1}		1.02	1.29	1.58	1.30	3.05	5.05
{2,1,2,1}		1.25	1.53	1.83	1.41	2.67	4.12
{3,2,1,2}		1.02	1.22	1.49	1.04	1.35	1.68
{3,1,2}		1.05	1.30	1.56	1.12	1.49	1.81

an optimal schedule must also be determined.

Hence, our current research aims at combining the heuristics introduced in this research with solution methods for the traditional flexible job shop scheduling problem. The objective is still to minimize the total energy cost with a constraint on the maximum allowed makespan, but we believe studying the minimization of both the total energy cost and the makespan is a relevant research avenue.

Chapter 6

Conclusions

In this thesis, to address recent developments in industry, the integration of job delivery times into machine scheduling was studied. More specifically, three related optimization problems in the context of parallel machine environments were formulated which are motivated by the new manufacturing paradigm *cloud manufacturing*. Additionally, a special case of the flexible job shop scheduling problem was investigated, where the objective is to minimize the total energy cost in the presence of time-of-use energy pricing schemes, subject to a bound on the makespan of the schedule.

For the studied problems, mathematical programming formulations were proposed as a means to derive optimal solutions for small-sized instances. Furthermore, the problems were analyzed from a theoretical perspective. To tackle instances of practical size, multiple (meta-)heuristic approaches were proposed for each problem which integrate the findings from the theoretical analyses to increase efficiency and quality of the solution procedures. To evaluate these approaches, extensive computational experiments were conducted.

To be precise, for the identical parallel machine scheduling problem with machine-dependent delivery times and total weighted tardiness minimization in Chapter 2, structural properties were identified that can be exploited to significantly increase the efficiency of local search procedures. Those properties were integrated in a variable neighborhood search (VNS) algorithm.

After modification, this VNS could also be applied to the unrelated parallel machine problem with job-machine-dependent delivery times and total weighted completion time minimization in Chapter 3. However, computational experiments showed that this approach was outperformed by a tabu search algorithm employing a data structure tailored to the underlying assignment problem.

Furthermore, the VNS approach incorporating problem-specific properties could also be adapted for the unrelated parallel machine problem with job-machine-dependent delivery times, eligibility constraints, and total weighted tardiness minimization in Chapter 4, which generalizes the problem from Chapter 2. Not only are the previous findings applied to this problem, but also new properties are formulated that can help to improve local search based solution procedures.

With respect to the special case of the flexible job shop problem involving time-of-use energy prices in Chapter 5, the theoretical analysis identified certain cases of pricing structures, for which optimal solutions can be determined in polynomial time. Moreover, general properties of optimal solutions were formulated based on which heuristic approaches were devised. An extensive computational study compared the different approaches and investigated the potential of energy cost savings through a relaxation of the maximum makespan.

For the integration of delivery aspects into machine scheduling, there exist multiple perspectives for future research. While this thesis considers delivery times only in the context of parallel machine problems, other production environments such as flow shops or job shops are of interest as well. Furthermore, additional constraints should be taken into account such as a limited number of transportation vehicles and vehicle capacity to increase practical relevance.

In case of the flexible job shop problem with time-of-use energy prices, several aspects deserve further investigation, too. First of all, although the results suggest that the special case of the problem considered in Chapter 5 is NP-hard, it is not yet formally proven. Another aspect of interest is the number of price intervals during the planning horizon, which remains limited in this study. In this context, multiple pricing cycles deserve consideration. Furthermore, based on the results from this study, the development of comprehensive approaches, that do not just operate on a given solution with a fixed sequence, but also alter it, is required in order to solve the general flexible job shop problem as well.

Chapter 7

Bibliography

Afzalirad, M. and Rezaeian, J. (2016). Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. *Computers & Industrial Engineering*, 98:40–52.

Afzalirad, M. and Rezaeian, J. (2017). A realistic variant of bi-objective unrelated parallel machine scheduling problem: Nsga-ii and moaco approaches. *Applied Soft Computing*, 50:109–123.

Afzalirad, M. and Shafipour, M. (2018). Design of an efficient genetic algorithm for resource-constrained unrelated parallel machine scheduling problem with machine eligibility restrictions. *Journal of Intelligent Manufacturing*, 29(2):423–437.

Ahmadizar, F. and Farhadi, S. (2015). Single-machine batch delivery scheduling with job release dates, due windows and earliness, tardiness, holding and delivery costs. *Computers and Operations Research*, 53:194–205.

Akbar, M. and Irohara, T. (2018). Scheduling for sustainable manufacturing: A review. *Journal of Cleaner Production*, 205:866–883.

Alidaee, B. and Rosa, D. (1997). Scheduling parallel machines to minimize total weighted and unweighted tardiness. *Computers and Operations Research*, 24(8):775–788.

Anghinolfi, D. and Paolucci, M. (2007). Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. *Computers and Operations Research*, 34(11):3471–3490.

Azizoglu, M. and Kirca, O. (1998). Tardiness minimization on parallel machines. *International Journal of Production Economics*, 55(2):163–168.

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21.

Baker, K. R. and Bertrand, J. (1982). A dynamic priority rule for scheduling against due-dates. *Journal of Operations Management*, 3(1):37–42.

Barnes, J. W. and Chambers, J. B. (1996). Flexible job shop scheduling by tabu search. Technical report, University of Texas, Austin.

Behnamian, J., Zandieh, M., and Ghomi, S. F. (2009). Parallel-machine scheduling problems with sequence-dependent setup times using an aco, sa and vns hybrid algorithm. *Expert Systems with Applications*, 36(6):9637–9644.

Biel, K. and Glock, C. H. (2016). Systematic literature review of decision support models for energy-efficient production planning. *Computers & Industrial Engineering*, 101:243–259.

Biskup, D., Herrmann, J., and Gupta, J. N. (2008). Scheduling identical parallel machines to minimize total tardiness. *International Journal of Production Economics*, 115(1):134–142.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41:57–83.

Cakici, E., Mason, S. J., Geismar, H. N., and Fowler, J. W. (2014). Scheduling parallel machines with single vehicle delivery. *Journal of Heuristics*, 20(5):511–537.

Carlier, J. (1987). Scheduling jobs with release dates and tails on identical machines to minimize the makespan. *European Journal of Operational Research*, 29(3):298–306.

Centeno, G. and Armacost, R. L. (1997). Parallel machine scheduling with release time and machine eligibility restrictions. *Computers & industrial engineering*, 33(1-2):273–276.

Centeno, G. and Armacost, R. L. (2004). Minimizing makespan on parallel machines with release time and machine eligibility restrictions. *International Journal of Production Research*, 42(6):1243–1256.

Chang, Y.-C. and Lee, C.-Y. (2004). Machine scheduling with job delivery coordination. *European Journal of Operational Research*, 158(2):470–487.

Che, A., Zeng, Y., and Lyu, K. (2016). An efficient greedy insertion heuristic for energy-conscious single machine scheduling problem under time-of-use electricity tariffs. *Journal of Cleaner Production*, 129:565–577.

Che, A., Zhang, S., and Wu, X. (2017). Energy-conscious unrelated parallel machine scheduling under time-of-use electricity tariffs. *Journal of Cleaner Production*, 156:688–697.

Chen, C.-L. and Chen, C.-L. (2009). Hybrid metaheuristics for unrelated parallel machine scheduling with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 43(1-2):161.

Chen, Y., Lu, L., and Yuan, J. (2015). Preemptive scheduling on identical machines with delivery coordination to minimize the maximum delivery completion time. *Theoretical Computer Science*, 583:67–77.

Chen, Y., Lu, L., and Yuan, J. (2016). Two-stage scheduling on identical machines with assignable delivery times to minimize the maximum delivery completion time. *Theoretical Computer Science*, 622:45–65.

Chen, Y.-Y., Cheng, C.-Y., Wang, L.-C., and Chen, T.-L. (2013). A hybrid approach based on the variable neighborhood search and particle swarm optimization for parallel machine scheduling problems—a case study for solar cell industry. *International Journal of Production Economics*, 141(1):66–78.

Cheng, R., Gen, M., and Tozawa, T. (1995). Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithms. *Computers & Industrial Engineering*, 29(1-4):513–517.

Cheng, W., Guo, P., Zhang, Z., Zeng, M., and Liang, J. (2012). Variable neighborhood search for parallel machines scheduling problem with step deteriorating jobs. *Mathematical Problems in Engineering*, 2012.

Dauzère-Pérès, S. and Paulli, J. (1997). An integrated approach for modelling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70:281–306.

De Paula, M. R., Ravetti, M. G., Mateus, G. R., and Pardalos, P. M. (2007). Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics*, 18(2):101–115.

Demirkol, E., Mehta, S., and Uzsoy, R. (1998). Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109:137–141.

Ding, J.-Y., Song, S., Zhang, R., Chiong, R., and Wu, C. (2016). Parallel machine scheduling under time-of-use electricity prices: New models and optimization approaches. *IEEE Transactions on Automation Science and Engineering*, 13(2):1138–1154.

Dong, J., Zhang, A., Chen, Y., and Yang, Q. (2013). Approximation algorithms for two-machine open shop scheduling with batch and delivery coordination. *Theoretical Computer Science*, 491:94–102.

Driessel, R. and Mönch, L. (2011). Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times. *Computers & Industrial Engineering*, 61(2):336–345.

Fang, Y., Liu, P., and Lu, X. (2011a). Optimal on-line algorithms for one batch machine with grouped processing times. *Journal of Combinatorial Optimization*, 22(4):509–516.

Fang, Y., Lu, X., and Liu, P. (2011b). Online batch scheduling on parallel machines with delivery times. *Theoretical Computer Science*, 412(39):5333–5339.

Gahm, C., Denz, F., Dirr, M., and Tuma, A. (2016). Energy-efficient scheduling in manufacturing companies: a review and research framework. *European Journal of Operational Research*, 248(3):744–757.

Gao, J. (2010). A novel artificial immune system for solving multiobjective scheduling problems subject to special process constraint. *Computers & Industrial Engineering*, 58(4):602–609.

Gharbi, A. and Haouari, M. (2002). Minimizing makespan on parallel machines subject to release dates and delivery times. *Journal of Scheduling*, 5(4):329–355.

Gharbi, A. and Haouari, M. (2007). An approximate decomposition algorithm for scheduling on parallel machines with heads and tails. *Computers and Operations Research*, 34(3):868–883.

Gokhale, R. and Mathirajan, M. (2012). Scheduling identical parallel machines with machine eligibility restrictions to minimize total weighted flowtime in automobile gear manufacturing. *The International Journal of Advanced Manufacturing Technology*, 60(9-12):1099–1110.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.

Gurobi Optimization, L. L. C. (2021). Gurobi optimizer reference manual.

Hall, L. A. and Shmoys, D. B. (1992). Jackson's rule for single-machine scheduling: Making a good heuristic better. *Mathematics of Operations Research*, 17(1):22–35.

Hansen, P., Mladenović, N., and Pérez, J. A. M. (2010). Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407.

Hansen, P., Mladenović, N., Todosijević, R., and Hanafi, S. (2017). Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454.

He, Y., Zhong, W., and Gu, H. (2006). Improved algorithms for two single machine scheduling problems. *Theoretical Computer Science*, 363(3):257–265.

Hoogeveen, J. and Vestjens, A. P. (2000). A best possible deterministic on-line algorithm for minimizing maximum delivery time on a single machine. *SIAM Journal on Discrete Mathematics*, 13(1):56–63.

Hurink, J., Jurisch, B., and Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum*, 15:205–215.

Jackson, J. R. (1955). Scheduling a production line to minimize maximum tardiness. *Management Science Research Project*.

Kanet, J. J. (2007). New precedence theorems for one-machine weighted tardiness. *Mathematics of Operations Research*, 32(3):579–588.

Koulamas, C. (1997). Decomposition and hybrid simulated annealing heuristics for the parallel-machine total tardiness problem. *Naval Research Logistics (NRL)*, 44(1):109–125.

Koulamas, C. and Kyparisis, G. J. (2010). Single-machine scheduling problems with past-sequence-dependent delivery times. *International Journal of Production Economics*, 126(2):264–266.

Lee, C.-Y. and Chen, Z.-L. (2001). Machine scheduling with transportation considerations. *Journal of Scheduling*, 4(1):3–24.

Li, C.-L., Vairaktarakis, G., and Lee, C.-Y. (2005). Machine scheduling with deliveries to multiple customer locations. *European Journal of Operational Research*, 164(1):39–51.

Liao, L.-W. and Sheen, G.-J. (2008). Parallel machine scheduling with machine availability and eligibility constraints. *European Journal of Operational Research*, 184(2):458–467.

Liaw, C.-F., Lin, Y.-K., , C.-Y., and Chen, M. (2003). Scheduling unrelated parallel machines to minimize total weighted tardiness. *Computers and Operations Research*, 30(12):1777–1789.

Lin, C.-W., Lin, Y.-K., and Hsieh, H.-T. (2013). Ant colony optimization for unrelated parallel machine scheduling. *International Journal of Advanced Manufacturing Technology*, 67:35–45.

Lin, Y.-K., Pfund, M. E., and Fowler, J. W. (2011). Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. *Computers and Operations Research*, 38(6):901–916.

Liu, M., Zheng, F., Chu, C., and Xu, Y. (2012). New results on single-machine scheduling with past-sequence-dependent delivery times. *Theoretical Computer Science*, 438:55–61.

Liu, P. and Lu, X. (2011). An improved approximation algorithm for single machine scheduling with job delivery. *Theoretical Computer Science*, 412(3):270–274.

Liu, P. and Lu, X. (2015). Online scheduling on two parallel machines with release dates and delivery times. *Journal of Combinatorial Optimization*, 30(2):347–359.

Liu, Y., Dong, H., Lohse, N., and Petrovic, S. (2015). Reducing environmental impact of production during a rolling blackout policy–a multi-objective schedule optimisation approach. *Journal of Cleaner Production*, 102:418–427.

Lu, L. and Yuan, J. (2008a). Single machine scheduling with job delivery to minimize makespan. *Asia-Pacific Journal of Operational Research*, 25(01):1–10.

Lu, L. and Yuan, J. (2008b). Unbounded parallel batch scheduling with job delivery to minimize makespan. *Operations Research Letters*, 36(4):477–480.

Luo, H., Du, B., Huang, G. Q., Chen, H., and Li, X. (2013). Hybrid flow shop scheduling considering machine electricity consumption cost. *International Journal of Production Economics*, 146(2):423–439.

Maecker, S. and Shen, L. (2020). Solving parallel machine problems with delivery times and tardiness objectives. *Annals of Operations Research*, 285(1-2):315–334.

Maecker, S., Shen, L., Alidaee, B., and Wang, H. (2020). Unrelated parallel machine scheduling in distributed manufacturing systems. In *Proceedings OLA'2020 International Conference on Optimization and Learning*, pages 69–71.

Maecker, S., Shen, L., and Mönch, L. (2021). Unrelated parallel machine scheduling with eligibility constraints and delivery times to minimize total weighted tardiness. *Unpublished working paper: Currently under second review after the first revision at Computers and Operations Research journal.*

Maggu, P. and Das, G. (1980). On 2× n sequencing problem with transportation times of jobs. *Pure and Applied Mathematika Sciences*, 12(1):6.

Mateo, M., Teghem, J., and Tuyttens, D. (2018). A bi-objective parallel machine problem with eligibility, release dates and delivery times of the jobs. *International Journal of Production Research*, 56(3):1030–1053.

Mati, Y., Dauzère-Pérès, S., and Lahlou, C. (2011). A general approach for optimizing regular criteria in the job-shop scheduling problem. *European Journal of Operational Research*, 212(1):33–42.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100.

Mönch, L. (2008). Heuristics to minimize total weighted tardiness of jobs on unrelated parallel machines. In *Proceedings of the 4th IEEE Conference on Automation Science and Engineering*, pages 572–577.

Mönch, L. and Shen, L. (2020). Parallel machine scheduling with the total weighted delivery time performance measure in distributed manufacturing. *Computers and Operations Research*, 127:105126.

Moon, J.-Y. and Park, J. (2014). Smart production scheduling with time-dependent and machine-dependent electricity cost by considering distributed energy resources and energy storage. *International Journal of Production Research*, 52(13):3922–3939.

Moon, J.-Y., Shin, K., and Park, J. (2013). Optimization of production scheduling with time-dependent and machine-dependent electricity cost for industrial energy efficiency. *The International Journal of Advanced Manufacturing Technology*, 68(1-4):523–535.

Moscato, P. and Norman, M. G. (1992). A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. *Parallel Computing and Transputer Applications*, 1:177–186.

Panwalkar, S., Smith, M., and Koulamas, C. (1993). A heuristic for the single machine tardiness problem. *European Journal of Operational Research*, 70(3):304–310.

Pei, J., Pardalos, P. M., Liu, X., Fan, W., and Yang, S. (2015). Serial batching scheduling of deteriorating jobs in a two-stage supply chain to minimize the makespan. *European Journal of Operational Research*, 244(1):13–25.

Pfund, M., Fowler, J. W., and Gupta, J. N. (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.

Pinedo, M. L. (2016). *Scheduling: Theory, Algorithms, and Systems*. Springer.

Potts, C. N. (1980). Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research*, 28(6):1436–1441.

Potts, C. N. and Van Wassenhove, L. (1982). A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters*, 1(5):177–181.

Radcliffe, N. J. and Surry, P. D. (1994). Formal memetic algorithms. In *AISB Workshop on Evolutionary Computing*, pages 1–16.

Rubaiee, S., Cinar, S., and Yildirim, M. B. (2018). An energy-aware multiobjective optimization framework to minimize total tardiness and energy cost on a single-machine nonpreemptive scheduling. *IEEE Transactions on Engineering Management*, 66(4):699–714.

Schulz, S., Buscher, U., and Shen, L. (2020). Multi-objective hybrid flow shop scheduling with variable discrete production speed levels and time-of-use energy prices. *Journal of Business Economics*, pages 1–29.

Sheen, G.-J., Liao, L.-W., and Lin, C.-F. (2008). Optimal parallel machines scheduling with machine availability and eligibility constraints. *The International Journal of Advanced Manufacturing Technology*, 36(1-2):132–139.

Shen, L., Dauzère-Pérès, S., and Maecker, S. (2021). Energy efficient scheduling for a fixed sequence in flexible job-shop manufacturing systems. *Unpublished working paper: Currently in second revision at European Journal of Operational Research.*

Shen, L., Dauzère-Pérès, S., and Neufeld, J. S. (2018). Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 265:503–516.

Shim, S.-O. and Kim, Y.-D. (2007). Scheduling on parallel identical machines to minimize total tardiness. *European Journal of Operational Research*, 177(1):135–146.

Shrouf, F., Ordieres-Meré, J., García-Sánchez, A., and Ortega-Mier, M. (2014). Optimizing the production scheduling of a single machine to minimize total energy consumption costs. *Journal of Cleaner Production*, 67:197–207.

Smith, W. E. et al. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66.

Srinivasa Raghavan, N. and Venkataramana, M. (2009). Parallel processor scheduling for minimizing total weighted tardiness using ant colony optimization. *The International Journal of Advanced Manufacturing Technology*, 41(9-10):986–996.

Su, C.-S., Pan, J. C.-H., and Hsu, T.-S. (2009). A new heuristic algorithm for the machine scheduling problem with job delivery coordination. *Theoretical Computer Science*, 410(27-29):2581–2591.

Su, H., Pinedo, M., and Wan, G. (2017). Parallel machine scheduling with eligibility constraints: A composite dispatching rule to minimize total weighted tardiness. *Naval Research Logistics (NRL)*, 64(3):249–267.

Tao, F., Zhang, L., Venkatesh, V., Luo, Y., and Cheng, Y. (2011). Cloud manufacturing: a computing and service-oriented manufacturing model. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 225(10):1969–1976.

Tasgetiren, M. F., Pan, Q.-K., and Liang, Y.-C. (2009). A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. *Computers and Operations Research*, 36(6):1900–1915.

Tian, J., Cheng, T., Ng, C., and Yuan, J. (2012). An improved on-line algorithm for single parallel-batch machine scheduling with delivery times. *Discrete Applied Mathematics*, 160(7-8):1191–1210.

Tian, J., Fu, R., and Yuan, J. (2007). On-line scheduling with delivery time on a single batch machine. *Theoretical Computer Science*, 374(1-3):49–57.

Tian, J., Fu, R., and Yuan, J. (2008). A best on-line algorithm for single machine scheduling with small delivery times. *Theoretical Computer Science*, 393(1-3):287–293.

Tian, J., Fu, R., and Yuan, J. (2011). An on-line algorithm for the single machine unbounded parallel-batching scheduling with large delivery times. *Information Processing Letters*, 111(21-22):1048–1053.

Vepsalainen, A. P. and Morton, T. E. (1987). Priority rules for job shops with weighted tardiness costs. *Management Science*, 33(8):1035–1047.

Wang, H. and Alidaee, B. (2019). Effective heuristic for large-scale unrelated parallel machines scheduling problems. *Omega*, 83:261–274.

Wang, I.-L., Wang, Y.-C., and Chen, C.-W. (2013). Scheduling unrelated parallel machines in semiconductor manufacturing by problem reduction and local search heuristics. *Flexible Services and Manufacturing Journal*, 25(3):343–366.

Wang, S., Wang, X., Yu, J., Ma, S., and Liu, M. (2018). Bi-objective identical parallel machine scheduling to minimize total energy consumption and makespan. *Journal of Cleaner Production*, 193:424–440.

Wang, X. and Cheng, T. E. (2007). Machine scheduling with an availability constraint and job delivery coordination. *Naval Research Logistics (NRL)*, 54(1):11–20.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics*, 1(6):80–83.

Woeginger, G. J. (1994). Heuristics for parallel machine scheduling with delivery times. *Acta Informatica*, 31(6):503–512.

Wu, D., Greer, M. J., Rosen, D. W., and Schaefer, D. (2013). Cloud manufacturing: Strategic vision and state-of-the-art. *Journal of Manufacturing Systems*, 32(4):564–579.

Wu, D., Rosen, D. W., Wang, L., and Schaefer, D. (2015). Cloud-based design and manufacturing: A new paradigm in digital manufacturing and design innovation. *Computer-Aided Design*, 59:1–14.

Xu, H., Lü, Z., and Cheng, T. (2014). Iterated local search for single-machine scheduling with sequence-dependent setup times to minimize total weighted tardiness. *Journal of Scheduling*, 17(3):271–287.

Xu, X. (2012). From cloud computing to cloud manufacturing. *Robotics and Computer-Integrated Manufacturing*, 28(1):75–86.

Yalaoui, F. and Chu, C. (2002). Parallel machine scheduling to minimize total tardiness. *International Journal of Production Economics*, 76(3):265–279.

Yuan, J., Li, S., Tian, J., and Fu, R. (2009). A best on-line algorithm for the single machine parallel-batch scheduling with restricted delivery times. *Journal of Combinatorial Optimization*, 17(2):206–213.

Zeng, Y., Che, A., and Wu, X. (2018). Bi-objective scheduling on uniform parallel machines considering electricity cost. *Engineering Optimization*, 50(1):19–36.

Zhang, H., Dai, Z., Zhang, W., Zhang, S., Wang, Y., and Liu, R. (2017). A new energy-aware flexible job shop scheduling method using modified biogeography-based optimization. *Mathematical Problems in Engineering*, 2017.

Zhang, H., Fang, Y., Pan, R., and Ge, C. (2018). A new greedy insertion heuristic algorithm with a multi-stage filtering mechanism for energy-efficient single machine scheduling problems. *Algorithms*, 11(2):18.

Zhang, H., Zhao, F., Fang, K., and Sutherland, J. W. (2014a). Energy-conscious flow shop scheduling under time-of-use electricity tariffs. *CIRP Annals*, 63(1):37–40.

Zhang, L., Luo, Y., Tao, F., Li, B. H., Ren, L., Zhang, X., Guo, H., Cheng, Y., Hu, A., and Liu, Y. (2014b). Cloud manufacturing: a new manufacturing paradigm. *Enterprise Information Systems*, 8(2):167–187.

Zheng, X., Zhou, S., Xu, R., and Chen, H. (2020). Energy-efficient scheduling for multi-objective two-stage flow shop using a hybrid ant colony optimisation algorithm. *International Journal of Production Research*, 58(13):4103–4120.

Zhong, W., Dósa, G., and Tan, Z. (2007). On the machine scheduling problem with job delivery coordination. *European Journal of Operational Research*, 182(3):1057–1072.

Zhou, H., Li, Z., and Wu, X. (2007). Scheduling unrelated parallel machine to minimize total weighted tardiness using ant colony optimization. In *Proceedings of the 2007 IEEE International Conference on Automation and Logistics*, pages 132–136. IEEE.

Zhou, S., Li, X., Du, N., Pang, Y., and Chen, H. (2018). A multi-objective differential evolution algorithm for parallel batch processing machine scheduling considering electricity consumption cost. *Computers and Operations Research*, 96:55–68.