International Journal of Industrial Engineering Computations 12 (2021) 273–292

Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Customer order scheduling with job-based processing on a single-machine to minimize the total completion time

Ferda Can Çetinkaya[a*], Pınar Yeloğlu[a] and Hale Akkocaoğlu Çatmakaş[a]

[a]*Department of Industrial Engineering, Çankaya University, 06790, Etimesgut, Ankara, Turkey*

C H R O N I C L E

Article history:
Received January 15 2021
Received in Revised Format March 14 2020
Accepted March 15 2021
Available online March, 15 2021

Keywords:
Customer order scheduling
Order-based processing
Job-based processing
Total completion time
Mixed-integer linear programming
Tabu search

A B S T R A C T

This study considers a customer order scheduling (COS) problem in which each customer requests a variety of products (jobs) processed on a single flexible machine, such as the computer numerical control (CNC) machine. A sequence-independent setup for the machine is needed before processing each product. All products in a customer order are delivered to the customer when they are processed. The product ordered by a customer and completed as the last product in the order defines the customer order's completion time. We aim to find the optimal schedule of the customer orders and the products to minimize the customer orders' total completion time. We have studied this customer order scheduling problem with a job-based processing approach in which the same products from different customer orders form a product lot and are processed successively without being intermingled with other products. We have developed two mixed-integer linear programming models capable of solving the small and medium-sized problem instances optimally and a heuristic algorithm for large-sized problem instances. Our empirical study results show that our proposed tabu search algorithm provides optimal or near-optimal solutions in a very short time. We have also compared the job-based and order-based processing approaches for both setup and no-setup cases and observed that the job-based processing approach yields better results when jobs have setup times.

1. Introduction

Most of the existing research on classical scheduling problems, unlike the *customer order scheduling* (COS) problem, assumes that a single customer orders various products (jobs), or there are multiple customer orders, each of which consists of only a single product. However, in a real-world make-to-order manufacturing system, there are multiple customer orders. Each order is a collection of several products processed in a job lot consisting of many customer orders demanding the same product. In such a system, an order is shipped as a group to the customer, at the completion time of that order's last job (Liu, 2009). In the COS, the problem is to satisfy the demand of several customers, who give orders with a set of several products having different quantities, by optimizing the scheduling performance (objective).

In make-to-order environments, there are mainly two extreme processing approaches for production: (1) *order-based processing* (OBP) and (2) *job-based processing* (JBP). In the order-based processing, which is the most frequently used in previous COS studies in the literature, all various products in a customer order form an order lot (group), and all products in this order are processed successively without being intermingled with other customer orders' products (Yang, 2011). In other words, if the processing of a product in a customer order starts on a machine, then all various products within that customer order should be processed before switching the machine to process the products of another customer order. This processing approach follows the so-called *group technology* (GT) assumption. In order-based processing, the customer orders' sequence and the processing sequence of the products in each customer order lot need to be simultaneously determined. However, in

* Corresponding author
E-mail: cetinkaya@cankaya.edu.tr (F. C. Çetinkaya)

2021 Growing Science Ltd.
doi: 10.5267/j.ijiec.2021.3.001

the job-based processing, the same products from different customer orders form a product lot and are processed successively without being intermingled with other products. In other words, all customer orders for a product should be processed before switching the machine to process the customer orders for another product. In job-based processing, the sequence of the products and the customer orders' sequence in each job need to be simultaneously determined. While order-based processing aims to manage customer orders on the shop floor easily, job-based processing aims to reduce the negative effect of job setups, especially when the setup times required before processing the products are significantly longer.

The *multi-operation jobs scheduling* (MJS) problems are closely related to the customer order scheduling problems. In the MJS problem, multiple jobs consist of several operations belonging to different families of operations. Each job has at most one operation in each family and is being completed when all of its operations have been processed. Furthermore, a sequence-independent setup time is incurred whenever an operation is processed following a different family operation. The operations belonging to a family of operations are performed successively without being intermingled with other families' operations to achieve setup savings, i.e., the group technology assumption is considered. The MJS problem needs to simultaneously determine the operation families' sequence and the jobs' sequence in each operation to optimize an objective function (Gerodimos et al., 1999). Following the standard three-field notation proposed by Graham et al. (1979), the single machine MJS problem to minimize the total completion time of the jobs is identified as $1/s_f, \text{assembly}, \text{GT}/\sum C_j$, where s_f is the sequence-independent family setup time. The term 'assembly' describes that a job is completed when all of its operations have been processed, while GT represents the group technology assumption.

Within the context of the MJS problem, a *customer order* and a *product ordered by a customer* in the COS problem correspond to a *job* and an *operation of the job* respectively (Gerodimos et al., 1999). Thus, COS and MJS are two closely related areas of research in the literature on scheduling. The problems in our study fall in the intersection of these two main areas of research.

The optimal solution of the COS problem with order-based processing in a single-machine environment is easy and polynomial-time solvable, (as shown in Section 3.3) when the scheduling performance aims to minimize the total completion time, which is the sum of the completion times of the customer orders and is equivalent to minimizing the total work-in-process inventory focusing on increasing customer satisfaction. For the same scheduling performance, the COS problem with the job-based processing is not as easy as the order-based processing since it is equivalent to the problem $1/s_f, \text{assembly}, \text{GT}/\sum C_j$ and strongly *NP-hard* (Non-deterministic Polynomial-time – hard), as shown in Section 3.3. Thus, we will focus on the job-based processing problem in our study. We aim to determine a schedule that gives both the sequence of products (jobs) and the sequence of customer orders in each job to minimize the customer orders' total completion time. Furthermore, as the COS problem's objective function values with these two extreme processing approaches are expected to be different, the order-based and job-based processing approaches with both setup and no-setup cases will also be compared in our study.

To the best of our knowledge, in the literature on customer order scheduling and multi-operation job scheduling problems, there is no previous study dealing with the development of an optimization model and a heuristic algorithm to solve the COS problem with job-based processing on a single machine to minimize the total completion time when the setup and processing times take any value, not restricted to be one or a constant value as in the work of Su and Chen (2009). In other words, there is no work on solving the problem $1/s_f, \text{assembly}, \text{GT}/\sum C_j$. Our study aims to make several contributions in this regard. First, we formulate two mixed-integer linear programming (MILP) models to solve the COS problem with job-based processing optimally. Second, our proposed heuristic algorithm is easy to implement for finding optimal and near-optimal solutions for medium and large-sized problem instances that the MILP model cannot solve. Third, we compare the order-based and job-based processing approaches for the single-machine case.

The rest of this paper is organized as follows. Section 2 provides a brief review of the works most relevant to our study on customer order scheduling and multi-operation jobs scheduling. Section 3 defines the customer order scheduling problems with order-based and job-based processing on a single-machine in detail and presents some structural properties of the optimal schedules for both problems. Two MILP models and a tabu-search-based heuristic algorithm for solving the COS problem with job-based processing are presented in Sections 4 and 5, respectively. We present our empirical studies to evaluate the performances of the MILP models and the heuristic algorithm and compare the order- and job-based processing approaches in Section 6. Finally, in Section 7, we discuss our study's main findings and several future research directions.

2. Literature review

Although the concept of customer order scheduling was first introduced three decades ago by Julien and Magazine (1990) and Ahmadi and Bagchi (1990), the studies on the COS problems are scarce in the literature. As we can see from the previous studies, there are several variants of the COS problem with different scheduling criteria, such as the maximum completion time (makespan), the total completion time, and the maximum lateness, and in various machining environments such as parallel machines and the job shop. We now provide a brief overview of the most related works focusing on the single-machine

problems in an attempt to properly position our study in the literature. Julien and Magazine (1990) considered multiple customer orders containing several products processed on a single machine with a job-dependent setup time between two different types of jobs. They provided a dynamic programming algorithm to minimize the total completion time of orders when there are only two types of jobs, and the batch processing order is fixed. Subsequently, Bagchi et al. (1994) considered the COS problem on a single-machine to determine the due dates of the customer orders and to schedule all jobs to minimize a penalty function. Other early research efforts for the COS problems of the single-machine were made by Baker (1988), Coffman et al. (1989), and Vickson et al. (1993). One of the more recent studies by Erel and Ghosh (2007) considered a single-machine COS problem for customer orders with various products from different families in which a family-dependent setup time is incurred between different families of products. They discussed the complexity of the problem and proposed a dynamic programming algorithm for solving the problem. Hazır et al. (2008) investigated the COS problem for a single-machine, in which all products in a customer order are allowed to be processed in a fashion that allows intermingling with the products of other customer orders. They proposed four metaheuristics: simulated annealing, genetic algorithm, tabu search, and ant colony, and evaluated the performance of these heuristics to minimize the average customer order flow time. Yang (2017) addressed a similar COS problem on a single-machine whose lot description is considered a job in our study. In his study, orders are indivisible, and each order has to be processed on the same lot. He provided the problem's complexity, a binary integer programming model, and four efficient heuristics to minimize the makespan and the total completion time objectives. The main difference between the problem studied by Yang (2017) and the one in our study is the processing approach. He assumed that all orders in the same lot have the same processing and completion times. Furthermore, each lot has a capacity, and there are no setup times between different lots in his study. In contrast, our study tackles sequence-independent setup times that exist between different products (jobs).

Although the MJS problems date back to the pioneering work of Gerodimos et al. (1999), the studies on this type of scheduling problem are also quite limited in the literature. Gerodimos et al. (1999) proved that the problems $1/s_f, \text{assembly}/L_{max}$ and $1/s_f, \text{assembly}, \text{GT}/L_{max}$could be solved optimally in polynomial-time, where L_{max} is the maximum lateness of the jobs. Ng et al. (2002) proved that the problem $1/s_f = s, \text{assembly}, \text{GT}, p_{ij} = 0$ or $1/\sum C_j$ is strongly *NP-hard* when sequence-independent setup times are fixed whenever an operation is processed following an operation of a different family of operations, each of whose processing time is 0 or 1. Cheng et al. (2003) considered the single-machine MJS problem to minimize the number of tardy jobs and showed that the problem is strongly *NP-hard* when the due dates are the same, and all jobs have the same processing time. Su and Chen (2009) developed a polynomial-time algorithm to solve the problem $1/s_f, \text{assembly}, \text{GT}/L_{max}$ and a heuristic algorithm and a branch-and-bound algorithm to solve the problem $1/s_f = s, \text{assembly}, \text{GT}, p_{ij} = 0$ or $1/\sum C_j$. Çetinkaya et al. (2019) considered a single-machine scheduling problem of indivisible multi-operation jobs. In their study of the problem P_{OBP}, a job and all job operations correspond to a customer and products demanded by the customer in the COS problem with order-based processing approach, and there is a sequence-independent setup time between jobs in a customer order. It aims to avoid frequent product (job) switchovers to minimize the makespan and the total completion time. Hence, it is accomplished by combining the first job of a customer order with the last job of the immediately preceding customer order if these jobs are the same.

3. Problem definition and preliminary results

In this section, we first define our order-based and job-based processing problems in a joint statement in detail. Then, we present a numerical example to illustrate two extreme processing approaches and finally establish some preliminary results that provide the basis for our study.

3.1 Problem definition

Consider a scheduling problem of K customers ($i = 1,2, \dots, K$) in which each customer i gives order O_i with one or more products (jobs) from a set of N jobs. Each customer order O_i has a demand for $D_{i,j}$ units of identical items of product j. A sequence-independent setup with s_j time units is needed to set up the machine before processing the product j. In sequence-independent setup, the setup time is dependent only on the product to be processed next and independent of the previous product. Each job has only one operation to be processed by a single machine, and the unit-processing time of product j on the machine is p_j time units. All products ordered by the same customer must be processed consecutively if the order-based processing approach is used. However, when the job-based processing approach is used, all customer orders for the same product must be processed consecutively.

The following additional assumptions will be considered in describing our problem:

- All customer orders are available for processing at the same time, say time 0.
- The machine is available continuously from time zero onwards, with no breakdowns or maintenance delays, to process the products.
- The setup cannot be performed while the machine is processing a job.

- The machine can process, at most, one job at a time, i.e., the machine is not a batch processing machine.
- No precedence relations among the jobs exist.
- No priorities among the customers exist.
- Job processing cannot be interrupted, i.e., no pre-emption is allowed.
- All parameters are known with certainty and are not subject to any change, i.e., the scheduling problem is deterministic and static.

A completed product within a customer order must wait until all finished products are combined with other products belonging to the same customer order and shipped as a complete order. In other words, each order is delivered to the customer when all products within that customer order are processed. Thus, the completion time of the product processed as the last product of a customer order defines the completion time of that customer order. Our goal is to find an optimal schedule:

- with a sequence of customer orders and the sequence of jobs in each customer order when order-based processing approach is used,
- with a sequence of jobs and the sequence of customer orders in each job when job-based processing approach is used.

Our ultimate goal is to minimize the total completion time of the customer orders and increase customer satisfaction in both processing approaches.

3.2 An illustrative example

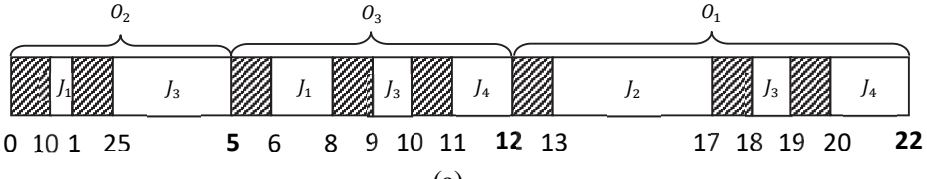
Before we proceed with our analysis, it seems appropriate to illustrate two extreme processing approaches with a numerical example. Consider a simple instance of a problem with three customer orders and four products. Each customer gives an order with several products having setup and unit processing times, as in Table 1. For example, Customer 1 gives an order with 10, 5, and 20 units of Products 2, 3, and 4, respectively.

Table 1
Customer orders, setup times, and unit-processing times

Jobs (Products)	Demand (in units) of the customer orders			Setup time	Unit-processing time
	O_1	O_2	O_3		
J_1	-	5	15	10	1
J_2	10	-	-	10	4
J_3	5	15	5	10	2
J_4	20	-	15	10	1

As mentioned in Çetinkaya et al. (2019), the order-based processing approach can be investigated in two forms: (1) *order-based processing without setup savings* in which setup time is required for each transition between products (jobs) while customer orders are processed successively on a machine, and (2) *order-based processing with setup savings* in which setup times are eliminated between products while customer orders are processed successively on a machine. In Fig. 1, the setup and processing times are illustrated by the gray and blank blocks, respectively. The optimal schedules for the order-based processing approach when there is no setup savings and a setup saving are given in Fig. 1(a) and Fig. 1(b), respectively. Fig. 1(c) illustrates the optimal schedule for the job-based processing approach.

The customer orders' optimal sequence is $O_2 - O_3 - O_1$, in which the orders of Customers 1, 2, and 3 are completed at 225, 55, and 125 time units, respectively, and the total completion time of the customer orders is 55 + 125 + 225 = 405 time units. However, if we process the common jobs successively when switching over customer orders on a machine, the total completion time is decreased due to setup-time savings of jobs. The customer orders' optimal sequence is $O_2 - O_3 - O_1$, in which the orders of Customers 1, 2, and 3 are completed at 205, 55, and 115 time units, respectively, and the total completion time is reduced to 55 + 115 + 205 = 375 time units. On the other hand, when we solve the problem with the job-based processing approach, the optimal job sequence is $J_1 - J_3 - J_4 - J_2$, in which the orders of Customers 1, 2, and 3 are completed at 185, 70, and 115 time units, respectively, and the total completion time of the customer orders is 185 + 70 + 115 = 370 time units.



(a)

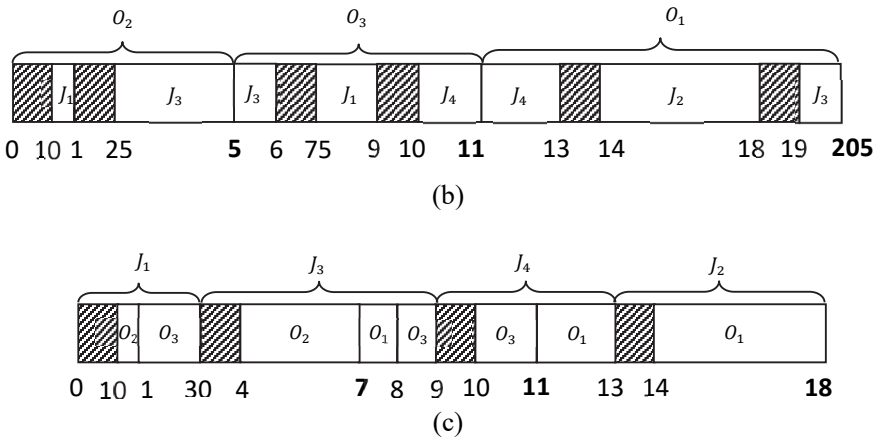


Fig. 1. Gantt chart for the extreme processing approaches: (a) order-based processing without setup savings; (b) order-based processing with setup savings; and (c) job-based processing

3.3 Preliminary results

We now give some definitions and theorems to investigate the complexities of the problems with different processing approaches and derive some structural properties of the optimal solutions that provide the basis for our analysis of these problems.

Definition 1: Let P_{JBP}, P_{OBP} and P'_{OBP} denote the job-based processing problem, the order-based processing problem with setup savings, and the order-based processing problem without setup savings, respectively.

Definition 2: *Total Time* (TT) of a customer order is the sum of the setup, if any, and processing times of all products (jobs) in this customer order.

Definition 3: The *Shortest Total Time* (STT) sequence is a sequence in which customer orders are sequenced in non-decreasing order of their total time (TT).

Since there are no restrictions that delay setups, jobs, and customer orders, we have the following result.

Lemma 1: *For all problems* P_{JBP}, P_{OBP}, and P'_{OBP}, *there is an optimal schedule in which the machine has no idle time; that is, the machine is busy either processing a job or being setup.*

The following theorem gives the optimal schedule for the problem P'_{OBP}.

Theorem 1: *The Shortest Total Time (STT) sequence gives the optimal schedule for the problem* P'_{OBP}.
Proof A sequence characterized by a string-based version of STT becomes optimal when each customer order may be treated as a pseudo-string of jobs, as defined by Pinedo (2008). □

Remark 1: It is evident that the minimum total completion time of customer orders in the problem P_{OBP} is always less than or equal to the minimum total completion time of customer orders in the problem P'_{OBP}. Furthermore, the problem P_{OBP}, when there is no-setup time, turns into the problem P'_{OBP}, which is optimally solved by Theorem 1.

The relevant definitions, theorems, and the mathematical model for solving the problem P_{OBP} are provided by Çetinkaya et al. (2019).

Theorem 2: *The problem* P_{JBP} *is strongly NP-hard.*
Proof The problem P_{JBP} is equivalent to the multi-operation job scheduling problem $1/s_f, \text{assembly}, \text{GT}/\sum C_j$. A special case of this problem is the problem $1/s_f = s, \text{assembly}, \text{GT}, p_{ij} = 0$ or $1/\sum C_j$, where the setup times are the same, and each operation's processing time is 0 or 1. Ng et al. (2002) proved that this special case is strongly *NP*-hard. Thus, the problem P_{JBP} is also strongly *NP*-hard. □

Definition 4: The *Smallest Demand* (SD) sequence is a sequence in which customer orders of a job are sequenced in the non-decreasing order of their demand for this job.

The following lemma, which is the extension of the theorem proposed by Su and Chen (2009) for the problem $1/s_f = 1, \text{assembly}, \text{GT}, p_{ij} = 0$ or $1/\sum C_j$, describes an optimal schedule for all customer orders of the job in the last position of the job sequence for the problem P_{JBP}.

Lemma 2: *For the problem P_{JBP}, there is an optimal schedule, in which all customer orders of the job in the last position of the job sequence are processed by the SD rule.*

Proof Note that the total completion time of the customer orders having no demand for the product (job) processed in the last position of the job sequence does not depend on the customer orders' sequence of the job in the last position of the job sequence. Thus, the problem of finding the customer orders' sequence of the job in the last position of the job sequence is equivalent to the basic single-machine scheduling problem $1//\sum C_j$. Smith (1956) showed that processing the jobs by the shortest processing time (SPT) rule minimizes the total completion time for the basic single-machine problem in which there are multiple jobs. In our problem, all customer orders of the job in the last position of the job sequence can be thought of as the jobs in the basic single-machine scheduling problem, and they are sequenced by the SD rule. □

Remark 2: As an extreme case, when there is a single customer order with multiple products, we observe that the problem P_{JBP} reduces to the scheduling problem of multiple products to minimize the maximum completion time (makespan) of the jobs in that customer order. In this reduced problem, the makespan minimization becomes trivial, and the arbitrary sequence of the products is the optimal solution.

Remark 3: When each customer gives an order consisting of only one product different from those ordered by the other customers, we observe that the problem P_{JBP} reduces to the scheduling problem of multiple products to minimize the total completion time of the customer orders, which is equivalent to the sum of the job completion times. This reduced problem is equivalent to the basic single-machine problem $1//\sum C_j$ in which there are multiple jobs. In this reduced problem, the STT rule minimizes the total completion time of the customer orders.

Remark 4: When each customer gives an order consisting of all products, it is clear that all customer orders in the problem P_{JBP} are completed within the job processed in the last position of the job sequence. From Lemma 2, we know that the SD rule processes all customer orders of the job in the last position of the job sequence. Thus, an optimal schedule for the problem P_{JBP} may be obtained by comparing the total completion times of the N schedules constructed by putting each of the jobs to the job sequence's last position.

From Remarks 2 through 4, for the problem P_{JBP}, we assume that:

(1) The number of customer orders is more than one.
(2) At least one customer gives an order with more than one different product.
(3) No customer gives an order consisting of all products.

The following theorem gives the optimal sequence of the customer orders in each job when a product (job) sequence is known for the problem P_{JBP}.

Theorem 3: *For a given product (job) sequence for the problem P_{JBP}, there is an optimal sequence of the customer orders in each job with the following properties:*

(a) In each job, the customer orders completed with this job precede all the customer orders completed with the successor jobs, as illustrated in Fig. 2.
(b) In each job, all the customer orders completed with this job are processed in the SD sequence, whereas all the customer orders completed with the successor jobs are sequenced in any order.

Proof The proof of the first property is straightforward. Suppose that a customer order completed with the current job is preceded by a customer order completed with the successor job. The quality of the schedule does not decrease by moving this customer order completed with the successor job to the end of the sequence of customer orders in the current job and shifting forward all the customer orders currently succeeding it. The repetition of this argument shows the correctness of the first property. The proof of the second property follows from Smith (1956), who showed that processing the jobs with the SPT rule minimizes the total completion time in the basic single-machine scheduling problem $1//\sum C_j$. Starting from the last minus one position of the job sequence, the repetitive use of Lemma 2 in the job shows the second property's correctness. □

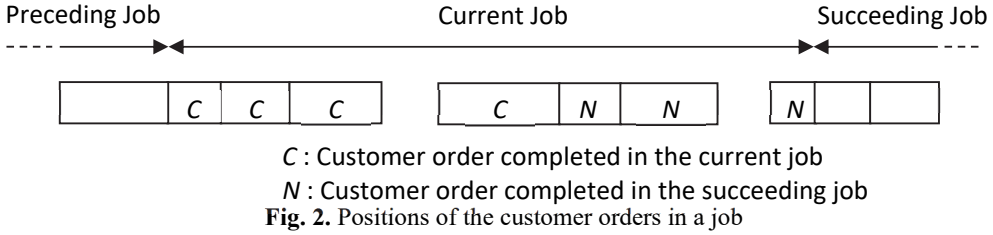


Fig. 2. Positions of the customer orders in a job

Based on Theorem 3, the algorithm SCO below gives the optimal sequence of the customer orders in each job when a job sequence is given for the problem P_{JBP}.

Algorithm SCO

Step 1 For the given job sequence, generate the initial sequence of the customer orders in each job by sorting each job's customer orders in the non-descending order of their demand for this job, i.e., apply the SD rule.

Step 2
- a Set $l = N - 1$.
- b Let the job in position l of the given job sequence be the current job.
- c Starting from the customer order in the first position of the customer orders' sequence in the current job, check whether the customer order will be completed in the succeeding jobs. If the answer is yes, then send this customer order to the last position of the customer orders' sequence in the current job; otherwise, keep this customer order in its current position. Repeat this step for all customer orders in the current job.
- d Set $l = l - 1$. If $l > 0$, then go to Step 2b; otherwise, stop.

4. Mathematical programming models

This section presents two MILP models, which will be denoted as MILP-1 and MILP-2, to solve the problem P_{JBP} optimally. In these models, we aim to minimize the sum of completion times of the customer orders. Both models' solution provides a schedule with the jobs' sequence and the customer orders' sequence in each job.

We first introduce the following parameters, indices, and sets commonly used for developing both models.

Parameters, indices and sets

K Number of customer orders.
o, m, u Indices for customer orders $(o, m, u = 1,2, \dots, K)$.
N Number of jobs.
j, k, l Indices for jobs $(j, k, l = 1,2, \dots, N)$.
$D_{o,j}$ Demand (number of identical items) for job j in customer order o.
SC_j Set of customer orders having demand for job j.
t_j Unit processing time of job j.
s_j Setup time of job j.

4.1 Model 1 (MILP-1)

This first model is positioned-based in which the binary variables are used to determine the position of a job in the job sequence and the position of each customer order in each job. Our additional parameters, indices, and decision variables are as follows:

Additional parameters and indices

$H_{o,j}$ 1, if customer order o has demand for job j; 0, otherwise.
$\|SC_j\|$ Cardinality of the set of customer orders having demand for job j. That is, the number of customer orders having demand for job j.
Q Sufficiently large positive number.
p Index for positions in the job sequence $(p = 1,2, \dots, N)$.
r Index for positions in the sequence of customer orders having demand for job j $(r = 1,2, \dots, \|SC_j\|)$.

Decision variables

$$Y_{j,p} = \begin{cases} 1 & \text{if job } j \text{ is assigned to position } p \text{ of the job sequence} \\ 0 & \text{otherwise} \end{cases}$$

$$X_{o,j,p,r} = \begin{cases} 1 & \text{if customer order } o \text{ in job } j \text{ at position } p \text{ of the job sequence is} \\ & \text{assigned to position } r \text{ of the customer orders' sequence in job } j \\ 0 & \text{otherwise} \end{cases}$$

$C_{o,j,p,r}$ Completion time of customer order o assigned to position r of the customer orders' sequence in job j at position p of the job sequence.
$CT_{j,p}$ Completion time of job j assigned to position p of the job sequence.
T_o Completion time of the customer order o.

The resulting MILP-1 model for solving the problem P_{JBP} is as follows:

$$\text{Minimize} \quad \sum_{o=1}^{K} T_o \tag{1}$$

$$\text{Subject to} \quad \sum_{j=1}^{N} Y_{j,p} = 1 \qquad \text{for } p = 1,2,\dots,N \tag{2}$$

$$\sum_{p=1}^{N} Y_{j,p} = 1 \qquad \text{for } j = 1,2,\dots,N \tag{3}$$

$$\sum_{o=1}^{K} H_{o,j}\, X_{o,j,p,r} = Y_{j,p} \qquad \text{for } j,p = 1,2,\dots,N;\ r = 1,2,\dots,\|SC_j\| \tag{4}$$

$$\sum_{r=1}^{\|SC_j\|} H_{o,j}\, X_{o,j,p,r} = Y_{j,p} \qquad \text{for } j,p = 1,2,\dots,N;\ o = 1,2,\dots,K \tag{5}$$

$$C_{o,j,1,1} \geq s_j + t_j D_{o,j} X_{o,j,1,1} - Q(1 - X_{o,j,1,1}) \qquad \text{for } o = 1,2,\dots,K;\ j = 1,2,\dots,N \tag{6}$$

$$C_{o,j,p,1} \geq CT_{k,p-1} + s_j + t_j D_{o,j} X_{o,j,p,1} - Q(1 - X_{o,j,p,1}) \qquad \text{for } o = 1,2,\dots,K;\ j,k = 1,2,\dots,N;\ p = 2,3,\dots,N;\ j \neq l \tag{7}$$

$$C_{o,j,p,r} \geq C_{m,j,p,r-1} + t_j D_{o,j} X_{o,j,p,r} - Q(1 - X_{o,j,p,r}) \qquad \text{for } o,m = 1,2,\dots,K;\ j = 1,2,\dots,N;\ p = 2,3,\dots,N;\ r = 2,3,\dots,\|SC_j\|;\ m \neq o \tag{8}$$

$$CT_{j,p} \geq C_{o,j,p,r} - Q(1 - Y_{j,p}) \qquad \text{for } o = 1,2,\dots,K;\ j = 1,2,\dots,N;\ p = 2,3,\dots,N;\ r = 1,2,\dots,\|SC_j\| \tag{9}$$

$$T_o \geq C_{o,j,p,r} \qquad \text{for } o = 1,2,\dots,K;\ j,p = 1,2,\dots,N;\ r = 1,2,\dots,\|SC_j\| \tag{10}$$

$$C_{o,j,p,r},\ CT_{j,p},\ T_o \geq 0 \qquad \text{for } \forall\, o,j,p,r \tag{11}$$

$$X_{o,j,p,r}, Y_{j,p} \in \{0,1\} \qquad \text{for } \forall\, o,j,p,r \tag{12}$$

In the above MILP-1 model, the objective in (1) is to minimize customer orders' total completion time. Constraint sets (2) and (3) ensure that each position in the sequence of jobs is occupied by one job only, and each job is assigned to one position only, respectively. Constraint set (4) guarantees that each position in the customer orders sequence in a job is occupied by one customer order only. Constraint set (5) ensures that each customer order in a job is assigned to a position in the sequence of customer orders in this job. Constraint sets (6) and (7) determine the completion time of the customer order assigned to the first position of customer orders in the job assigned to the job sequence's first and remaining positions. Constraint set (8) defines the completion times of the customer orders assigned to the remaining positions of the customer orders' sequence in a job. Constraint set (9) determines each job's completion time in each position of the job sequence. Constraint set (10) defines the completion time of each customer order. Constraint sets (11) and (12) impose the non-negativity and binary restrictions on the decision variables, respectively.

4.2 Model 2 (MILP-2)

This model is sequence-based in which the binary variables are used to determine the precedence relations among all jobs and all customer orders in each job. Our additional parameters and decision variables used in this model are as follows:

Additional parameters

L_j Lot size (total demand) for job j, where $L_j = \sum_{o \in SC_j} D_{o,j}$.

Decision variables

$$Y_{j,k} = \begin{cases} 1 & \text{if job } j \text{ precedes job } k \\ 0 & \text{otherwise} \end{cases}$$

$$X_{o,m,j} = \begin{cases} 1 & \text{if customer order } o \text{ in job } j \text{ precedes customer order } m \text{ in job } j \\ 0 & \text{otherwise} \end{cases}$$

$C_{o,j}$ Completion time of customer order o in job j.

T_o Completion time of the customer order o.

The MILP-2 model for solving the problem P_{JBP} is as follows:

$$\text{Minimize} \quad \sum_{o=1}^{K} T_o \tag{1}$$

$$\text{Subject to} \quad Y_{j,k} + Y_{k,j} = 1 \qquad \text{for } j,k = 1,2,\dots,N;\ j < k \tag{13}$$

$$Y_{j,k} + Y_{k,l} + Y_{l,j} \leq 2 \quad \text{for } j,k,l = 1,2,\dots,N;\ j \neq k \neq l \tag{14}$$

$$X_{o,m,j} + X_{m,o,j} = 1 \quad \text{for } o,m = 1,2,\dots,K;\ o < m;\ j = 1,2,\dots,N;\ D_{o,j} = D_{m,j} > 0 \tag{15}$$

$$X_{o,m,j} + X_{m,u,j} + X_{u,o,j} \leq 2 \quad \text{for } o,m,u = 1,2,\dots,K;\ j = 1,2,\dots,N;\ o \neq m \neq u;\ D_{o,j} = D_{m,j} = D_{u,j} > 0 \tag{16}$$

$$C_{o,j} = \sum_{\substack{k=1 \\ k \neq j}}^{N} (s_k + t_k L_k) Y_{k,j} + s_j + t_j D_{o,j} + \sum_{\substack{m=1 \\ m \neq o}}^{K} t_j D_{m,j} X_{m,o,j} \quad \text{for } o = 1,2,\dots,K;\ j = 1,2,\dots,N;\ D_{o,j} > 0 \tag{17}$$

$$T_o \geq C_{o,j} \quad \text{for } o = 1,2,\dots,K;\ j = 1,2,\dots,N \tag{18}$$

$$C_{o,j},\ T_o \geq 0 \quad \text{for } \forall\ o,j \tag{19}$$

$$X_{o,m,j}, Y_{j,k} \in \{0,1\} \quad \text{for } \forall\ o,m,j,k \tag{20}$$

In the above MILP-2 model, the objective is the same as in (1) of the MILP-1 model. Constraint set (13) ensures that for each pair of the jobs, one of them should precede the other, and similarly, the constraint set (15) guarantees that for each pair of the orders, one of them should precede the other. Constraint sets (14) and (16) are triangular inequalities. Constraint set (17) calculates the completion time of customer order i that demands job j. Constraint set (18) defines the completion time of each customer order. Constraint sets (19) and (20) impose the non-negativity and binary restrictions on the decision variables, respectively.

4.3 Size complexity of the models

In our MILP-1 model, there are three sets of continuous variables, and the number of these variables is $N^2(K^2+1)+K$. Also, there are two sets of binary variables, and the number of binary decision variables is $N^2(K^2+1)$. Thus, there is a total of $2N^2(K^2+1)+K$ decision variables in this model. On the other hand, the MILP-1 model has a total of $N^3 - 2N^2 - 3N + NK(K^2 - 2K + 3)$ constraints. However, in our MILP-2 model, there are two sets of continuous variables, and the number of these variables is $K(N+1)$. Also, there are two sets of binary variables, and the number of binary decision variables is $N(K^2+N)$. Thus, there is a total of $K(N+1+KN)+N^2$ decision variables in this model. Also, the MILP-2 model has a total of $K + N + NK(1 + N(1+N) + NK(NK+1))$ constraints.

When both models are compared in terms of the number of decision variables and the number of constraints, we may conclude that the MILP-2 model is more efficient than the MILP-1 model. We have also observed this result in our computational study in Section 6.

5. Heuristic algorithm

The size of our MILP models increases tremendously as the number of products and the number of customer orders increase. As we will show later in Section 6, the MILP models cannot provide optimal solutions for large-sized problem instances in reasonable times. However, an optimal solution may not be essential in many real-life problems as a high-quality approximate solution may be sufficient. This situation motivated us to develop a heuristic algorithm that provides optimal or near-optimal solutions for the large-sized problem instances within relatively short times.

Our heuristic algorithm for solving problem P_{JBP} is a *tabu search* (*TS*) algorithm and consists of two phases: *finding an initial schedule by the insertion algorithm* and *improving the initial schedule by the tabu search algorithm*. The detailed descriptions of each phase in our heuristic algorithm are given below.

5.1 Phase 1 - Finding an initial schedule by the insertion algorithm

This phase finds an initial schedule of jobs by applying the insertion algorithm, a neighborhood algorithm. Insertion algorithm is also known as the *NEH* algorithm since it was proposed first by Nawaz et al. (1983) for the makespan minimization problem in a flow shop. The *NEH* algorithm has been widely used to solve various scheduling problems with different scheduling criteria other than makespan. The algorithm generates $(N(N+1)/2) - 1$ different sequences of jobs, where N of them are complete, and the rest are partial sequences. The *NEH* algorithm assumes that a job with a long total processing time is given higher priority than the job with a small total processing time. Our algorithm modifies this assumption as the job with more customer orders being given a higher priority than the job with fewer customer orders.

The stepwise description of Phase 1 in our algorithm is given below.

Step 1 Generate an initial job sequence by sorting the jobs in descending order of their number of customer orders.

Step 2 In the initial job sequence, generate the initial sequence of the customer orders in each job by sorting each job's customer orders in the ascending order of their total demand.

Step 3
- a Select the jobs $J_{[1]}$ and $J_{[2]}$, which are in the first two positions of the initial job sequence obtained in Step 2.
- b Form two partial job sequences in such a way that the first selected job $J_{[1]}$ is in the first and second positions in these partial sequences, respectively. That is,

  Partial sequence 1: $J_{[1]} - J_{[2]}$ Partial sequence 2: $J_{[2]} - J_{[1]}$
- c Let the first partial sequence among all partial sequences be the current partial sequence.

Step 4
- a Let the job in the last position of the current partial sequence be the current job. Sort all customer orders of the current job in ascending order of their total demands.
- b Consider the previous job as the new current job.
- c Starting from the customer order in the first position of the customer orders' sequence in the current job, check whether the customer order has the jobs processed after the current job of the current partial job sequence. If the answer is yes, send this customer order to the last position of the customer orders in the current job; otherwise, keep this customer order in its current position. Repeat this step for all remaining customer orders in the current job.
- d If the current job is in the first position of the current partial sequence, compute the total completion time of the customer orders in the current partial sequence, and go to Step 4e; otherwise, go to Step 4b.
- e After all partial sequences have been considered, select the best partial sequence giving the minimum total completion time, and go to Step 5a; otherwise, consider the next partial sequence as the current partial sequence and go to Step 4a.

Step 5
- a If all jobs of the initial job sequence obtained in Step 2 have not been considered yet, go to Step 4a; otherwise, go to Phase 2.
- b Pick the job in the next position of the initial job sequence obtained in Step 2, generate all possible partial sequences by placing this new job in all possible positions (beginning, between, and end) in the best partial sequence developed so far, and go to Step 4a.

5.2 Phase 2 - Improving the initial schedule by the tabu search algorithm

Tabu Search (*TS*), which was first proposed by Glover (1989), is a local-search-based metaheuristic for solving many combinatorial optimization problems, including scheduling problems widely used in the literature. It starts with an *initial solution* (schedule) generated randomly or obtained by a simple rule or a heuristic algorithm. The initial solution is considered the best solution. A local search mechanism is then applied to find a better solution in the current solution neighborhood, which is defined as *all solutions* (also called *mutations*) obtained by an alternative solutions generation mechanism using the current solution. This *neighborhood generation mechanism* can be an adjacent pairwise interchange of the jobs or insertion of every job in every position in the current schedule. In our *TS* algorithm, we use the schedule obtained by Phase 1 of our algorithm as the initial schedule. The neighborhood is generated by adjacent pairwise interchanges of the jobs in this initial schedule. The mutation with the lowest objective function (total completion time of the customer orders) value is selected as a *candidate solution*. The local changes providing the candidate solution among the solutions in a current solution's neighborhood is called a *move*. A *tabu* list is used to keep the search history and avoid cycling, i.e., returning to a solution that has been visited before and guide the search towards unexplored regions of the problem's solution space. The move providing the candidate solution is put into the tabu list if this move is not tabu. The candidate solution becomes the new best solution if the candidate solution's objective function value is better than the current best solution's objective function value. This is the *aspiration criterion* used in our TS procedure. Once a move is entered into the tabu list, its oldest move is deleted since it has a fixed size called *tabu list size*, say l. The tabu list is also called *tabu tenure*, which allows the new move added to the tabu list to remain in the list for the next l iterations.

Tabu-search iterations are conducted until one of the *stopping criteria* is reached. In the literature, there are several applications of the *TS* algorithm using different stopping criteria, which determine the search length. One approach is to set the number of iterations to a pre-specified value. That is, the *TS* procedure stops when no improvement can be obtained after several iterations. Setting the number of iterations to a high number may increase the search space and solution time. Our algorithm lets the *TS* procedure run for the $NI = 2 \times N$ iterations, where N is the number of jobs. Our second stopping criterion is that the *TS* procedure terminates when all possible mutations are worse than the parent.

Tabu tenure is also an important parameter that affects the *TS* algorithm's performance since the tabu list directs the search. The tabu tenure can be fixed (usually preferred in the literature) or variable. Setting the tabu tenure to a low number may cause cycling, i.e., returning to the solution already determined before. It is tough to escape from local optima when the tabu tenure is too low. However, setting the tabu tenure to a high number may result in deterioration in the solutions found. In other words, the algorithm spends more time comparing with the current solution one by one. In our algorithm, we set the tabu list size to 5.

The stepwise description of Phase 2 in our algorithm is given below.

Step 1 Set the iteration counter ic to 1, i.e., set $ic = 1$. Set the initial schedule σ_1 to the schedule obtained in Phase 1 of the algorithm. Set the best schedule σ_B to σ_1, i.e., set $\sigma_B = \sigma_1$.

Step 2
- a Generate the neighborhood of the schedule σ_{ic} by adjacent pairwise interchanges of the jobs in the schedule σ_{ic}.
- b For each of the mutations in the neighborhood of the schedule σ_{ic}, apply the algorithm *SCO*.
- c If the total completion time value of each mutation is bigger than the total completion time of the parent schedule σ_{ic}, then stop; otherwise, from the neighborhood of the schedule σ_{ic}, select the schedule with the lowest total completion time value as the candidate schedule σ_C.

Step 3
- a If the move $\sigma_{ic} \rightarrow \sigma_C$ is prohibited by a mutation on the tabu list, set $\sigma_{ic+1} = \sigma_{ic}$ and go to step 4; otherwise,
  - i Delete the entry at the bottom of the tabu list.
  - ii Pull all other entries in the tabu list one position down.
  - iii. Enter reverse mutation at the top of the tabu list.
  - iv. Set $\sigma_{ic+1} = \sigma_C$.
  - v. Set the new best schedule to the candidate schedule (i.e., set $\sigma_B = \sigma_C$) if the total completion time value of the candidate schedule is smaller than the total completion time value of the current best schedule, i.e., $TCT(\sigma_C) < TCT(\sigma_B)$.
  - vi. Go to step 4.

Step 4
- a Increment the iteration counter ic by 1. i.e., set $ic = ic + 1$.
- b If the iteration counter ic is equal to the pre-specified value NI for the number of iterations (i.e., $ic = NI$), then stop; otherwise, go to step 2.

A numerical example to demonstrate our proposed heuristic algorithm is provided in the Appendix.

6. Computational experiments

This section describes our computational experiments to evaluate the performances of the mathematical programming models and the heuristic algorithm to find optimal or near-optimal schedules. The MILP models for the problems P_{JBP} and P_{OBP} are solved by using version 24.1 of the software package General Algebraic Modeling System (GAMS). The proposed heuristic algorithm for solving the problem P_{JBP} is programmed in Python in Visual Studio Code, and the optimal schedule for the problem P'_{OBP} is obtained in Microsoft Excel VBA. All computations are conducted on a computer with an Intel(R) Core(TM) i7-9750H processor running at 2.60GHz, with 16 GB of RAM under the Windows 10 operating system.

6.1 Problem instances design

We generate the values of the parameters used in our experiments as in Çetinkaya et al. (2019):

1. *Number of customer orders (K)*: They are taken as 5, 10, 15, and 20.
2. *Number of jobs (N)*: They are taken as 5, 10, 15, and 20.
3. *Number of customer orders having demand for each job ($\|SC_j\|$)*: They are randomly generated from a DU [1, K].
4. *Demand (number of identical items) for each job in each customer order ($D_{o,j}$)*: They are randomly generated from a discrete uniform distribution DU [1, 10].
5. *Unit processing times (t_j)*: They are randomly generated from a discrete uniform distribution DU [1, 10].
6. *Setup times (s_j)*: They are randomly generated from a discrete uniform distribution DU [0, $100f$], where f is taken as 0.5, 1.0, 1.5, and 2.0.

Problem size is mainly determined by the number of customer orders and the number of jobs (products). For each possible combination of the number of customer orders and the number of jobs, 25 replicates (problem instances) are generated, and a total of 400 problem instances are tested for the setup case. Also, 25 replicates are generated for each possible combination excluding setup times, and a total of 400 problem instances are tested. Hence, the total number of problem instances generated for both setup and no-setup cases is 800.

6.2 Comparison of the models MILP-1 and MILP-2

To measure the efficiency and effectiveness of the two MILP models, namely MILP-1 and MILP-2, they are solved using GAMS with a time limitation of 10,800 seconds. The models MILP-1 and MILP-2 are compared in terms of execution time and the solution quality of the tested problems. Let PD_k^t denotes the percent deviation of the execution time of the MILP-1 model from the MILP-2 model, then $PD_k^t = 100 \times (ET_k^{MILP-1} - ET_k^{MILP-2})/ET_k^{MILP-2}$ where ET_k^{MILP-1} and ET_k^{MILP-2} are the execution times of the tested problems solved by the models MILP-1 and MILP-2, respectively. For the problem instances, the percent deviations of the objective function values are calculated as $PD_k^o = 100 \times (OFV_k^{MILP-1} - OFV_k^{MILP-2})/OFV_k^{MILP-2}$ where OFV_k^{MILP-1} and OFV_k^{MILP-2} are the objective function values of the tested problems solved by the models MILP-1 and MILP-2, respectively.

The percentage of optimally solved instances by both MILP formulations and the average percent deviations in terms of solution quality and time are tabulated in Table 2. Almost none of the problem instances can be solved optimally by the MILP-1 model within 10,800 seconds, whereas the MILP-2 model solves all problems up to 10 jobs and 58.5% of all problem instances optimally. This finding indicates that the MILP-2 model outperforms the MILP-1 model with regard to the number of optimal solutions obtained. When we compare both models according to their quality of the objective function values, it is observed that the MILP-2 model is better than the MILP-1 model. The grand average percent deviation of the solution quality is 8.33%, which means that the MILP-1 resulted in 8.33% lower quality solutions than the MILP-2 model. Besides, we have observed that the solution time of the MILP-1 model takes longer than the MILP-2 model. The MILP-2 model finds the optimal solution faster than the MILP-1 model, in which the solution time is 71% greater than the MILP-2 model on average.

Table 2
Comparison of the MILP-1 and MILP-2 models

K	*N*	*Percentage of Optimally Solved Instances by MILP-1*	*Percentage of Optimally Solved Instances by MILP-2*	*Average percent gap for solution quality*	*Average percent gap for solution time*
5	5	5	100	0.23	100.00
	10	0	100	1.00	100.00
	15	0	76	2.49	98.93
	20	0	0	7.23	26.31
Averages		1.25	69	2.74	81.31
10	5	0	100	0.26	99.99
	10	0	100	5.67	99.99
	15	0	52	11.35	93.48
	20	0	0	14.92	16.29
Averages		0	63	8.05	77.44
15	5	0	100	3.76	99.99
	10	0	100	6.49	99.93
	15	0	8	13.33	84.70
	20	0	0	15.14	1.04
Averages		0	52	9.68	71.42
20	5	0	100	2.64	99.99
	10	0	100	10.13	99.93
	15	0	0	27.31	17.51
	20	0	0	11.38	0.34
Averages		0	50	12.86	54.44
Grand Averages		0.31	58.50	8.33	71.15

According to these results, the MILP-2 model outperforms the MILP-1 model in both solution quality and execution time. Therefore, we have selected the MILP-2 model as the more efficient and effective leading formulation, which we will refer to as MILP hereafter considering it as our sole model of analysis.

6.3 Performance measures

To measure the effectiveness of the two solution approaches, we compared the objective function values obtained with the MILP model solved by GAMS and the proposed heuristic algorithm. For the problem instances in which the MILP model achieves the best integer solution despite the fact that an optimal solution is not reached, we apply this best integer solution in our comparison. The average, maximum, and minimum deviations of objective function values over the optimal solutions (or best integer solutions) are used as the performance measures.

For a problem instance k, in which the MILP model obtains the optimal solution, we define the percent deviation PD_k of the total completion time obtained by the proposed heuristic algorithm from the optimal solution's total completion time. That is, $PD_k = 100 \times (TC_k^H - TC_k^O)/TC_k^O$ where TC_k^H and TC_k^O are the total completion times of the solutions obtained by the heuristic algorithm and the MILP model, respectively. For the problem instances in which the optimal solution is not obtained by the MILP model but the best integer solution exists, TC_k^O is replaced by TC_k^B where TC_k^B is the total completion time of the best integer solution obtained by the MILP model.

The computational (execution) time also serves as an efficient measure to compare the MILP and the heuristic algorithm. The average computing time is in CPU seconds in our experiments. The GAMS's CPLEX solver's running time limit is set to 10,800 seconds (3 hours), and this time limit is exceeded for solving the large-sized problem instances. The proposed heuristic algorithm's running time is recorded for all test problems and is relatively minimal. The experiments' results, which are given in the next section, demonstrate that the heuristic algorithm's computational time increases as both customer orders and jobs increase. However, the computational time, which is less than a minute, is minimal in general. On the other hand, the MILP model has a much longer computational time than the heuristic algorithm.

6.4 Discussion of the results

6.4.1 Performance of the MILP model

We now discuss our computational tests' results to investigate the MILP model's performance for both setup and no-setup cases when we solve problem instances with the job-based processing approach. From Table 3, we observe that the MILP returns optimal solutions for all small-sized problem instances with up to 10 jobs for the setup case, but it cannot find optimal solutions for most of the problem instances in a three-hour time limit as the number of jobs increases. When the number of customer orders is five and the number of jobs is 15, there are 19 problem instances optimally solved. For the problem instances with ten customer orders and 15 jobs, there are 13 problem instances optimally solved. Optimal solutions are obtained for only two problem instances when there are 15 customer orders and 15 jobs. However, the MILP model doesn't provide optimal solutions in any of the cases when the number of jobs is 20. Similarly, we observe that no optimal solution for the problem instances with 20 customer orders and 15 or 20 jobs. In summary, Table 3 shows that the MILP model returns optimal solutions for 234 out of 400 problem instances.

Table 3
Performance of the MILP model

K	*N*	*NPI*	*Setup case*			*No-setup case*		
			NOS	*NBIS*	*AG*	*NOS*	*NBIS*	*AG*
5	5	25	25	0	0	25	0	0
	10	25	25	0	0	25	0	0
	15	25	19	6	12	21	4	4
	20	25	0	25	24	0	25	17
Total & Averages		100	69	31	18	71	29	10.5
10	5	25	25	0	0	25	0	0
	10	25	25	0	0	25	0	0
	15	25	13	12	12	19	6	10
	20	25	0	25	32	0	25	29
Total & Averages		100	63	37	22	69	31	19.5
15	5	25	25	0	0	25	0	0
	10	25	25	0	0	25	0	0
	15	25	2	23	22	7	18	21
	20	25	0	25	45	1	24	42
Total & Averages		100	52	58	33.5	58	42	31.5
20	5	25	25	0	0	25	0	0
	10	25	25	0	0	25	0	0
	15	25	0	25	48	1	24	45
	20	25	0	25	58	0	25	56
Total & Averages		100	50	50	53	51	49	50.5
Total & Grand Averages		400	234	166	31.6	249	151	28

Notes: *K*: number of customer orders; *N*: number of jobs; *NPI*: number of problem instances; *NOS*: number of optimally solved instances; *NBIS*: number of best integer solutions obtained only; *AG*: average gap value in percentage.

When we consider the MILP model's performance for the no-setup case, we observe from Table 3 it cannot find the optimal solutions for the problem instances with 15 and 20 jobs regardless of the number of customer orders as in the setup case. When the number of customer orders is five and the number of jobs is 15, there are 21 problem instances optimally solved out of 25, while there is no optimal solution when the number of jobs is increased to 20. For the problem instances with ten customer orders, 19 test problems are optimally solved out of 25 when the number of jobs is 15, but there is no optimal solution when the number of jobs is 20 for the same problem set. The MILP finds the optimal solution for seven problem instances with 15 customer orders and 15 jobs; however, only one optimal solution is obtained when the number of jobs is 20. Lastly, when the number of customer orders is 20, and the number of jobs is 15, the MILP finds the optimal solution for only one problem instance. However, no optimal solution is found when the number of jobs is 20.

To emphasize the MILP model's performance, we have also investigated the quality of solutions that are not optimal. It is a common phenomenon that GAMS's CPLEX ends up in a gap between the best integer solution and the best possible solution. Therefore, when we examined the gap values to investigate the percent deviation of the integer solution from the theoretical optimum, we observed that the gap values for 166 non-optimally solved problem instances with 15 jobs and 20 jobs for the setup case are significantly high. The gap values increase as the number of jobs increases from 15 to 20. GAMS performs more iterations for these problems the closer the integer solutions get to the theoretical optimum. However, GAMS is terminated because of the time limitation before reaching the optimum solution. On the other hand, when the number of jobs is 5 or 10 for all problem instances, all of the gap values equal zero, proving that the MILP can solve all problem instances

optimally. When we investigate the gap values for 151 non-optimally solved problems for the no-setup case in Table 3, we observe that the gap values for the no-setup case are also significantly high when the number of jobs is increased to 15 or 20 as in the setup case.

6.4.2 Performance of the heuristic algorithm

This section presents computational test results to examine the quality of solutions obtained by the proposed heuristic algorithm and its computational time requirement. As explained before, we use the best integer solutions obtained by the MILP model for comparison between the heuristic algorithm and the MILP model when the MILP finds no optimal solution.

Table 4 reports the average, maximum, and minimum percent deviations from the optimal or best integer solution returned by the MILP model solution. We observe negative percent deviations for some problem instances having 15 and 20 jobs, which indicates that the heuristic algorithm yields better solutions than the best integer solutions of the MILP model when the MILP model gives no optimal solution. It can be observed in Table 4 that all average percent deviations increase as the number of customers increases, except in the case with 20 customers. This occurs because there are some problem instances with negative percent deviations, especially when we have 15 and 20 jobs. Another observation from Table 4 is that the overall percent deviations for all problem instances of setup and no-setup cases are 2.41 and 2.50, respectively. These are relatively low and indicate that the heuristic algorithm is robust for finding near-optimal solutions.

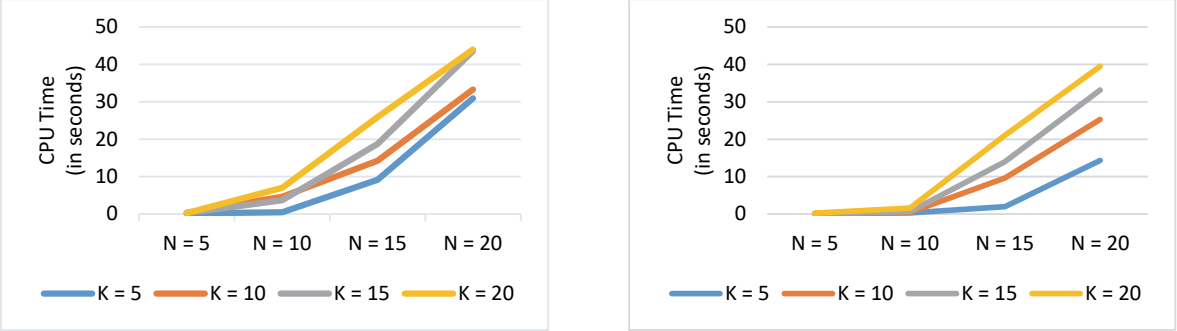
Table 4
Performance of the heuristic algorithm

K	*N*	*NPI*	*Setup case*			*No-setup case*		
			AVE	*MAX*	*MIN*	*AVE*	*MAX*	*MIN*
5	5	25	1.84	11.60	0.00	2.06	19.20	0.00
	10	25	2.40	10.30	0.00	2.40	10.30	0.00
	15	25	3.57	9.02	0.00	3.57	8.86	0.00
	20	25	2.17	7.42	-0.05	2.17	6.70	0.00
Total & Averages		100	2.49	9.59	-0.01	2.55	11.27	0.00
10	5	25	1.15	8.01	0.00	2.58	8.95	0.00
	10	25	3.26	19.10	0.00	2.11	12.69	0.00
	15	25	3.28	28.03	0.00	3.28	10.58	0.00
	20	25	2.96	12.19	-1.34	2.27	13.84	0.00
Total & Averages		100	2.66	16.83	-0.33	2.56	11.52	0.00
15	5	25	1.77	8.28	0.00	2.10	9.42	0.00
	10	25	2.12	7.02	0.00	2.30	6.60	0.00
	15	25	3.35	10.07	0.17	3.34	9.86	-0.04
	20	25	3.69	12.40	-0.69	4.08	16.43	-0.23
Total & Averages		100	2.73	9.44	-0.13	2.95	10.58	-0.07
20	5	25	1.14	2.80	0.00	1.31	5.50	0.00
	10	25	1.91	7.87	0.00	1.78	8.49	0.00
	15	25	1.45	7.41	-4.68	2.16	7.09	-2.56
	20	25	2.48	6.07	-0.39	2.52	9.38	-6.65
Total & Averages		100	1.74	6.04	-1.27	1.94	7.62	-2.30
Total & Grand Averages		400	2.41	10.48	0.44	2.50	10.24	-0.59

Notes: *K*: number of customer orders; *N*: number of jobs; *NPI*: number of problem instances; *AVE*: average percent deviation, i.e., $\overline{PD}$; *MAX*: maximum percent deviation; *MIN*: minimum percent deviation.

Fig. 3 demonstrates the heuristic algorithm's average CPU times for different problem sizes for the setup and no-setup cases, respectively. The average CPU time in both cases tends to increase as the number of jobs increases; however, it is relatively small in general. We can conclude that the heuristic algorithm yields significantly higher quality solutions with relatively shorter computational time than the MILP model. We have finally investigated the effect of each phase in the heuristic algorithm. As shown in Section 4, the heuristic algorithm works in two phases: Phase 1 provides an initial solution to Phase 2 in which the initial solution is improved by the tabu search method. Table 5 shows the summary results for improving the Phase 1 solutions by Phase 2 for the setup and no-setup cases. 217 problems out of 400 problem instances for the setup case are improved in Phase 2 of the algorithm. The average improvement percentages are relatively low, and the overall improvement percentage is %1.32 due to the robust initial solution provided in Phase 1. On the other hand, as shown in Table 4, the scenario is similar for the no-setup case. 189 problems out of 400 problem instances for the no-setup case are improved

in Phase 2 of the algorithm. The overall improvement percentage is %1.56, which is relatively low again for the improvement.



(a) Setup case (b) No-setup case

Fig. 3. Average CPU time of the heuristic algorithm

Table 5
Improvement in Phase-2

K	*N*	*NPI*	*Setup case*		*No-setup case*	
			NIS	*API*	*NIS*	*API*
5	5	25	14	1.95	13	3.29
	10	25	14	0.98	11	0.77
	15	25	12	0.57	12	0.86
	20	25	16	0.51	13	0.57
Total & Averages		100	56	1.00	49	1.37
10	5	25	17	4.08	14	4.10
	10	25	8	0.65	8	0.87
	15	25	10	0.52	11	0.58
	20	25	15	0.59	14	0.58
Total & Averages		100	50	1.46	47	1.53
15	5	25	12	3.22	14	4.77
	10	25	16	0.79	11	0.53
	15	25	11	0.52	9	0.46
	20	25	10	0.22	6	0.12
Total & Averages		100	49	1.18	40	1.47
20	5	25	20	4.48	18	6.04
	10	25	13	0.66	12	0.72
	15	25	16	0.93	11	0.39
	20	25	13	0.43	12	0.25
Total & Averages		100	62	1.63	53	1.85
Total & Grand Averages		400	217	1.32	189	1.56

Notes: *K*: number of customer orders; *N*: number of jobs; *NPI*: number of problem instances; *NIS*: number of instances with improved solutions in Phase-2; *API*: average percent improvement.

6.4.3 Comparison of the job-based and order-based processing approaches

In this section, we discuss our experiments' results to compare both job-based and order-based processing approaches. As mentioned in Section 3, the solution of the problem P_{OBP} gives the optimal sequence of the order-based processing problem with setup savings. From the results of our experiments, we have observed that the solutions of the problem P_{JBP} outperforms P_{OBP} even though there is a significant amount of reduction in setup times in the problem P_{OBP}. The job-based processing approach yields better solutions for 310 problems out of 400 problem instances when there are setup times. Our results also demonstrate that the job-based processing approach does not optimally solve large-size problems; however, this approach provides the best integer solutions, which are still better than the solutions obtained for the problems solved by the order-based processing approach. For example, for the problem set with ten customer orders and 15 jobs, 12 non-optimal solutions are found by the job-based approach. The total completion time values of these solutions are smaller than those of the order-based approach. As shown in Table 6, the job-based processing approach yields negative average percent deviations, which can be considered as an indication that the results obtained by the job-based processing approach are better than those obtained by the order-based processing approach with setup savings. As the number of customer orders and jobs increases, the percent deviation gets more significant and the job-based approach provides better solutions for the setup case.

Table 6
Comparison of the job-based and order-based processing approaches

K	*N*	*NPI*	*Average percent deviation*	
			Setup case	*No-setup case*
5	5	25	-9.59	26.31
	10	25	-10.34	28.61
	15	25	-11.58	32.20
	20	25	-10.48	33.57
Total & Averages		100	-10.50	30.17
10	5	25	-11.05	23.36
	10	25	-14.28	34.49
	15	25	-22.22	35.32
	20	25	-36.41	36.71
Total & Averages		100	-20.99	32.47
15	5	25	-15.66	29.73
	10	25	-17.90	38.49
	15	25	-23.93	41.51
	20	25	-26.58	42.49
Total & Averages		100	-21.02	38.06
20	5	25	-21.88	30.77
	10	25	-24.28	41.05
	15	25	-29.35	44.26
	20	25	-31.02	45.02
Total & Averages		100	-26.63	40.28
Total & Grand Averages		400	-19.78	35.24

However, the results differ when there is no setup time between jobs. As described in Remark 1 in Section 2, the problem P_{OBP} turns into the problem P'_{OBP} when we ignore setup times. Thus, both problems provide the same sequence of customer orders for the problem instances. In contrast, as shown in Table 6, the difference between job-based and order-based processing approaches now yields high positive average percent deviations between the solutions. This difference indicates that the order-based processing approach is better than the job-based processing approach when we ignore setup times in the same problem instances.

From the above analysis, we can conclude that customer order scheduling with the job-based processing approach yields better results when the jobs have setup times. However, the order-based processing approach is preferable when there is no setup time. Setup times are essential in production scheduling. Therefore, manufacturers or decision-makers need to employ convenient processing methods for the effective scheduling of customer orders.

7. Conclusions and future research

This study considers a customer order scheduling problem in a single machine to find a schedule with a sequence of jobs and the sequence of customer orders in each job when the job-based processing approach is used to minimize the customer orders' total completion time. We have proved that the order-based processing problem without setup savings in a single-machine environment is easy and polynomial-time solvable. Furthermore, we have developed two MILP models and a tabu search algorithm that respectively obtain optimal and near-optimal solutions for the job-based processing problem. Our empirical study shows that the second model (MILP-2) finds optimal solutions in a 3-hour limit of CPU time for problems up to 10 jobs regardless of the number of customer orders. However, there are problems with 15 and 20 jobs, which are not solved optimally owing to their numbers. From these observations, it is clear that solving the problem with a standard MILP solver seems ineffective especially for large-sized problem instances. The results also show that our proposed heuristic algorithm provides satisfactory solutions. It solves small and medium-sized problem instances optimally and finds near-optimal solutions for large-sized instances in a short computational time.

We have also compared the order-based and job-based processing approaches and observed that the job-based processing approach gives better results than the order-based processing approach when a setup on the machine is needed before starting to process each job (product). On the other hand, our observation was reversed towards the order-based processing as expected if there is no setup.

We believe that there are several fruitful issues for future research in the customer order scheduling problem with job-based processing:

1. A more elaborated metaheuristic, such as a genetic algorithm, could be developed and compared with our tabu search algorithm.
2. Total tardiness and the number of tardy customer orders could be other scheduling criteria to be investigated if there are due dates for the customer orders.
3. Considering the job-based processing approach on more complex machining environments, including parallel machines, flow shop, job shop, and open shop, would be other future study subjects in the customer order scheduling and multi-operation jobs scheduling literature.

References

Ahmadi, R.H., & Bagchi, U. (1990). Scheduling of multi-job customer orders in multi-machine environments. *ORSA/TIMS*, Philadelphia.

Bagchi, U., Julien, F.M., & Magazine, M.J. (1994). Note: due-date assignment to multi-job customer orders. *Management Science*, *40*(10), 1389–1392.

Baker, K.R. (1988). Scheduling the production of components at a common facility. *IIE Transactions*, *20*(1), 32–35.

Cheng, T.C.E., Ng, C.T., & Yuan, J.J. (2203). A stronger complexity result for the single machine multi-operation jobs scheduling problem to minimize the number of tardy jobs. *Journal of Scheduling*, *6*, 551–555.

Coffman, E.G., Nozari, A., & Yannakakis, M. (1989). Optimal scheduling of products with two subassemblies on a single machine. *Operations Research*, *37*(3), 426-436.

Çetinkaya, F.C., Çatmakaş, H.A., & Görür, A.K. (2019). Single-machine scheduling of indivisible multi-operation jobs. *South African Journal of Industrial Engineering*, *30*(3), 78–93.

Erel, E., & Ghosh, J.B. (2007). Customer order scheduling on a single machine with family setup times: Complexity and algorithms. *Applied Mathematics and Computation*, *185*(1), 11–18.

Gerodimos, A.E., Glass, C.A., Potts, C.N., & Tautenhahn, T. (1999). Scheduling multi-operation jobs on a single machine. *Annals of Operations Research*, *92*(1–4), 87–105.

Glover F. (1989). Tabu search - Part I. *ORSA Journal of Computing*, *1*, 190–206.

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

Hazır, O., Günalay, Y., & Erel, E. (2008). Customer order scheduling problem: A comparative metaheuristics study. *International Journal of Advanced Manufacturing Technology*, *3*(5-6), 589–598.

Julien, F.M., & Magazine, M.J. (1990). Scheduling customer orders: an alternative production scheduling approach. *Journal of Manufacturing and Operations Management*, *3*, 177–199.

Liu, C.H. (2009). Lot streaming for customer order scheduling problem in job shop environments. *International Journal of Computer Integrated Manufacturing 22*(9), 890–907.

Nawaz, M., Enscore, E., & Ham, I. (1983). A heuristic for the m-machine, n-job flow shop sequencing problem, *Omega*, *5*(11), 91.

Ng, D., Cheng, T.C.E., & Yuan, J.J. (2002). Strong NP-hardness of the single machine multi-operation jobs total completion time scheduling problem. *Information Processing Letters*, *82*, 187–191.

Pinedo, M.L. **(**2008). *Scheduling: Theory, Algorithms and Systems*. Springer: New York.

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

Su, L.H. & Chen, Y.H. 2009. Scheduling multi-operation jobs on a single flexible machine. *International Journal of Advanced Manufacturing Technology*, *42*, 1165–1174.

Vickson, R.G., Magazine, M.J., & Santos, C.A. (1993) Batching and sequencing of components at a single facility. *IIE Transactions*, *25*(2), 65–70.

Yang, D.L., Hou, Y.T., & Kuo, W.H. (2017). A note on a single-machine lot scheduling problem with indivisible orders. *Computers and Operations Research*, *79*, 34–38.

Yang, J. (2011). Customer order scheduling in a two machine flowshop, *International Journal of Management Science*, *17*(1), 1921–1939.

Appendix

A.1 Numerical example

We consider a problem instance in which five customers give orders for five products (jobs). Products demanded by the customer orders, the sequence-independent setup times, and the unit-processing times are given in Table A.1.

Table A.1
Data set for the numerical example

Jobs (Products)	Demand (in units) of the customer orders					Setup Time	Unit-processing Time
	O_1	O_2	O_3	O_4	O_5		
J_1	9	3	1	6	3	41	4
J_2	-	-	-	5	-	48	6
J_3	-	-	3	8	7	5	4
J_4	-	6	5	-	-	40	6
J_5	-	-	-	5	4	47	8

Phase 1 - Finding an initial schedule by the insertion algorithm

Step 1 Sorting the jobs in descending order of their number of customer orders gives the initial job sequence as:
$J_1\{O_1, O_2, O_3, O_4, O_5\} - J_3\{O_3, O_4, O_5\} - J_4\{O_2, O_3\} - J_5\{O_4, O_5\} - J_2\{O_4\}$

Step 2 In the initial job sequence, sorting the customer orders of each job in ascending order of their total demand yields the following initial sequence of jobs with the sorted customer orders:
$J_1\{O_3[1], O_2[3], O_5[3], O_4[6], O_1[9]\} - J_3\{O_3[3], O_5[7], O_4[8]\} - J_4\{O_3[5], O_2[6]\} - J_5\{O_5[4], O_4[5]\} - J_2\{O_4[5]\}$

Step 3 From the initial job sequence $J_1 - J_3 - J_4 - J_5 - J_2$ obtained in Step 2, we select the first two jobs J_1 and J_3, and form two partial sequences $\boldsymbol{J_1} - J_3$ and $J_3 - \boldsymbol{J_1}$.

Step 4 In the first partial sequence $\boldsymbol{J_1} - J_3$, the optimal sequence of the customer orders in each job is:
$J_1\{O_2[3], O_1[9], O_3[1], O_5[3], O_4[6]\} - J_3\{O_3[3], O_5[7], O_4[8]\}$
with the total completion time of customer orders
$TCT(J_1 - J_3) = CT_2 + CT_1 + CT_3 + CT_5 + CT_4 = 53 + 89 + 146 + 174 + 206 = 668.$
In the second partial sequence $J_3 - \boldsymbol{J_1}$, the optimal sequence of the customer orders in each job is:
$J_3\{O_3[3], O_5[7], O_4[8]\} - J_1\{O_3[1], O_2[3], O_5[3], O_4[6], O_1[9]\}$
with the total completion time of customer orders
$TCT(J_3 - J_1) = CT_3 + CT_2 + CT_5 + CT_4 + CT_1 = 122 + 134 + 146 + 174 + 206 = 778.$
We select the partial sequence $J_1 - J_3$ since its total completion time is less than that of the partial sequence $J_3 - J_1$.

Step 5 All jobs of the initial job sequence obtained in Step 2 are not considered yet. Thus, we go to Step 4.

Step 4 We select the next job, which is job J_4, from the initial job sequence obtained in Step 2, and form three partial sequences $\boldsymbol{J_4} - J_1 - J_3$, $J_1 - \boldsymbol{J_4} - J_3$, and $J_1 - J_3 - \boldsymbol{J_4}$ from the selected partial sequence $J_1 - J_3$.

Step 5 In the first partial sequence $\boldsymbol{J_4} - J_1 - J_3$, the optimal sequence of the customer orders in each job is:
$J_4\{O_3[5], O_2[6]\} - J_1\{O_2[3], O_1[9], O_3[1], O_5[3], O_4[6]\} - J_3\{O_3[3], O_5[7], O_4[8]\}$
with the total completion time of customer orders
$TCT(J_4 - J_1 - J_3) = CT_2 + CT_1 + CT_3 + CT_5 + CT_4 = 159 + 195 + 252 + 280 + 312 = 1{,}198.$
In the second partial sequence $J_1 - \boldsymbol{J_4} - J_3$, the optimal sequence of the customer orders in each job is:
$J_1\{O_1[9], O_3[1], O_2[3], O_5[3], O_4[6]\} - J_4\{O_2[6], O_3[5]\} - J_3\{O_3[3], O_5[7], O_4[8]\}$
with the total completion time of customer orders
$TCT(J_1 - J_4 - J_3) = CT_1 + CT_2 + CT_3 + CT_5 + CT_4 = 77 + 205 + 252 + 280 + 312 = 1{,}126.$
In the third partial sequence $J_1 - J_3 - \boldsymbol{J_4}$, the optimal sequence of the customer orders in each job is:
$J_1\{O_1[9], O_3[1], O_2[3], O_5[3], O_4[6]\} - J_3\{O_5[7], O_4[8], O_3[3]\} - J_4\{O_3[5], O_2[6]\}$
with the total completion time of customer orders
$TCT(J_1 - J_3 - J_4) = CT_1 + CT_5 + CT_4 + CT_4 + CT_2 = 77 + 162 + 194 + 276 + 312 = 1{,}021.$
Among these three partial sequences, we select the partial sequence $J_1 - J_3 - J_4$ since its total completion time is less than those of other partial sequences.

Step 4 We select the next job, which is job J_5, from the initial job sequence obtained in Step 2, and form four partial sequences $\boldsymbol{J_5} - J_1 - J_3 - J_4$, $J_1 - \boldsymbol{J_5} - J_3 - J_4$, $J_1 - J_3 - \boldsymbol{J_5} - J_4$, and $J_1 - J_3 - J_4 - \boldsymbol{J_5}$ from the selected partial sequence $J_1 - J_3 - J_4$.

Step 5 In the first partial sequence $\boldsymbol{J_5} - J_1 - J_3 - J_4$, the optimal sequence of the customer orders in each job is:
$J_5\{O_5[4], O_4[5]\} - J_1\{O_1[9], O_3[1], O_2[3], O_5[3], O_4[6]\} - J_3\{O_5[7], O_4[8], O_3[3]\} - J_4\{O_3[5], O_2[6]\}$
with the total completion time of customer orders
$TCT(J_5 - J_1 - J_3 - J_4) = CT_1 + CT_5 + CT_4 + CT_3 + CT_2 = 196 + 281 + 313 + 395 + 431 = 1{,}616.$
In the second partial sequence $J_1 - \boldsymbol{J_5} - J_3 - J_4$, the optimal sequence of the customer orders in each job is:
$J_1\{O_1[9], O_3[1], O_2[3], O_5[3], O_4[6]\} - J_5\{O_5[4], O_4[5]\} - J_3\{O_5[7], O_4[8], O_3[3]\} - J_4\{O_3[5], O_2[6]\}$
with the total completion time of customer orders
$TCT(J_1 - J_3 - J_4 - J_5) = CT_1 + CT_5 + CT_4 + CT_3 + CT_2 = 77 + 281 + 313 + 395 + 431 = 1{,}497.$
In the third partial sequence $J_1 - J_3 - \boldsymbol{J_5} - J_4$, the optimal sequence of the customer orders in each job is:

$J_1\{O_1[9], O_3[1], O_2[3], O_5[3], O_4[6]\} - J_3\{O_3[3], O_5[7], O_4[8]\} - J_5\{O_5[4], O_4[5]\} - J_4\{O_3[5], O_2[6]\}$
with the total completion time of customer orders
$TCT(J_1 - J_3 - J_5 - J_4) = CT_1 + CT_5 + CT_4 + CT_3 + CT_2 = 77 + 285 + 325 + 395 + 431 = 1{,}513.$
In the fourth partial sequence $J_1 - J_3 - J_4 - \boldsymbol{J_5}$, the optimal sequence of the customer orders in each job is:
$J_1\{O_1[9], O_3[1], O_2[3], O_5[3], O_4[6]\} - J_3\{O_3[3], O_5[7], O_4[8]\} - J_4\{O_3[5], O_2[6]\} - J_5\{O_5[4], O_4[5]\}$
with the total completion time of customer orders
$TCT(J_1 - J_3 - J_4 - J_5) = CT_1 + CT_3 + CT_2 + CT_5 + CT_4 = 77 + 276 + 312 + 391 + 431 = 1{,}487.$
Among these four partial sequences, we select the partial sequence $J_1 - J_3 - J_4 - J_5$ since its total completion time is less than those of other partial sequences.

Step 4 We select the next job, which is job J_2, from the initial job sequence obtained in Step 2, and form five complete sequences $\boldsymbol{J_2} - J_1 - J_3 - J_4 - J_5$, $J_1 - \boldsymbol{J_2} - J_3 - J_4 - J_5$, $J_1 - J_3 - \boldsymbol{J_2} - J_4 - J_5$, $J_1 - J_3 - J_4 - \boldsymbol{J_2} - J_5$, and $J_1 - J_3 - J_4 - J_5 - \boldsymbol{J_2}$ from the selected partial sequence $J_1 - J_3 - J_4 - J_5$.

Step 5 In the first complete sequence $\boldsymbol{J_2} - J_1 - J_3 - J_4 - J_5$, the optimal sequence of the customer orders in each job is:
$J_2\{O_4[5]\} - J_1\{O_1[9], O_3[1], O_2[3], O_5[3], O_4[6]\} - J_3\{O_3[3], O_5[7], O_4[8]\} - J_4\{O_3[5], O_2[6]\} - J_5\{O_5[4], O_4[5]\}$
with the total completion time of customer orders
$TCT(J_2 - J_1 - J_3 - J_4 - J_5) = CT_1 + CT_3 + CT_2 + CT_5 + CT_4 = 1{,}877.$
In the second complete sequence $J_1 - \boldsymbol{J_2} - J_3 - J_4 - J_5$, the optimal sequence of the customer orders in each job is:
$J_1\{O_1[9], O_3[1], O_2[3], O_5[3], O_4[6]\} - J_2\{O_4[5]\} - J_3\{O_3[3], O_5[7], O_4[8]\} - J_4\{O_3[5], O_2[6]\} - J_5\{O_5[4], O_4[5]\}$
with the total completion time of customer orders
$TCT(J_1 - J_2 - J_3 - J_4 - J_5) = CT_1 + CT_3 + CT_2 + CT_5 + CT_4 = 1{,}799.$
In the third complete sequence $J_1 - J_3 - \boldsymbol{J_2} - J_4 - J_5$, the optimal sequence of the customer orders in each job is:
$J_1\{O_1[9], O_3[1], O_2[3], O_5[3], O_4[6]\} - J_3\{O_3[3], O_5[7], O_4[8]\} - J_2\{O_4[5]\} - J_4\{O_3[5], O_2[6]\} - J_5\{O_5[4], O_4[5]\}$
with the total completion time of customer orders
$TCT(J_1 - J_3 - J_2 - J_4 - J_5) = CT_1 + CT_3 + CT_2 + CT_5 + CT_4 = 1{,}799.$
In the fourth complete sequence $J_1 - J_3 - J_4 - \boldsymbol{J_2} - J_5$, the optimal sequence of the customer orders in each job is:
$J_1\{O_1[9], O_3[1], O_2[3], O_5[3], O_4[6]\} - J_3\{O_3[3], O_5[7], O_4[8]\} - J_4\{O_3[5], O_2[6]\} - J_2\{O_4[5]\} - J_5\{O_5[4], O_4[5]\}$
with the total completion time of customer orders
$TCT(J_1 - J_3 - J_4 - J_2 - J_5) = CT_1 + CT_3 + CT_2 + CT_5 + CT_4 = 1{,}643.$
In the fifth complete sequence $J_1 - J_3 - J_4 - J_5 - \boldsymbol{J_2}$, the optimal sequence of the customer orders in each job is:
$J_1\{O_1[9], O_3[1], O_2[3], O_5[3], O_4[6]\} - J_3\{O_3[3], O_5[7], O_4[8]\} - J_4\{O_3[5], O_2[6]\} - J_5\{O_5[4], O_4[5]\} - J_2\{O_4[5]\}$
with the total completion time of customer orders
$TCT(J_1 - J_3 - J_4 - J_5 - J_2) = CT_1 + CT_3 + CT_2 + CT_5 + CT_4 = 1{,}565.$
Among these five complete sequences, we select the sequence $J_1 - J_3 - J_4 - J_5 - J_2$ since its total completion time, which is 1,565 time units, is less than those of other complete sequences. Thus, the initial schedule obtained by Phase 1 is
$J_1\{O_1[9], O_3[1], O_2[3], O_5[3], O_4[6]\} - J_3\{O_3[3], O_5[7], O_4[8]\} - J_4\{O_3[5], O_2[6]\} - J_5\{O_5[4], O_4[5]\} - J_2\{O_4[5]\}$

Phase 2 – Improving the initial schedule by the tabu search algorithm

Step 1 We set $ic = 1$, the initial schedule σ_1 to the schedule obtained in Phase 3 of the algorithm, and the best schedule σ_B to σ_1, i.e., $\sigma_B = \sigma_1 = J_1 - J_3 - J_4 - J_5 - J_2$ with $TCT(\sigma_B) = 1{,}565$.

Step 2 When we apply the adjacent pairwise interchanges of the jobs in the current schedule σ_1, we generate four mutations $\boldsymbol{J_3} - \boldsymbol{J_1} - J_4 - J_5 - J_2$, $J_1 - \boldsymbol{J_4} - \boldsymbol{J_3} - J_5 - J_2$, $J_1 - J_3 - \boldsymbol{J_5} - \boldsymbol{J_4} - J_2$, and $J_1 - J_3 - J_4 - \boldsymbol{J_2} - \boldsymbol{J_5}$. When we apply the algorithm SCO for each of these mutations, the candidate schedule σ_C becomes $J_1 - \boldsymbol{J_4} - \boldsymbol{J_3} - J_5 - J_2$ since $\min\{TCT(\boldsymbol{J_3} - \boldsymbol{J_1} - J_4 - J_5 - J_2),\ TCT(J_1 - \boldsymbol{J_4} - \boldsymbol{J_3} - J_5 - J_2),\ TCT(J_1 - J_3 - \boldsymbol{J_5} - \boldsymbol{J_4} - J_2),\ TCT(J_1 - J_3 - J_4 - \boldsymbol{J_2} - \boldsymbol{J_5})\} = \min\{1{,}642;\ 1{,}434; 1697; 1{,}643\} = 1{,}434$.

Step 3 The tabu list is updated with a pair of (J_3, J_4). We set $\sigma_2 = \sigma_C = J_1 - J_4 - J_3 - J_5 - J_2$, and the new best schedule to the current schedule since the total completion time of the current schedule is less than that of the best schedule. That is, $\sigma_B = \sigma_C = J_1 - J_4 - J_3 - J_5 - J_2$ since $TCT(\sigma_C) = TCT(J_1 - \boldsymbol{J_4} - \boldsymbol{J_3} - J_5 - J_2) = 1{,}434 < TCT(\sigma_B) = TCT(J_1 - J_3 - J_4 - J_5 - J_2) = 1{,}565$.

Step 4 We set $ic = ic + 1 = 1 + 1 = 2$, and go to Step 2 since the iteration counter ic is smaller than the pre-specified value NI for the number of iterations, i.e., $ic = 2 < NI = 2 \times N = 2 \times 5 = 10$.

Step 2 When we apply the adjacent pairwise interchanges of the jobs in the current schedule $\sigma_2 = J_1 - J_4 - J_3 - J_5 - J_2$, we generate three mutations $\boldsymbol{J_4} - \boldsymbol{J_1} - J_3 - J_5 - J_2$, $J_1 - J_4 - \boldsymbol{J_5} - \boldsymbol{J_3} - J_2$, and $J_1 - J_4 - J_3 - \boldsymbol{J_2} - \boldsymbol{J_5}$. Note that the mutation $J_1 - \boldsymbol{J_3} - \boldsymbol{J_4} - J_5 - J_2$ is not possible since the pair of (J_3, J_4) is in the tabu list. When we apply the algorithm SCO for each of the three possible mutations, we observe that $\min\{TCT((\boldsymbol{J_4} - \boldsymbol{J_1} - J_3 - J_5 - J_2)),\ TCT(J_1 - J_4 - \boldsymbol{J_5} - \boldsymbol{J_3} - J_2),\ TCT(J_1 - J_4 - J_3 - \boldsymbol{J_2} - \boldsymbol{J_5})\} = \min\{1{,}530; 1{,}561; 1{,}512\} = 1{,}512 > TCT(\sigma_B) = TCT(J_1 - J_4 - J_3 - J_5 - J_2) = 1{,}434$. Thus, the TS algorithm terminates before reaching the tabu-search iteration-size of $NI = 2 \times N = 2 \times 5 = 10$.

The total completion time of the schedule obtained by the heuristic algorithm is 1,434 time units, which equals that of the optimal schedule obtained by solving the MILP model. Fig. A.1 illustrates the Gantt chart for this optimal schedule.

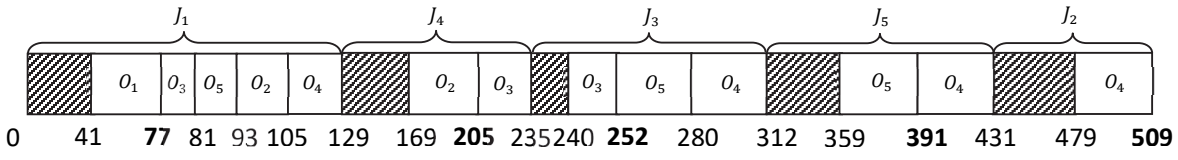


Fig. A.1. Gantt chart of the schedule for the numerical example problem