Single Machine Scheduling Problems With Sequence-dependent Setup Times and Precedence Delays

Shih-Wei Lin
Chang Gung University
Kuo-Ching Ying (kcying@ntut.edu.tw)
National Taipei University of Technology

Research Article

Keywords: Sequence-dependent setup times (SDSTs), delayed precedence (DP), Single Machine Scheduling Problems, manufacturing settings

Posted Date: December 13th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-1148434/v1

Single Machine Scheduling Problems with Sequence-dependent Setup Times and Precedence Delays

Shih-Wei Lin[1,2,3], and Kuo-Ching Ying[4,*]

[1] Department of Information Management, Chang Gung University, Taoyuan 333, Taiwan, R.O.C.
[2] Linkou Chang Gung Memorial Hospital, Taoyuan 333, Taiwan, R.O.C.
[3] Ming Chi University of Technology, New Taipei City 243, Taiwan, R.O.C.
[4] Department of Industrial Engineering and Management, National Taipei University of Technology, Taipei 106, Taiwan, R.O.C.
* corresponding: kcying@ntut.edu.tw

Abstract

Sequence-dependent setup times (SDSTs) and delayed precedence (DP) occur commonly in various manufacturing settings. This study investigated the single machine scheduling problem with SDSTs and DP constraints arising in an amplifier assembly company. A mixed-integer linear programming model and a lean iterated greedy (LIG) algorithm is proposed to search for the best job sequence with minimum makespan. Based on the characteristic of delayed precedence constraints of the problem, the proposed LIG algorithm implements a straightforward but effective lean construction mechanism, which can keep the search process within the feasible solution space and quickly converge toward the (near-) global optimum. Computational results reveal that LIG significantly outperforms the state-of-the-art algorithm in terms of solution quality and computational efficiency. This study mainly contributes to providing a simple, effective, and efficient algorithm that can facilitate industrial applications and act as a new benchmark approach for future research.

Introduction

Scheduling jobs on a single machine/facility/resource, production line, entire manufacturing system, or manufacturing system with a bottleneck station can all be considered as examples of the single machine scheduling problem (SMSP)[1]. In the past decades, SMSPs have been receiving much interest. This study aims to optimize the makespan of the SMSP with sequence-dependent setup times (SDSTs) and delayed precedence (DP) constraints, which is a significant (sub-)problem in production scheduling[2]. The SDSTs constraint occurs when the setup times of jobs are dependent on their immediate predecessors. Explicitly considering the SDSTs constraint in scheduling decisions can significantly save setup costs, eliminate waste, and increase productivity in many real-world industrial settings, especially in the textiles, printing, chemical, pharmaceuticals, and metallurgical industries[3,4].

Nonetheless, DP constraint means that specific pairs of jobs require a delay between the completion time of the predecessor and the start time of the successor. This constraint usually occurs when the workpiece needs to be processed multiple times on the machine/facility based on precedence constraints and waits for its temperature to drop, or its glue/color painted to dry[2]. Using the three-field notation scheme proposed by Graham[5], the addressed problem can be signified as $1|s_{ij}, prec(d_{ij})|C_{\max}$, where 1 indicates that the shop type is a single machine; s_{ij} and $prec(d_{ij})$ represent the scheduling problem with the SDSTs and DP constraints, respectively; and $C_{\max}$ designates the scheduling objective is to minimize the makespan, which is directly related to machine utilization and commonly used in the most scheduling problems.

Compared with other restrictions, there has been limited research on SMSPs with DP constraints. To the best of our knowledge, Wikum et al.[6] were the first study to identify and systematically treat SMSPs with DP constraints explicitly. They considered minimizing different objective functions in SMSPs subject to minimum DP constraints, maximum DP constraints, or a combination of the two DP constraints for various types of precedence relations. Wikum et al.[6] proved that except for the simplest of precedence relations, most of these SMSPs are NP-hard and then proposed a variety of results, including polynomial algorithms for some special cases, heuristic methods, and the worst-case bounds of two SMSPs with minimum DP constraints. Nevertheless, their study did not consider the SDSTs constraints.

Balas et al.[7] investigated an SMSP with release dates, delivery times, and DP constraints without SDSTs constraints. The authors first indicated that this problem is a relaxation of the jobshop scheduling problem (JSP), which is tighter than the standard SMSP relaxation. Then, Balas et al.[7] proposed a modified shifting bottleneck algorithm that used the relaxation to minimize makespan in JSPs. Computational results revealed consistently better solutions using this approach compared to other algorithms on all classes of JSPs, at a higher computing time.

Finta and Liu[8] studied an SMSP with PD constraints with an aim to minimize the makespan. For a scheduling problem with DP constraints, the release dates (RDs) of jobs are unknown before scheduling. The authors applied the concept of DP constraints to model RDs of jobs and calculated them depending on a given feasible schedule. Finta and Liu[8] proved that the problem is NP-hard in the strong sense when the delay times and execution times are integer lengths and units, respectively. While the delay times and execution times are unit lengths and arbitrary integers, the problem is polynomial solvable using an $O(n^2)$ optimal algorithm. However, their study, too, did not consider the SDSTs constraints.

Brucker et al. [9] investigated an SMSP with start-start DP constraints, which allow positive and negative time-lags between the start times of two jobs. Unfortunately, their study ignored SDSTs. They indicate that some complex jobshop and openshop scheduling problems with multi-processor tasks, multi-purpose machines, or changeover times can be reduced to such an SMSP. Brucker et al. [9] showed that SMSPs with start-start relations is a very complex problem and developed a branch and bound (B&B) algorithm for solving the SMSP with arbitrary time-lags and the reduction problems of classical jobshop and openshop scheduling problems. However, computational results revealed that the transformation and the B&B algorithm could not efficiently solve the classical jobshop and openshop scheduling problems.

Motivated by a practical problem in an amplifier assembly company, Kuo et al.[2] proposed the first study regarding the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem. The amplifier assembly procedure consists of several steps, in which each step is executed at a time. Since different parts/products need different tools, fixtures, and assembly components, SDSTs are required in advance processing of all parts/products. As the covers of amplifiers are assembled with special glue, DP constraints exist in the assembly procedure, waiting for the glue to dry. Kuo et al.[2] presented a makespan calculation model, a non-linear programming model, a variable neighborhood search (VNS) metaheuristic, and five simulated annealing (SA) algorithms for solving the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem. Computational results based on 16 randomly generated scenarios revealed that VNS outperformed the five SA algorithms in solving the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem and provided robust solutions for different scenarios of DPs.

The above review of the related literature indicates that there is a dearth of research concerning SMSPs that consider both SDSTs and DP constraints. This study proposes a mixed-integer linear programming (MILP) model and a more effective and efficient lean iterated greedy (LIG) algorithm for solving the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem further to bridge the gap between scheduling theory and practical applications. Since VNS is the best-to-date metaheuristic, we set it as the benchmarking algorithm in this study. The rest of this paper begins with defining the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem and formulates its MILP model in section 2, and section 3 elaborates on the proposed LIG algorithm. The experimental results and statistical analysis based on different scenarios are reported in section 4. Finally, concluding remarks and recommendations for future research are proposed in section 5.

Problem description and MILP model formulation

This section gives a formal definition of the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem and formulates it as an MILP model discussed below.

Problem description. The $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem involves a finite set *J* of *n* independent jobs that have to be processed on a single machine/facility/resource. The processing time of the job j $(j=1,...,n)$ on the machine is p_j. An SDSTs, s_{ij}, is required before processing job *j*, if it is processed immediately after job *i*. Let *A* be the set of DP constraints of all jobs. For certain pairs of jobs $(i,j)\in A$, the succeeding job *j* can only be executed after its predecessor has completed the job and the corresponding delay time, d_{ij}, between the completion time of the preceding job *i* and the start time of the job has elapsed. The objective is to find a feasible schedule of these *n* jobs that satisfied the precedence delays to minimize the makespan, $C_{\max}$, i.e., the completion time of the last executed job. Besides, the following assumptions are made in the addressed $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem.

- The processing times, SDSTs, and delay times are assumed to be non-negative integers.
- Each job is assumed to be available for processing immediately at the commencement of the scheduling period.
- Each job has to be processed on the machine exactly once without preemption.
- Each job is independent of the other and its processing time is known, fixed, and finite.
- Each job is processed as early as possible. Thus, there is no intentional job waiting or machine idle time.
- The machine is available at the beginning of the scheduling period.
- The machine can process at most one job at a time.
- The machine is continuously available for processing jobs throughout the scheduling period, and there are no interruptions due to breakdowns, maintenance, or any such cause.
- The machine is provided with adequate waiting space for allowing jobs to wait before starting their processing.

MILP formulation. This study developed an MILP model for the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem. The following notations were used in the proposed formulation.

Indices

j , i	Job tags, $j = 1,2,...,n;\ i = 0,1,...,n$, where 0 is a dummy job
A	Set of delayed precedence constraints of all jobs

Parameters

n	Number of jobs
b_j	Starting time of job j
s_{ij}	Sequence-dependent setup times required for processing of job j after job i
d_{ij}	Delay time needed for the processing of job j after job i
p_j	Processing time of job j
M	A sufficiently large positive number

Decision variables

x_{ij}	Binary variable, $x_{ij} = \begin{cases} 1, & \text{if job } j \text{ is processed immediately after job } i; \\ 0, & \text{otherwise.} \end{cases}$
c_i , c_j	Completion time of job i and job j

With the above preliminaries and notations, the MILP model of the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem can be formulated as follows.

Minimize $C_{\max}$ (1)

subject to

$$C_{\max} \geq c_j;\ j = 1,2,...,n, \tag{2}$$

$$\sum_{j=0}^{n} x_{ij} = 1;\ \ i = 0,1,...,n\ \ (i \neq j), \tag{3}$$

$$\sum_{i=0}^{n} x_{ij} = 1;\ \ j = 1,2,...,n\ \ (i \neq j), \tag{4}$$

$$b_j \geq c_i + s_{ij} - M(1 - x_{ij});\ \ i = 1,2,...,n;\ j = 1,2,...,n\ \ (i \neq j), \tag{5}$$

$$b_j \geq c_i + d_{ij};\ \forall (i,j) \in A, \tag{6}$$

$$c_j = b_j + p_j;\ j = 1,2,...,n, \tag{7}$$

$$x_{ij} \in \{0,1\};\ i = 0,1,...,n;\ \ j = 1,2,...,n\ (i \neq j). \tag{8}$$

Constraint (1) sets the objective function of the problem. Constraint set (2) calculates the makespan. Constraint sets (3) and (4) ensure that each job is dispatched to only one position in the sequence. Constraint sets (5) and (6) outline the relationship between the starting time of a job and the completion time of its predecessor. Constraint set (7) describes the relationship between the completion time of a job with its starting time and processing time. Finally, constraint set (8) defines the domains of the decision variable x_{ij} .

The lean iterated greed algorithm

In the past decades, researchers have proposed different meta-heuristics for solving intractable combinatorial optimization problems. Among these meta-heuristics, the iterative greedy (IG) algorithm[10] is one of the most efficient and effective methods applied successfully in solving various scheduling problems[11-15]. The original IG algorithm proposed earlier[10] consists of two key phases: destruction and reconstruction. In each iteration, some incumbent solution elements are first selected and removed through the destruction phase. Consequently, in the following reconstruction phase, all the removed elements are sequentially reinserted into the remaining partial solution using a greedy heuristic until a new solution is reassembled again. After a new solution is constructed, some acceptance criteria are used to judge whether or not the incumbent solutions will be replaced by it. The above process is iterated until certain termination conditions are met.

Based on the basic frame of IG and the characteristic of DP constraints, this study presents an LIG algorithm to solve the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem. The proposed LIG algorithm proposes a straightforward but effective lean construction mechanism, which can avoid generating infeasible solutions and quickly converge to the (near-) global optimum. In the following subsections, we will further discuss the solution representation, the construction of the DP constraints table, the makespan calculation method,

the lean construction mechanism, and the detailed procedures of the LIG algorithm.

Solution representation. In the proposed LIG algorithm, a feasible solution Π is coded as a string of n numbers representing the job permutation without violating the DP constraints. For example, a solution coded as [1 7 2 3 5 8 6 4] indicates that it has eight jobs, and the job permutation is 1-7-2-3-5-8-6-4. In the following sub-sections we discuss how to ensure that the generated solution does not violate the DP constraints.

Construction of the DP constraints table. This study created a DP constraints table to quickly determine whether or not inserting a removed job into a specific position in the partial solution will violate the DP constraint. Assuming that each DP constraint is expressed as $DPC(i, j) = d_{ij}$, which means that job *j* cannot be processed before the job *i* and its start time should be delayed at least d_{ij} time units after the completion time of job *i*. Then, the DP constraints table with n jobs can be constructed as follows.

Step 1. As shown in Table 1, if the predecessor job i is not equal to the successor job j, the grid (*i*, *j*) is filled with *U* (undetermined) as the initial value; otherwise, it is filled with *P* (prohibited) as the initial value.

Step 2. For each pair of jobs with $DPC(i, j) = d_{ij}$, the grid (*i, j*) and its corresponding grid (*j, i*) are refilled with d_{ij} and *P* (prohibited), respectively. Also, if any job *k* must be processed after job *j*, the grid (*k, i*) is refilled with *P* (prohibited).

Table 1. Initial values fill in the DP constraints table.

Predecessor\Successor	Job 1	Job 2	L	Job *n*-1	Job *n*
Job 1	*P*	*U*	*U*	*U*	*U*
Job 2	*U*	*P*	*U*	*U*	*U*
M	M	M	M	M	M
Job *n*-1	*U*	*U*	*U*	*P*	*U*
Job *n*	*U*	*U*	*U*	*U*	*P*

For example, assuming there are eight jobs with five DP constraints, as shown in Table 2. Since the first DP constraint is $DPC(1,2) = 35$, the grid (1, 2) and its corresponding grid (2, 1) are refilled with 35 and *P*, respectively. The second DP constraint is $DPC(2,3) = 0$, the grids (2, 3) and (3, 2) are refilled as 0 and *P*, respectively. Since job 3 also must be processed after job 1, the grid (3, 1) is refilled with *P*. In the same way, processing the three remaining DP constraints gives the final lag times of the DP constraints table as shown in Table 3.

Table 2. DP constraints of the example.

No	Constraint
1	$DPC(1,2) = 35$
2	$DPC(2,3) = 0$
3	$DPC(3,4) = 13$
4	$DPC(5,6) = 24$
5	$DPC(7,8) = 46$

Table 3. Final lag times of the DP constraints table.

Predecessor\Successor	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7	Job 8
Job 1	*P*	35	*U*	*U*	*U*	*U*	*U*	*U*
Job 2	*P*	*P*	0	*U*	*U*	*U*	*U*	*U*
Job 3	*P*	*P*	*P*	13	*U*	*U*	*U*	*U*
Job 4	*P*	*P*	*P*	*P*	*U*	*U*	*U*	*U*
Job 5	*U*	*U*	*U*	*U*	*P*	24	*U*	*U*
Job 6	*U*	*U*	*U*	*U*	*P*	*U*	*U*	*U*
Job 7	*U*	*U*	*U*	*U*	*U*	*U*	*P*	46
Job 8	*U*	*U*	*U*	*U*	*U*	*U*	*P*	*P*

The makespan calculation method. Assume that J_s is the set of jobs that have been scheduled, $L_{j[k+1]}$ which means the lag time of the grid (*j*, *k*+1) in the DP constraints table, and $\pi_{[j]}$ ($j = 0,1,2,...,n$) denotes the job sequenced in the j^{th} position of a given feasible solution $\Pi = (\pi_{[0]}, \pi_{[1]}, ..., \pi_{[n]})$, where $\pi_{[0]}$ is a dummy job with $p_{[0]} = 0$ and $s_{[0][1]} = 0$. Then, the makespan of a given feasible solution Π can be calculated using the procedure presented in Figure 1.

Set $J_s = \varnothing$

For $(k = 0;\ k \le n;\ k++)$ {

$b_{[k+1]} = \max\{C_{[k]} + s_{[k][k+1]},\ \max_{j \in J_s}\{C_j + L_{j[k+1]}\}\}$;

$c_{[k+1]} = b_{[k+1]} + s_{[k][k+1]} + p_{[k+1]}$;

$J_s =: J_s \cup \pi_{[k+1]}$;

}

Output $C_{\max} = c_{[n]}$

Figure 1. The procedure of the makespan calculation.

Using the DP constraints table and SDSTs data listed in Tables 3 and 4, respectively, as an example, the makespan of a given feasible solution $\Pi = (1, 7, 2, 3, 5, 8, 6, 4)$ can be calculated as shown in Table 5.

Table 4. The sequence-dependent setup times of the example.

Predecessor\Successor	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7	Job 8
Job 1	-	8	8	9	5	5	4	6
Job 2	6	-	9	9	7	8	6	4
Job 3	9	5	-	7	7	6	4	6
Job 4	8	8	7	-	6	6	5	4
Job 5	4	5	9	5	-	9	8	3
Job 6	6	6	6	7	6	-	4	8
Job 7	5	5	6	4	4	3	-	3
Job 8	7	5	6	5	6	4	4	-

Table 5. Calculation of the makespan for given feasible solution.

Job	SDST	Lag Time	Start Time	Processing Tine	Completion Time
1	0	0	0	12	12
7	4	0	16	12	28
2	5	14	47	11	58
3	9	0	67	14	81
5	7	0	88	16	104
8	3	0	107	15	122
6	4	2	128	14	142
4	7	0	149	17	166

The lean construction mechanism. To avoid generating infeasible solutions that violate any DP constraints and to quickly converge to the (near-) optimal solution, the proposed LIG algorithm implements a straightforward but effective lean construction mechanism, and is described below.

Assuming that there are k jobs in the current partial solution, there will be k+1 possible positions to insert the next job removed in the previous destruction phase. Some positions may violate the DP constraints, therefore, there is no need to test all positions and select the best one to insert the removed job. The following two properties are implemented in the lean construction mechanism to reduce the waste of unnecessary searching. First, if we insert the removed job into a position immediately before the job J_b in the partial solution, it will violate any DP constraints, then we insert it into any position before J_b will also violate DP constraints and is unnecessary. On the other hand, if we insert the removed job into a position immediately after the job J_a in the partial solution, it will violate any DP constraints, then we insert it into any position after J_a will also violate DP constraints and is unnecessary. Using the lean construction mechanism, many infeasible positions that violate DP constraints can be discarded in the *reconstruction* phase of LIG and make it quickly converge to (near-) optimum.

Procedures of the proposed LIG algorithm. The procedures of the proposed LIG algorithm are detailed as follows.

Step 1. Generate feasible initial solution

The method of constructing the initial solution is to add jobs one by one to the last position of the current partial solution until the complete schedule is rebuilt. To ensure that the generated initial solution is feasible, in each iteration, the job that satisfies DP constraints and with the shortest completion time is selected next. The detailed procedure of constructing the initial

solution is shown in Figure 2, where J_c the set of candidate jobs meet the DP constraints in each iteration; and J_u denotes the set of unscheduled jobs.

Set $J_s = \varnothing$
For $(k = 0;\ k \le n;\ k++)$ {
  For each job $i \in J_c$ {
  $b_i = \max\{C_{[k]} + s_{[k]i},\ \max_{j \in J_s}\{C_j + L_{j[k+1]}\}\}$;
  $c_i = b_i + s_{[k]i} + p_i$;
  }
  $\pi_{[k+1]} = \arg\{\underset{i \in J_c}{Min}\ c_i\}$;
  $J_s =: J_s \cup \pi_{[k+1]}$;
  $J_u =: J_u - \{\pi_{[k+1]}\}$;
  $J_c =: J_c - \{\pi_{[k+1]}\}$;
  For each job $i \in J_u$, add it to J_c if all its precedence jobs are in J_s;
  }

Figure 2. The detailed procedure of constructing the initial solution.

Step 2: Destruction and reconstruction phases

(a) *Destruction* phase: Randomly select D jobs from the incumbent solution $\Pi_{incumbent}$; then, remove them into Π_d and sort them in the order in which the jobs are selected. Set the remaining jobs in the incumbent solution as the current partial solution Π_p.

(b) *Reconstruction* phase: Apply the lean construction mechanism to sequentially insert the jobs from Π_d into Π_p until a new completed solution Π_{new} is rebuilt.

Step 3: Acceptance criteria

Apply the following criteria to judge whether or not the incumbent solution $\Pi_{incumbent}$ and the best-found solution Π_{best} will be updated by Π_{new}.

IF $C_{\max}(\Pi_{new}) \le C_{\max}(\Pi_{best})$, set $\Pi_{best} := \Pi_{new}$ and $\Pi_{incumbent} := \Pi_{new}$;
ELSE_IF $C_{\max}(\Pi_{new}) \le C_{\max}(\Pi_{incumbent})$, set $\Pi_{incumbent} := \Pi_{new}$;
ELSE_IF $C_{\max}(\Pi_{new}) > C_{\max}(\Pi_{incumbent})$, generate $r \sim$ U (0,1);
  IF $r < Exp(-\Delta E)$, set $\Pi_{incumbent} := \Pi_{new}$
  Otherwise, discard Π_{new}.

Here, $C_{\max}(\bullet)$ denotes the makespan of a specific solution $(\bullet)$; $r \in [0,1]$ is a random number generated from the uniform distribution U(0,1); $\Delta E = [C_{\max}(\Pi_{new}) - C_{\max}(\Pi_{incumbent})] / [SF \times C_{\max}(\Pi_{incumbent})]$, wherein, SF is a scale factor used to control the probability of accepting a worse solution.

Step 4: Stopping criterion

Repeat Steps 2 to 3 until the maximum allowable computational time $T_{\max}$ is reached. In this study, $T_{\max} = \tau \times n$ (CPU time in seconds), in which τ is a parameter that controls the maximum allowable computational time.

Using the data shown in Tables 2-4 as an example, one iteration of the proposed LIG algorithm is presented in Figure 3 to clearly illustrate the procedure. In Step 1, a feasible initial solution $\Pi = (1, 7, 2, 3, 5, 8, 6, 4)$ with $C_{\max}(\Pi) = 166$ was generated. In Step 2, the *destruction* and *reconstruction* phases, which can be regarded as a perturbation mechanism, were executed. In Steps 2(a), three jobs (*i.e.*, jobs 8, 5, 1) were randomly selected and sequentially removed from the incumbent solution to Π_d. In Step 2(b), a new complete solution $\Pi_{new} = (1, 7, 5, 2, 3, 5, 4, 8)$ with $C_{\max}(\Pi_{new}) = 150$ was rebuilt by sequentially inserting the jobs in $\Pi_d = (8, 5, 1)$ into $\Pi_p = (7, 2, 3, 5, 4)$. Then, in Step 3, Π_{best} and $\Pi_{incumbent}$ were updated by Π_{new} according to the acceptance criteria. If the quality of Π_{new} is inferior to that of $\Pi_{incumbent}$, the Boltzmann function ($e^{[C_{\max}(\Pi_{new}) - C_{\max}(\Pi_{incumbent})]/[SF \times C_{\max}(\Pi_{incumbent})]}$) was used to determine whether or not $\Pi_{incumbent}$ could be replaced by Π_{new}. This mechanism is usually implemented in the annealing process of the SA algorithm to facilitate the incumbent solution to escape from the local optimum.

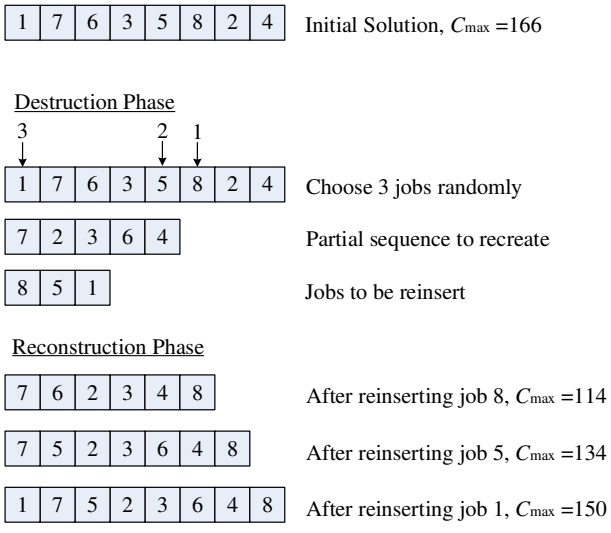


Figure 3. An example for one iteration of the LIG algorithm.

Computational experimentation and results

An extensive computational experiment was conducted to evaluate the performance of the proposed IG algorithm in solving the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem. The following subsections describe the benchmark problem set, parameter calibration, and computational results of LIG compared with the state-of-the-art algorithm.

The benchmark problem set. To fairly compare the performance of the proposed LIG algorithm with that of the existing best solution algorithm, *i.e.*, VNS, the benchmark problem set in a similar approach considered by Kuo et al.[2] was generated and used in this study. The benchmark problem set consisted of 540 test instances that were produced as described herein. The number of jobs $n = \{10, 15, 20, 25, 50, 75, 100\}$; the processing times were generated randomly using the uniform distributions [10, 20] and [20, 30], respectively; the SDSTs were generated randomly using the uniform distributions [5, 10] and [10, 15], respectively; the delay times were generated randomly using the uniform distributions [20, 40] and [40, 60], respectively; $\lceil n/2 \rceil$ number of randomly selected jobs were with precedence constraints, in which $\lceil n/4 \rceil$ number of jobs were with zero delay time. On this basis, ten distinct test instances for each of the $7\times2\times2\times2\times1\times1=56$ configurations were generated to analyze the performance of LIG under various operational scales and workloads. The files of these test instances can be downloaded from http://swlin.cgu.edu.tw/data/SDSTsDP_Data.7z.

The proposed LIG algorithm and VNS were coded and compiled in Visual C++ (2017), in which the source code of VNS was provided by Kuo et al.[2]. All the experiments were carried out using a personal computer (PC) with the following specs: Intel® Core™ Xeon CPU E5-1620v2 @ 3.70GHz processor, 64 GB RAM, and a Windows 10 operating system. The MILP mathematical model was solved via Gurobi version 9.0 on the same PC with a maximal computing time of 3600 seconds for each test instance. The final incumbent solution generated by the Gurobi MILP solver was recorded as the feasible solution.

Parameter calibration. Before beginning with the computational experiments, 12 test instances used the same data generation procedures as mentioned above were generated to calibrate the three parameters, *D*, *SF*, and τ, of the LIG algorithm. For this purpose, as shown in Table 6, four alternatives of each parameter were considered to analyze which results were better outcomes. Each of the 12 test instances was solved 20 times in the preliminary testing. The resulting maximum (Max.), average (Ave.), and minimum (Min.) makespan values for each parameter combination were recorded, and their respective relative percent of deviations (*RPDs*) were computed as follows.

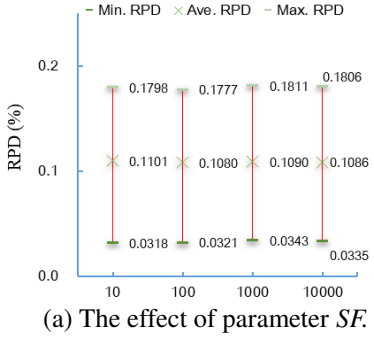
$$RPD = (C_{\max}(\Pi)^{\min} - C_{\max}(\Pi)^{P}) / C_{\max}(\Pi)^{\min} \times 100\%$$

where $C_{\max}(\Pi)^{P}$ is the makespan value yielding by the parameter combination *P*, and $C_{\max}(\Pi)^{\min}$ denotes the minimum makespan value that was yielded among all parameter combinations.

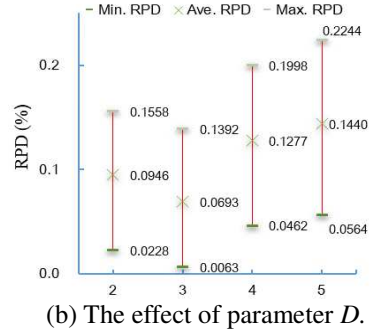
Table 6. Parameter values in the calibration experiments.

Parameter	Value tested
SF	10, 100, 1000, 10000
D	2, 3, 4, 5
τ	0.05, 0.10, 0.15, 0.20

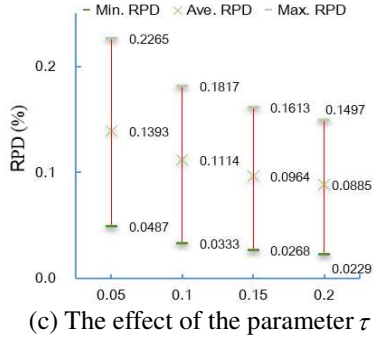
Figure 4 displays the effect of each parameter on the solution quality. The parameter *SF* affects the probability of accepting a worse solution. In general, the higher the value of *SF*, the higher likelihood of accepting a worse solution, and therefore, the slower the convergence speed of the LIG algorithm. However, assuming that the value of SF is too small, the possibility of accepting a worse new solution will also be very small, and the algorithm will not be able to escape from the local optimum. As shown in Figure 4(a), when the value of *SF* is too small, the corresponding Ave. *RPD* will be slightly larger. However, there is no significant difference between different *SF* values. The value of *D* signifies the number of jobs removed in each iteration of the destruction phase, which may affect the range of searching for neighborhood solutions. Figure 4(b) demonstrates that when the value of *D* is too high or too low, the average *RPD* is larger. Therefore, when the value of *D* is too small, the search range of the neighborhood solution is too narrow, resulting in poor quality of the solution. However, when too much jobs are removed, a better solution may not be obtained under the limitation of the maximum computation time. Figure 4(c) demonstrates that the higher the value of τ, the smaller the value of Ave. *RPD*. That is, a better solution can be obtained by using more computational time. Based on the computational time and the quality of solutions in preliminary testing, $SF = 100$, $D = 3$, and $\tau = 0.1$ were selected for the final experiments.



(a) The effect of parameter *SF*.



(b) The effect of parameter *D*.



(c) The effect of the parameter τ .

Figure 4. The effect of each parameter on the solution quality

Results and discussion. This study used the average relative percentage deviation (Ave. *RPD*) as a performance metric to compare the performance of the LIG and VNS algorithms. For each small-scale instance ($n = 10$ and 15), the *RPD* of the solution obtained using LIG, VNS, and the MILP mathematical model were calculated according to the following expression:

$$RPD = \frac{C_{\max}^{Method} - C_{\max}^{LB}}{C_{\max}^{LB}} \times 100\%$$

where $C_{\max}^{Method}$ is the best makespan value of the feasible solution obtained using a specific method among five trials, and $C_{\max}^{LB}$ is the lower bound (LB) of the makespan value obtained by the MILP mathematical model with a maximal computing time of 3600 seconds for each test instance. A feasible solution is optimum when the gap between its makespan value and the LB is zero.

Table 7 summarizes the computational results for the small-scale instances, showing that the total average *RPD* value of the solutions obtained using LIG was smaller than those obtained using the VNS and MILP model. The MILP model achieved optimal solutions in 80 out of 80 and 1 out of 80 test instances for the test instances with $n = 10$ and $n = 15$, respectively. Notably, all optimal solutions obtained using the MILP model were also obtained using LIG. Overall, LIG got 8 and 93 better solutions than those obtained using the MILP model and VNS, respectively, in 160 small-scale instances. The CPU times consumed by LIG and VNS were significantly shorter than those used by the MILP model; hence, one can conclude that the tested metaheuristics may be desired more for solving small-scale instances when the efficiency of the approach is the key indicator. These analytical results confirm that the proposed LIG algorithm exhibits excellent convergence to optimal solutions than those of the VNS algorithm and the MILP model.

Table 7. Average *RPD*s (%) and running time for small-scale instances.

	Processing		Delay	MILP		LIG		VNS	
n	Time	SDST	Time	RPD	Time	RPD	Time	RPD	Time
10	[10, 20]	[5,10]	[20,40]	0.000	3.76	0.000	1.002	0.597	4.56
10	[10, 20]	[5,10]	[40,60]	0.000	2.96	0.000	1.010	0.052	4.45
10	[10, 20]	[10,15]	[20,40]	0.000	5.51	0.000	1.009	0.238	4.44
10	[10, 20]	[10,15]	[40,60]	0.000	3.12	0.000	1.009	0.574	4.53
10	[10, 20]	[5,10]	[20,40]	0.000	4.83	0.000	1.009	0.196	4.47
10	[20, 30]	[5,10]	[40,60]	0.000	3.73	0.000	1.009	0.663	4.42
10	[20, 30]	[10,15]	[20,40]	0.000	6.91	0.000	1.009	0.000	4.48
10	[20, 30]	[10,15]	[40,60]	0.000	4.35	0.000	1.007	0.433	4.49
15	[20, 30]	[5,10]	[20,40]	1.181	3600.02	1.079	1.509	2.801	5.44
15	[10, 20]	[5,10]	[40,60]	1.141	3359.48	1.105	1.509	3.416	5.47
15	[10, 20]	[10,15]	[20,40]	0.782	3600.02	0.756	1.510	1.723	5.49
15	[10, 20]	[10,15]	[40,60]	0.950	3600.02	0.894	1.510	2.288	5.44
15	[20, 30]	[5,10]	[20,40]	0.618	3600.03	0.618	1.509	1.412	5.42
15	[20, 30]	[5,10]	[40,60]	0.776	3600.02	0.731	1.510	1.984	5.45
15	[20, 30]	[10,15]	[20,40]	0.531	3600.02	0.512	1.509	1.256	5.41
15	[20, 30]	[10,15]	[40,60]	0.611	3600.02	0.611	1.508	1.573	5.48
Total Ave.				0.412	1787.18	0.394	1.259	1.200	4.96

We now elaborate on the numerical results for solving larger test instances. The $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem is very complex, therefore, high-quality feasible solutions and LBs cannot be obtained using the MILP model for the test instances with $n = 20$, 25, 50, 75, and 100. Therefore, the *RPD* value calculated for each larger test instance is computed using the following expression:

$$RPD = \frac{C_{\max}^{Method} - C_{\max}^{best}}{C_{\max}^{best}} \times 100\% ,$$

where $C_{\max}^{Method}$ is the best makespan value of the solution obtained using a specific method among five trials, and $C_{\max}^{best}$ is the best makespan value among the solutions obtained using LIG and VNS.

Table 8 lists the computational results for the larger test instances, showing that the best and mean *RPD*s of the solutions obtained using LIG were significantly smaller than those obtained using VNS in terms of all categories of the larger test instances. The total average best and mean *RPD*s obtained using the proposed LIG algorithm were 0.000% and 0.050%, respectively, while those obtained using VNS were 3.279% and 3.580%, respectively. The mean *RPD*s obtained by LIG were smaller than those of the VNS, showing that the performance of LIG was more robust than VNS under all the larger production

scales and workloads. Furthermore, the computational times consumed by LIG were much smaller than those of VNS. The gap between computational times of LIG and VNS was observed to increase with an increase in the number of jobs.

As a final step to the numerical analysis, one-sided paired *t*-tests in terms of best and mean *RPD*s were conducted to verify LIG, and was found significantly superior to VNS. The statistical results are summarized in Table 9. According to Table 9, at a confidence level of $\alpha = 0.05$, statistical tests supported the proposed LIG algorithm that significantly outperformed the VNS algorithm in terms of best and mean *RPD*s for both small and larger test instances.

Table 8. Average *RPD*s (%) and running time for larger test instances.

	LIG			VNS		
Category	Best *RPD*s	Mean *RPD*s	Time (s)	Best *RPD*s	Mean *RPD*s	Time (s)
n						
25	0.000	0.059	2.50	2.891	3.229	8.43
50	0.000	0.066	4.98	4.842	5.194	19.54
75	0.000	0.045	7.48	5.218	5.490	34.16
100	0.000	0.045	9.98	0.620	0.806	40.01
Processing Time						
[10, 20]	0.000	0.056	5.39	3.841	4.224	22.54
[20, 30]	0.000	0.037	5.41	2.567	2.803	21.25
SDSTs						
[5, 10]	0.000	0.053	5.39	3.533	3.866	22.00
[10, 15]	0.000	0.040	5.41	2.874	3.159	21.79
Delay Times						
[20, 40]	0.000	0.045	5.39	3.140	3.439	21.85
[40, 60]	0.000	0.048	5.41	3.265	3.585	21.94
Total Ave.	0.000	0.050	5.73	3.279	3.580	23.35

Table 9. Statistical results from paired-*t* tests ($\alpha = 0.05$).

Problem Size	Type	IG *vs.*	VNS
Small	Best RPD	Paired difference (*RPD*)	−0.1717
		t-value	−7.6120
		Degree of freedom	159
		P-value	0.0000
Small	Mean RPD	Paired difference (*RPD*)	−0.6172
		t-value	−18.5008
		Degree of freedom	159
		P-value	0.0000
Larger	Best RPD	Paired difference (*RPD*)	−0.6774
		t-value	−49.9303
		Degree of freedom	399
		P-value	0.0000
Larger	Mean RPD	Paired difference (*RPD*)	−0.9443
		t-value	−59.2770
		Degree of freedom	399
		P-value	0.0000

Concluding remarks

The $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem is a theoretical and practical subject that nearly matches the conditions in the existing production system. The present study contributes to the literature by proposing an MILP model and a very efficient and effective LIG algorithm, filling an essential gap in solution methodology development for solving this understudied problem. Numerical

experiments showed that LIG finds consistently better schedules than the benchmarking algorithm in all test instances at less computational costs, making it a viable solution approach for the relevant industrial applications.

Despite its significance in real-world applications, the study of the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem is still in the early stages and remains an area for potentially future research. Several directions for future research warrant further exploitation. First, more effective and efficient exact methods, approximation algorithms, and matheuristics are the areas for further studies. Second, an alternative performance measure or multiple criteria SMSP with SDSTs and DP constraints is worth considering. Third, from a practical perspective, new extensions to the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem, such as multi-agent problems, can be further exploited to address other industry-specific situations. Finally, extending the $1|s_{ij}, prec(d_{ij})|C_{\max}$ problem to more complex issues like stochastic and dynamic models is worth further research.

References

[1] Pereira J. The robust (minmax regret) single machine scheduling with interval processing times and total weighted completion time objective. Comput Oper Res 2016;66:141–152. https://doi.org/10.1016/j.cor.2015.08.010

[2] Kuo Y, Chen SI, Yeh YH. Single machine scheduling with sequence-dependent setup times and delayed precedence constraints. Oper Res Int J 2020;20:927–942. https://doi.org/10.1007/s12351-017-0349-y

[3] Choobineh FF, Mohebbi E, Khoo H. A multi-objective tabu search for a single-machine scheduling problem with sequence-dependent setup times. Eur J Oper Res 2006;175:318–337. https://doi.org/10.1016/j.ejor.2005.04.038

[4] Allahverdi A. The third comprehensive survey on scheduling problems with setup times/costs. Eur J Oper Res 2015;246:345–378. https://doi.org/10.1016/j.ejor.2015.04.004

[5] Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discret Math 1979;5:287–326. https://doi.org/10.1016/S0167-5060(08)70356-X

[6] Wikum ED, Llewellyn DC, Nemhauser GL. One-machine generalized precedence constrained scheduling problems. Oper Res Lett 1994;16:87–99. https://doi.org/10.1016/0167-6377(94)90064-7

[7] Balas E, Lenstra JK, Vazacopoulos A. The one-machine problem with delayed precedence constraints and its use in job scheduling. Manage Sci 1995;41:94–109. https://doi.org/10.1287/mnsc.41.1.94

[8] Finta L, Liu Z. Single machine scheduling subject to precedence delays. Discrete Appl Math 1996;70:247–266. https://doi.org/10.1016/0166-218X(96)00110-2

[9] Brucker P, Hilbig T, Hurink J. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. Discrete Appl Math 1999;70:247–266. https://doi.org/10.1016/S0166-218X(99)00015-3

[10] Jacobs LW, Brusco MJ. A local-search heuristic for large set-covering problems. Nav Res Logist 1995;42:1129–1140. https://doi.org/10.1002/1520-6750(199510)42:7

[11] Ying KC, Lee ZJ. Lu CC, Lin SW. Metaheuristics for scheduling a no-wait flowshop manufacturing cell with sequence dependent family setups. Int. J. Adv. Manuf. Technol 2012;58:671–682. https://doi.org/10.1007/s00170-011-3419-y

[12] Lin SW, Ying KC, Wu WJ, Chiang YI. Multi-objective unrelated parallel machine scheduling: a Tabu-enhanced iterated Pareto greedy algorithm, Int J Prod Res 2016;54: 1110–1121. https://doi.org/10.1080/00207543.2015.1047981

[13] Ying KC, Lin SW, Cheng CY, He CD. Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems, Comput Ind Eng 2017;110:413–423. https://doi.org/10.1016/j.cie.2017.06.025

[14] Ying KC, Lin SW. Minimizing makespan in distributed blocking flowshops using hybrid iterated greedy algorithms. IEEE Access 2017;5:15694–15705. https://doi.org/10.1109/ACCESS.2017.2732738

[15] Cheng, C.Y., Pourhejazy, P., Ying, K.C. & Huang, S. Y. New benchmark algorithm for minimizing total completion time in blocking flowshops with sequence-dependent setup times. Appl. Soft. Comput. 104, no.107229, DOI: https://doi.org/10.1016/j.asoc.2021.107229 (2021).

Author contributions statement

S.W. Contributed in funding acquisition, conceptualization, methodology, and formal analysis. K.C. contributed in conceptualization, methodology, and writing original draft. All authors reviewed the manuscript.

Acknowledgment

The authors would like to acknowledge Professor Yiyo Kuo for providing the source code of the VNS algorithm. The work of S.-W. Lin is partially supported by the Ministry of Science and Technology, Taiwan, under Grant MOST109-2410-H-182-009MY3, and the Linkou Chang Gung Memorial Hospital under Grant BMRPA19. The work of the K-C Ying is partially supported by the Ministry of Science and Technology, Taiwan, under Grant MOST109-2221-E-027-073.

Competing interests

The authors declare no competing interests.