Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

A novel hybrid algorithm of cooperative variable neighborhood search and constraint programming for flexible job shop scheduling problem with sequence dependent setup time

Yajie Wu[a], Shiming Yang[a], Leilei Meng[a*], Weiyao Cheng[a], Biao Zhang[a] and Peng Duan[a*]

[a]*School of Computer Science, Liaocheng University, Liaocheng 252000, China*

C H R O N I C L E

Article history:
Received September 16 2024
Received in Revised Format October 26 2024
Accepted November 23 2024
Available online November 23 2024

Keywords:
Flexible job shop scheduling problem
Sequence dependent setup time
Constraint programming
Variable neighborhood search

A B S T R A C T

This study focuses on the flexible job shop scheduling problem with sequence-dependent setup times (FJSP-SDST), and the goal is minimizing the makespan. To solve FJSP-SDST, first, we develop a constraint programming (CP) model to obtain optimal solutions. Due to the NP-hardness of FJSP-SDST, a CP assisted meta-heuristic algorithm (C-VNS-CP) is designed to make use of the advantages of both CP model and cooperative variable neighborhood search (C-VNS). The C-VNS-CP algorithm consists of two stages. The first stage involves C-VNS, for which eight neighborhood structures are defined. In the second stage, CP is used to further optimize the good solution obtained from C-VNS. In order to prove the efficiency of the C-VNS algorithm, CP model, and C-VNS-CP algorithm, experiments of 20 instances are conducted.

1. Introduction

The job shop scheduling problem (JSP), as a classic NP-hard problem, has been extensively researched by many scholars. Compared with JSP, the flexible job shop scheduling problem (FJSP) considers machine flexibility. In a JSP problem, each operation can only be processed on only one machine. In the FJSP problem, one operation can be processed on multiple machines. Therefore, the FJSP problem must determine two sub-problems: (1) machine selection sub-problem. (2) operations sequencing sub-problem. Thus, the FJSP problem is a more complex NP-hard problem than the JSP problem (Meng and Zhang et al., 2020).Traditional FJSP does not consider the actual production situation of the job shop. For example, in an actual productive process, a machine needs to conduct some adjusting tasks when it processes two different jobs, such as installation of job or tool replacement. Therefore, the time spent on handling these tasks cannot be ignored. Based on the characteristics and actual demand of FJSP, studying the FJSP-SDST is of great research significance.

Obtaining the optimal solutions is very important for scheduling problems. Therefore, a constraint programming (CP) model is formulated to obtain optimal solutions. Variable neighborhood search (VNS) has been widely implemented to solve shop scheduling problems. It alternates between different neighborhood structures for searching. When one neighborhood structure gets stuck at local optimum, it will jump to the next neighborhood to continue searching (Meng and Cheng et al., 2024). Thus, it can quickly obtain a near optimal solution to the problem. For FJSP-related problems, cross operators have been proved to be very effective. Therefore, in this paper, a cooperative VNS (C-VNS) algorithm is designed upon the cooperation of two VNSs. Specifically, C-VNS takes the advantages of both VNS and effective cross operators for FJSP-SDST. Moreover, to make use of both the advantages of C-VNS and CP model, a CP assisted C-VNS (C-VNS-CP) is designed. The C-VNS-CP algorithm consists of two stages. The first stage involves C-VNS, for which eight neighborhood structures are defined. In the second stage, CP is used to further optimize the good solution obtained from C-VNS. Comparing this work with existing studies, the contributions can be summed up as follows:

* Corresponding author
E-mail mengleilei@lcu-cs.com (L. Meng) duanpeng@lcu-cs.com (P. Duan)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2025 Growing Science Ltd.
doi: 10.5267/j.ijiec.2024.11.003

(1) A CP model is formulated for obtaining optimal solutions.
(2) A novel cooperative meta-heuristic C-VNS is proposed for obtaining approximate optimal solutions.
(3) A hybrid algorithm C-VNS-CP of the CP model and the C-VNS algorithm is proposed to make use of their advantages.

The remainder of the paper is described as below: Section 2 introduces the related research about FJSP-SDST. Section 3 describes the FJSP-SDST and designs the CP model. Section 4 introduces the C-VNS-CP algorithm in detail. Section 5 presents the experimental setups and findings. Section 6 provides a comprehensive summary of the conclusions drawn and future research.

2 Literature review

2.1 Literature review of FJSP-SDST

For the FJSP-SDST focused on minimizing the makespan, Saidi-Mehrabad and Fattahi (2007) proposed a mixed integer linear programming (MILP) model based on adjacent sequence modeling idea and a tabu search (TS) algorithm. Subsequently, Shen et al.(2018) formulated a MILP model based on sequence modeling idea along with an improved TS algorithm, incorporating specific neighborhood functions and a diversity of structures. Moreover, a sequence-based MILP model was evaluated to be more effective than adjacent sequence-based MILP model. Azzouz et al. (2017) developed a novel hybrid algorithm that combines a genetic algorithm (GA) and a local search-based algorithm to solve the FJSP-SDST problem, aiming to minimize the makespan. For multi-objective FJSP-SDST, Sun et al. (2021) devised a hybrid many-objective evolutionary algorithm aimed at minimizing the makespan, total machine workload, maximum machine workload, and earliness/tardiness penalties. Zhang al.(2020) investigated the FJSP-SDST with transportation time (FJSP-SDST-T), and formulated an improved multi-objective GA, aiming to minimize three objectives. Zhang et al. (2024) proposed an evolutionary algorithm that combines reinforcement learning aimed at minimizing four objectives at the same time. For the FJSP-SDST-T problem, Li et al.(2020) designed an improved version of the multi-objective Jaya algorithm, and minimized both energy consumption and makespan. For the FJSP-SDST-T issue, Li and Lei (2021) formulated a multi-objective algorithm with considering feedback mechanism, and minimized the makespan, total delay and total energetic consumption simultaneously. Meng et al. (2024) formulated several MILP models for FJSP-SDST and FJSP-SDST-T with minimizing the energy consumption. In addition, Meng et al. (2023) presented a MILP model of FJSP-SDST-T with considering Off/On strategy for minimizing energy consumption. Zhang et al.(2020) presented a collaborative migratory bird optimization algorithm for solving the FJSP-SDST by considering batch splitting.

2.2 Literature review of CP modeling for FJSP-related problems

Zeballos et al. (2010) designed a CP model for tool allocation in FJSP, and aimed to minimize the number of total tool copies and makespan. And, Ham and Cakici (2016) built a parallel CP model for FJSP to batch processing with minimizing the makespan. For minimizing the makespan, Novas et al. (2019) formulated a CP model for the FJSP with considering lot splitting. Ham et al. (2021) developed a CP model for jobs the FJSP with multi-AGV, aiming to minimize the makespan. Ham et al. (2020) extended the CP model for FJSP with multi-AGV. Ham et al. (2021) created a CP model for the FJSP with an energy-saving strategy of shutdown, and minimized the makespan and energy consumption. For minimizing the makespan, Meng et al. (2020) created a CP model for the distributed FJSP.

2.3 Literature of VNS for FJSP-related problems

For solving the FJSP with worker flexibility, Meng et al.(2019) proposed an improved VNS to minimize the energy consumption. Moreover, eight neighborhood structures were designed. Bagheri and Zandieh (2011) presented a multi-objective VNS aimed at minimizing the makespan and mean tardiness of the FJSP-SDST. With the knowledge of FJSP, Karimi et al. (2012) developed an improved VNS with minimizing the makespan, and designed seven neighborhood structures. Yazdani et al.(2010) built a parallel VNS for FJSP with minimizing the makespan. For solving specific open shop scheduling problems, de Abreu et al. (2022) formulated an efficient VNS with considering CP search. Alicastro et al. (2021) presented a VNS algorithm that considers Q-learning for solving additive manufacturing machine scheduling problems.

2.4 Summary of existing researches

As can be seen above, FJSP-SDST has gained significant attention in recent times. However, the existing research is focused on meta-heuristic algorithms and MILP models. No specialized research focuses on the CP model and the hybrid algorithm combining CP and meta-heuristic algorithms for solving FJSP-SDST. Moreover, existing research about VNS only considers one VNS and cannot make use of the effectiveness of crossover operators for FJSP-related problems. Therefore, to make up for the gaps of existing research, the CP model, cooperative VNS, and hybrid algorithm of CP and cooperative VNS are developed and studied in this paper.

3. CP modeling for FJSP-SDST

3.1 FJSP-SDST definition

The definition of FJSP-SDST with minimizing the makespan, is as follows: in the workshop, a certain number of jobs need to be processed, and each job has several operations. The operation of the same job must be processed in a specific sequence. Each operation can be handled by several machines. SDST is considered when different operations of various jobs are processed on the same machine. In addition, FJSP-SDST needs to satisfy the following constraints:

(1) Initial states: All jobs and machines can be processed and used at time 0;
(2) Machine assignment: Each operation can solely be machined by one specific machine of eligible machines;
(3) Non-overlapping of machine processing: at any time, a machine can process at most one operation;
(4) Operations sequence of a job: the processing of the operations of the same job must follow the determined order;
(5) No preemption of all operations: for each operation, it can be interrupted once it is started.

The symbols in the CP model are as follows：

Table 1
Definitions of symbols in the CP model

Symbols	Definitions
i, i'	job indices
n	total number of jobs
n_i	operations' number of job i
I	set of jobs, and $I = \{1,2,3,...,n\}$
j, j'	operation indices
N	total number of operations, and $N = \sum_{i \in I} n_i$
J_i	set of operations of job $_i$,and $J_i = \{1,2,3,...,n_i\}$
k	machine indices
m	total number of machines
M	a very large positive integer
K	set of machines, and $K = \{1,2,3,...,m\}$
$O_{i,j}$	the j -th operation of job i
$K_{i,j}$	machine set for processing operation $O_{i,j}$
$pt_{i,j,k}$	processing time for operation $O_{i,j}$ on machine k
$st_{i,j,i',j',k}$	setup time of operations $O_{i,j}$ and $O_{i',j'}$ when they are adjacently processing on machine k . Specifically, when $i = i'$, $st_{i,j,i',j',k} = 0$
T_k	SDST matrix of $st_{i,j,i',j',k}$

3.2 CP model

Regarding CP, it is an effective method to obtain optimal solutions (Meng and Gao et al., 2022). CP determines the machine selection sub-problem by defining optional interval decision variables, and determines the operations sequencing sub-problem by defining sequence decision variables.

Decision variables:

$Ops_{i,j}$	It denotes interval variable for operation $O_{i,j}$.
$mod_{i,j,k}$	It denotes optional interval variable for operation $O_{i,j}$.
$mchs_k$	It denotes sequence decision variable for optional interval variable $mod_{i,j,k}$ of machine k .

The objective of minimizing the makespan can be seen from function (1).

$$\min C_{\max} = \max_{i \in I}(endOf(Ops_{i,j})) \tag{1}$$

$$endBeforeStart(Ops_{i,j}, Ops_{i,j+1}), \forall i \in I, \forall j \in \{1,2,...,n_i - 1\} \tag{2}$$

$$alternative(Ops_{i,j}, mod_{i,j,k}), \forall i \in I, \forall j \in J_i \quad (3)$$

$$noOverlap(mchs_k, T_k, 1), \forall k \in K \quad (4)$$

where, function (1) shows the objective of minimizing the makespan, and function $endOf(Ops_{i,j})$ returns the completion time of interval variable $Ops_{i,j}$ .Constraint set (2) indicates that for each operation of a job, it can be started only when its previous operation of the job is finished. Function $endBeforeStart(Ops_{i,j}, Ops_{i,j+1})$ means that the starting time of interval variable $Ops_{i,j+1}$ is no less the ending time of interval variable $Ops_{i,j}$. Constraint set (3) indicates that only one machine is selected to process one operation, and function $alternative(Ops_{i,j}, mod_{i,j,k})$ indicates that there is only one optional interval variable $mod_{i,j,k}$ is present for interval variable $Ops_{i,j}$. For instance, if operation O_{11} can be processed on machines 1 and 2, then only one of $mod_{1,1,1}$ and $mod_{1,1,2}$ can be present for $Ops_{1,1}$. Constraint set (4) indicates that for each machine can process at most one operation at any time. Moreover, when two operations of different jobs are processed adjacently on a machine, SDST must be considered. Function $noOverlap(mchs_k, T_k, 1)$ means the non-overlapping of the optional interval variables $mod_{i,j,k}$ of $mchs_k$ with considering setup times. For instance, if $mod_{1,1,1}$ and $mod_{2,1,1}$ are present and adjacent on machine 1, then the starting time of $mod_{2,1,1}$ is no less the sum of the starting time of $mod_{1,1,1}$ and SDST $st_{1,1,2,1,1}$.

3.3An example

In this article, in order to better introduce CP model, Fig. 1 provides an example. This example includes three jobs and each job contains two operations. The operations O_{21} and O_{32} are processed on machine 1, the operations O_{11} and O_{12} are processed on machine 2, and the operations O_{31} and O_{22} are processed on machine 3. Then, the optional decision variables $mod_{2,1,1}, mod_{3,2,1}, mod_{1,1,2}, mod_{1,2,2}, mod_{3,1,3}$ and $mod_{2,2,3}$ are present.

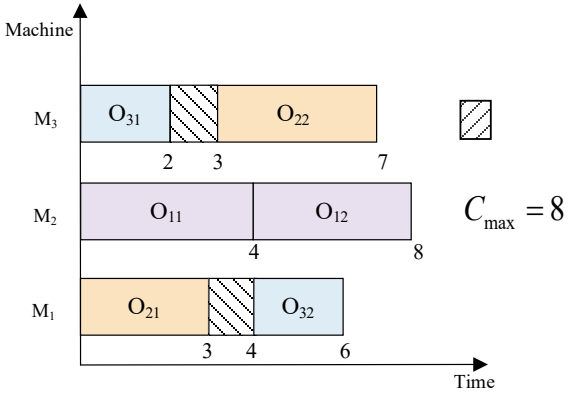


Fig. 1. An example for FJSP-SDST

4. The C-VNS-CP algorithm for FJSP-SDST

This section provides a detailed introduction to the C-VNS-CP from five aspects: initialization, encoding, decoding, neighborhoods, and how to connect C-VNS with CP. The C-VNS-CP algorithm consists of two stages. The first stage part involves C-VNS, for which eight neighborhood structures are defined. In the second stage, CP is used to further optimize the good solution obtained from the C-VNS algorithm.

4.1 Workflow of the proposed C-VNS-CP

Fig. 2 presents the flowchart of the devised C-VNS-CP algorithm, with the following detailed steps:

Step 1 (Initialization): According to Section 4.2, generate the initial solutions x_1 and x_2 for two VNSs respectively, and go to Step 2.

Step 2: Set $gen = 0$, and repeat the Steps 3-6 until the stopping criteria of C-VNS is met.

Step 3 (Shaking operation): Determining one neighborhood randomly, and produce the neighborhood solutions x'_1 and x'_2 of solutions x_1 and x_2 . Go to Step 4.

Step 4: Set $k = 1$, and repeat the following Steps 4-1, and 4-2 until $k > 8$.

Step 4-1 (Local Search): Generate neighborhood solutions of solutions x'_1 and x'_2 by neighborhood k . Specifically,

solutions x''_1 and x''_2 are the best solutions of SN neighborhood solutions of solutions x'_1 and x'_2 respectively. Go to step 4-2.

Step 4-2 (Updating): If solution x''_1 outperforms x_1, update x_1 by using x''_1. If solution x''_2 is better than solution x_2, update x_2 by using x''_2. If solutions x_1 or x_2 is updated, set $k=1$; otherwise, set $k=k+1$.

Step 5: If solutions x_1 or x_2 are improved, then set $gen=0$; otherwise, set $gen=gen+1$. Go to Step 6.

Step 6 (Restart operation): If the $gen > RN$ is met, conduct the restart operation in Section 4.4 and set $gen=0$.

Step 7 (CP search): CP search is performed on the best solution of x_1 and x_2.

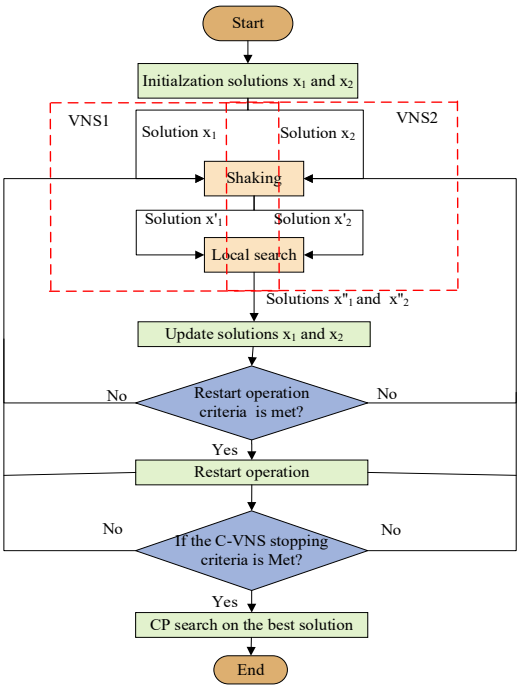


Fig. 2. The flowchart of C-VNS-CP algorithm

4.2 Initialization

To generate a good initial solution of VNS, 3000 solutions are randomly generated and two best solutions with the smallest makespan are selected as the initial solutions of two VNSs respectively.

4.3 Encoding scheme

The encoding uses two-layer strings in this paper. Specifically, the two-layer strings are named as OS string and MS string. The OS string determines operations sequencing sub-problem. The OS string's length corresponds to the total count of all operations of all jobs. Each number in the OS string represents the jobs number. The sequence of numbers indicates the processing order of the operations. The same number indicates different operations of the same job. The MS string determines the machine selection sub-problem. The MS string represents the chosen machines for all operations of all jobs. Its length equals the OS string. To clearly introduce the encoding, Fig. 3 provides an example. In Fig. 3, for the OS string, the first number '1' denotes the first operation of job 1, while the second number '1' denotes the second operation of job 1. For MS string, the first three numbers represent the machine selections of three operations of job 1. Specifically, the first number '1' denotes Machine 1 is chosen for machining the first operation of job 1.

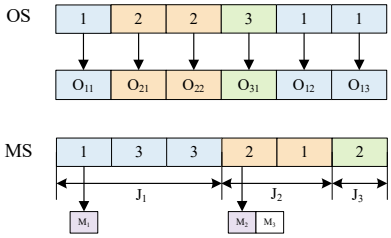


Fig. 3. An example of encoding schemes

4.4 Decoding scheme

In the decoding phase, based on the OS string and MS string obtained from the encoding phase, all operations are allocated to appropriate machines for processing with active decoding methods. Active decoding is used in this paper. In active decoding,

each operation is allocated to machines based on the OS and MS strings. For each operation, the active decoding methods involve the following detailed steps:

Step 1: For each operation $O_{i,j}$, determine its machine $k_{i,j}$ by MS string.

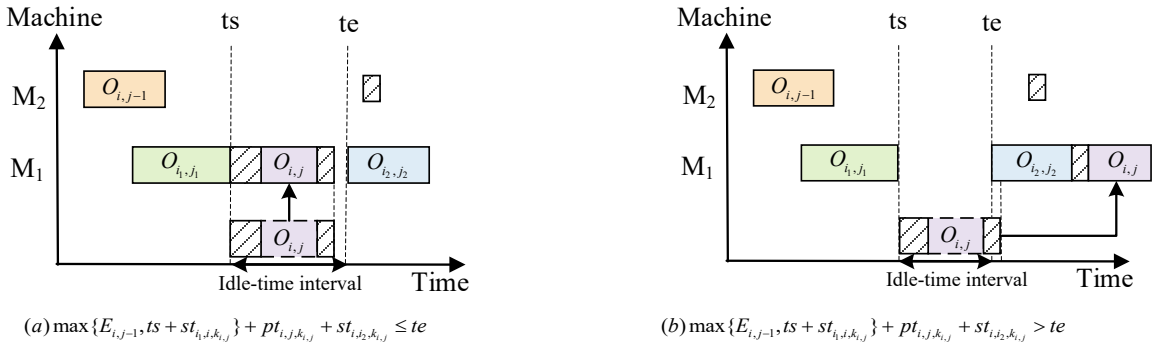
Step 2: Search for the idle-time intervals on machine $k_{i,j}$ in order, and get the idle-time interval $[ts,te]$. Check whether this idle-time interval has enough time to process operation $O_{i,j}$. If the idle time is enough, operation $O_{i,j}$ is inserted into the idle interval $[ts,te]$ and Eq. (5) is true. And the start time $B_{i,j}$ is obtained according to Eq. (6). Fig. 4(a) shows an example of active decoding. If the idle-time interval $[ts,te]$ is not enough, continue to check for the next idle-time interval.

$$\max\{E_{i,j-1}, ts + st_{i_1,i,k_{i,j}}\} + pt_{i,j,k_{i,j}} + st_{i,i_2,k_{i,j}} \le te \tag{5}$$

$$B_{i,j} = \max\{E_{i,j-1}, ts + st_{i_1,i,k_{i,j}}\} \tag{6}$$

where, $E_{i,j-1}$ is the finish time of operation $O_{i,j-1}$ when $O_{i,j}$ is not the first operation of job i. If $O_{i,j}$ is the initial operation of job i, then $E_{i,j-1}$ is 0. Jobs i_1 and i_2 are the jobs before and after the idle-time interval, respectively. If $O_{i,j}$ is the first operation processed on machine $k_{i,j}$, then $st_{i_1,i,k_{i,j}}$ is 0.

Step 3: If no suitable idle-time interval is found, operation $O_{i,j}$ is inserted after the last operation processed by machine $k_{i,j}$. Fig. 4(b) shows an example of this situation.



$(a)\ \max\{E_{i,j-1}, ts + st_{i_1,i,k_{i,j}}\} + pt_{i,j,k_{i,j}} + st_{i,i_2,k_{i,j}} \le te$ $(b)\ \max\{E_{i,j-1}, ts + st_{i_1,i,k_{i,j}}\} + pt_{i,j,k_{i,j}} + st_{i,i_2,k_{i,j}} > te$

Fig. 4. Active decoding of FJSP-SDST

4.5 Neighborhoods

For C-VNS, eight neighborhoods are specifically selected. Specifically, the eight neighborhood structures include four normal ones and four cooperative ones.

4.5.1 Normal neighborhoods

For the normal neighborhoods, four neighborhood structures are used. The first three neighborhood structures are Swap neighborhood, Insert neighborhood and Inversion neighborhood. These three neighborhood structures are applied to the OS string. And the fourth neighborhood structure, referred to as Reassign neighborhood, is applied to the MS string. Specifically, the details of these four neighborhood structures are given as follows (P_1 and P_2 are the two parents; O_1 and O_2 are the two offsprings):

For Swap neighborhood structure, it is shown in Fig. 5, and its steps are given as below:

Step 1: Randomly choose two different positions in the P_1 and P_2 respectively. Go to Step 2.

Step 2: Swap the values of these two positions to generate O_1 and O_2.

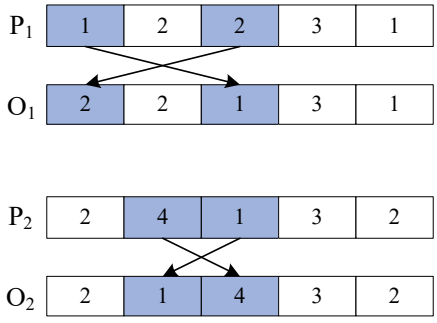


Fig. 5. Example of Swap neighborhood structure for OS

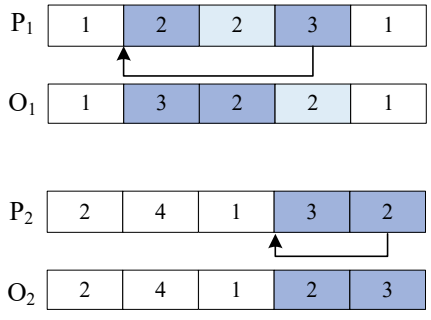


Fig. 6. Example of Insert neighborhood structure for OS

For Insert neighborhood structure, it is shown in Fig. 6, and its steps are given as below:

Step 1: Randomly select two different positions (the first position is smaller than the second position) in the P_1 and P_2 respectively. Go to Step 2.

Step 2: Shift the value in the second position to the front of the first position to generate O_1 and O_2.

The third neighborhood structure is the Inversion neighborhood structure, and it is shown in Fig. 7. Moreover, the steps of Inversion are outlined as below:

Step 1: Randomly choose two positions (the first position smaller less than the second position) in the P_1 and P_2 respectively. Go to step 2.

Step 2: Invert the values between these two positions to generate O_1 and O_2.

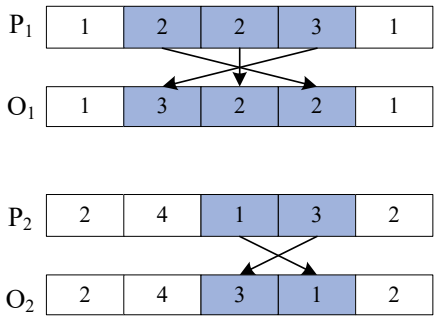


Fig. 7. Example of Inversion neighborhood structure for OS

An example of the Reassign neighborhood structure is illustrated in Fig. 8, and the steps are outlined as below:

Step 1: Randomly choose one position in the P_1 and P_2. Go to Step 2.
Step 2: Replace this position with another eligible machine to generate O_1 and O_2.

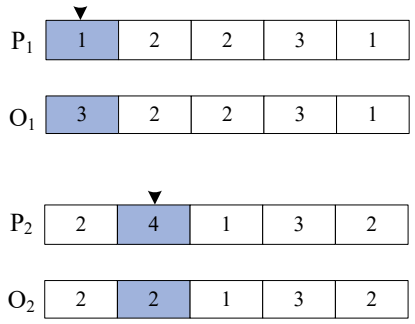


Fig. 8. Example of Reassign neighborhood structure for MS

4.5.2 Cooperative Neighborhoods

The normal neighborhood structures do not communicate the information of two solutions of two VNSs. Therefore, to let the solutions of two VNSs communicate with each other, four cooperative neighborhood structures, namely priority operation crossover (POX), job-based crossover (JBX), two-points crossover (TPX) and uniform crossover (UC) are used. Specifically, POX, JBX and TPX are used for exchanging information of OS strings, and UC is used for exchanging information of MS strings.

Fig. 9 gives an example of the POX neighborhood structure, and the steps are outlined below:

Step 1: Randomly separate the job into two parts, Jt_1 and Jt_2. Go to Step 2.
Step 2: Copy the values of P_1 belonging to Jt_1 to the corresponding positions of O_1 ; and copy the values of P_2 belonging to Jt_1 to the corresponding positions of O_2. Go to Step 3.
Step 3: Copy the values belong to Jt_2 in P_2 to the remaining positions in O_1, keeping the operations order unchanged; and copy the values belong to Jt_2 in P_1 to the remaining positions in O_2, keeping the operations order unchanged.

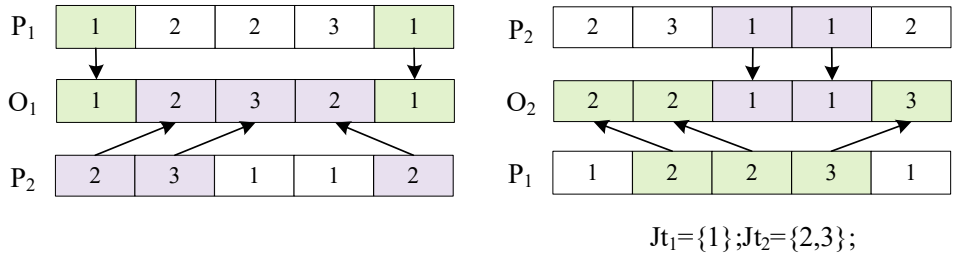


Fig. 9. Example of POX neighborhood structure for OS

For JBX neighborhood structure, Fig. 10 gives an example and its steps are outlined as below:

Step 1: Randomly separate the job into two parts, Jt_1 and Jt_2. Go to Step 2.
Step 2: Copy the values belong to Jt_1 in P_1 to the same position of O_1 ; and copy the values belong to Jt_2 in P_2 to the identical position of O_2 . Go to Step 3.
Step 3: Copy the values belong to Jt_2 in P_2 to the remaining positions in O_1, keeping the operations order unchanged; and copy the values belong to Jt_1 in P_1 to the remaining positions in O_2, keeping the operations order unchanged.

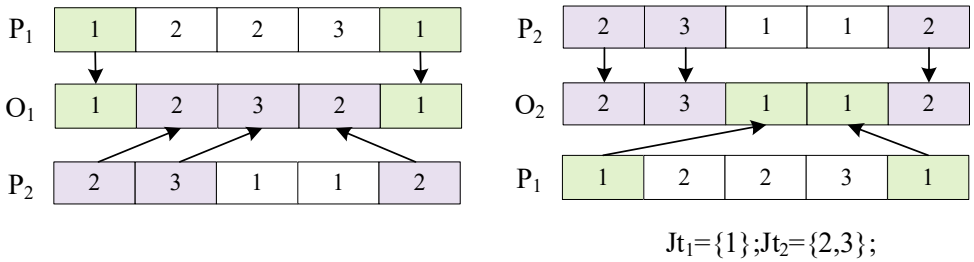


Fig. 10. Example of JBX neighborhood structure for OS

For TPX neighborhood structure, Fig. 11 shows an example and its steps are outlined as below:

Step 1: Randomly choose two positions r_1 and r_2 (r_1 is smaller than r_2), and split P_1 and P_2 into three sections. Go to Step 2.
Step 2: Copy the values before r_1 and after r_2 from P_1 to O_1 at the same position. Copy the values before r_1 and after r_2 from P_2 to O_2 at the same position. Go to Step 3.
Step 3: Assign values that belong to P_2 but don't exist in O_1 to the empty positions in O_1 sequentially, and assign values that exist in P_1 but don't exist in O_2 to the empty positions in O_2 sequentially.

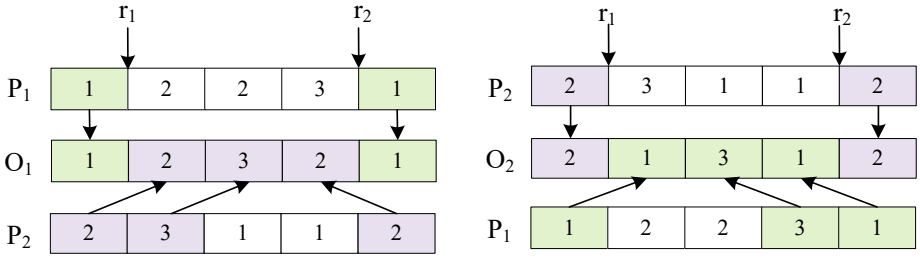


Fig. 11. Example of TPX neighborhood structure for OS

For UC neighborhood structure, Fig. 12 presents an example and its steps are outlined as below:

Step 1: Randomly generate a set of binary sequences, where the lengths of the binary sequences are equal to P_1 and P_2. Go to step 2.

Step 2: Copy the position of "1" from the binary sequence in P_1 and P_2 to the corresponding position in O_1 and O_2, respectively. Go to step 3.

Step 3: Copy the position of "0" from the binary sequence in P_2 to the corresponding position in O_1. Copy the position of "0" from the binary sequence in P_1 to the corresponding position in O_2.

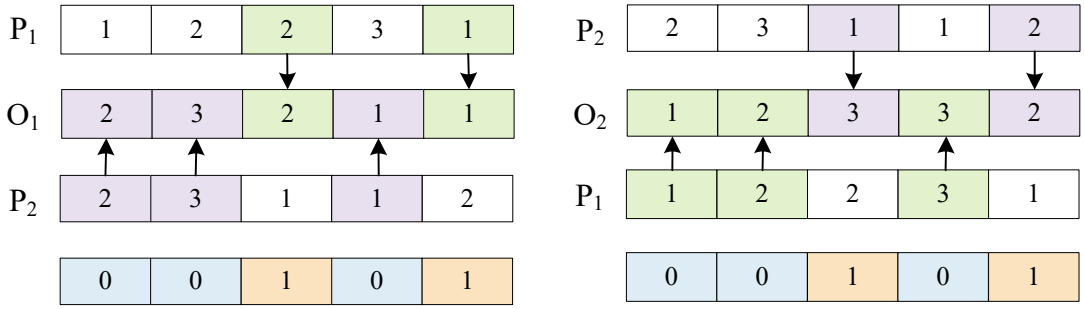


Fig. 12. Example of UC neighborhood structure for MS

4.6 Restart operation

When the C-VNS is not updated for *RN* iterations, replace the worst solution of the two VNSs by using the initialization method in Section 4.2.

4.7 Connecting C-VNS and CP

As described above, CP is used to further optimize the good solution obtained from C-VNS. The following constraints (5)-(8) are used to transforming the information of the solution obtained by C-VNS to CP model.

$$sol = newIloOplCPSolution() \tag{5}$$

$$cp.setStartingPoint(sol) \tag{6}$$

$$sol.setStart(Ops_{i,j}, start_{i,j}), \forall i \in I, \forall j \in J_i \tag{7}$$

$$sol.setPresent(mod_{i,j,MS_{i,j}}), \forall i \in I, \forall j \in J_i \tag{8}$$

where, constraint function (5) defines the initial solution sol. Constraint function (6) means that CP model starts from the initial sol. Constraint function (7) transforms the starting time of each operation to the corresponding interval decision variable, and $start_{i,j}$ is the starting time for operation $O_{i,j}$ of the initial sol. Constraint function (8) transforms the machine selection information to the CP model, and $MS_{i,j}$ is the machine selection for operation $O_{i,j}$ of the initial sol.

5. Experimental results

Through testing 20 instances of MFJS01-10 and MK01-10, the validity of MILP model, CP model and C-VNS-CP algorithm are proved. The setup times are generated according to processing times. According to the proportion of setup time and processing time, the test instances are divided into three groups: s/p=0.1, s/p=0.3 and s/p=0.5. All meta-heuristic algorithms are designed in C++. Both the MILP model and CP model are coded in the OPL language with using IBM CPLEX Studio IDE 12.7.1. The maximum CPU time (timelimit) is set to 2*Nop* seconds for all algorithms. For C-VNS-CP algorithm, the runtime of both C-VNS algorithm and CP model are set to *Nop* seconds. Specifically, *Nop* is the total number of all operations. All instances in the algorithms are repeated 10 times. In the following tables, the best results of all the compared algorithms in the table are highlighted in bold.

5.1 Effectiveness of the CP model

The comparison results of the proposed CP model and the existing MILP model (Shen and Dauzère-Pérès et al., 2018) are given in Table 2. In Table 2, “NB”, “NC” and “NCT” denote the numbers of binary decision variables, continuous decision variables and constraints respectively. “NV” is the number of interval decision variables of the CP model. “Gap” represents the optimal gap. If the value of “Gap” is equal to 0, it means that the optimal solution is found. “Cmax” indicates a solution found within the time limit. “Time” is the CPU time. Specifically, when an optimal solution is obtained, the value of “Time” is no more than time limit, otherwise, it is equal to the time limit. According to Table 2, as the size of the instance increases, NB, NC and NTC of the MILP model increase greatly. The MILP model can solve eighth instances, namely MFJS01-07 and MK01 to optimality within the time limit. The sizes of MK06, MK09 and MK10 are relatively large, and no feasible solution is found by MILP model within the time limit. The proposed CP model can obtain the same optimal solution as the MILP model for MFJS01-07 and MK01. For the other instances, the CP model can obtain better solutions than the MILP model. Moreover, CP model outperforms the MILP model in speed for all the instances where their optimal solutions are obtained (MFJS01-07 and MK01). In conclusion, the CP model demonstrates superior performance compared to the existing MILP model.

Table 2
The results of MILP model and CP model

Inst.	MILP model						CP model			
	NB	NC	NCT	Cmax	Time	Gap	NV	NCT	Cmax	Time
MFJS01	103	16	185	**497***	0.1	0	54	34	**497***	0.1
MFJS02	128	16	223	**470***	0.3	0	61	35	**470***	0.1
MFJS03	190	19	338	**497***	0.8	0	73	40	**497***	0.1
MFJS04	250	22	451	**598***	2.6	0	84	45	**598***	1.0
MFJS05	243	22	439	**563***	1.0	0	83	45	**563***	0.1
MFJS06	307	25	562	**686***	24.3	0	93	50	**686***	1.6
MFJS07	475	33	890	**967***	17.4	0	117	66	**967***	4.4
MFJS08	519	37	974	975	72.0	0.13	130	74	**963***	10.6
MFJS09	751	45	1428	1241	88.0	0.33	155	88	**1181**	88.0
MFJS10	899	49	1718	1403	96.0	0.31	168	95	**1353**	96.0
MK01	1279	56	2493	**50***	27.3	0	176	109	**50***	4.6
MK02	4495	59	8688	37	116.0	0.16	302	115	**33**	116.0
MK03	12512	151	24572	290	300.0	0.44	609	296	**248**	300.0
MK04	2225	91	4376	77	180.0	0.29	270	176	**75**	180.0
MK05	4077	107	8110	233	212.0	0.67	291	204	**210**	212.0
MK06	12865	151	25200	-	300.0	-	650	303	**72**	300.0
MK07	7906	101	15546	208	200.0	0.65	388	188	**171**	200.0
MK08	6514	226	13059	660	450.0	0.70	557	443	**622**	450.0
MK09	19444	241	38396	-	480.0	-	856	473	**362**	480.0
MK10	25870	241	51028	-	480.0	-	967	474	**252**	480.0

5.2 Parameter calibration of C-VNS-CP

To determine the parameters, Taguchi method of design of experiment (DOE) is designed for the MK07, and mean value is used as the response value. For the proposed algorithm, we must determine two parameters, namely *SN* and *RN*. Where *SN* represents the search number of solutions in a neighborhood operation and *RN* denotes the number of generations, the solution employs a restart strategy when the current generation exceeds *RN*. For the *SN* parameter, four levels of [8,16,32,64] are selected. For the *RN* parameter, four levels of [100,200,300,400] are selected. 16 combinations are obtained, and C-VNS-CP repeats 10 times for each combination. The comparison results of different combinations of parameters are shown in Table 3. Fig. 13 shows the trends of different parameters. As shown in Fig. 13, the best combination is: S*N*=8, *RN*=400.

Table 3
Results of DOE test

Test	*SN*	*RN*	*Mean*
1	8	100	170.1
2	8	200	167.7
3	8	300	169.8
4	8	400	167.5
5	16	100	168.3
6	16	200	169.0
7	16	300	169.5
8	16	400	170.7
9	32	100	169.3
10	32	200	170.0
11	32	300	168.8
12	32	400	168.8
13	64	100	169.8
14	64	200	170.6
15	64	300	169.8
16	64	400	169.6

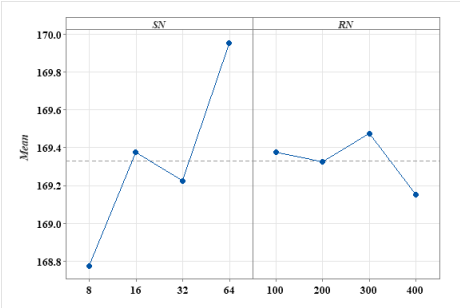


Fig. 13. Two parameters of main effect plots

5.3 Effectiveness of the CP, restart operation and cooperative neighborhoods

To demonstrate the efficacy of the introduced algorithms, experiments on VNS, C-VNS, C-VNS-R and C-VNS-CP are conducted. Specifically, VNS is the classical algorithm and only with normal neighborhoods. C-VNS is with two VNSs, all

neighborhoods and does not consider the restart operation. C-VNS-R is with two VNSs, all neighborhoods and considers the restart operation. GA-VNS-CP combines C-VNS-R and CP search. The comparison results of VNS, C-VNS, C-VNS-R and C-VNS-CP with s/p=0.1, s/p=0.3 and s/p=0.5 are displayed in Tables 4-6 respectively. In Tables 4-6, "Best" refers to the best solution achieved over 10 repetitions, “AV” indicates the average value of solutions obtained from the 10 repetitions. The values presented in bold represent the best among all algorithms. From Table 4, it can be obtained that for the “Best” metric, comparing C-VNS and VNS, the results for 5 instances are the same. Moreover, the former can obtain better and worse results than the latter for 12 and 3 instances respectively. C-VNS-R can obtain the same results as C-VNS for 3 instances, and the former can obtain better results than the latter for 17 instances. C-VNS-CP obtains 9 better and 11 equal solutions than/to C-VNS-R. For the “AV” metric, C-VNS obtains the same results as VNS for 2 instances, and the former can obtain better and worse results than the latter for 14 and 4 instances respectively. C-VNS-R obtains better results than C-VNS for 18 instances. For the other two instances, C-VNS-R and C-VNS obtains the same results. Comparing C-VNS-CP and C-VNS-R, the former obtains better and equal instances than/to the latter for 16 and 4 instances respectively.

Table 4
Comparison result of instances with s/p=0.1

Inst.	VNS		C-VNS		C-VNS-R		C-VNS-CP	
	Best	AV	Best	AV	Best	AV	Best	AV
MFJS01	**497**	498.0	**497**	498.0	**497**	**497.0**	**497**	**497.0**
MFJS02	**470**	484.1	482	484.5	**470**	**470.0**	**470**	**470.0**
MFJS03	504	516.0	518	525.4	**497**	498.4	**497**	**497.0**
MFJS04	611	625.6	611	631.7	**598**	608.4	**598**	**598.0**
MFJS05	583	599.8	575	591.0	**563**	565.0	**563**	**563.0**
MFJS06	711	738.3	702	736.1	**686**	686.2	**686**	**686.0**
MFJS07	1024	1064.6	988	1040.5	968	979.4	**967**	**967.1**
MFJS08	1052	1084.5	990	1044.5	975	993.5	**963**	**968.3**
MFJS09	1275	1327.6	1264	1312.9	1216	1238.1	**1179**	**1194**
MFJS10	1432	1502.7	1453	1474.3	1387	1412.8	**1351**	**1364.6**
MK01	52	52.0	52	52.1	**50**	50.2	**50**	**50**
MK02	36	36.9	35	36.3	34	34.6	**33**	**33.2**
MK03	**248**	**248.0**	**248**	**248.0**	**248**	**248.0**	**248**	**248.0**
MK04	79	83.9	78	82.2	76	76.9	**75**	**75.8**
MK05	216	218.3	215	218	213	214.0	**208**	**208.5**
MK06	91	93.1	84	87.3	80	82.5	**73**	**74.6**
MK07	177	182.2	173	177.5	170	172.6	**166**	**168.1**
MK08	**622**	622.1	**622**	**622.0**	**622**	**622.0**	**622**	**622.0**
MK09	372	379.1	366	371.6	362	364.2	**361**	**361.8**
MK10	279	282.6	269	276.1	261	267.8	**243**	**248.6**

From Table 5, it can be obtained that for the “Best” metric, C-VNS obtains equal, better and worse results to/than VNS for 5,13 and 2 instances respectively. C-VNS-R obtains equal and better results to/than C-VNS for 5 and 15 instances respectively. C-VNS-CP obtains equal and better results to/than C-VNS-R for 9 and 11 instances respectively. For the “AV” metric, C-VNS obtains equal, better and worse results to/than VNS for 1,18 and 1 instances respectively. C-VNS-R obtains equal and better results to/than C-VNS for 2 and 18 instances respectively. C-VNS-CP obtains equal and better results to/than C-VNS-R for 4 and 16 instances respectively.

Table 5
Comparison result of instances with s/p=0.3

Inst.	VNS		C-VNS		C-VNS-R		C-VNS-CP	
	Best	AV	Best	AV	Best	AV	Best	AV
MFJS01	**593**	**593.3**	**593**	**593.0**	**593**	**593.0**	**593**	**593**
MFJS02	537	544.4	537	543.2	**516**	**516.0**	**516**	**516**
MFJS03	**593**	609.9	**593**	606.0	**593**	**593.0**	**593**	**593**
MFJS04	698	740.6	694	741.0	**681**	684.5	**681**	**681**
MFJS05	678	695.6	671	690.5	**652**	664.3	**652**	**652**
MFJS06	807	860.5	821	848.5	807	807.0	**793**	**793**
MFJS07	1218	1264.3	1190	1251.0	1167	1179.0	**1137**	**1141.2**
MFJS08	1232	1289.1	1179	1248.9	1150	1178.0	**1124**	**1133.7**
MFJS09	1526	1585.0	1517	1563.2	1466	1503.7	**1412**	**1437.8**
MFJS10	1750	1832.1	1750	1792.9	1666	1685.3	**1615**	**1639**
MK01	62	62.3	**60**	61.8	**60**	60.8	**60**	**60**
MK02	44	46.0	43	45.3	41	41.8	**40**	**40**
MK03	**325**	**325.0**	**325**	**325.0**	**325**	**325.0**	**325**	**325.0**
MK04	99	104.4	98	101.9	95	96.7	**94**	**94.4**
MK05	266	271.5	262	266.4	260	260.9	**251**	**254.2**
MK06	104	108.5	101	102.9	92	95.1	**85**	**87.8**
MK07	217	226.6	218	224.2	210	214.0	**208**	**210.2**
MK08	796	798.6	**794**	794.6	**794**	794.1	**794**	**794**
MK09	470	480.0	463	469.7	**460**	463.5	**460**	**460**
MK10	356	364.0	343	352.1	328	334.6	**308**	**313.8**

From Table 6, it can be obtained that for the “Best” metric, C-VNS obtains equal, better and worse results to/than VNS for

5,14 and 1 instances respectively. C-VNS-R obtains equal, and better results to/than C-VNS for 4 and 16 instances respectively. C-VNS-CP obtains equal and better results to/than C-VNS-R for 7 and 13 instances respectively. For the "AV" metric, C-VNS obtains equal and better results to/than VNS for 1 and 19 instances respectively. C-VNS-R obtains equal and better results to/than C-VNS for 1 and 19 instances respectively. C-VNS-CP obtains equal and better results to/than C-VNS-R for 1 and 19 instances respectively.

Table 6
Comparison result of instances with s/p=0.5

Inst.	VNS		C-VNS		C-VNS-R		C-VNS-CP	
	Best	AV	Best	AV	Best	AV	Best	AV
MFJS01	677	680.9	677	680.7	**670**	674.2	**670**	**670**
MFJS02	578	602.1	578	597.9	578	579.2	**570**	**570**
MFJS03	690	704.3	690	695.8	690	690.0	**687**	**687**
MFJS04	815	862.6	783	844.5	**755**	760.6	**755**	**755**
MFJS05	791	818.0	748	789.0	**739**	742.4	**739**	**739**
MFJS06	893	965.6	883	943.9	**859**	861.4	**859**	**859**
MFJS07	1417	1474.3	1358	1416.5	1315	1335.0	**1259**	**1281.4**
MFJS08	1409	1470.5	1401	1445.4	1327	1340.6	**1282**	**1282**
MFJS09	1819	1899.7	1750	1837.0	**1663**	1732.7	**1633**	**1650.3**
MFJS10	1993	2133.3	2020	2109.0	1912	1951.8	**1866**	**1876.6**
MK01	72	72.2	71	71.9	70	70.3	**69**	**69**
MK02	48	52.8	48	50.1	46	47.2	**43**	**44.4**
MK03	**391**	**391.0**	**391**	**391.0**	**391**	**391.0**	**391**	**391**
MK04	122	123.4	113	117.4	**108**	109.3	**108**	**108.1**
MK05	313	322.3	309	314.0	304	308.4	**290**	**292.5**
MK06	120	122.5	111	116.0	104	106.4	**96**	**100.2**
MK07	262	271.2	255	265.7	246	252.8	**243**	**245.4**
MK08	964	964.0	**955**	956.5	**955**	955.6	**955**	**955**
MK09	564	574.4	547	556.9	545	547.3	**544**	**544.3**
MK10	417	432.0	407	418.7	398	401.0	**362**	**367.6**

Table 7 shows summarized results of Tables 4-6. Moreover, in Table 8, for the "AV" values reported in Tables 4-6, a paired-t test at 95% confidence level is conducted. Specifically, the p values obtained for the C-VNS vs. VNS, C-VNS-R vs. C-VNS, and C-VNS-CP vs. C-VNS-R are all significantly less than 0.05. Therefore, C-VNS is statistically better than VNS, and this shows the effectiveness of the cooperative operation (cooperative neighborhoods). C-VNS-R is statistically better than C-VNS, and this shows the effectiveness of the restart operation. C-VNS-CP is statistically better than C-VNS-R, and this shows the advantages of combining the C-VNS-R and CP search.

Table 7
Summarized results of Tables 4-6

Comparisons	s/p	Indices	better	worse	equal
C-VNS vs. VNS	0.1	Best	12	3	5
C-VNS-R vs. C-VNS	0.1	Best	17	0	3
C-VNS-CP vs. C-VNS-R	0.1	Best	11	0	9
C-VNS vs. VNS	0.1	AV	14	4	2
C-VNS-R vs. C-VNS	0.1	AV	18	0	2
C-VNS-CP vs. C-VNS-R	0.1	AV	16	0	4
C-VNS vs. VNS	0.3	Best	13	2	5
C-VNS-R vs. C-VNS	0.3	Best	15	0	5
C-VNS-CP vs. C-VNS-R	0.3	Best	11	0	9
C-VNS vs. VNS	0.3	AV	18	1	1
C-VNS-R vs. C-VNS	0.3	AV	18	0	2
C-VNS-CP vs. C-VNS-R	0.3	AV	16	0	4
C-VNS vs. VNS	0.5	Best	14	1	5
C-VNS-R vs. C-VNS	0.5	Best	16	0	4
C-VNS-CP vs. C-VNS-R	0.5	Best	13	0	7
C-VNS-R vs. C-VNS	0.5	AV	19	0	1
C-VNS-CP vs. C-VNS-R	0.5	AV	19	0	1
C-VNS vs. VNS	0.5	AV	19	0	1

Table 8
Paired-t test for the AV values

Comparisons	s/p	p-value	Remark
C-VNS vs. VNS	0.1	0.001	<0.05
C-VNS-R vs. C-VNS	0.1	0.002	<0.05
C-VNS-CP vs. C-VNS-R	0.1	0.006	<0.05
C-VNS vs. VNS	0.3	0.003	<0.05
C-VNS-R vs. C-VNS	0.3	0.001	<0.05
C-VNS-CP vs. C-VNS-R	0.3	0.005	<0.05
C-VNS vs. VNS	0.5	0.001	<0.05
C-VNS-R vs. C-VNS	0.5	0.002	<0.05
C-VNS-CP vs. C-VNS-R	0.5	0.006	<0.05

5.4 Effectiveness of C-VNS-CP

To demonstrate the efficacy of the C-VNS-CP algorithm, it is compared against GA and CP models. Tables 9-11 gives the comparisons of instances with s/p=0.1, s/p=0.3, and s/p=0.5 respectively. From Table 9, it can be obtained that, for the "Best" metric, CP, GA, and C-VNS-CP can obtain 14, 4, and 19 best solutions respectively. For the "AV" metric, C-VNS-CP performs equally to GA for 3 instances, and the former performs better and worse than the latter for 16 and 1 instances respectively. From Table 10, it can be obtained that for the "Best" metric, CP, GA, and C-VNS-CP can obtain 16, 7, and 19 best solutions respectively. For the "AV" metric, C-VNS-CP performs equally to GA on 3 instances and better than GA for 17 instances. In Table 11, for the "Best" metric, CP, GA, and C-VNS-CP can obtain 14, 3, and 20 best solutions respectively. For the "AV" metric, C-VNS-CP performs equally to GA for 1 instance and better than GA for 19 instances. Table 12 shows summarized results of Tables 9-11. Furthermore, Table 13 shows the results of a paired-t test at 95% confidence level for the "AV" values reported in Tables 9-11. Specifically, the p values obtained for the C-VNS-CP vs. GA with s/p=0.1, s/p=0.3, and s/p=0.5 are 0.003, 0.006 and 0.005 respectively, and they are all less than 0.05. All of these show that C-VNS-CP outperforms GA.

Table 9
Comparison result of instances with s/p=0.1

Inst.	CP	GA		C-VNS-CP	
	Best	Best	AV	Best	AV
MFJS01	**497**	**497**	**497.0**	**497**	**497.0**
MFJS02	**470**	**470**	481.5	**470**	**470.0**
MFJS03	**497**	512	525.6	**497**	**497.0**
MFJS04	**598**	611	613.6	**598**	**598.0**
MFJS05	**563**	570	583.4	**563**	**563.0**
MFJS06	**686**	687	720.1	**686**	**686.0**
MFJS07	**967**	969	1000.8	**967**	**967.1**
MFJS08	**963**	993	1031.4	**963**	**968.3**
MFJS09	1181	1242	1260.3	**1179***	**1194**
MFJS10	1353	1425	1435.8	**1351***	**1364.6**
MK01	**50**	52	52.0	**50**	**50**
MK02	**33**	35	35.2	**33**	**33.2**
MK03	**248**	**248**	**248.0**	**248**	**248.0**
MK04	**75**	77	78.2	**75**	**75.8**
MK05	210	212	212.8	**208***	**208.5**
MK06	**72**	77	79.6	73	**74.6**
MK07	171	171	172.3	**166***	**168.1**
MK08	**622**	**622**	**622.0**	**622**	**622.0**
MK09	362	362	363.3	**361***	**361.8**
MK10	252	244	**246.8**	**243***	**248.6**

Table 10
Comparison result of instances with s/p=0.3

Inst.	CP	GA		C-VNS-CP	
	Best	Best	AV	Best	AV
MFJS01	**593**	**593**	**593.0**	**593**	**593**
MFJS02	**516**	**516**	532.0	**516**	**516**
MFJS03	**593**	**593**	604.7	**593**	**593**
MFJS04	**681**	689	705.9	**681**	**681**
MFJS05	**652**	661	680.2	**652**	**652**
MFJS06	**793**	807	811.3	**793**	**793**
MFJS07	**1137**	1163	1199.7	**1137**	**1141.2**
MFJS08	**1124**	1159	1231.8	**1124**	**1133.7**
MFJS09	1423	1516	1529.4	**1412***	**1437.8**
MFJS10	1651	1689	1734.3	**1615***	**1639**
MK01	**60**	62	62.0	**60**	**60**
MK02	**40**	42	44.1	**40**	**40**
MK03	**325**	**325**	**325.0**	**325**	**325.0**
MK04	**94**	98	98.8	**94**	**94.4**
MK05	**250**	257	260.1	251	**254.2**
MK06	86	90	92.1	**85***	**87.8**
MK07	**208**	210	217.2	**208**	**210.2**
MK08	**794**	**794**	**794.0**	**794**	**794**
MK09	**460**	**460**	460.4	**460**	**460**
MK10	334	**308**	315.6	**308**	**313.8**

Table 11
Comparison result of instances with s/p=0.5

Inst.	CP	GA		C-VNS-CP	
	Best	Best	AV	Best	AV
MFJS01	**670**	677	680.9	**670**	**670**
MFJS02	**570**	**570**	593.0	**570**	**570**
MFJS03	**687**	690	690.0	**687**	**687**
MFJS04	**755**	781	808.5	**755**	**755**
MFJS05	**739**	755	777.4	**739**	**739**
MFJS06	**859**	869	885.2	**859**	**859**
MFJS07	**1259**	1363	1406.7	**1259**	**1281.4**
MFJS08	**1282**	1348	1411.5	**1282**	**1282**
MFJS09	1663	1720	1789.0	**1633***	**1650.3**
MFJS10	1871	1981	2049.0	**1866***	**1876.6**
MK01	**69**	72	72.0	**69**	**69**
MK02	44	48	50.1	**43***	**44.4**
MK03	**391**	**391**	**391.0**	**391**	**391**
MK04	**108**	110	114.6	**108**	**108.1**
MK05	294	306	310.0	**290***	**292.5**
MK06	**96**	99	104.3	**96**	**100.2**
MK07	246	253	255.2	**243***	**245.4**
MK08	**955**	**955**	956.0	**955**	**955**
MK09	**544**	545	545.0	**544**	**544.3**
MK10	374	375	382.2	**362***	**367.6**

Table 12

Summarized results of Tables 9-11

Comparisons	s/p	Indices	better	worse	equal
C-VNS-CP vs. GA	0.1	Best	16	0	4
C-VNS-CP vs. GA	0.3	Best	13	0	7
C-VNS-CP vs. GA	0.5	Best	17	0	3
C-VNS-CP vs. CP	0.1	Best	6	1	13
C-VNS-CP vs. CP	0.3	Best	4	1	15
C-VNS-CP vs. CP	0.5	Best	6	0	14
C-VNS-CP vs. GA	0.1	AV	16	1	3
C-VNS-CP vs. GA	0.3	AV	17	0	3
C-VNS-CP vs. GA	0.5	AV	19	0	1

Table 13
Paired-t test for the AV values of C-VNS-CP and GA

s/p	p-value	Remark
0.1	0.003	<0.05
0.3	0.006	<0.05
0.5	0.005	<0.05

6. Conclusions and future research

In this paper, FJSP-SDST with minimizing the makespan is studied. A CP model is formulated for obtaining optimal solutions, and it is verified by using CPLEX. The results show that the CP model is more efficient than the existing MILP model. Due to the NP-hardness of FJSP-SDST, C-VNS-CP of combining C-VNS and CP is designed. Specifically, C-VNS-CP first obtains a very good solution by using C-VNS, and then sets the solution as the initial solution of the CP model. Experimental results show that C-VNS-CP outperforms GA, C-VNS and CP model, and C-VNS outperforms VNS.

In future research, energy-efficient FJSP-SDST with minimizing the energy consumption and makespan simultaneously will be studied (Meng and Zhang et al., 2023). Moreover, deep reinforcement learning will be studied for solving FJSP-SDST with different constraints, such as predictive maintenance and limited number of automatic guided vehicles (Meng and Cheng et al., 2023; Han and Cheng et al., 2024).

Acknowledgements

This research is supported by the Funds for the National Natural Science Foundation of China [grant numbers 52205529 and 62303204], the Natural Science Foundation of Shandong Province [grant numbers ZR2021QE195 and ZR2021QF036], the Youth Innovation Team Program of Shandong Higher Education Institution (2023KJ206), the Guangyue Youth Scholar Innovation Talent Program support received from Liaocheng University [LCUGYTD2022-03], the Foundation of Young Talent of Lifting engineering for Science and Technology in Shandong, China (No. SDAST2024QTA074).

References

Alicastro, M., Ferone, D., Festa, P., Fugaro, S., & Pastore, T. (2021). A reinforcement learning iterated local search for makespan minimization in additive manufacturing machine scheduling problems. *Computers & Operations Research*, *131*, 105272.

Azzouz, A., Ennigrou, M., & Said, L. B. (2017). A self-adaptive hybrid algorithm for solving flexible job-shop problem with sequence dependent setup time. *Procedia computer science*, *112*, 457-466.

Bagheri, A., & Zandieh, M. (2011). Bi-criteria flexible job-shop scheduling with sequence-dependent setup times—Variable neighborhood search approach. *Journal of Manufacturing Systems*, *30*(1), 8-15.

de Abreu, L. R., Araújo, K. A. G., de Athayde Prata, B., Nagano, M. S., & Moccellin, J. V. (2022). A new variable neighbourhood search with a constraint programming search strategy for the open shop scheduling problem with operation repetitions. *Engineering Optimization*, *54*(9), 1563-1582.

Ham, A. (2020). Transfer-robot task scheduling in flexible job shop. *Journal of Intelligent Manufacturing*, *31*(7), 1783-1793.

Ham, A. (2021). Transfer-robot task scheduling in job shop. *International Journal of Production Research*, *59*(3), 813-823.

Ham, A. M., & Cakici, E. (2016). Flexible job shop scheduling problem with parallel batch processing machines: MIP and CP approaches. *Computers & Industrial Engineering*, *102*, 160-165.

Ham, A., Park, M. J., & Kim, K. M. (2021). Energy-Aware Flexible Job Shop Scheduling Using Mixed Integer Programming and Constraint Programming. *Mathematical Problems in Engineering*, *2021*(1), 8035806.

Han, X., Cheng, W., Meng, L., Zhang, B., Gao, K., Zhang, C., & Duan, P. (2024). A dual population collaborative genetic algorithm for solving flexible job shop scheduling problem with AGV. *Swarm and Evolutionary Computation*, *86*, 101538.

Karimi, H., Rahmati, S. H. A., & Zandieh, M. (2012). An efficient knowledge-based algorithm for the flexible job shop scheduling problem. *Knowledge-Based Systems*, *36*, 236-244.

Li, J. Q., Deng, J. W., Li, C. Y., Han, Y. Y., Tian, J., Zhang, B., & Wang, C. G. (2020). An improved Jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times. *Knowledge-Based Systems*, *200*, 106032.

Li, M., & Lei, D. (2021). An imperialist competitive algorithm with feedback for energy-efficient flexible job shop scheduling

with transportation and sequence-dependent setup times. *Engineering Applications of Artificial Intelligence*, *103*, 104307.
Meng, L., Zhang, B., Gao, K., & Duan, P. (2022). An MILP model for energy-conscious flexible job shop problem with transportation and sequence-dependent setup times. *Sustainability*, *15*(1), 776.
Meng, L., Zhang, C., Zhang, B., & Ren, Y. (2019). Mathematical modeling and optimization of energy-conscious flexible job shop scheduling problem with worker flexibility. *IEEE Access*, *7*, 68043-68059.
Meng, L., Zhang, C., Ren, Y., Zhang, B., & Lv, C. (2020). Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem. *Computers & industrial engineering*, *142*, 106347.
Meng, L., Zhang, C., Zhang, B., Gao, K., Ren, Y., & Sang, H. (2023). MILP modeling and optimization of multi-objective flexible job shop scheduling problem with controllable processing times. *Swarm and Evolutionary Computation*, *82*, 101374.
Meng, L., Gao, K., Ren, Y., Zhang, B., Sang, H., & Chaoyong, Z. (2022). Novel MILP and CP models for distributed hybrid flowshop scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation*, *71*, 101058.
Meng, L., Duan, P., Gao, K., Zhang, B., Zou, W., Han, Y., & Zhang, C. (2024). MIP modeling of energy-conscious FJSP and its extended problems: From simplicity to complexity. *Expert Systems with Applications*, *241*, 122594.
Meng, L., Cheng, W., Zhang, B., Zou, W., Fang, W., & Duan, P. (2023). An improved genetic algorithm for solving the multi-AGV flexible job shop scheduling problem. *Sensors*, *23*(8), 3815.
Meng, L., Cheng, W., Zhang, B., Zou, W., & Duan, P. (2024). A novel hybrid algorithm of genetic algorithm, variable neighborhood search and constraint programming for distributed flexible job shop scheduling problem. *International Journal of Industrial Engineering Computations*, *15*(3), 813-832.
Novas, J. M. (2019). Production scheduling and lot streaming at flexible job-shops environments using constraint programming. *Computers & Industrial Engineering*, *136*, 252-264.
Saidi-Mehrabad, M., & Fattahi, P. (2007). Flexible job shop scheduling with tabu search algorithms. *The international journal of Advanced Manufacturing technology*, *32*, 563-570.
Shen, L., Dauzère-Pérès, S., & Neufeld, J. S. (2018). Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European journal of operational research*, *265*(2), 503-516.
Sun, J., Zhang, G., Lu, J., & Zhang, W. (2021). A hybrid many-objective evolutionary algorithm for flexible job-shop scheduling problem with transportation and setup times. *Computers & operations research*, *132*, 105263.
Yazdani, M., Amiri, M., & Zandieh, M. (2010). Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications*, *37*(1), 678-687.
Zeballos, L. J., Quiroga, O. D., & Henning, G. P. (2010). A constraint programming model for the scheduling of flexible manufacturing systems with machine and tool limitations. *Engineering Applications of Artificial Intelligence*, *23*(2), 229-248.
Zhang, G., Yan, S., Song, X., Zhang, D., & Guo, S. (2024). Evolutionary algorithm incorporating reinforcement learning for energy-conscious flexible job-shop scheduling problem with transportation and setup times. *Engineering Applications of Artificial Intelligence*, *133*, 107974.
Zhang, G., Hu, Y., Sun, J., & Zhang, W. (2020). An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints. *Swarm and evolutionary computation*, *54*, 100664.
Zhang, M., Tan, Y., Zhu, J., Chen, Y., & Chen, Z. (2020). A competitive and cooperative Migrating Birds Optimization algorithm for vary-sized batch splitting scheduling problem of flexible Job-Shop with setup time. *Simulation Modelling Practice and Theory*, *100*, 102065.