Li, Xiaoping; Chen, Long; Xu, Haiyan; Gupta, Jatinder N.

Article

Trajectory Scheduling Methods for minimizing total tardiness in a flowshop

Operations Research Perspectives

Provided in Cooperation with:
Elsevier

Suggested Citation: Li, Xiaoping; Chen, Long; Xu, Haiyan; Gupta, Jatinder N. (2015) : Trajectory Scheduling Methods for minimizing total tardiness in a flowshop, Operations Research Perspectives, ISSN 2214-7160, Elsevier, Amsterdam, Vol. 2, pp. 13-23, https://doi.org/10.1016/j.orp.2014.12.001

This Version is available at:
https://hdl.handle.net/10419/178246

Operations Research Perspectives 2 (2015) 13–23

Contents lists available at ScienceDirect

Operations Research Perspectives

journal homepage: www.elsevier.com/locate/orp

Trajectory Scheduling Methods for minimizing total tardiness in a flowshop

Xiaoping Li [a,b,*], Long Chen [a,b], Haiyan Xu [a,b], Jatinder N.D. Gupta [c]

[a] School of Computer Science and Engineering, Southeast University, Nanjing 211189, China

[b] Key Laboratory of Computer Network and Information Integration, Ministry of Education, Nanjing, 211189, China

[c] College of Business Administration, University of Alabama in Huntsville, Huntsville, Al, USA

ARTICLE INFO

Article history:
Received 13 May 2014
Received in revised form
29 December 2014
Accepted 29 December 2014
Available online 13 January 2015

Keywords:
Scheduling
Heuristic
Permutation flow shop
Total tardiness

ABSTRACT

In this paper, Trajectory Scheduling Methods (TSMs) are proposed for the permutation flowshop scheduling problem with total tardiness minimization criterion. TSMs belong to an iterative local search framework, in which local search is performed on an initial solution, a perturbation operator is deployed to improve diversification, and a restart point mechanism is used to select the new start point of another cycle. In terms of the insertion and swap neighborhood structures, six composite heuristics are introduced, which exploit the search space with a strong intensification effect. Based on purely insertion-based or swap-based perturbation structures, three compound perturbation structures are developed that construct a candidate restart point set rather than just a single restart point. The distance between the current best solution and each start point of the set is defined, according to which the diversification effect of TSMs can be boosted by choosing the most appropriate restart point for the next iteration. A total of 18 trajectory scheduling methods are constructed by different combinations of composite heuristics. Both the best and worst combinations are compared with three best existing sequential meta-heuristics for the considered problem on 540 benchmark instances. Experimental results show that the proposed heuristics significantly outperform the three best existing algorithms within the same computation time.

1. Introduction

The permutation flow shop scheduling problem (PFSP) is important and prevalent in modern manufacturing systems (for example, the flexible manufacturing environment) and in traditional industry settings (such as chemical, food, and metal processing). Not completing a job by its due date would lead to: (1) incurring tardiness costs which depend on the penalty clauses in the contract if there are any; (2) loss of goodwill which results in an increased probability of losing the customer for some or all future jobs; and (3) a damaged reputation which would turn other customers away [1]. Therefore, minimizing total tardiness which is closely related to the due dates agreed by all partners is of great importance in manufacturing systems. In this paper, the PFSP to minimize total tardiness considered, which is known to be NP-hard in the strong sense [2] and can be denoted as $F|prmu|\sum T_j$ [3].

For decades, many exact methods, heuristics, and meta-heuristics have been proposed for the considered problem [4]. Exact methods are effective only for small size problems. Branch & bound procedures [5–9] are exact methods for few jobs (usually less than 20 jobs) being scheduled on two machines. However, exact methods are seldom efficient in practical environments because usually there are more than 20 jobs to be scheduled on more than three machines. Therefore, heuristics and meta-heuristics have been investigated. Generally, there are two types of heuristics: constructive heuristics and composite ones. The constructive heuristic NEH_{EDD} [10] has been the most widely used, which was adapted from NEH [11] using the EDD (Earliest Due Date) rule to produce the seed. NEH_{EDD} is always adopted by composite heuristics or meta-heuristics to generate initial solutions. For example, NEH_{EDD} is utilized to generate initial solutions of the typical composite heuristics [12]. Meta-heuristics are always adopted for combinatorial optimization problems, which provide high level strategies for exploring search spaces using different methods [13]. They generally obtain better solutions than simple constructive heuristics but require significantly more computation time. Meta-heuristics can be classified into population-based and trajectory (or single point) methods [13].

Genetic algorithms (GAs) are the most common population-based methods for the $F|prmu|\sum T_j$ problem. The GA developed in [14] generates initial individuals randomly and outperforms

* Corresponding author at: School of Computer Science and Engineering, Southeast University, Nanjing 211189, China. Tel.: +86 25 52090916; fax: +86 25 52090916.
E-mail address: xpli@seu.edu.cn (X. Li).

http://dx.doi.org/10.1016/j.orp.2014.12.001

the DE (Differential Evolutionary) algorithm [15] proposed later. GAPR, GAPR2 and GADV [16] are three GA methods presented recently, which seem to be the best existing sequential algorithms for the considered problem. These three GAs use the EDD rule or both the EDD dispatching rule and the NEH_{EDD} heuristic to generate one or two initial individual(s) while the other initial individuals are generated randomly. Furthermore, based on these three GAs, three cooperative genetic algorithms (CGAPR, CGAPR2, and CGADV) were investigated in [17]. These investigations were performed on 4, 8, and 12 parallel computers and found to be relatively more effective than the GAPR, GAPR2, and GADV, which used only one computer.

The popular Trajectory Scheduling Methods (TSMs) start with an initial solution and improve it by a suitable strategy. Tabu Search (TS) and Simulated Annealing (SA) are commonly adopted complex strategies in TSMs. The two TS algorithms proposed in [18] and [10] utilize the heuristic developed in [19] and the EDD rule, respectively, to generate initial solutions. The four TS and four SA algorithms presented in [12] produce initial solutions using NEH_{EDD} and apply several local search methods for further improvement. The TS constructed in [20] adopts the Modified Due Dates rule to generate the initial solution. The two SAs introduced in [21] use the Earliest Apportioned Due Date rule as the initial heuristic. The SA algorithm developed in [22] generates the seed by a constructive heuristic and improves the current solution by several local search methods. A complex strategy was introduced in [23], which uses Ow's algorithm [19] to generate the initial solution and integrates SA with TS to obtain a high quality solution. Besides SA and TS, Iterated Local Search (ILS) is an effective trajectory meta-heuristic for combinatorial optimization. Though ILS has been applied to job shop scheduling problems [24], the PFSP with makespan minimization [25], and the PFSP with total flow time minimization [26], it has not yet been used to solve the $F|prmu|\sum T_j$ problem according to the extensive and comprehensive review on heuristics and metaheuristics for the m-machine flowshop problem with total tardiness minimization by E. Vallada, R. Ruiz and G. Minella [4].

In this paper, Trajectory Scheduling Methods (TSMs) are proposed for PFSP with total tardiness minimization. There are three components in each of the TSMs: Composite Heuristic, Adaptive Perturbation, and Restart Point Selection. For the considered problem, six composite heuristics and three compound perturbation methods are developed and compared. NEH_{EDD}, the most widely used rule for initial solutions, is adopted to generate the start point. An adaptive perturbation operator is presented to produce a set of candidate restart points. By defining the distance between a pair of solutions, a restart point selection criterion is introduced to select the most promising restart point of the next iteration from the candidate set.

The rest of the paper is organized as follows. Section 2 gives the description of the $F|prmu|\sum T_j$ problem. Section 3 discusses the proposed Trajectory Scheduling Methods. Empirical evaluation and comparison results of the proposed heuristics with existing algorithms are shown in Section 4. Finally, Section 5 concludes the paper with a summary of our findings and some fruitful directions for future research.

2. Problem description

To define the $F|prmu|\sum T_j$ problem, consider the following scenario: a set of n jobs are processed on m machines where each job requires m operations processed on m machines $M_1, \ldots, M_m$ sequentially with the same order. Each operation has a predetermined processing time and each machine can process one operation exclusively at a time. Preemption of jobs is not allowed.

Let $\mathbb{J} = \{J_1, \ldots, J_n\}$ be the job set and $\pi(n)$ be a schedule of the n jobs, i.e., a permutation of the n jobs, denoted as $(\pi_{[1]}, \ldots, \pi_{[n]})$. $\pi_{[k]} \in \mathbb{J}$ is the kth ($k = 1, \ldots, n$) job in $\pi(n)$. For convenience, a dummy job $\pi_{[0]}$ is added to the beginning of $\pi(n)$ with zero processing time and zero due date, i.e., the sequence can also be represented as $\pi(n) = (\pi_{[0]}, \pi_{[1]}, \ldots, \pi_{[n]})$. All the permutations of the n jobs are denoted as Ω, i.e., $\Omega = \{\pi(n)\}$. Let $C_{i,\pi_{[k]}}$ denote the completion time of job $\pi_{[k]}$ on machine i, and $C_{i,\pi_{[0]}} = 0$. $t_{i,j}$ represents the processing time of job j ($j = 1, 2, \ldots, n$) on machine i ($i = 1, 2, \ldots, m$). For $k = 1, \ldots, n$, $C_{i,\pi_{[k]}} = C_{1,\pi_{[k-1]}} + t_{1,\pi_{[k]}}$ when $i = 1$ and $C_{i,\pi_{[k]}} = max\{C_{i-1,\pi_{[k]}}, C_{i,\pi_{[k-1]}}\} + t_{i,\pi_{[k]}}$ when $i = 2, \ldots, m$. The tardiness of job $\pi_{[k]}$ is $T_{\pi_{[k]}} = max\{C_{m,\pi_{[k]}} - d_{\pi_{[k]}}, 0\}$, where $d_{\pi_{[k]}}$ is the due date of job $\pi_{[k]}$. The total tardiness of $\pi(n)$ can be denoted as

$$\widetilde{T}(\pi(n)) = \sum_{k=1}^{n} T_{\pi_{[k]}} = \sum_{k=1}^{n} max\{C_{m,\pi_{[k]}} - d_{\pi_{[k]}}, 0\}. \quad (1)$$

Obviously, the time complexity of calculating $\widetilde{T}(\pi(n))$ is $O(mn)$. The objective of the considered problem is to find the permutation $\pi^*(n) = \arg\min_{\pi(n)\in\Omega}\{\widetilde{T}(\pi(n))\}$ among the $n!$ solutions.

3. The proposed trajectory scheduling methods

Trajectory Scheduling Method (TSM) is composed of three components: Composite Heuristic, Adaptive Perturbation, and Restart Point Selection. A Composite Heuristic starts from an initial solution which is iteratively improved by an Iterated Improvement until it is stalled at a local optimum. The local optimum is perturbed by the Adaptive Perturbation. A new restart point is selected by the Restart Point Selection and the Iterated Improvement is performed again. The procedure is repeated until a given termination condition is satisfied.

Initially, both the current solution π^c and the current best solution π^b are generated by NEH_{EDD}. An Iterated Improvement procedure in a Composite Heuristic starts from π^c where the neighborhood is constructed by a neighborhood structure. If the best solution π^ℓ of the neighborhood is better than π^c, π^ℓ is selected as the new π^c. In every iteration, π^b is replaced with π^c if π^c is better than π^b. This neighborhood searching procedure is repeated until π^c is not better than π^b. Distinct from traditional perturbation operators each of which generates only one restart point, an Adaptive Perturbation method is developed to produce a set of candidate restart points. To select the most appropriate restart point from the candidate set, the distance from π^b to every candidate is calculated by the Restart Point Selection. If the termination condition is not satisfied, the procedure is repeated. The framework of the proposed trajectory scheduling methods is depicted as Algorithm 1.

Algorithm 1 Framework of Proposed Trajectory Scheduling Methods

1: Generate the start point π^c by NEH_{EDD}. $\pi^b \leftarrow \pi^c$.
2: **repeat**
3: Improve π^c by some *Composite Heuristic*. $\pi^b \leftarrow \pi^c$ if π^c is better than π^b.
4: Produce a set of candidate restart points by *Adaptive Perturbation* on π^c.
5: The best solution of the candidate set is selected as the new start point π^c according to the *Restart Point Selection*.
6: **until** (The termination criterion is satisfied)
7: **return** π^b.

3.1. Composite heuristics

According to the framework given in [27], a heuristic contains three phases: index development, solution construction

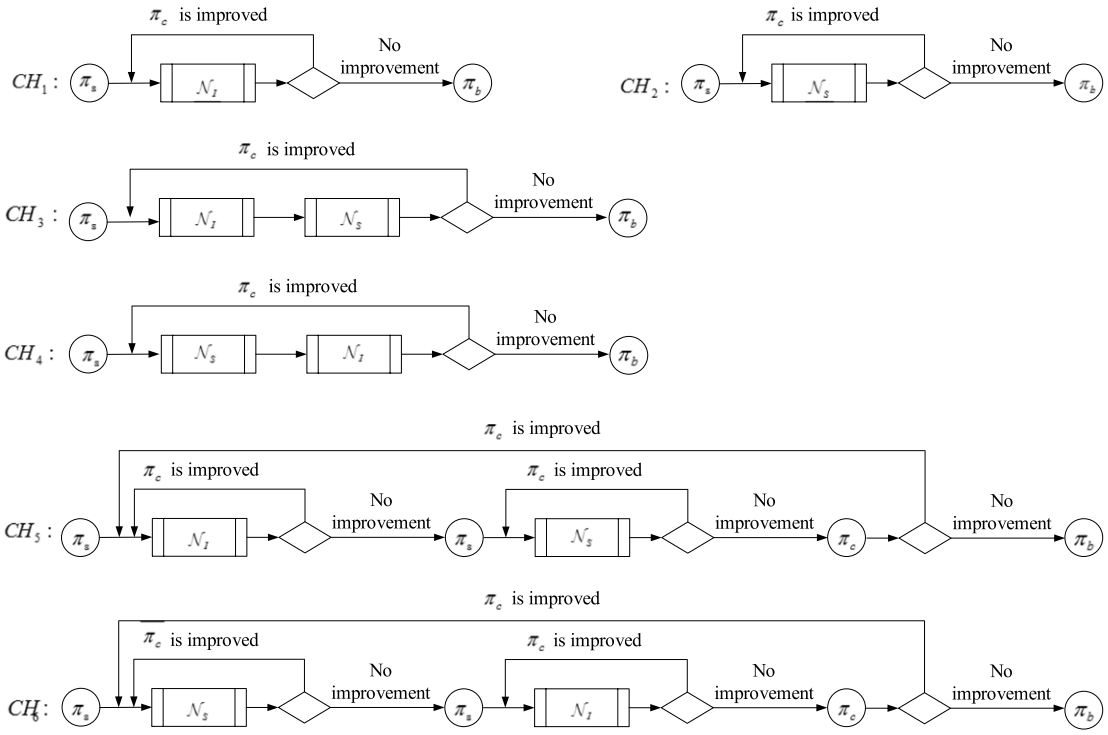

Fig. 1. Schematic figures of the composite heuristics.

and solution improvement. A heuristic is regarded as composite if it employs another heuristic for one or more of the three above-mentioned phases. In this paper, six composite heuristics, $CH_1, \ldots, CH_6$, are developed. The initial solution for each of these six heuristics is constructed by NEH_{EDD} [10]. This initial solution is improved by iteratively searching a neighborhood constructed by utilizing well-defined neighborhood structures. The six composite heuristics differ in terms of the manner in which the neighborhood structures are defined and iterative search is conducted.

3.1.1. Start point & neighborhood structures

NEH [11] is a simple but very good algorithm for permutation flow shops with makespan minimization. In the original NEH algorithm, a seed is generated by sorting all jobs in non-increasing order of the sum of the processing times on machines. A final solution is obtained in a constructive way. At each step, a new job in the order of the seed is added and inserted to the best slot to produce the best partial solution. There are two fundamental components in NEH: the seed generation and the constructive insertion. NEH_{EDD} [10] adopts the same constructive insertion but different seed generations, in which the seed is produced by sorting all jobs in non-decreasing order of the due dates. EDD means the Earliest Due Date. Since NEH_{EDD} is the most used rule to generate initial solutions for flowshop scheduling problems with due dates, it is also adopted to generate the start point π^c in this paper. For a n-job m-machine problem, the time complexity of NEH_{EDD} is $O(mn^3)$. The final best solution π^b is initialized as π^c.

An initial solution is always improved by searching its neighborhood, which is usually constructed by a neighborhood structure defined as [13]: a neighborhood structure is a function $\mathcal{N} : \Omega \rightarrow 2^{\Omega}$ that assigns to every $\pi \in \Omega$ a set of solutions $\mathcal{N}(\pi) \subseteq \Omega$. $\mathcal{N}(\pi)$ is called the neighborhood of π. For the neighborhood $\mathcal{N}(\pi)$, there are two commonly used extreme strategies to choose the improved solution: the first improvement and the best improvement. The former scans the neighborhood $\mathcal{N}(\pi)$ and chooses the first solution that is better than π while the latter exhaustively explores the neighborhood and returns the solution with the best objective function value (returns any one to break the tie if there is more than one solution with the best objective function value). In this paper, the best improvement strategy is adopted.

A neighborhood structure is crucial for the search trajectory of an algorithm because it determines the topological properties of the search landscape. Each neighborhood defines one landscape. Different heuristics have various search landscapes and search trajectories because of their distinct neighborhood structures. Generally, insertion and swap are two fundamental operations of heuristics, which can also be regarded as fundamental neighborhood structures, $\mathcal{N}_I$ and $\mathcal{N}_S$. Insertion neighborhood structure $\mathcal{N}_I$ on π generates neighborhood $\mathcal{N}_I(\pi)$ with $n(n-1)$ neighbors by sequentially taking out $\pi_{[i]}$ $(0 < i \leq n)$ and inserting it back to all possible $n-1$ slots (position i is excluded) of the remaining sequence. Swap neighborhood structure $\mathcal{N}_S$, however, produces the neighborhood $\mathcal{N}_S(\pi)$ of π by exchanging jobs $\pi_{[i]}$ and $\pi_{[j]}$ $(0 < i < j \leq n)$, i.e., $|\mathcal{N}_S(\pi)| = \frac{1}{2}n(n-1)$.

3.1.2. Composite heuristics

Generally, the result of NEH_{EDD} is always improved by local search procedures. The search landscape of each local search is determined by the neighborhoods resulted in the procedure. A neighborhood is generated by conducting a neighborhood structure. Each neighborhood defines one landscape. All local optimums of the defined landscapes by an algorithm forms a trajectory. The neighborhood structure is iteratively performed on the current solution π^c until there is no improvement, i.e., all the neighbors of π^c are not better than π^c. According to the two neighborhood structures ($\mathcal{N}_I$ and $\mathcal{N}_S$) and the iterated improvement schema, there are six combinations. And six composite neighborhood structures are defined: iterated $\mathcal{N}_I$, iterated $\mathcal{N}_S$, iterated combination of $\mathcal{N}_I$ with $\mathcal{N}_S$, iterated combination of $\mathcal{N}_S$ with $\mathcal{N}_I$, iterated combination of iterated $\mathcal{N}_I$ with iterated $\mathcal{N}_S$, and iterated combination of iterated $\mathcal{N}_S$ with iterated $\mathcal{N}_I$.

In this paper, six composite heuristics CH_i $(i = 1, \ldots, 6)$ are constructed based on six composite neighborhood structures. Fig. 1

shows the schematic of the corresponding six composite heuristics, denoted as $CH_1 \sim CH_6$.

CH_1 is identical to CH_2 except that CH_1 adopts $\mathcal{N}_I$ as the neighborhood structure while CH_2 uses $\mathcal{N}_S$. In fact, CH_1 has been adopted for some flowshop scheduling problems [28–31]. That CH_1 outperforming CH_2 for flowshop scheduling has also been demonstrated in [30]. We adapt them to the considered problem in this paper. Only the procedure of CH_1 is given here. For the returned solution π^c of NEH_{EDD}, the solution with the minimum total tardiness among $\mathcal{N}_I(\pi^c)$ is searched and denoted as π^ℓ. If π^ℓ is better than π^c, π^c is replaced with π^ℓ. The neighborhood structure $\mathcal{N}_I(\pi^c)$ is performed again. The process is repeated until π^ℓ is not better than π^c. π^b is updated to π^c if it is worse than π^c. CH_1 is formally described in Algorithm 2.

Algorithm 2 Composite Heuristic $CH_1(\pi^c)$

repeat
  $flag \leftarrow$ **false**.
  $\pi^\ell \leftarrow \arg\min\{\widetilde{T}(\pi)|\pi \in \mathcal{N}_I(\pi^c)\}$.
  if $\widetilde{T}(\pi^\ell) < \widetilde{T}(\pi^c)$ **then**
    $\pi^c \leftarrow \pi^\ell$, $flag \leftarrow$ **true**.
until ($flag =$ **false**)
$\pi^b \leftarrow \arg\min\{\widetilde{T}(\pi^b), \widetilde{T}(\pi^c)\}$.
return π^b.

CH_3 and CH_4 iterate the same way but use different orders of $\mathcal{N}_I$ and $\mathcal{N}_S$. For simplicity, only the procedure of CH_3 is given. For the returned solution π^c of NEH_{EDD}, the solution with the minimum total tardiness among $\mathcal{N}_I(\pi^c)$ is searched and denoted as π^ℓ. If π^ℓ is better than π^c, it is assigned to π^c. The solution with the minimum total tardiness among $\mathcal{N}_S(\pi^c)$ is searched and denoted as π^ℓ. If π^ℓ is better than π^c, it is assigned to π^c. The neighborhood structures $\mathcal{N}_I(\pi^c)$ and $\mathcal{N}_S(\pi^c)$ are performed again if π^c is improved by either of the neighborhood structures. The process is repeated until there is no improvement on π^c. π^b is updated to π^c if it is worse than π^c. CH_3 is formally described in Algorithm 3.

Algorithm 3 Composite Heuristic $CH_3(\pi^c)$

repeat
  $flag \leftarrow$ **false**.
  $\pi^\ell \leftarrow \arg\min\{\widetilde{T}(\pi)|\pi \in \mathcal{N}_I(\pi^c)\}$.
  if $\widetilde{T}(\pi^\ell) < \widetilde{T}(\pi^c)$ **then**
    $\pi^c \leftarrow \pi^\ell$, $flag \leftarrow$ **true**.
  $\pi^\ell \leftarrow \arg\min\{\widetilde{T}(\pi)|\pi \in \mathcal{N}_S(\pi^c)\}$.
  if $\widetilde{T}(\pi^\ell) < \widetilde{T}(\pi^c)$ **then**
    $\pi^c \leftarrow \pi^\ell$, $flag \leftarrow$ **true**.
until ($flag =$ **false**)
$\pi^b \leftarrow \arg\min\{\widetilde{T}(\pi^b), \widetilde{T}(\pi^c)\}$.
return π^b.

CH_5 is similar to CH_6, both of which are iterated procedures of two components and each component is still an iterated $\mathcal{N}_I$ or iterated $\mathcal{N}_S$. Here just CH_6 is illustrated because CH_5 is similar to it. The returned solution π^c of NEH_{EDD} is assigned to the start point. π^ℓ is obtained by $CH_2(\pi^c)$. If π^ℓ is better than π^c, it is assigned to π^c. $CH_1(\pi^c)$ is then conducted and returns its solution π^ℓ. Again, π^c is replaced with π^ℓ if the latter is better than the former. $CH_2(\pi^c)$ and $CH_1(\pi^c)$ are performed again if π^c is improved by either of the procedures, i.e., CH_6 stops only there is no improvement on π^c. π^b is updated to π^c if it is worse than π^c. CH_6 is formally described in Algorithm 4.

It is obvious that CH_1 and CH_2 usually spend the shortest computation time while CH_5 and CH_6 require the longest computation time among the six composite heuristics. Though the time complexity of either of the two neighborhood structures ($\mathcal{N}_I$ and $\mathcal{N}_S$)

Algorithm 4 Composite Heuristic $CH_6(\pi^c)$

repeat
  $flag \leftarrow$ **false**.
  $\pi^\ell \leftarrow CH_2(\pi^c)$.
  $\pi^\ell \leftarrow CH_1(\pi^\ell)$.
  if $\widetilde{T}(\pi^\ell) < \widetilde{T}(\pi^c)$ **then**
    $\pi^c \leftarrow \pi^\ell$, $flag \leftarrow$ **true**.
until ($flag =$ **false**)
$\pi^b \leftarrow \arg\min\{\widetilde{T}(\pi^\ell), \widetilde{T}(\pi^c)\}$.
return π^b.

is $O(mn^3)$, the computational complexity of the above composite heuristics is hard to estimate because of the unknown iteration number. Though it is hard to decrease the worst time complexity of the six methods, the simple speed-up method given in [16] can be adapted to save a lot of computation time for the total tardiness calculation.

3.2. Adaptive perturbation

3.2.1. Compound perturbation operators

The diversification effect should be increased when a search algorithm traps into a local optimum. A perturbation operator is always adopted to boost diversification, which helps the algorithm to jump out of a local optimum and reach a new solution. Appropriate perturbation strength is important: too weak a perturbation might not enable the algorithm to escape from the basin of attraction of the local optimum just found. On the other side, too strong a perturbation would make the algorithm similar to a random restart local search. Inappropriate restart points seriously deteriorate the efficiency of an algorithm because the search process would spend too much time on repetitive searches.

A perturbation operator guides the local optimum to a new solution, which can be regarded as a function $\mathcal{F} : \Omega \rightarrow \Omega$ which maps a solution to another. Insertion and adjacent exchange are the most commonly used perturbation operators, which are denoted as $\mathcal{F}_i$ and $\mathcal{F}_s$, respectively. $\mathcal{F}_i$ gets a new solution by extracting a job from a sequence π and randomly reinserting it into another slot of π. $\mathcal{F}_s$ obtains a new solution by exchanging a random pair of adjacent jobs in π. $\mathcal{F}_c$ is a linear combination of $\mathcal{F}_i$ and $\mathcal{F}_s$ which performs $\mathcal{F}_i$ with probability p_c or conducts $\mathcal{F}_s$ with probability $1 - p_c$ on π.

It is obvious that $\mathcal{F}_c$ becomes $\mathcal{F}_i$ if $p_c = 1$ and turns into $\mathcal{F}_s$ if $p_c = 0$. Generally, a simple perturbation operator generates a solution by conducting a perturbation structure d independent rounds (which is called perturbation strength). The processes performing $\mathcal{F}_i$, $\mathcal{F}_s$ and $\mathcal{F}_c$ are called SIPP (Simple Insertion Perturbation Process), SEPP (Simple Exchange Perturbation Process) and IPP (Integrated Perturbation Process), respectively. So the adjacent pairwise interchange perturbation developed by Dong et al. [26] is SEPP. The solution obtained by SIPP is denoted as $\mathcal{F}_i(d, \pi)$, that by SEPP is represented as $\mathcal{F}_s(d, \pi)$, while that by IPP is shown as $\mathcal{F}_c(d, p_c, \pi)$. Each of them is the only candidate restart point of the next iteration and is accepted even if its total tardiness value is worse than that of the current best solution π^b.

Naturally, we want to accept a solution better than π^b as the new restart point because accepting a solution worse than π^b would require considerable computation time for repetitive searches. Therefore, an appropriate restart point is desirable. In this paper, three compound perturbation operators are constructed, each of which performs one of the three above operators ω independent cycles. So ω candidate restart points are produced by each compound perturbation operator, which is denoted as Φ. For example, $\Phi = \bigcup_{k=1}^{\omega}\{\mathcal{F}_c(d, p_c, \pi^b)\}$ when IPP conducts ω $\mathcal{F}_c$ cycles. The selection of the appropriate candidate as the new restart point is important for the search trajectory of the proposed algorithms. Therefore, we investigate the following Restart Point Selection mechanism.

Table 1
Job pairs of the two sequences.

Sequence	Job-pairs									
x	**(1, 2)**	**(3, 1)**	(1, 4)	**(1, 5)**	**(3, 2)**	(2, 4)	(2, 5)	**(3, 4)**	**(3, 5)**	(5, 4)
y	**(2, 1)**	**(1, 3)**	(1, 4)	**(5, 1)**	**(2, 3)**	(2, 4)	(2, 5)	**(4, 3)**	**(5, 3)**	(5, 4)

3.2.2. Restart point selection

If the best solution $\mathcal{F}^*$ in Φ is better than $\mathcal{F}^b$, i.e., $\widetilde{T}(\pi^*) < \widetilde{T}(\pi^b)$, π^b is replaced with $\mathcal{F}^*$ as the restart point of the next iteration. Otherwise, an appropriate solution should be selected from Φ as the new restart point. Measuring the difference between $\mathcal{F}^*$ and $\mathcal{F}^b$ is key in this selection process. Different measurements are suitable for distinct operators, such as six pairs of adjacent jobs being used in the perturbation operator [26], four removed jobs in the destruction phase in [28,29], and eight removed jobs in the destruction phase in [32]. In this paper, a new distance between two sequences is defined, based on which the most appropriate restart point is selected.

For two n-job sequences x and y, we define the distance $D(x, y)$ from x to y by a number of different job-pairs (a job-pair is a pair of jobs, different order means different job-pairs.) between x and y. For example, $x = (3, 1, 2, 5, 4)$ and $y = (2, 5, 1, 4, 3)$. Table 1 shows the job-pairs of the two sequences.

Job-pairs $(1, 2)$ in sequence x means 1 is in front of 2. Job-pairs $(2, 1)$ in sequence y means 2 is in front of 1. Table 1 illustrates that there are four identical job-pairs between the two sequences: (1, 4), (2, 4), (2, 5), and (5, 4). The other six are different. Therefore, $D(x, y) = 6$.

To calculate $D(x, y)$, the job-pairs of x are inversed to x'. Then $D(x, y)$ is the number of identical job-pairs between x' and y. The inverse matrix $E = [e_{i,j}]_{n\times n}$ and the sequential matrix $F = [f_{i,j}]_{n\times n}$ are introduced, in which $e_{x_{[k]},x_{[l]}} = \begin{cases} 1 & if k > l \\ 0 & otherwise \end{cases}$ and $f_{y_{[k]},y_{[l]}} = \begin{cases} 1 & if k < l \\ 0 & otherwise \end{cases}$. The value $e_{x_{[k]},x_{[l]}} = 1$ implies that $x_{[l]}$ locates before $x_{[k]}$ in x when $k > l$, or there is a job-pair $(x_{[l]}, x_{[k]})$ in x. The value $f_{y_{[k]},y_{[l]}} = 1$ indicates that $y_{[l]}$ locates after $y_{[k]}$ when $k < l$ or there is a job-pair $(y_{[k]}, y_{[l]})$ in y. A binary variable $g_{i,j}$ is defined as $g_{i,j} = e_{i,j} \otimes f_{i,j} = \begin{cases} 1 & if e_{i,j} = 1 and f_{i,j} = 1 \\ 0 & otherwise \end{cases}$. In the matrix $G = E \otimes F = [g_{i,j}]_{n\times n}$, $g_{i,j}$=1 means that the positive job order of the pair (i, j) in y is identical to the inverse one in x, i.e., the job orders of x and y at positions i and j are just reversed. Therefore, $D(x, y)$=$\sum_{i=1}^{n}\sum_{j=1}^{n} g_{i,j}$.

For the above example, the corresponding matrixes E and F are

$$E = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}, \quad F = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

So

$$G = E \otimes F = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

and $D(x, y) = \sum_{i=1}^{5}\sum_{j=1}^{5} g_{i,j} = 6$.

We note that $D(x, y) = D(y, x)$ and $D(x, y) = 0$ if $x = y$. However, the probability that $x = y$ is low, particularly for the first iterations. The time complexity of computing $D(x, y)$ is $O(n^2)$.

In this paper, non-zero $\min_{\pi\in\Phi}\{D(\pi, \pi^b)\}$ is just adopted as the measure the difference between two solutions. In other words, the candidate solution $arg\ \min_{\pi\in\Phi}\{D(\pi, \pi^b)\}$ is selected as the restart point if $\min_{\pi\in\Phi}\{D(\pi, \pi^b)\} \neq 0$.

3.2.3. Proposed compound perturbation methods

Based on the three perturbation structures, three compound perturbation operators $CP_1, \ldots, CP_3$ are developed. CP_1 performs SIPP ω independent cycles, CP_2 performs SEPP ω independent cycles, while CP_3 performs IPP ω independent cycles. In every cycle, one of the three perturbation structures is conducted for d rounds. Because of their similarity, we just show the process of CP_3.

A candidate restart point $\mathcal{F}_c(d, p_c, \pi)$ is generated by IPP, which conducts $\mathcal{F}_c$ d independent rounds. In every round, $\mathcal{F}_i$ is performed with probability p_c or $\mathcal{F}_s$ is done with probability $1 - p_c$, i.e., a random job is removed from a sequence π^c and reinserted into another slot with probability p_c whereas a random pair of adjacent jobs of π^c are swapped with probability $1 - p_c$. IPP is performed ω independent cycles and $\Phi = \bigcup_{k=1}^{\omega}\{\mathcal{F}_c(d, p_c, \pi^c)\}$ is obtained. For each $\pi \in \Phi$, the distance $D(\pi, \pi^b)$ between π and the current best solution π^b is calculated. If the solution π^* with the minimum total tardiness in Φ is better than π^b, i.e., $\widetilde{T}(\pi^*) < \widetilde{T}(\pi^b)$, π^b is replaced with π^* and π^* is assigned as the new restart point. Otherwise, the solution with minimum $D(\pi, \pi^b)$ is selected as the new restart point π^ℓ. $\min_{\pi\in\Phi}\{D(\pi, \pi^b)\} = 0$ is not allowed because it implies that the two solutions are identical. So a candidate $\mathcal{F}_c(d, p_c, \pi)$ is abandoned if $D(\pi, \pi^b)$ and $\mathcal{F}_c$ is run again to produce another candidate. The procedure CP_3 is formally described in Algorithm 5.

Algorithm 5 Compound Perturbation Method CP_3

```
Dist ← n², Φ ← ∅, i ← 1, Flag ← false.
repeat
  π^c ← π^b.
  for j = 1 to d do
    Generate a random number λ ∈ [0, 1].
    if (λ ≤ p_c) then
      π^c is changed by randomly removing a job and reinserting it into another slot.
    else
      π^c is changed by randomly exchanging a pair of adjacent jobs.
  if D(π^c, π^b) = 0 then
    Continue. \\* Go to Step 4 and generate a candidate solution again.*\\
  else
    i ← i + 1, Φ ← Φ ∪ {π^c}.
until (i > ω)
for (each π ∈ Φ) do
  if T̃(π) < T̃(π^b) then
    π^b ← π, π^ℓ ← π, Flag ← true.
if (Flag = false) then
  for (each π ∈ Φ) do
    if D(π, π^b) < Dist then
      Dist ← D(π, π^b), π^ℓ ← π.
return π^b and π^ℓ.
```

The perturbation strength d, probability p_c and ω are three important parameters for $\mathcal{F}_c$. CP_3 would become IPP if $d = 1$ while it would be similar to a random search if d is too big. CP_3 turns into CP_1 if $p_c = 1$ whereas it becomes CP_2 if $p_c = 0$. When $\omega = 1$, there is only one candidate which is the restart point. However, the minimum distance would be very close to π^b with a high probability if ω is big enough, which results in too weak a perturbation. These parameters will be calibrated in Section 4.

The computational complexity of algorithm CP_3 depends on the number of π^c with $D(\pi^c, \pi^b) = 0$ generated between Steps 4–9 of Algorithm 5, which is uncertain. However, the time complexity between Steps 2–14 is only $O(d\omega)$ if only a few solutions π^c with $D(\pi^c, \pi^b) = 0$ are generated. Because the time complexity between Steps 19–21 is $O(n^2\omega)$ and $d < n$, the lower bound of the time complexity of CP_3 is $O(n^2\omega)$.

3.3. Trajectory scheduling methods

There are 18 combinations for the six proposed composite heuristics and three perturbation methods, i.e., 18 trajectory scheduling methods can be constructed for the considered problem. For simplicity, we denote the trajectory scheduling method as TSM_{ij} which combines CH_i with CP_j ($i = 1, \ldots, 6$; $j = 1, \ldots, 3$), where NEH_{EDD} is adopted to generate the initial solution π^c. π^c is improved by conducting CH_i. A new restart point is generated by performing CP_j. Then, heuristic CH_i is applied again. The process is repeated until the terminal criterion is satisfied. TSM_{ij} is constructed just by instantiation the Composite Heuristic with CH_i ($i = 1, \ldots, 6$) and the Adaptive Perturbation with CP_j ($j = 1, \ldots, 3$) for Algorithm 1. Similar to other meta-heuristics, the time complexity of TSM algorithms is hard to estimate because of the unknown iteration numbers between Step 2 and Step 6 in Algorithm 1.

4. Experimental results

We now describe the computational experiments conducted to compare the effectiveness of the proposed composite algorithms with the existing algorithms to solve the PFSP to minimize total tardiness. For this purpose, the best heuristics out of the 18 proposed algorithms are compared with GAPR, GAPR2, and GADV, the best existing sequential algorithms for the considered problem developed in [16]. All the 21 algorithms are implemented in Java and performed on the same virtual machine with Intel i5-3470 CPU (four cores, 2.2 GHz) and 1 GB Memory. Though CGAPR, CGAPR2 and CGADV proposed in [17] are the most effective existing algorithms for the considered $F|prmu|\sum T_j$ problem, they are not compared with the proposed trajectory scheduling methods in this paper because they are cooperative algorithms performed on more than one computer while our proposed algorithms are sequential ones conducted only on one computer.

Different testing instance sets are used for calibrating the involved parameters and comparing the algorithms in this paper. For calibrating parameters, problem instances are randomly generated according to [16,17], where instances with size $n \in \{50, 150, 250, 350\}$ and $m \in \{10, 30, 50\}$ are tested. The processing times are uniformly distributed in [1, 99]. Due dates are generated in terms of the Tardiness Factor F and the Due Date Range R, with a uniform distribution between $B(1 - F - R/2)$ and $B(1 - F + R/2)$ where B is a tight lower bound of the makespan given by Taillard [33]. For each instance size, the following combinations of F and R are concerned, $F \in \{0.2, 0.4, 0.6\}$ and $R \in \{0.2, 0.6, 1\}$. For a given F and R, there are five instances for each combination of n and m. Therefore, there are $4 \times 3 \times 3 \times 3 \times 5 = 540$ instances in total. However, the 540 benchmark instances given at http://soa.iti.es are adopted to compare the involved algorithms.

In this paper, effectiveness of methods is measured by RDI (Relative Deviation Index), which is commonly used to evaluate performance of scheduling problems with tardiness criterion [10,12,16]. RDI is defined by

$$RDI = \frac{\widetilde{T}(M) - \widetilde{T}(B)}{\widetilde{T}(W) - \widetilde{T}(B)} \times 100\% \quad (2)$$

where $\widetilde{T}(M)$ denotes the total tardiness of algorithm M, $\widetilde{T}(B)$ and $\widetilde{T}(W)$ are respectively the best and the worst solutions of the involved algorithms on each instance. The index lies between 0 and 100. The closer to 0 the better the algorithm is. Note that if the worst and the best solutions are similar, all the combinations would provide the best (same) solution and hence, the index value would be 0 (the best index value).

Generally, there are four possible terminal criteria for iterative methods [13]: maximum computation time, maximum number of iterations, finding a solution with the objective function value less than a predefined threshold value, or reaching the maximum number of iterations without improvements. Use of the maximum number of iterations is unfair for the different size problems. In view of the NP-hardness of the considered problem, it is unreasonable to assume that a solution with the objective function value less than a predefined threshold value found in a reasonable amount of computational effort. The maximum number of iterations without improvements is only used to compare the proposed composite heuristics among themselves. For comparing the performance of the proposed composite algorithms with existing algorithms, the maximum number of iterations without improvements is hard to find since a big number would consume huge computation time and a small number may miss better solutions for many cases. Therefore, a maximum computation time is adopted as the termination criterion, which were also used in [16,17,28,29]. The maximum computation time of $(n \times m/2 \times t)$ milliseconds is set as the termination criterion where t is the total number of iterations being considered. This function is related to the number of job n and the number of machine m. More computation time is allocated for bigger n and m.

4.1. Parameter calibration

To calibrate the three parameters (d, p_c, and ω) in the compound perturbation methods, the three parameters are restricted to {3,4,5,6,7}, {0,0.2,0.4, 0.6,0.8,1} and {1,10, 20,30,40}, respectively. The different value of p_c represents a different compound perturbation method. e.g., $p_c = 0$ indicates CP_1, $p_c = 1$ indicates CP_2 and other values represent CP_3. Every instance is conducted 5 independent times. Therefore, there are $5 \times 6 \times 5 = 150$ parameter combinations and $150 \times 540 \times 5 = 405\,000$ tests in total for the calibration. The maximum computation time $(n \times m/2 \times 120)$ milliseconds is set as the terminal criterion.

RDI is calculated for each algorithm on all the instances. The three parameters are analyzed experimentally using the multi-factor analysis of variance (ANOVA) method. First, the three main hypotheses (normality, homoscedasticity, and independence of the residuals) are checked by residuals from the experiments. Since all the p-values in the experiments are close to zero, they are not analyzed in this paper. Greater F-ratio implies more effective the factor. Interactions between (or among) any two (or more than two) factors are not considered because the corresponding F-ratios are quite small. The results are shown in Table 2, from which it can be observed that parameters for CP_1 and CP_2 exerts little influence while those for CP_3 exert great influence on any composite heuristic CH_i. In other words, a trajectory scheduling method is more robust by integrating a composite heuristic with the hybrid perturbation structure than just integrating it with the simple inserting or exchanging structure.

To further illustrate the impact of different values of every parameter on each trajectory method, TSM_{53} is taken as an example. The Means plot and the Tukey HSD intervals of p_c, ω and d at the 95% confidence level are depicted in Fig. 2. Fig. 2 shows that p_c exerts great influence on TSM_{53}, of which the performance achieves the best when $p_c = 0.4$. Fig. 2 indicates that $\omega = 40$ is the best among the five tested parameters. Fig. 2 illustrates that TSM_{53} obtains the best RDI when $d = 4$.

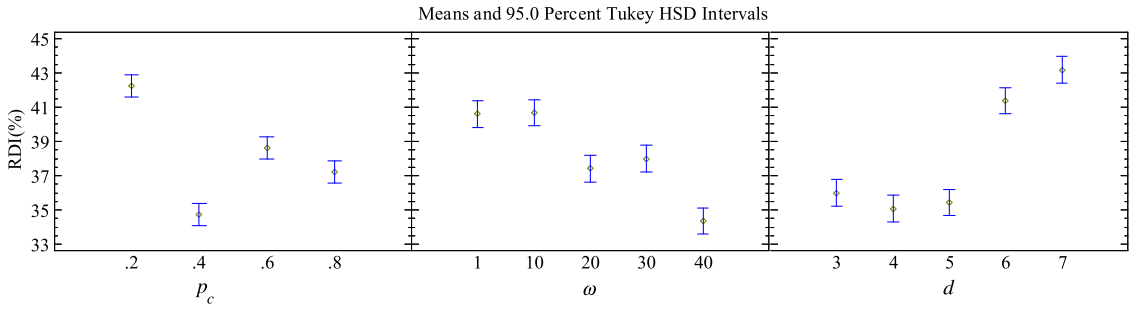

Fig. 2. Means plot and the Tukey HSD intervals at the 95% confidence level for p_c, ω and d.

Table 2
Best combination of parameters for TSM_{ij}.

Algorithm	CP_1		CP_2		CP_3		
	ω	d	ω	d	ω	d	p_c
CH_1	30	3	10	6	30	4	0.2
CH_2	30	3	1	8	20	4	0.4
CH_3	40	3	20	6	30	3	0.8
CH_4	30	4	20	5	30	3	0.2
CH_5	10	3	40	5	40	4	0.4
CH_6	20	4	30	4	30	3	0.2

4.2. Performance comparison on the proposed composite heuristics

In this section, performance of various CH_i heuristics is compared with that of NEH_{EDD}. Different from the other comparisons in this paper, every instance is performed only once because the heuristics are deterministic. Therefore, there are 540 tests in total. The terminal criterion is set as the maximum number of iterations without improvements. In the performance measurement RDI, $\widetilde{T}(B)$ and $\widetilde{T}(W)$ are the best and the worst solutions selected from the seven compared methods on each instance. Effectiveness and efficiency of the compared heuristics are shown in Tables 3 and 4, respectively.

From Table 3, it can be observed that NEH_{EDD} has the worst effectiveness among the seven algorithms. CH_1 and CH_2 are the worst while CH_3 and CH_4 are the best on average of the 6 composite heuristics. The average RDI of CH_4 is only 4.78% while that of CH_1 is 32.67%. CH_3 and CH_4 outperform CH_1 and CH_2 on all instances. Though CH_5 and CH_6 also outperform CH_1 and CH_2 on most instances, they are outperformed by CH_3 and CH_4 on all instances. As well, the methods with $\mathcal{N}_S$ ahead of $\mathcal{N}_I$ are always better than those with $\mathcal{N}_I$ ahead of $\mathcal{N}_S$, e.g., CH_2 is better than CH_1, CH_4 outperforms CH_1, and CH_6 is not worse than CH_5 both on average and on most instances.

Table 4 shows that NEH_{EDD} without local search is the fastest among the 7 compared algorithms with an average CPU time 1.635 s. For the composite heuristics, CH_1 and CH_2 are faster than CH_5 and CH_6, which are also faster than CH_3 and CH_4. So better performance implies more CPU time for the compared heuristics.

Table 3
Effectiveness comparison on the six composite heuristics and NEH_{EDD}.

Instance	CH_1	CH_2	CH_3	CH_4	CH_5	CH_6	NEH_{EDD}
50 × 10	33.77	32.57	6.43	**4.68**	23.29	15.88	100.00
50 × 30	59.25	34.97	**8.34**	10.20	26.03	28.98	100.00
50 × 50	67.92	39.10	10.67	**9.90**	33.31	28.66	100.00
150 × 10	20.80	27.77	4.02	**1.40**	15.41	11.07	100.00
150 × 30	40.61	32.92	**3.46**	7.22	24.46	21.07	100.00
150 × 50	53.68	37.19	**4.72**	6.19	30.16	29.59	100.00
250 × 10	14.40	26.11	1.51	**1.38**	12.26	8.26	100.00
250 × 30	32.07	30.68	4.37	**2.67**	21.80	18.15	100.00
250 × 50	22.13	20.13	3.09	**2.69**	37.08	32.16	100.00
350 × 10	17.05	26.33	6.08	**5.94**	10.22	7.80	100.00
350 × 30	14.18	16.22	3.51	**2.08**	32.36	27.99	100.00
350 × 50	16.12	15.49	4.75	**3.01**	41.09	38.44	100.00
Average	32.67	28.29	5.08	**4.78**	25.62	22.34	100.00

Table 4
Average CPU time (s) comparison on the six composite heuristics and NEH_{EDD}.

Instance	CH_1	CH_2	CH_3	CH_4	CH_5	CH_6	NEH_{EDD}
50 × 10	**0.016**	0.017	0.133	0.114	0.028	0.036	0.000
50 × 30	**0.044**	0.053	0.307	0.291	0.075	0.093	0.010
50 × 50	**0.070**	0.078	0.457	0.410	0.117	0.140	0.016
150 × 10	0.611	**0.550**	2.967	3.108	0.802	0.822	0.183
150 × 30	1.384	**1.359**	9.176	8.784	2.004	2.028	0.444
150 × 50	2.108	**2.104**	14.467	15.824	3.165	3.246	0.666
250 × 10	**2.345**	2.462	10.392	10.465	3.559	3.621	0.683
250 × 30	**5.787**	6.429	34.435	34.951	9.193	9.591	1.713
250 × 50	**8.675**	9.836	61.718	60.631	14.616	15.316	2.713
350 × 10	**5.498**	6.306	16.621	16.665	9.630	9.822	1.596
350 × 30	**14.412**	16.782	52.529	52.330	25.372	26.860	4.401
350 × 50	**22.189**	25.861	90.133	90.048	39.779	42.165	7.200
Average	**5.262**	5.986	24.445	24.469	9.028	9.478	1.635

Table 5
Effectiveness comparison on 12 trajectory scheduling methods with the maximum computational time $n \times m/2 \times 120$.

Instance	TSM_{31}	TSM_{32}	TSM_{33}	TSM_{41}	TSM_{42}	TSM_{43}	TSM_{51}	TSM_{52}	TSM_{53}	TSM_{61}	TSM_{62}	TSM_{63}
50 × 10	25.62	30.53	22.55	19.34	18.07	18.98	23.91	30.72	21.27	17.11	**15.39**	17.00
50 × 30	24.04	28.77	21.73	18.23	19.57	19.18	23.12	29.49	21.64	**17.41**	18.03	17.53
50 × 50	25.83	29.22	25.53	21.11	21.23	20.31	24.09	30.20	23.04	19.62	18.79	**18.06**
150 × 10	29.61	33.72	24.79	18.58	19.21	19.24	25.82	31.86	19.63	16.07	16.88	**15.20**
150 × 30	31.43	34.89	26.40	20.92	20.14	20.64	25.50	33.05	19.37	14.03	14.14	**13.59**
150 × 50	29.86	35.81	25.51	21.09	20.54	22.93	26.56	33.10	18.27	**14.52**	15.58	15.64
250 × 10	31.35	35.49	28.65	18.94	18.32	18.37	25.59	29.73	21.12	14.00	14.61	**11.98**
250 × 30	33.71	35.12	28.81	21.77	21.90	21.10	26.49	32.10	19.73	11.82	12.11	**11.18**
250 × 50	30.19	34.00	27.45	21.48	20.70	22.25	24.16	32.37	17.87	**12.56**	12.94	12.68
350 × 10	31.46	34.25	30.92	21.55	23.35	22.84	24.72	27.76	22.00	13.70	14.46	**11.54**
350 × 30	35.19	37.93	32.73	23.01	22.41	22.89	27.40	33.68	22.48	11.17	13.29	**10.95**
350 × 50	33.50	36.65	31.21	24.86	24.55	25.35	25.03	31.63	19.50	12.43	13.22	**12.33**
Average	30.15	33.87	27.19	20.91	20.83	21.17	25.20	31.31	20.49	14.54	14.95	**13.97**

4.3. Performance comparison on the proposed trajectory scheduling methods

To compare performance of the proposed trajectory scheduling methods, five replications are executed on each instance. Therefore, there are $540 \times 5 = 2700$ tests in total. The maximum computation time ($(n \times m/2 \times t)$ milliseconds) is set as the termination criterion where t takes a value from {60, 90, 120}. Because CH_1 and CH_2 heuristics show the worst effectiveness among the six composite heuristics, they are not integrated with the three compound perturbation methods, i.e., 12 trajectory scheduling methods (TSM_{ij}, $i = 3, \ldots, 6$, $j = 1, \ldots, 3$) are compared by RDI. The parameters shown in Table 2 are adopted. In the measurement RDI, $\widetilde{T}(B)$ and $\widetilde{T}(W)$ are the best and the worst solutions selected from the 12 compared trajectory scheduling methods performed 5 independent times on each instance, i.e., the best and worst among the 60 values. From the experimental results, we observed the three maximum computation time termination conditions show similar performance on the 12 methods. We just give the results with the maximum computation time $n \times m/2 \times 120$ in Table 5.

Table 5 illustrates that the TSMs based on CH_6 outperform the other 9 trajectory scheduling methods on all instances. For example, the average RDI of TSM_{33}, TSM_{43}, and TSM_{53} is 27.19%, 21.17%, and 20.49%, respectively whereas that of TSM_{63} is only 13.97%. On the other hand, among the three trajectory scheduling methods based on each composite heuristic CH_i ($i = 3, \ldots, 6$), TSM_{i3} (perturbed by CP_3) always outperforms the other two and TSM_{i2} (perturbed by CP_2) is often the worst method. For example, the average RDI of TSM_{61}, TSM_{62}, and TSM_{63} is 14.54%, 14.95%, and 13.97%, respectively. This implies that the hybrid perturbation structure is more effective than the only insertion-based and the only swap-based ones.

Details about the performance for each trajectory scheduling methods with different numbers of jobs and machines are shown in Figs. 3 and 4. Figs. 3 and 4 show that TSM_{63} is the best one while TSM_{32} is the worst one for each job number n and machine number m. With the increasing of n, the difference between each TSM is increasing while that for m is more or less the same. This phenomenon implies that the performance of TSM is more sensitive to the job number n.

Table 6 shows the comparisons of the best three methods (TSM_{61}, TSM_{62} and TSM_{63}) with different termination criteria ($t \in$ {60, 90, 120}). It can be seen that TSM_{63} is the best for each termination criterion. i.e., the average RDI for TSM_{63} is 47.98, 37.85 and 30.65, respectively. The performance of all the three algorithms is improved with the increasing of the termination criterion. i.e., the average RDI for TSM_{61} is 50.20, 39.31 and 31.76, respectively.

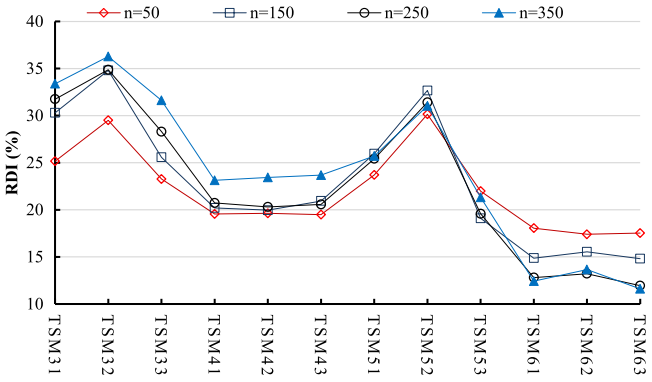

Fig. 3. Average RDI (%) of the 12 trajectory scheduling methods with different n.

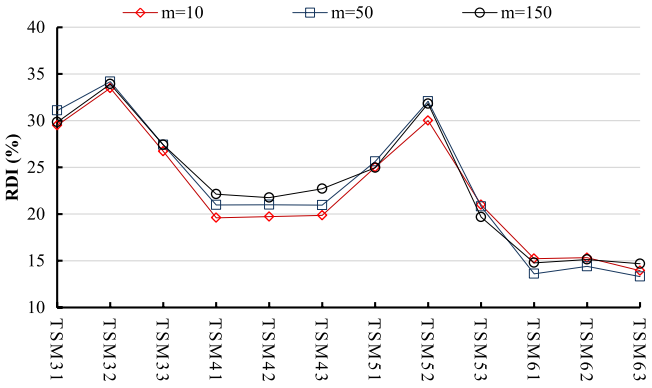

Fig. 4. Average RDI (%) of the 12 trajectory scheduling methods with different m.

4.4. Performance comparison with existing algorithms

Both the best and worst trajectory scheduling methods TSM_{63} and TSM_{32} in Table 5 are selected to compare with the best existing sequential algorithms GAPR, GAPR2 and GADV. Every instance is performed 5 independent times, i.e., there are $540 \times 5 = 2700$ tests in total. The maximum computation time ($n \times m/2 \times$ {60, 90, 120} milliseconds) is set as the termination criterion. As well, in the performance measurement RDI of the compared algorithms. $\widetilde{T}(B)$ and $\widetilde{T}(W)$ are respectively the best and the worst solutions selected among the five compared algorithms running five times with three termination criteria on each instance. In other words, $\widetilde{T}(B)$ and $\widetilde{T}(W)$ are the best and the worst among the $5 \times 5 \times 3 = 75$ values of each instance size. The results are shown in Table 7.

Table 7 indicates that RDI of each algorithm decreases with an increase in the computation time for each of the 12 subsets of

Table 6
Comparisons of the best three methods with different termination criteria.

Instance	TSM_{61}			TSM_{62}			TSM_{63}		
	$t = 60$	$t = 90$	$t = 120$	$t = 60$	$t = 90$	$t = 120$	$t = 60$	$t = 90$	$t = 120$
50 × 10	43.88	37.03	31.55	37.96	31.63	**27.21**	42.77	36.52	31.15
50 × 30	50.02	43.21	**39.10**	54.15	45.74	40.05	52.94	44.67	39.85
50 × 50	54.84	48.24	43.22	53.37	46.13	41.03	50.36	44.56	**39.86**
150 × 10	45.90	36.51	30.47	46.56	37.81	31.69	42.52	33.97	**28.80**
150 × 30	54.87	43.05	34.81	55.45	42.36	35.34	51.54	40.81	**33.60**
150 × 50	60.48	45.92	**36.66**	61.79	48.31	39.51	58.31	47.62	39.60
250 × 10	42.34	32.88	25.78	42.81	33.48	26.79	37.12	28.37	**21.77**
250 × 30	48.87	36.92	28.13	50.44	37.99	28.61	44.04	33.94	**26.46**
250 × 50	56.59	42.41	**32.39**	58.21	43.49	33.58	55.27	42.25	32.62
350 × 10	39.41	28.92	22.17	41.02	30.94	23.51	37.76	26.54	**19.18**
350 × 30	48.52	35.45	25.68	53.08	39.13	30.77	48.21	34.96	**24.72**
350 × 50	56.72	41.20	31.13	59.78	43.32	32.76	54.90	39.94	**30.19**
Average	50.20	39.31	31.76	51.22	40.03	32.57	47.98	37.85	**30.65**

Table 7
Performance comparison of *TSM* against the best existing meta-heuristics.

Instance	$t = 60$					$t = 90$					$t = 120$				
	TSM_{63}	TSM_{32}	GAPR	GAPR2	GADV	TSM_{63}	TSM_{32}	GAPR	GAPR2	GADV	TSM_{63}	TSM_{32}	GAPR	GAPR2	GADV
50 × 10	**39.75**	48.13	54.31	53.78	52.93	**38.20**	45.79	53.57	51.10	50.52	**36.86**	44.33	51.93	49.37	49.37
50 × 30	**42.31**	49.80	56.45	55.41	54.50	**41.01**	47.47	55.21	52.99	52.00	**39.93**	46.20	53.73	50.57	50.29
50 × 50	**43.39**	50.88	56.80	53.81	54.15	**41.78**	49.26	55.05	51.28	51.43	**40.56**	47.85	53.55	49.69	49.54
150 × 10	**39.90**	47.05	55.03	59.62	59.09	**37.74**	45.15	52.53	57.36	57.05	**36.49**	43.95	51.29	55.99	55.74
150 × 30	**41.66**	51.73	52.38	61.30	60.28	**39.30**	50.53	49.84	58.24	57.87	**37.01**	49.85	48.07	56.70	56.68
150 × 50	**42.73**	54.90	53.39	60.50	60.00	**39.95**	53.46	51.05	58.93	58.18	**37.56**	52.16	49.93	57.99	57.04
250 × 10	**38.16**	44.05	52.88	55.02	53.68	**36.22**	42.85	50.21	53.04	52.36	**34.98**	41.56	49.16	52.03	51.43
250 × 30	**39.73**	52.72	53.72	58.92	58.14	**37.33**	51.23	51.33	57.25	56.85	**35.48**	49.73	50.11	56.38	55.97
250 × 50	**40.66**	51.27	54.13	58.58	58.14	**38.06**	49.01	51.53	55.87	56.45	**36.18**	47.74	49.57	55.00	54.88
350 × 10	**37.92**	42.93	51.78	53.32	53.59	**36.30**	41.51	48.95	52.18	51.47	**35.17**	40.14	47.63	50.99	50.56
350 × 30	**39.86**	50.47	51.39	55.13	54.86	**37.60**	48.13	49.90	52.50	53.33	**35.48**	47.71	49.07	51.64	52.45
350 × 50	**39.16**	48.13	50.02	53.72	54.66	**37.05**	46.77	48.83	51.62	52.57	**35.09**	45.63	47.60	50.40	52.02
Average	**40.44**	49.34	53.52	56.59	56.17	**38.38**	47.60	51.50	54.36	54.17	**36.73**	46.40	50.14	53.06	53.00

problem instances, i.e., the more computation time is, the better are the solutions. For example, the average RDI of TSM_{63} for instance 50×10 with $t \in \{60, 90, 120\}$ is 39.75%, 38.20% and 36.86%, respectively. TSM_{63} and TSM_{32} achieve the best performance on all instance groups. The average RDIs of TSM_{63} and TSM_{32} are much better than that of the other three for the corresponding three computation time cases. For example, the average RDI of TSM_{32} is only 46.40% while that of GAPR, GAPR2, and GADV is 50.14%, 53.06%, and 53.00%, respectively, when $t = 120$. The size of instances exerts little influence on effectiveness of *TSM*. It is hard to explicitly figure out the exact tendency. However, the general tendency is that RDI increases with m, e.g., RDIs of TSM_{63} are 36.86%, 39.93% and 40.56% for different m when $n = 50$, $t = 120$. Therefore, *TSM* usually finds better solutions for the small instances.

Means plot, showing the significance difference in effectiveness, is depicted in Fig. 5 for all the compared algorithms with $t = 60$, $t = 90$ and $t = 120$. From Fig. 5, it can be observed that TSM_{63} and TSM_{32} obtain the best performance among the compared algorithms for all the termination criterion. As well, the average RDI differences of all the algorithms between two termination conditions are almost the same, which indicates that the algorithms converge similarly.

To further investigate the influence of the number of jobs and machines to the performance of each method, test results with $t = 90$ are selected. Details are shown in Figs. 6 and 7. Figs. 6 and 7 show that TSM_{63} much better than other existing algorithms while TSM_{32} is more or less the same as them. With the increasing of n and m, the difference between the compared algorithms are similar. This phenomenon implies that the performances of compared algorithms are not sensitive to the job number and the machine number.

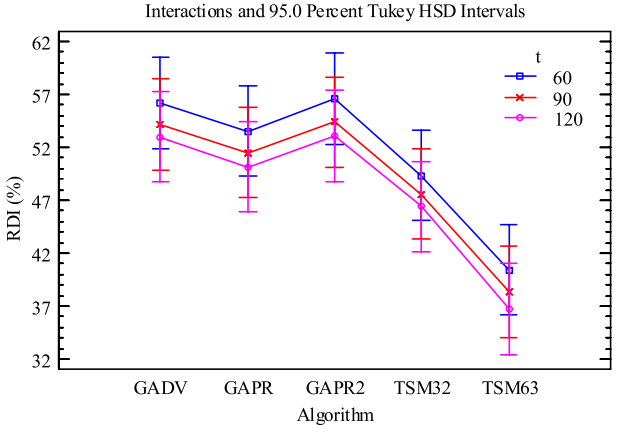

Fig. 5. Interactions and 95.0 Percent Tukey HSD Intervals for all the compared algorithms with $t = 60$, $t = 90$ and $t = 120$.

From a global point of view, Table 8 shows the result of comparisons with the same best and worst solutions. The best and worst composite heuristics (CH_4 and CH_1), the best trajectory scheduling methods TSM_{63} and the best existing algorithm *GAPR* are compared. From the table, it can be seen that TSM_{63} is best (14.98) while CH_1 is the worst (94.48) among the compared algorithms. *GAPR* is much better than both the composite heuristics (CH_4 and CH_1) with the average RDI 30.88. The average RDI of all the compared algorithms have no obvious trends when the machine number m increases. However, the average RDIs of CH_1 and CH_4 become worse while those of TSM_{63} and *GAPR* become better with the increasing of the job number n.

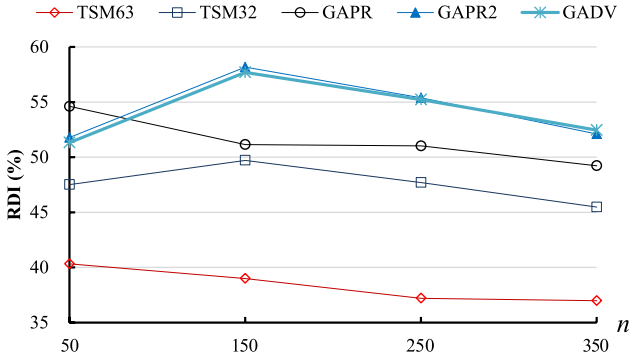

Fig. 6. Average RDI (%) of the compared methods with different n and $t = 90$.

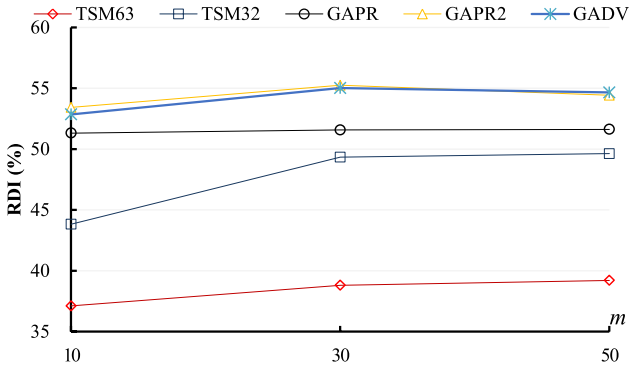

Fig. 7. Average RDI (%) of the compared methods with different m and $t = 90$.

Table 8
Comparisons with the same best and worst solutions.

Instance	CH_1	CH_4	TSM_{63}	$GAPR$
50 × 10	89.93	65.38	**28.42**	26.35
50 × 30	88.08	68.38	**20.59**	62.04
50 × 50	88.43	59.89	**9.77**	34.90
150 × 10	98.45	42.34	**11.61**	46.61
150 × 30	96.33	80.59	**27.46**	37.36
150 × 50	99.86	76.75	**27.62**	35.20
250 × 10	97.23	36.68	**12.80**	39.29
250 × 30	89.31	56.69	**15.77**	35.55
250 × 50	99.67	95.32	**1.46**	4.01
350 × 10	97.89	93.54	**0.93**	6.66
350 × 30	90.32	89.72	**3.63**	10.30
350 × 50	98.24	98.24	**19.72**	32.29
Average	94.48	71.96	**14.98**	30.88

5. Conclusion

Trajectory Scheduling Methods (*TSM*) are presented in this paper for the Permutation Flow Shop Scheduling Problem (PFSP) to minimize total tardiness. Six composite heuristics, $CH_1, \ldots, CH_6$, are constructed based on the Insertion and Swap neighborhood structures to boost the intensification of *TSM*. Three compound perturbation methods, $CP_1, \ldots, CP_3$, are developed to generate a set of candidate starting points with different perturbation strengths. The distance of each candidate from the current best solution is defined. The candidate with the shortest distance is selected as the new starting point of the next iteration to get a balance between diversification and intensification. 18 trajectory scheduling methods TSM_{ij} are constructed by combining CH_i and CH_j.

For each TSM_{ij}, three critical parameters are tested on 540 random instances and the best values are determined according to the experiments. By testing the proposed composite heuristics $CH_1, \ldots, CH_6$ on the benchmark individually, the best four $CH_3, \ldots, CH_6$ are selected to combine with the different compound perturbation methods $CP_1, \ldots, CP_3$. Results show that TSM_{63} and TSM_{32} are the best and worst ones among the 12 combinations and the hybrid perturbation structure is more effective. Both TSM_{63} and TSM_{32} heuristics are compared with the best existing sequential algorithms GAPR, GAPR2 and GADV for PFSP. Although all the compared algorithms show a similar convergency, TSM_{63} and TSM_{32} improve the effectiveness of the algorithms for the considered problem significantly, which implies that not only the composite heuristics but also the proposed perturbation structure play an important role in the proposed trajectory scheduling methods.

This development in this paper also suggests some fruitful directions for future research. For example, the proposed trajectory methods can also be used to solve several other scheduling problems like the hybrid flowshop problems and manufacturing cell scheduling problems. Further, scheduling problems with several other performance measures, like the total earliness and tardiness penalties, can be solved using the proposed methods. Therefore, it is worthwhile to further explore and develop the proposed composite heuristics and the perturbation techniques to solve a variety of scheduling problems.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grants 61272377) and the Specialized Research Fund for the Doctoral Program of Higher Education (20120092110027).

References

[1] Sen T, Gupta SK. A state-of-art survey of static scheduling research involving due dates. Omega 1984;12(1):63–76.
[2] Du J, Leung J. Minimizing total tardiness on one machine is np-hard. Math Oper Res 1990;15(3):483–95.
[3] Pinedo M. Scheduling: theory, algorithms, and systems. Springer; 2012.
[4] Vallada E, Ruiz R, Minella G. Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. Comput Oper Res 2008;35(4):1350–73.
[5] Sen T, Dileepan P, Gupta JN. The two-machine flowshop scheduling problem with total tardiness. Comput Oper Res 1989;16(4):333–40.
[6] Kim Y. A new branch and bound algorithm for minimizing mean tardiness in two-machine flowshops. Comput Oper Res 1993;20(4):391–401.
[7] Pan J, Fan E. Two-machine flowshop scheduling to minimize total tardiness. Internat J Systems Sci 1997;28(4):405–14.
[8] Pan J, Chen J, Chao C. Minimizing tardiness in a two-machine flow-shop. Comput Oper Res 2002;29(7):869–85.
[9] Schaller J. Note on minimizing total tardiness in a two-machine flowshop. Comput Oper Res 2005;32(12):3273–81.
[10] Kim Y. Heuristics for flowshop scheduling problems minimizing mean tardiness. J Oper Res Soc 1993;19–28.
[11] Nawaz M, Enscore E, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 1983;11(1):91–5.
[12] Kim Y, Lim H, Park M. Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process. European J Oper Res 1996;91(1):124–43.
[13] Blum C, Roli A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput Surv (CSUR) 2003;35(3):268–308.
[14] Onwubolu G, Mutingi M. Genetic algorithm for minimizing tardiness in flow-shop scheduling. Production Planning & Control 1999;10(5):462–71.
[15] Onwubolu G, Davendra D. Scheduling flow shops using differential evolution algorithm. European J Oper Res 2006;171(2):674–92.
[16] Vallada E, Ruiz R. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. Omega 2010;38(1):57–67.
[17] Vallada E, Ruiz R. Cooperative metaheuristics for the permutation flowshop scheduling problem. European J Oper Res 2009;193(2):365–76.
[18] Adenso-Díaz B. Restricted neighborhood in the tabu search for the flowshop problem. European J Oper Res 1992;62(1):27–37.
[19] Ow P. Focused scheduling in proportionate flowshops. Manage Sci 1985;31(7):852–69.
[20] Armentano V, Ronconi D. Tabu search for total tardiness minimization in flowshop scheduling problems. Comput Oper Res 1999;26(3):219–35.
[21] Parthasarathy S, Rajendran C. Scheduling to minimize mean tardiness and weighted mean tardiness in flowshop and flowline-based manufacturing cell. Comput Ind Eng 1998;34(2):531–46.

[22] Hasija S, Rajendran C. Scheduling in flowshops to minimize total tardiness of jobs. Int J Prod Res 2004;42(11):2289–301.
[23] Adenso-Díaz B. An sa/ts mixture algorithm for the scheduling tardiness problem. European J Oper Res 1996;88(3):516–24.
[24] Lourenčo H. Job-shop scheduling: Computational study of local search and large-step optimization methods. European J Oper Res 1995;83(2):347–364.
[25] Stützle T.. Applying iterated local search to the permutation flow shop problem. FG Intellektik. TU Darmstadt, Darmstadt. Germany.
[26] Dong X, Huang H, Chen P. An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. Comput Oper Res 2009;36(5):1664–9.
[27] Framinan JM, Leisten R, Ruiz-Usano R. Comparison of heuristics for flowtime minimisation in permutation flowshops. Comput Oper Res 2005;32(5):1237–54.
[28] Ruiz R, Stützle T. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European J Oper Res 2008;187(3):1143–59.
[29] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European J Oper Res 2007;177(3):2033–49.
[30] Osman I, Potts C. Simulated annealing for permutation flow-shop scheduling. Omega 1989;17(6):551–7.
[31] Taillard E. Some efficient heuristic methods for the flow shop sequencing problem. European J Oper Res 1990;47(1):65–74.
[32] Pan Q, Tasgetiren M, Liang Y. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Comput Ind Eng 2008;55(4):795–816.
[33] Taillard E. Benchmarks for basic scheduling problems. European J Oper Res 1993;64(2):278–85.