Journal of Intelligent Manufacturing (2020) 31:1051–1078
https://doi.org/10.1007/s10845-019-01518-4

Time/sequence-dependent scheduling: the design and evaluation of a general purpose tabu-based adaptive large neighbourhood search algorithm

Lei He[1,2] · Mathijs de Weerdt[2] · Neil Yorke-Smith[2]

Received: 21 May 2019 / Accepted: 2 December 2019 / Published online: 20 December 2019

Abstract
In intelligent manufacturing, it is important to schedule orders from customers efficiently. Make-to-order companies may have to reject or postpone orders when the production capacity does not meet the demand. Many such real-world scheduling problems are characterised by processing times being dependent on the start time (time dependency) or on the preceding orders (sequence dependency), and typically have an earliest and latest possible start time. We introduce and analyze four algorithmic ideas for this class of time/sequence-dependent over-subscribed scheduling problems with time windows: a novel hybridization of adaptive large neighbourhood search (ALNS) and tabu search (TS), a new randomization strategy for neighbourhood operators, a partial sequence dominance heuristic, and a fast insertion strategy. Through factor analysis, we demonstrate the performance of these new algorithmic features on problem domains with varying properties. Evaluation of the resulting general purpose algorithm on three domains—an order acceptance and scheduling problem, a real-world multi-orbit agile Earth observation satellite scheduling problem, and a time-dependent orienteering problem with time windows—shows that our hybrid algorithm robustly outperforms general algorithms including a mixed integer programming method, a constraint programming method, recent state-of-the-art problem-dependent meta-heuristic methods, and a two-stage hybridization of ALNS and TS.

Keywords Time/sequence-dependent scheduling · Time windows · Hybrid algorithms · Adaptive large neighbourhood search · Tabu search · Factor analysis

Introduction

Manufacturing planning is an essential element of supply chain management (Jacobs et al. 2010). In intelligent manufacturing, efficient scheduling of orders from customers plays an important role to maximise productivity and profit. For many make-to-order companies, orders have to be rejected or postponed when the company's capacity cannot meet the demand. This makes the problem become an over-subscribed scheduling problem, consisting of simultaneously selecting a subset of jobs to be processed as well as the associated schedule. This problem is important because it represents a class of real-world problems including the Earth observation satellite scheduling problem (Augenstein et al. 2016; Akturk and KiliÇ 1999), the order acceptance and scheduling problem (Oğuz et al. 2010; Wang et al. 2017), the orienteering problem (Verbeeck et al. 2017), and selective maintenance scheduling (Duan et al. 2018). Many real-world instances in this class have time windows (e.g., from the time when the factory receives the raw material to the user-specified deadline (Rebai et al. 2012)) and time/sequence-dependent setup times (e.g., the time to prepare next batches of products in an intelligent manufacturing system): the scheduled start time of each job must be in its time window, and the setup time between every two jobs depends on the specific pair of jobs (Mirsanei et al. 2011) or their scheduled start times (Dong et al. 2014).

✉ Lei He
l.he@tudelft.nl

Mathijs de Weerdt
M.M.deWeerdt@tudelft.nl

Neil Yorke-Smith
N.Yorke-Smith@tudelft.nl

1 College of Systems Engineering, National University of Defense Technology, Changsha 410073, China

2 Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands

For the varying problem instances in this class with time windows and time/sequence-dependent setup times, general approaches such as mixed integer programming do not perform very well (Cesaret et al. 2012; Verbeeck et al. 2017; Liu et al. 2017). There are also problem-specific methods that use highly specialised subroutines (Liu et al. 2017; Silva et al. 2018; Poggi et al. 2010). The design and implementation of such methods requires expertise and effort. Our goal is to provide an intermediate approach: a generic algorithm that performs well on different problem variants.

The major contributions of this paper are as follows:

1. We propose a general algorithm with four algorithmic features: a hybridization of adaptive large neighbourhood search (ALNS) and tabu search (TS), a new randomization strategy for neighbourhood operators, a partial sequence dominance heuristic, and a fast insertion strategy.
2. Through factor analysis, we show the robust performance of the new algorithmic features, and we derive useful conclusions on how to use tabu search for problem domains with different properties.
3. By means of our algorithm, ALNS/TPF, we illustrate a methodology for solving a larger class of problems with high efficiency, without specifying a specialised method for each domain.
4. There exist a limited number of publicly available source codes and benchmark instances for this class of problems. We publish the source codes of the algorithms and the benchmark instances to enable future studies.[1]

Two early versions of this hybrid algorithm have been introduced in our preliminary work (He et al. 2018, 2019). We first applied the algorithm to the time-dependent agile Earth observation satellite scheduling problem (He et al. 2018) and then generalized the algorithm to the order acceptance and scheduling problem (He et al. 2019). The method in this paper is a further novel extension of the previous methods. The tabu heuristic, the partial sequence dominance heuristic and the insertion strategy have been upgraded and more neighbourhood operators are introduced. A more complete factor analysis of individual contributions of the new algorithmic features and the correlation between the performances of the algorithm and the properties of problem instances are included. Besides, in this paper we evaluate the proposed algorithm on more problem domains.

The remainder of this article is summarized as follows: “Background” section provides background information; “ALNS/TPF: tabu-based ALNS algorithm” section introduces the ALNS/TPF algorithm; “Algorithmic analysis” section introduces the algorithmic analysis of the proposed algorithm. In “Comparison with state-of-the-art algorithms” section, the algorithm is compared with state-of-the-art methods on three problem domains; the conclusions are summarized in “Conclusions” section.

Background

This section describes the mathematical formulation of the problem, gives a review of approaches to instance domains, and lastly describes the standard ALNS and TS algorithms.

Mathematical formulation

Here we present a high-level abstraction of the common aspects of instances of the problem class. This model is proposed based on the models of different problems from Oğuz et al. (2010); He et al. (2018); Verbeeck et al. (2017). Detailed specific constraints of problems are introduced in later sections.

Consider a set of jobs $O = \{o_0, o_1, \ldots, o_n, o_{n+1}\}$ that can be potentially scheduled, where o_o and o_{n+1} are two dummy jobs representing the start and end job in the solution sequence respectively. The order of other jobs is not fixed. Each job has a revenue r_i, a processing duration time d_i, and a time window $[b_i, e_i]$. Let x_i be a binary variable representing whether job o_i is selected, y_{ij} be a binary variable representing whether job o_i directly precedes job o_j, p_i be a decision variable representing the start time of o_i, and M be a sufficient large constant. The problem can be formulated as a mixed integer linear programming (MILP) model:

$$\max \sum_{i=1}^{n} x_i r_i \tag{1}$$

subject to

$$p_i + d_i + s_{ij} + (y_{ij} - 1)M \leq p_j \forall i \in \{0, \ldots, n\}, j \in \{1, \ldots, n+1\}, i \neq j \tag{2}$$

$$\sum_{j=1, i\neq j}^{n+1} y_{ij} = x_i \quad \forall i \in \{0, \ldots, n\} \tag{3}$$

$$\sum_{j=0, i\neq j}^{n} y_{ji} = x_i \quad \forall i \in \{1, \ldots, n+1\} \tag{4}$$

$$b_i \leq p_i + (1 - x_i)M \quad \forall i \in \{0, \ldots, n+1\} \tag{5}$$

$$p_i \leq e_i + (1 - x_i)M \quad \forall i \in \{0, \ldots, n+1\} \tag{6}$$

$$x_0 = 1, x_{n+1} = 1 \tag{7}$$

$$x_i \in \{0, 1\} \quad \forall i \in \{1, \ldots, n\} \tag{8}$$

$$y_{ij} \in \{0, 1\} \quad \forall i \in \{0, \ldots, n\}, j \in \{1, \ldots, n+1\}, i \neq j \tag{9}$$

[1] The source code of our algorithm is available https://doi.org/10.4121/uuid:3a23b216-3762-4a61-ba2c-d3df6dc53268, and the datasets used in this paper are available https://doi.org/10.4121/uuid:1a4e5895-7dae-4b6a-9315-a9e8cb463d73.

The objective function (1) maximizes the total revenue of scheduled jobs. Constraints (2) restrict the time between every two adjacent jobs should be long enough for the setup, where s_{ij} is the setup time between jobs o_i and o_j. The value of s_{ij} depends on i and j for the sequence-dependent setup time case (e.g., a table of setup times for all pairs of o_i and o_j) and depends on p_i and p_j for the time-dependent setup time case (e.g., a function of p_i and p_j or a table of setup times for all pairs of p_i and p_j). Constraints (3) and (4) restrict that each accepted job is succeeded by only one job and precedes only one job. Constraints (5) and (6) define the earliest and latest start time of a job. Constraints (7) restrict that the two dummy jobs, the start and end jobs must be selected. Constraints (8) and (9) define the domains of the variables x_i and y_{ij} respectively F.

Domain instances

Due to the large number of problem variants and solution approaches, the reader is referred to Slotnick (2011) and Gunawan et al. (2016) for comprehensive surveys on this class of over-subscribed scheduling problems with time windows and time/sequence-dependent setup times.

The order acceptance and scheduling problem (OAS) is a typical representative of this problem class. The problem happens for instance when a manufacturing system does not have the capacity to meet demand. Oğuz et al. (2010) proposed the OAS problem with a penalty for late completion, time windows and sequence-dependent setup times. The problem was approached by MIP (Cesaret et al. 2012), TS (Cesaret et al. 2012), genetic algorithm (Nguyen et al. 2015; Chen et al. 2014; Chaurasia and Singh 2017), artificial bee colony algorithm (Lin and Ying 2013; Chaurasia and Kim 2019), hyper-heuristic based methods (Nguyen 2016) and iterated local search (Silva et al. 2018). Recently, Silva et al. (2018) used Lagrangian relaxation and column generation to find tight upper bounds of problem instances.

The Earth observation satellite scheduling (EOSS) problem is another important problem instance of this problem class. In a limited scheduling horizon, the satellite can usually observe only a subset of the user-specified jobs. Besides, the transition time for the satellite to change its observation angle between two adjacent jobs is time/sequence-dependent (Lemaître et al. 2002). The time windows in the EOSS are usually much shorter than those in the OAS problem (Liu et al. 2017). Akturk and KiliÇ (1999) studied the observation scheduling problem of Hubble Space Telescope, in which the setup times jobs are sequence-dependent. A dispatching rule considering weighted shortest processing time, setup times and nearest neighbour was proposed. Liu et al. (2017) and Peng et al. (2018) studied the agile satellite observation scheduling problem with time-dependent setup times. Liu et al. (2017) proposed a mixed integer programming (MIP) method and an ALNS algorithm, where they also integrated ALNS with an insertion algorithm considering time-dependency by introducing forward/backward time slacks. Peng et al. (2018) proposed an iterated local search (ILS) algorithm. They further calculated the minimal transition time, the neighbours and earliest/latest start time of each job to accelerate the insertion.

The selected travelling salesman problem (STSP), also referred to as the orienteering problem (OP), defines a problem where the salesman visits a subset of cities to maximize the total collected reward within a limited travel time. Compared with the above two problem variants, the travelling time (i.e., setup time) between cities is much longer, and there exists a stronger correlation among cities depending on the locations of them. Existing methods include genetic algorithms (Abbaspour and Samadzadegan 2011), iterated local search (Garcia et al. 2013), MIP (Verbeeck et al. 2017), and act colony optimization (ACO) (Verbeeck et al. 2017). Verbeeck et al. (2017) proposed the time-dependent OP with time windows (TDOPTW). In their ACO metaheuristic, they used the max shift to fast determine the feasibility of an insertion, which is similar to the ideas of Peng et al. (2018).

Despite all this work, there is no method capable of finding good solutions to diverse real life instances within the available solving time. In this paper, we propose a general method which performs well on all above problem instances in terms of solution quality and running time.

Standard ALNS and TS

Adaptive large neighbourhood search (ALNS) is one of the most promising approaches for scheduling problems (Ropke and Pisinger 2006). Since ALNS is less sensitive to the initial solution than general local search (Demir et al. 2012), the initial solution is usually generated by a simple greedy heuristic. The solution is updated by removing some jobs from the current solution with the removal operators and inserting some jobs back to the solution with the insertion operators. Multiple removal and insertion operators can be defined according to the problem characteristics. In each iteration, an adaptive mechanism based on a roulette wheel is used to select a pair of removal and insertion operators according to their weights. The weight of the operator w_i is updated according to its accumulated score π_i in the previous iterations, $w_i = (1 - \lambda)w_i + \lambda\pi_i / \sum_j \pi_j$, where λ is a constant weight update parameter in $[0, 1]$. When a new solution is generated, it is accepted if it is better than the current solution, otherwise its acceptance is determined by a simulated annealing (SA) criterion with probability: $\rho = \exp\left(\frac{100}{T}\left(\frac{f(S')-f(S)}{f(S)}\right)\right)$, where $f(S)$ and $f(S')$ are the reward of current solution S and new solution S' respectively, and T is the temperature.

Tabu search (TS) was first proposed by Glover (1986). In TS, some simple local search moves are defined to update the solutions. To prevent short-term cycling, recently visited solutions or recently used local search moves are stored in a tabu list, and may not be tried again while in the list. The forbidden solutions or moves lose their tabu status after a certain short time.

Žulj et al. (2018) proposed the first hybridization of ALNS and TS and found that a simple hybridization of these two metaheuristics outperformed both the standalone methods for the order batching problem. Their method combines the diversification capabilities of ALNS and the intensification capabilities of TS. It uses ALNS to search for better solutions and, if a certain number of ALNS iterations have passed, invokes TS. Thus ALNS and TS are alternated in a simple two-stage manner.

ALNS/TPF: tabu-based ALNS algorithm

In this section, we introduce four algorithmic features in our approach: a more advanced tabu search hybridization (TS), randomized heuristic neighbourhood operators, partial sequence dominance (PSD), and a fast insertion algorithm (FI) considering time/sequence-dependent setup times. The resulting algorithm, called ALNS/TPF, is shown as Algorithm 1.

Algorithm 1: Overview of ALNS/TPF

```
Input: Candidate job set: $O$
Output: Best solution: $S^*$
1  function ALNSTPF($O$)
2      $S^I \leftarrow$ GenerateInitialSolution($O$)[2]
3      Set $S^I$ as the current and the best solution: $S \leftarrow S^I$, $S^* \leftarrow S^I$
4      repeat
5          Choose removal, insertion operators $D_i$, $R_i$ based on weights
6          $S' \leftarrow R_i(D_i(S))$
7          $S_c \leftarrow$ GenerateCompoundSolution($S$, $S'$)
8          if $f(S_c) > f(S') \wedge S_c \neq S$ then
9              $S' \leftarrow S_c$
10         if $f(S') > f(S) \vee$ SA accepts $S'$ then
11             $S \leftarrow S'$
12         if $f(S) > f(S^*)$ then
13             $S^* \leftarrow S$
14         Update removal and insertion tabu attributes of all jobs
15         Update scores and weights of operators
16     until Terminal condition is met
17     return $S^*$
```

Tabu search hybridization

Although ALNS has been widely successful (Thomas and Schaus 2018), a main drawback is that its search efficiency can founder due to re-visiting recent solutions. Although in standard ALNS, large portions of the solution are destroyed and rebuilt, and it would seem less likely a previous solution would be visited than other local search algorithms with simpler moves, the probability of short cycling still exists and could be higher when 'good' jobs are identified and selected a lot by the algorithm. Sometimes although jobs are inserted in different orders by different insertion operators, the resulting solutions are the same. This motivates us to propose the hybrid algorithm of ALNS and TS.

As noted earlier, Žulj et al. (2018) proposed the first hybridization of ALNS and TS. However, since ALNS and TS are used in separate stages, this hybridization does not change the short-term cycling nature of ALNS. In contrast, we propose a tight integration of ALNS with TS, which includes three types of tabu heuristics: the removal tabu, the insertion tabu and the instant tabu.

Removal tabu For each job, we declare a removal tabu attribute. If a new solution is accepted, the removal of newly-inserted jobs is forbidden for the following θ iterations; otherwise, the removal of the jobs removed in this iteration is forbidden for the following θ iterations.

Insert tabu For each job, we declare an insertion tabu attribute. If a new solution is accepted, the reinsertion of newly-removed jobs is forbidden for the following θ iterations. Here we do not forbid the reinsertion of the jobs inserted in this iteration if the new solution is rejected as we do in the removal tabu, because in this case the removal tabu is enough to prevent visiting a recent solution. Preventing the reinsertion of jobs can lead to a lower solution quality, because this problem attempts to schedule as many jobs as possible.

Instant tabu this tabu heuristic is used to prevent producing a new solution same as the current solution. When inserting a job to the destroyed solution by the insertion algorithm introduced in "Fast insertion algorithm" section, if the resulting solution assuming the candidate job is inserted is same as the current solution, this insertion will be forbidden.

All the three heuristics are used in the solution update process (Algorithm 1, line 6). The removal and tabu attributes are updated in Algorithm 1, line 14. For the values of the tabu attribute θ, we follow Cordeau and Laporte (2005) and set it to a random value in $[0, \sqrt{n/2}]$, where n is the number of jobs. Since the instant tabu is used only once in one iteration, it does not have a tabu attribute.

We compare the three tabu types and the two ALNS–TS hybridizations in "Tabu search" section.

[2] We sort the jobs by an ascending order of begin times of their time windows and we attempt to start each job as early as possible under all the constraints. Jobs that cannot be inserted are given up.

Randomized generic neighbourhood operators

ALNS uses multiple neighbourhood operators to update solutions (Algorithm 1, lines 5–6). To make sure that the algorithm is efficient for a diverse range of problem instances with varying properties, we use the following ten generic removal operators and seven generic insertion operators, and introduce a simple but effective randomization strategy to diverse the search. These operators are adapted from Pisinger and Ropke (2007) and Demir et al. (2012) to fit our problem, while the randomization strategy is new. The randomization strategy is similar to the idea of adding noise to the insertion operators by Pisinger and Ropke (2007). However, we extend this idea to both removal and insertion operators.

The ten removal operators are:

1. *Random removal (RR)* p_d jobs are removed randomly from the current solution. The worst-case time complexity of this operator is $O(n)$;
2. *Min revenue removal (MRR)* p_d jobs with lower revenue are removed. The worst-case time complexity of this operator is $O(n^2)$;
3. *Min unit revenue removal (MURR)* p_d jobs with lower unit revenue are removed. The unit revenue is the job's revenue divided by its processing time. The worst-case time complexity of this operator is $O(n^2)$;
4. *Max setup time removal (MSTR)* p_d jobs with longer setup time are removed. The worst-case time complexity of this operator is $O(n^2)$;
5. *Max opportunity removal (MOR)* For the problems where jobs have multiple time windows, p_d jobs with more time windows are removed. The rationale of this operator is that these jobs can be scheduled in other time windows easily. The worst-case time complexity of this operator is $O(n^2)$;
6. *Max conflict removal (MCR)* p_d jobs with higher conflict degree are removed. The conflict degree of job o_i is calculated by comparing its time window with those of other jobs: $(\sum_{o_j\in O, i\neq j} TimeSpan(o_i, o_j))/(e_i + d_i - b_i)$, where the function $TimeSpan$ calculates the time span that the time windows of two jobs overlap with each other. The worst-case time complexity of this operator is $O(n^2)$;
7. *Worst route removal (WRR)* This removal operator evaluates a small sequence of p_d jobs in the current solution by calculating the unit revenue of the sequence, which is the revenue of the p_d jobs divided by the time period of the sequence. The worst sequence of p_d jobs is removed. The worst-case time complexity of this operator is $O(n)$;
8. *Max wait removal (MWR)* p_d jobs with higher waiting time are removed. Let o_i be the immediate precursor of o_j. The waiting time of job o_j is calculated by $p_j - (p_i + d_i) - s_{ij}$. This operator reduces the time wasted by waiting for the release time of a job. The worst-case time complexity of this operator is $O(n^2)$;
9. *Historical setup removal (HSR)* This removal operator keeps track of the shortest setup time of each job. p_d jobs with setup times that have larger distance to their best setup times are removed. The worst-case time complexity of this operator is $O(n^2)$;
10. *Historical unit revenue removal (HURR)* This removal operator is similar to the min unit revenue removal operator. The main difference is that when calculating the unit revenue, the setup times before and after the job are considered. Let o_i, o_j, o_k be a sequence in the solution. The unit revenue of o_j in the sequence is calculated by $r_j/(p_k - (p_i + d_i))$. This operator keeps track of the best unit revenue each job in the sequence. p_d jobs with lower unit revenue in the sequence are removed. The worst-case time complexity of this operator is $O(n^2)$.

All the jobs removed by the above removal operators and other unscheduled operators are stored in a candidate job bank. Seven insertion operators are used to sort the jobs in the job bank and insert them one by one from the top of the list back to the solution. The insertion method is introduced in "Fast insertion algorithm" section. The insertion stops until no jobs can be inserted, or the total revenue of the repaired solution plus the total revenue in the job bank (i.e., the upper bound of the new solution in this iteration) is lower than the current solution.

The seven insertion operators are:

1. *Max revenue insertion (MRI)* The jobs in the job bank are sorted according to descending revenue. The worst-case time complexity of this operator is $O(n^2)$;
2. *Max unit revenue insertion (MURI)* The jobs in the job bank are sorted according to descending unit revenue. The worst-case time complexity of this operator is $O(n^2)$;
3. *Min setup time insertion (MSTI)* Due to the time/sequence-dependency, the accurate setup time cannot be calculated until the job is inserted in the solution; therefore for this operator, the average setup time of jobs is calculated and used to rank the jobs. The worst-case time complexity of this operator is $O(n^2)$;
4. *Min opportunity insertion (MOI)* For the problems where jobs have multiple time windows, the jobs are sorted according to ascending numbers of time windows. The jobs with fewer time windows are considered first. The worst-case time complexity of this operator is $O(n^2)$;
5. *Min conflict insertion (MCI)* The jobs in the job bank are sorted according to ascending conflict degree. The worst-case time complexity of this operator is $O(n^2)$;
6. *Historical unit revenue insertion (HURI)* Opposite to the historical unit revenue removal, this insertion operator sorts the jobs according to descending historical unit rev-

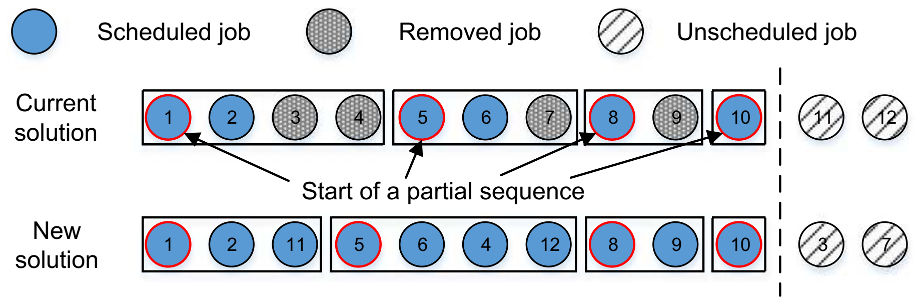


Fig. 1 An example of the partial sequence

enue in the sequence. The worst-case time complexity of this operator is $O(n^2)$;

7. *Min distance insertion (MDI)* This operator tries to insert the jobs that are closest to the jobs in the solution. First, the shortest distance to the current solution (i.e., the shortest setup time to jobs in the current solution) of each candidate job is calculated. Then the jobs are sorted according to ascending nearest distance. The worst-case time complexity of this operator is $O(n^2)$;

After selecting the removal/insertion operators by the roulette wheel mentioned in "Standard ALNS and TS" section, standard ALNS ranks the jobs according to the heuristic values of the operators: e.g., for the min revenue removal operator, the revenue is regarded as the heuristic value h, the jobs are ranked in an ascending order of h, and the jobs on the top of the list are removed. In order to diverse the search, we add randomness to the heuristic values of selected certain operators: $h \leftarrow h \times (1 + r)$, where r is a random value in [0, 1]. Here we differ from the common approach of selecting jobs randomly according to a probability that depends on h, because we want to add limited randomness and while keeping emphasis on following h. Our approach here thus introduces a random component without neglecting the heuristic.

When removing and inserting a job, if the job is forbidden to be removed or inserted, the operator skips it and check the next one. However, through an aspiration criterion, the tabu status of a job can be revoked if the number of removed jobs is smaller than p_d for the removal tabu, and all other jobs that are not forbidden by tabu have been tested for insertion and there is still open space in the solution for the candidate job for the insertion tabu. This aspiration criterion ensures that there are enough jobs to remove and insert.

Partial sequence dominance

Besides solution cycling, a further drawback of ALNS is that it evaluates a new solution depending on the quality of the whole solution sequence. Hence, during the search process, solutions with some good parts (i.e., parts of the solution might have higher objective value while consume less time)

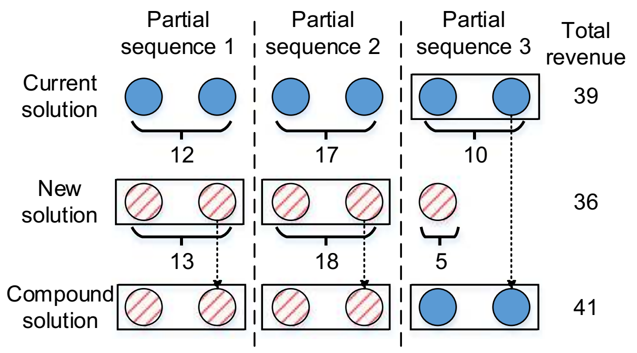


Fig. 2 An example of partial sequence dominance

are rejected due to the low quality of the whole sequence—thus neglecting potentially valuable in-process information. Due to the time-dependency and sequence-dependency, the quality of a solution is influenced significantly by these small parts.

Inspired by genetic algorithms, we propose the partial sequence dominance (PSD) heuristic (Algorithm 1, lines 7–9): we construct a compound solution which keeps better partial sequences from the new solution and the current solution. The partial sequence is defined in Definition 1. An example of the definition of the partial sequence is shown in Fig. 1. The quality of the partial sequence is evaluated by unit revenue: the objective function value divided by the total period of the partial sequence. If the compound solution is better than the new solution and different from the current solution, it replaces the new solution. Figure 2 shows one example of PSD. In standard ALNS, the new solution is given up. However, partial sequence 1 and partial sequence 2 of the new solution are better than the current solution. So according to PSD, we keep partial sequence 1 and partial sequence 2 of the new solution and partial sequence 3 of the current solution, and we get the compound solution, which is better than the current solution.

Definition 1 A partial sequence is a sequence of jobs, starting with the first job of each continuous sequence of unchanged jobs.

The detailed process of constructing a compound solution from two solutions is shown in Algorithm 2. When a

Algorithm 2: Process of constructing a compound solution

```
Input: Current solution $S$, New solution $S'$
Output: Compound solution $S_c$
1  function GenerateCompoundSoluion($S$, $S'$)
2      $S_c \leftarrow \emptyset$
3      Partition $S$ and $S'$ into two sets of partial sequences: $S_p$ and $S'_p$
4      for $i \leftarrow 1$ to $\#S_p$ do
5          $U \leftarrow$ Calculate the unit revenue of the $i^{th}$ partial sequence in $S_p$
6          $U' \leftarrow$ Calculate the unit revenue of the $i^{th}$ partial sequence in $S'_p$
7          if $U > U'$ then
8              Add $S_p$ into $S_c$
9          else
10             Add $S'_p$ into $S_c$
11     Update jobs in $S_c$ to remove duplicate ones and start each job as early as possible
12     return $S_c$
```

new solution is produced, we partition it into multiple partial sequences. We compare each partial sequence of the new solution with the corresponding partial sequence of the current solution. The partial sequence with higher unit revenue is stored in the compound solution.

A challenge for the PSD heuristic is that one job can appear in different partial sequences of the current solution and the new solution. Thus one job might be processed twice in the compound solution. To maintain the feasibility, all the repetitive jobs in the compound solution are removed. Then the start times of jobs in the solution are updated so that all jobs in the compound solution start as early as possible.

Fast insertion algorithm

The last algorithmic feature of the algorithm is a fast insertion algorithm, which is used to insert jobs back to the solution by the insertion operators. We first proposed this method for a scheduling problem with sequence-dependent setup times in a previous paper (He et al 2019). To make sure the algorithm is complete and clear to the reader, we give a short description of the basic ideas of the insertion algorithm, and then we generalize the fast insertion algorithm to the time-dependent case.

The fast insertion algorithm uses the following two steps to insert a candidate job. First, the feasibility of the insertion at all positions of the solution is evaluated rapidly by a concept called time slack. The time slack is the time a job can be postponed before the solution becomes infeasible. Because all the jobs in the current solution are started as early as possible, the candidate job can be inserted before a job if the time that the following job needs to be postponed is smaller than its time slack. Note that in this step, all positions at which if the candidate job is inserted and the resulting solution is the same as the current solution are also forbidden because of the new instant tabu. Then in the second step, the best possible position is selected to insert the task. We select the position which increases the least setup time and adds the least penalty to the postponed jobs in the current solution. More details of the fast insertion algorithm are given in Appendix A.

For the problems put forward in this paper, which have time-dependent setup times, there are two challenges for the above insertion algorithm: (1) how to calculate the latest start time of a job so that we can calculate its time slack? This is more complicated than the sequence-dependent case because the setup time changes with the start time; (2) how to calculate the increased setup time? We cannot know the exact increased setup time unless we calculate the start times of all the postponed jobs and the changed setup times, which is too complex. For challenge 1, we can calculate the latest start time of job o_i by solving the equation: $p_i^{Late} = p_j^{Late} - Setup(p_i^{Late}, p_j^{Late})$, where p_j^{Late} is the latest start time of the following job o_j and $Setup(time_1, time_2)$ is a function to calculate the time-dependent setup times. However, for the problem where such an analytic function does not exist, such as the agile Earth observation satellite scheduling problem where the setup time is given by a table, the dichotomy method by Liu et al. (2017) can be used to find the latest start time of a job given the latest start time of its successor. For challenge 2, we use the increased setup time of the insertion position (e.g., if we insert candidate job o_c between o_i and o_j, the increased setup time of the insertion position is $s_{ic} + s_{cj} - s_{ij}$) to approximate the actual increased setup time. On a time-dependent agile Earth observation satellites scheduling problem benchmark (Liu et al. 2017) and a time-dependent orienteering problem benchmark (Verbeeck et al. 2017), the average error of this approximation of the setup time is 12.43% and 16.12% respectively.

Algorithmic analysis

The proposed ALNS/TPF algorithm consists of four features (i.e., tabu search, randomized generic operators, partial sequence dominance and fast insertion algorithm). In this section, we study how the discussed algorithmic features perform on different problem domains in order to understand how and when to use them depending on the domain properties.

We first introduce three problem domains and the datasets we used to test our algorithm. Then, we analyze the algorithmic features individually. For each new feature, we compare the algorithm without this feature against the full algorithm to understand its performance on the three problem domains. For each removed algorithmic feature, we calculate the decrease in solution quality (the percentage of the decrease of the objective function value compared with the full algorithm) and the increase in running time (the percentage of the increase of the running time compared with the full algorithm). A higher decrease in solution quality and increase in running time shows that the corresponding algorithmic feature performs better.

Problem datasets

We choose three representative problem domains, the order acceptance and scheduling (OAS) problem with time windows and sequence-dependent setup times, the agile Earth observation satellite scheduling problem (AEOSS), and the time-dependent orienteering problem with time windows (TDOPTW).

We consider the OAS problem from Cesaret et al. (2012). In this problem, the setup time between two jobs is sequence-dependent. Besides, the setup of a job can only start after its release, which makes the setup constraint become: $\max\{b_j, p_i + d_i\} + s_{ij} + (y_{ij} - 1)M \leq p_j$. This problem is also special because of the tardiness penalty: if a job o_i starts after its due time $\bar{e_i}$, it receives penalty $\omega_i T_i$ on its revenue, where ω_i is the penalty weight and T_i is the tardiness, $T_i = \max\{p_i - \bar{e_i}, 0\}$. Therefore, the objective function becomes $\max \sum_{i=1}^{n} x_i(r_i - \omega_i T_i)$. We use the benchmark dataset published by Cesaret et al. (2012). Three main parameters were used to generate these instances. The first parameter is the number of jobs $n = 10, 15, 20, 25, 50, 100$ and we only test the larger instances with 25, 50 and 100 jobs; the second parameter, τ, influences the length of time windows: when τ is larger, the time windows are smaller; the third parameter, R, influences the range of the end time and the due time of time windows: when R is larger, the deadlines spread broadly, so the overlap of time windows gets smaller. Both τ and R have five values: 0.1, 0.3, 0.5, 0.7, 0.9. Ten random instances are generated for each parameter setting, giving 750 instances in total.

We consider the AEOSS problem from Liu et al. (2017). The transition time between two adjacent jobs o_i and o_j is calculated by: $s_{ij} = t + |A_i(p_i) - A_j(p_j)|/v$, where t is constant time for stabilizing the satellite, function A_i is represented by a table returning the angle of the satellite for observing o_i at the start time of the observation, and v is the satellite transition velocity. The scheduling horizon is 24 hours, which means there are multiple time windows for each observation job. Let w_i be the total number of time windows for job i, and k be the k-th time window of job i. The objective function becomes $\max \sum_{i=1}^{n} \sum_{k=1}^{w_i} x_{ik} r_i$, where $x_{ik} \in \{0, 1\}$ is a decision variable, meaning whether the k-th time window is selected. Since each job can only be processed once, an additional constraint should be considered: $\sum_{k=1}^{w_i} x_{ik} \leq 1 \ \forall i$. The original dataset of AEOSS problem was not published, we generate the dataset following the configuration from Liu et al. (2017). The jobs are generated according to a uniform random distribution over two geographical regions: China and the whole world. For the Chinese area distribution mode, fifteen instances are designed and the number of jobs contained in these instances changes from 50 to 400 with an increment of 25. For the worldwide distribution mode, twelve instances are designed and the number of jobs contained in these instances changes from 50 to 600 with an increment of 50.

We consider the TDOPTW problem from Verbeeck et al. (2017). In this problem, the scheduling horizon is divided into several 15-minute time slots and for each time slot, a travelling velocity starting in this time slot is given. Therefore, the travelling time (i.e., setup time) is given by a piecewise linear function, dependent on the end time of the preceding job: $s_{ij} = \mu(p_i + d_i) + \upsilon$, where μ and υ are two parameters, and the values of them depend on the time slot of the end time. This dataset was generated by Verbeeck et al. (2017). It has instances with job number 20, 50 and 100. For each set of instances with the same number of jobs, there are four variations in scheduling horizon t_{max} (8, 10, 12 and 14h) and three random variations in the lengths of time windows (small, medium and large). Therefore, in total there are 36 instances.

Although these are sequence/time-dependent scheduling problems with time windows, there are in fact some differences. Table 1 compares all the differences of the three problem domains we have noticed.

1. The number of time windows: in the AEOSS problem, each job has multiple time windows, while in the other two problems, each job has only one time window;
2. The length of time windows: the length of time windows is evaluated by the average length of time windows divided by the length of the scheduling horizon. It is clear that the AEOSS problem has very short time windows, the TDOPTW problem has medium time windows, and the OAS problem has time windows with more varying lengths;
3. Congestion ratio: this value is calculated by adding up the processing time and the average setup time of each job, divided by the horizon, to evaluate how congested the instance is. In the three problems, the TDOPTW problem is the most congested, and the AEOSS problem is the least;
4. Earliest start time: the OAS problem is quite special for its start time. The setup time of a job can only start after its release (i.e., the start time of its time window);
5. Late penalty: the OAS problem has a late penalty on the revenue of a job if it starts after its due time;

Table 1 Comparison of the three problem domains

Difference	OAS	AEOSS	TDOPTW
Number of time windows	Single	Multiple	Single
Length of time windows	0.09–0.87	0.0007–0.0009	0.22–0.38
Congestion ratio	0.93–1.66	0.03–0.43	2.42–22.09
Earliest start time of t_j	$\max\{b_j, c_i\} + s_{ij}$	$\max\{b_j, c_i + s_{ij}\}$	$\max\{b_j, c_i + s_{ij}\}$
Late penalty	Yes	No	No
Setup time dependency	i and j	p_i and p_j	p_i
Length of setup time	0.43–0.84	1.36–1.47	4.26–5.49
Triangle inequality	No	Yes	Yes
Conflict of jobs	3.82–96.83	0.4–24.84	8.72–75.61
Correlation among jobs' locations	Weak	Strong	Strong

Job o_i is the immediate precursor of job o_j. Let p_i and p_j be the start time, and c_i and c_j be the completion time of o_i and o_j respectively

6. Setup time: the setup time of the OAS problem is sequence-dependent, while the setup time of the other two is time-dependent;
7. The length of setup time: the length of setup time is evaluated by the average of setup time divided by the processing time of a job. Among the three problems, the TDOPTW problem has the longest setup times, and the OAS problem has the shortest setup times;
8. Triangle inequality: the setup times of the OAS problem are generated randomly, which means it does not follow the triangle inequality. The setup time from job o_i to job o_j might be longer than the time for the sequence o_i, o_k, o_j. As a result, when removing a job for the OAS problem, the start time of the job after the removed job should be checked to maintain the feasibility;
9. Conflict of jobs: we use the formula in the max conflict removal operator to calculate the conflict of jobs. The TDOPTW problem has the highest conflict degree, and the AEOSS has the least;
10. Correlation among jobs: since the setup times of the OAS problem are generated randomly, there exists little correlation if two jobs are not adjacent in the solution. However, for the AEOSS and the TDOPTW problems, the setup times are calculated based on real locations of the jobs, and jobs influence each other if they are close. We believe such property makes the location-related operators such as the worst route removal and the min distance insertion work well on these two problems.

Tabu search

In this section we aim to answer the following questions: how the three tabu types perform on problem domains with varying properties; whether TS helps to avoid recent solutions; whether TS can improve the performance of ALNS on the three problem domains; whether our tight hybridization is better than the two-stage hybridization? These studies help us to understand the performance of the tabu search heuristics so that we can use them efficiently.

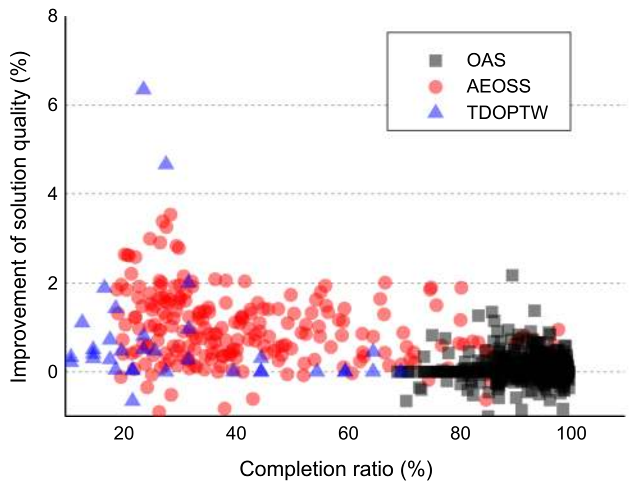
Performance of three tabu types

In "Tabu search hybridization" section, we introduced three types of tabu: the insertion tabu, the removal tabu and the instant tabu. The first type is more common in over-subscribed problems (Bianchessi et al. 2007; Cordeau et al. 2001; Cordeau and Laporte 2005; Rogers et al. 2006; Prins et al. 2007). The only removal tabu we found in the literature is from Rogers et al. (2006). However, their strategy is for updating an infeasible solution by inserting jobs first and then removing jobs. Therefore, their removal tabu is used in the intermediate solution (i.e., the infeasible solution) while ours is used in the repaired solution (i.e., the feasible solution). The third tabu type, instant tabu, is new according to our knowledge.

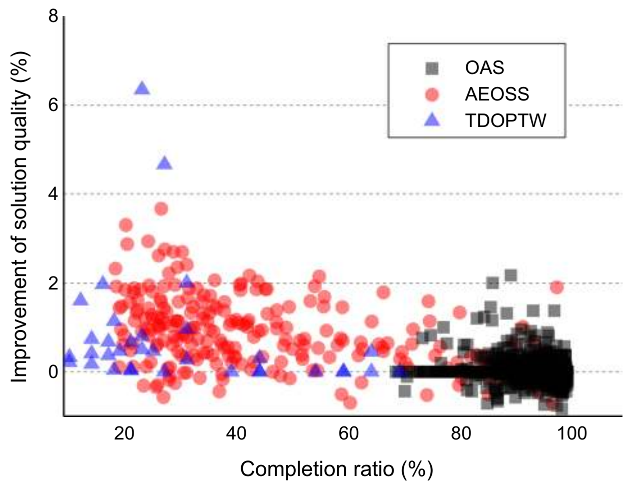
In this section, we study how the three tabu types perform on problem domains with varying properties. We compare the standard ALNS with each tabu type against the standard ALNS without any new algorithmic features except the fast insertion algorithm, which is necessary in the insertion process of the ALNS algorithm. We find that the performance of the removal tabu and insertion tabu correlates with the proportion of jobs that can be fulfilled, which we call the completion ratio,[3] and the performance of the instant tabu correlates with the number of jobs in the solution.

According to Fig. 3, the removal and insertion tabu works better when the completion ratio is low. Since all the OAS

[3] The completion ratio of the OAS problem and the AEOSS problem is high and the completion ratio of the TDOPTW problem is low. To cover a larger range of completion ratio, besides the three problem datasets that we have, we further generate an AEOSS dataset to study the tabu heuristics. The dataset follows the same configuration as the Chinese area distribution mode. The number of jobs changes from 50 to 1000 with an increment of 25. For each number of jobs, six instances are generated.



(a) Removal tabu



(b) Insertion tabu

Fig. 3 The improvement of solution quality (i.e., the value of the objective function) by two types of tabu for different completion ratios on the three problem domains

instances have relatively high completion ratio, these two tabu heuristics do not show obvious improvements. The Pearson correlation coefficient is -0.60 between the completion ratio and the performance of the removal tabu, and -0.63 between the completion ratio and the performance of the insertion tabu, showing negative correlation between the performance of these two tabu heuristics and the completion ratio. However, when the completion ratio is too low, the performance of the removal and insertion tabu starts to decrease. This is because our tabu length θ is calculated according to the total number of jobs n. If the completion is too low, the jobs that are forbidden to remove or insert are too many for the number of jobs in the solution sequence. This can be improved by decreasing the value of θ.

Although the performance of the removal and insertion tabu is similar, the difference between them is statistical significant: the p value in a paired t test is 4.25×10^{-3}. We find that the performance of the insertion tabu decreases with the completion ratio faster than the removal tabu. As a result, the insertion tabu works better than the removal tabu when the completion ratio is low (i.e., in Table 2, the the insertion tabu is better for the TDOPTW problem and the AEOSS problem when the completion ratio is low), while the removal tabu works better than the insertion tabu when the completion ratio is high (i.e., in Table 2, the removal tabu is better for the OAS problem and the AEOSS problem when the completion ratio is high).

Then we study the instant tabu. According to Fig. 4, the instant tabu works well when there are fewer jobs in the solution. Therefore, it works much better on the TDOPTW problem than it does on the other two problems. This is because when there are fewer jobs in the solution, the probability that the new repaired solution is the same as the old solution is higher, and the instant tabu can prevent this.

Overall, we conclude that it is better (although marginally so) to include the insertion tabu and the removal tabu for

Table 2 The difference between the performance of the removal and insertion tabu

Problem	Completion ratio (%)	Average improvement by tabu	
		Removal (%)	Insertion (%)
AEOSS	$>$ 50%	0.69	0.47
AEOSS	$<$ 50%	1.05	1.07
OAS	$>$ 50%	0.06	0.03
TDOPTW	$<$ 50%	0.66	0.71

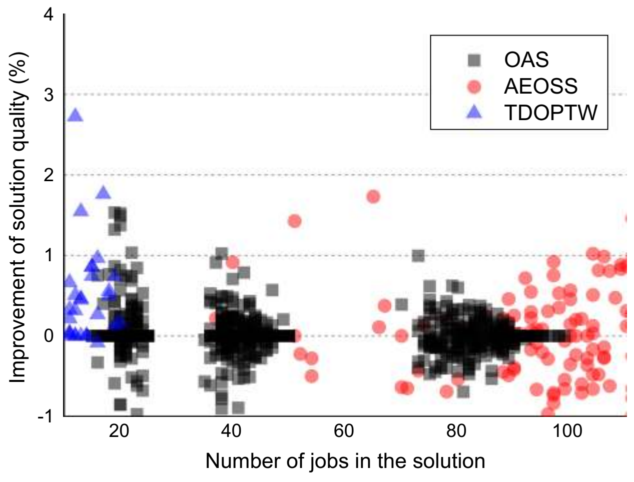


Fig. 4 The improvement of solution quality (i.e., the value of the objective function) by the instant tabu for different numbers of jobs in the solution on the three problem domains

problems with low completion ratio. But since these two tabu heuristics also help increasing the performance of the algorithm when the completion ratio is high, we suggest using them also for problems with high completion ratio such as the OAS problem. The instant tabu works better when the solution sequence is short; on problems with longer solution sequences, such as the AEOSS and OAS problems, performance is not improved. Further, as the length of the solution sequence increases, the time used for comparing solutions by the instant tabu increases. The instant tabu should not be used in this case. Therefore, in the following experiments, we use the insertion tabu and the removal tabu for the OAS problem and the AEOSS problem, and all the three tabu heuristics for the TDOPTW problem. We refer to it as the *TS strategy*.

Percentage of revisited recent solutions

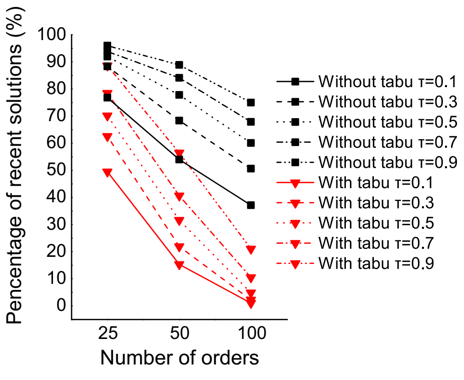
In this section, we test whether the TS strategy helps to reduce the probability of short-term cycling search process of ALNS. We run the algorithm with and without TS on the three problems, and the average percentages of iterations visiting a solution which has been visited within the previous 50 iterations are shown in Fig. 5.

According to Fig. 5, it is obvious that the probability that the algorithm re-visits a recent solution is reduced considerably by TS. This proves that the short-term cycling of the standard ALNS is reduced by TS. For the AEOSS problem, the instance with 50 jobs of the worldwide distribution is too simple for the algorithm and it can schedule all the jobs in the initial solution and terminate. Therefore, the percentage of recent solutions in this instance is 0.

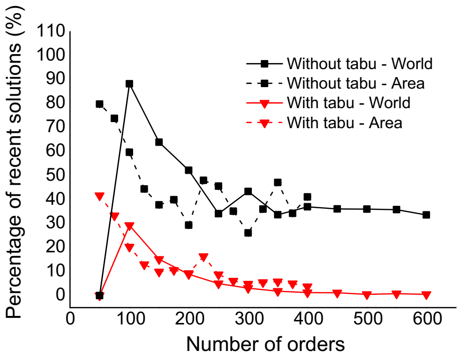
We also observe that the gap between the percentages of revisited recent solutions with and without TS tends to increase when the problem grows in size. This is because when the problem instance grows in size, the completion ratio gets lower. According to "Performance of three tabu types" section, the insertion tabu and the removal tabu work better when the completion ratio is lower.

Performance of the TS strategy

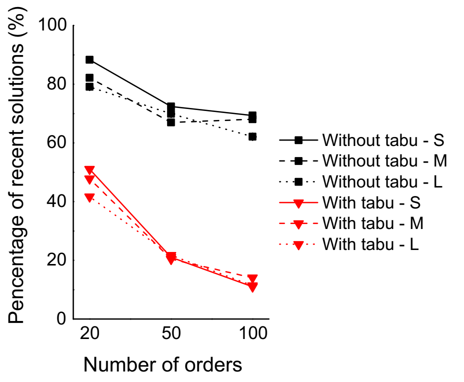
We test the performance of the TS strategy as follows. We compare the algorithm without the TS strategy (ALNS/PF) with the full ALNS/TPF algorithm. For the three problems, we divide the instances into three groups (i.e., small, medium and large) according to the number of jobs. Figure 6 shows that without TS, all the gaps are increased on all the three problems. For the three problem, TS contributes to the solution quality more when the instance grows in size. This is consistent with the observations in "Performance of three tabu types" and "Percentage of revisited recent solutions" sections. The performance of TS on the TDOPTW is not very obvious, because the algorithm with our other techniques is already quite efficient for the TDOPTW, especially for small instances (i.e., the decrease in the solution quality when removing TS is 0). For the OAS problem and the AEOSS problem, TS also reduces the CPU time much by forbidding useless removal of jobs from the solution. For the TDOPTW problem, TS increases the CPU time, because the instant tabu is time-consuming for comparing the new solution with the current solution.



(a) OA Sproblem



(b) AEOSS problem



(c) TDOPTW problem

Fig. 5 The average percentages of iterations visiting a recent solution. For the OAS problem, each point shows the average value of 50 instances with the same number of orders and the same τ. For the TDOPTW problem, each point shows the average value of four instances with the same number of orders and the same size of time windows

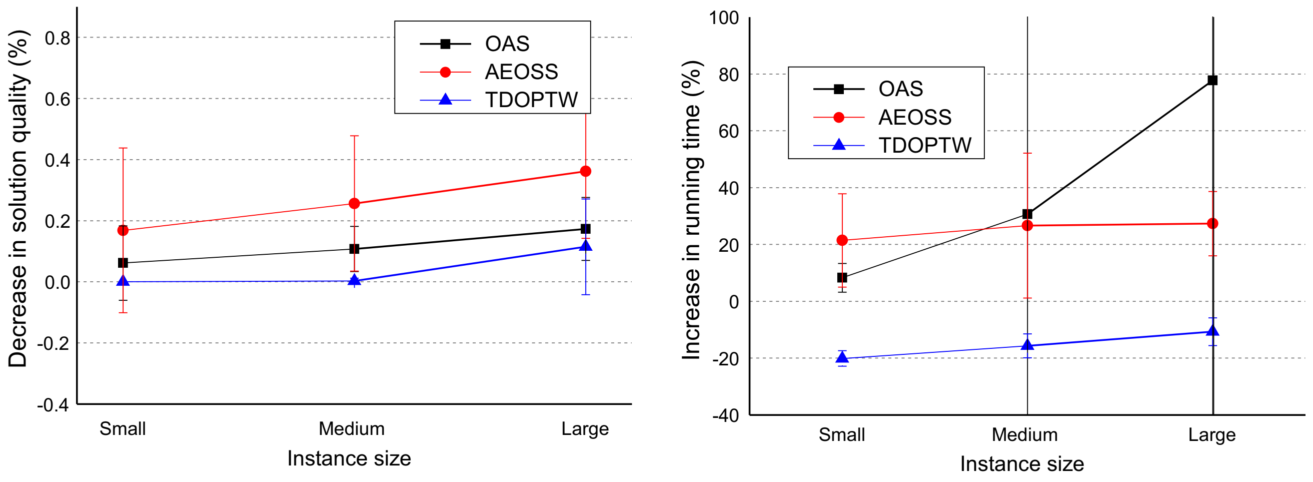


Fig. 6 The performance of ALNS/PF compared to the full ALNS/TPF algorithm

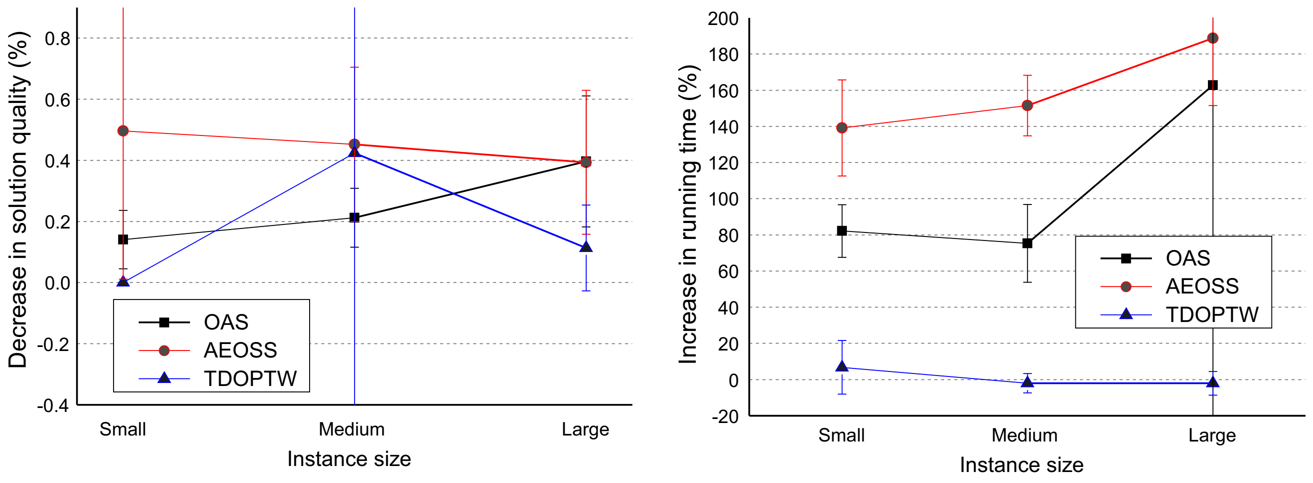
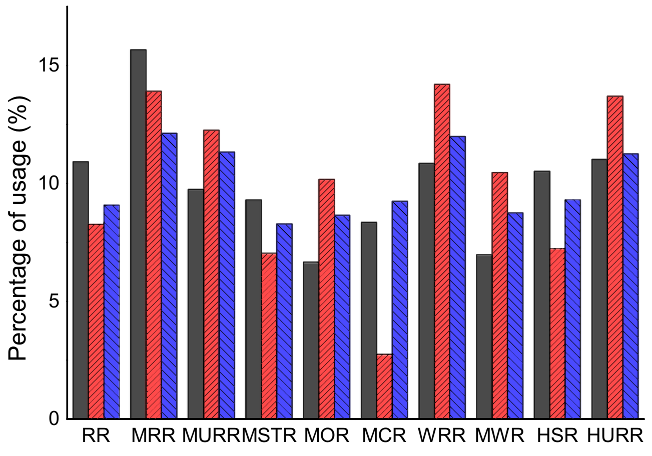
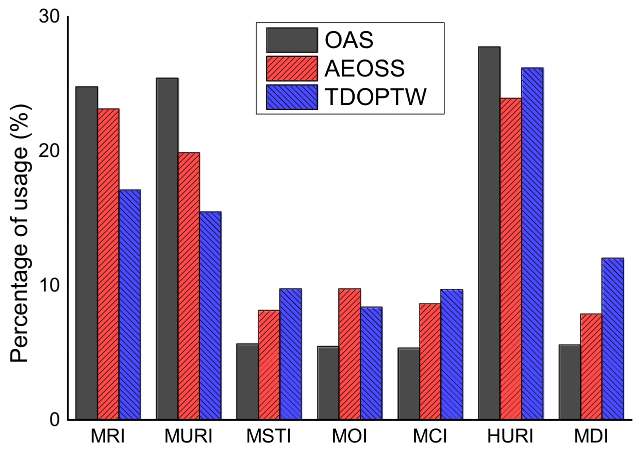


Fig. 7 The performance of ALNS-TS compared to the full ALNS/TPF algorithm



(a) Removal operators



(b) Insertion operators

Fig. 8 The performance of different neighborhood operators on three problem domains

Comparison with the two-stage hybridization

We further compare our tight hybridization with the two-stage hybridization of ALNS and TS (ALNS-TS). Let us assume for the full ALNS/TPF, the maximum iteration is N. In ALNS-TS, TS is run for $0.015N$ iterations after every $0.1N$ iterations of ALNS. In each TS iteration, 10 new neighbourhoods by our removal and insertion operators are examined to find the best local move. The whole process is run four times, hence for N neighbourhood moves in total. Recently visited solutions are inserted in a tabu list for $\sqrt{n/2}$ iterations. Note that ALNS-TS might examine more neighbours than N in order to find unvisited ones.

From Fig. 7, it is obvious that the two-stage strategy produces much worse solutions. For the OAS problem, the gap increases when the instance gets larger. We also observe that the two-stage hybridization works less well than even the standalone ALNS, by comparing ALNS-TS with ALNS/PF. When ALNS and TS share a total number of iterations, the standalone ALNS performs better than the two-stage hybridization of them. This shows that ALNS has a higher search efficiency than TS does for the OAS problem. For the TDOPTW problem, ALNS-TS does not use as much CPU time as for the other two problems, because the solutions for the TDOPTW problem are relatively short, and the TS component which compares different solutions does not consume much time.

Randomized generic neighbourhood operators

In the proposed algorithm, we use ten removal operators and seven insertion operators to update solutions.

Comparison of the different operators

We compare the performance of these operators on the three problem domains. Figure 8 shows the percentage of usage of different operators. Operators which perform better are selected by the adaptive mechanism of ALNS more often, so that they have higher percentage of usage.

In the removal operators, the max revenue removal (MRR) operator performs best and the max conflict removal (MCR) performs worst on the three domains. In the insertion operators, the max revenue insertion (MRI), the max unit revenue insertion (MURI) and the historical unit revenue insertion (HURI) perform best. We can also observe that location-related operators (worst route removal, min distance insertion) do not work well on OAS, because the setup times of jobs in the OAS problem are generated randomly and the jobs have little dependency on the location. Setup-related operators (max setup time removal, historical setup removal, max setup time insertion) perform well on the TDOPTW problem. This is because the setup times of TDOPTW problem are very long compared with the processing time. The setup times are useful heuristic information. The opportunity operators (max opportunity removal, min opportunity insertion) perform better on the AEOSS problem than the other two problems, because the jobs in the AEOSS problem have multiple time windows.

From the above observations, we can see that the algorithm adapts itself well by selecting different operators according to the different properties of different problems. However, this selection strategy of ALNS can still be improved. For example, although the opportunity operators (max opportunity removal, min opportunity insertion) perform best on the AEOSS problem, they are also used a lot on the other two problems, where these operators should be very inefficient because there is only one time window for each job. If such selections can be avoided or reduced, the search efficiency of ALNS can be improved. We will investigate a better online-learning strategy for these instance-dependent neighbourhood operators in our future work.

Performance of the randomization strategy

Then, we test the performance of the randomization strategy in the operators. We compare the algorithm with neighbourhood operators without randomness (ALNS/TPF (No random)) with the full ALNS/TPF algorithm. Figure 9 shows that the randomization strategy helps to increase the performance on most of the instances of the three problems. For the three problems, the randomization strategy contributes more for larger instances. When there are more jobs, the randomization strategy can help visiting more states in the solution space.

Partial sequence dominance

We test the performance of PSD by comparing the algorithm without PSD (ALNS/TF) with ALNS/TPF. According to Fig. 10, PSD shows obvious improvements on AEOSS problems, while it does not improve the performance of the algorithm much on the OAS problem and the TDOPTW problem. There are two reasons for this. For the OAS problem and the TDOPTW problem, the solution sequences are relatively short. PSD works well when the solution sequence gets long, because more partial sequences can be ignored in the long solution sequence. The second reason is that the time windows of the OAS problem and the TDOPTW problem are relatively long. If the time window is long, one order can exist in different partial sequences in the current solution and in the new solution respectively, reducing the quality of the compound solution. This can be further proved in Fig. 11, where for the OAS problem, PSD works better when τ is larger (i.e., when the time window is shorter).

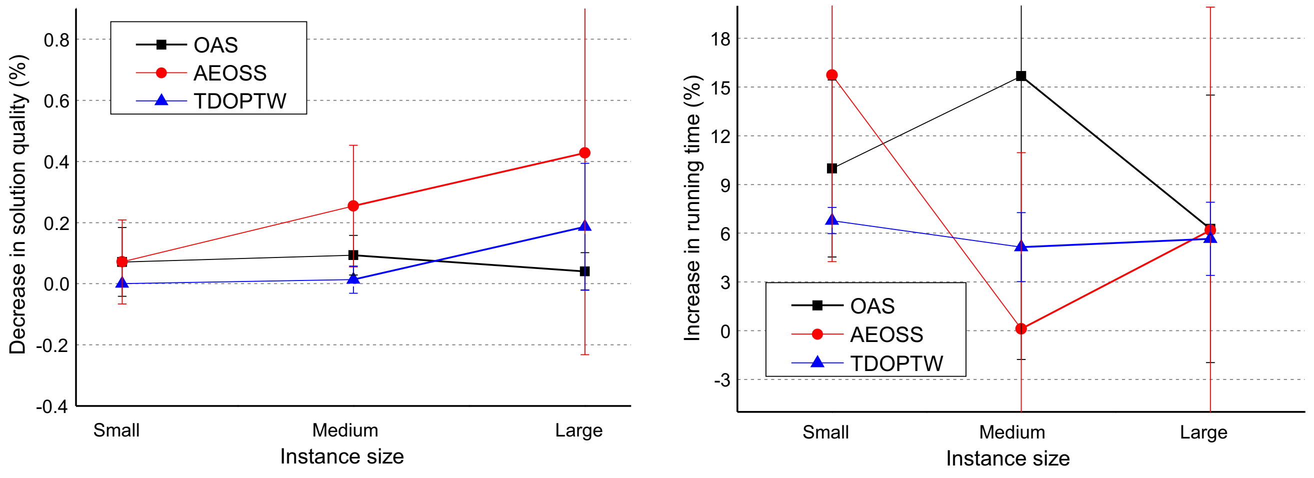


Fig. 9 The performance of ALNS/TPF (No random) compared to the full ALNS/TPF algorithm

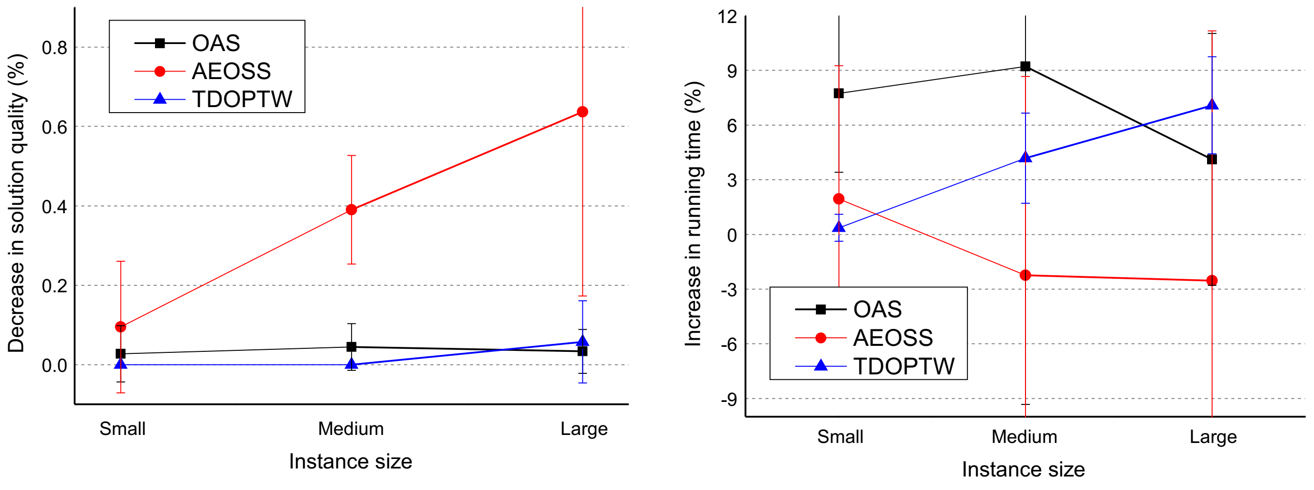


Fig. 10 The performance of ALNS/TF compared to the full ALNS/TPF algorithm

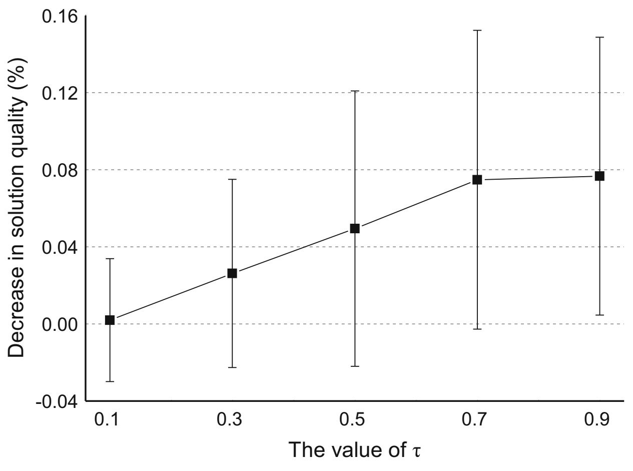


Fig. 11 The performance of ALNS/TF compared to the full ALNS/TPF algorithm on the OAS problem for different values of τ

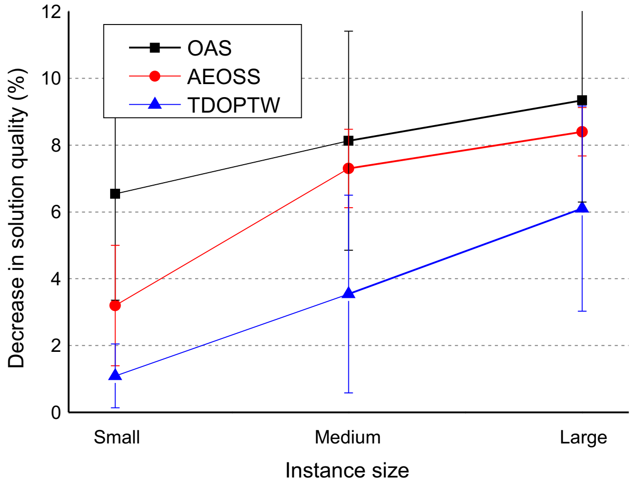
Fast insertion algorithm

The fast insertion algorithm contains two ideas: the time slack strategy and the best position selection strategy. We study each in turn.

Performance of the time slack strategy

First, to test the performance of the *time slack* strategy, we compare it with the *backward/forward time slack* strategy of Liu et al. (2017). Both the strategies have the same time complexity $O(n)$, but ours creates much more space in the schedule by considering postponing all the possible jobs in the solution, while Liu et al. (2017)'s method only creates limited space by moving two jobs.

In Fig. 12, ALNS/TPF with Liu et al. (2017)'s strategy is denoted as ALNS/TP (no time slack). It is obvious that our time slack strategy uses less time and has higher solution



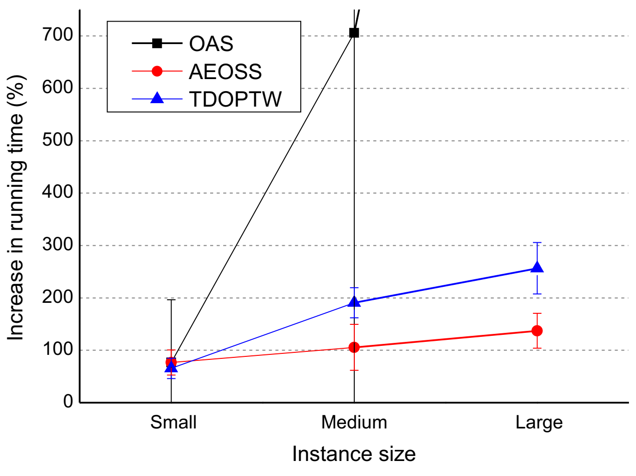
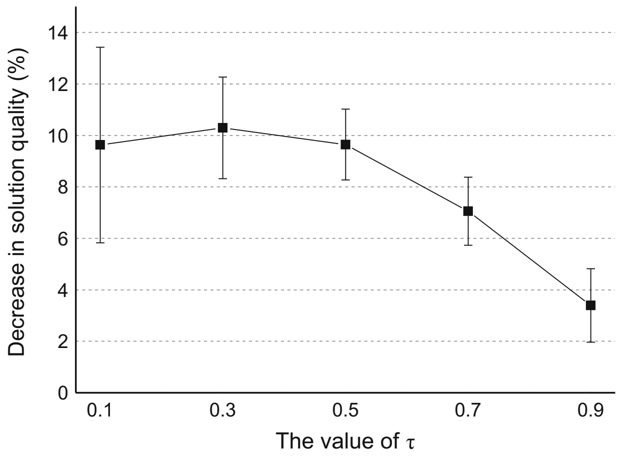


Fig. 12 The performance of ALNS/TP (no time slack) compared to the full ALNS/TPF algorithm



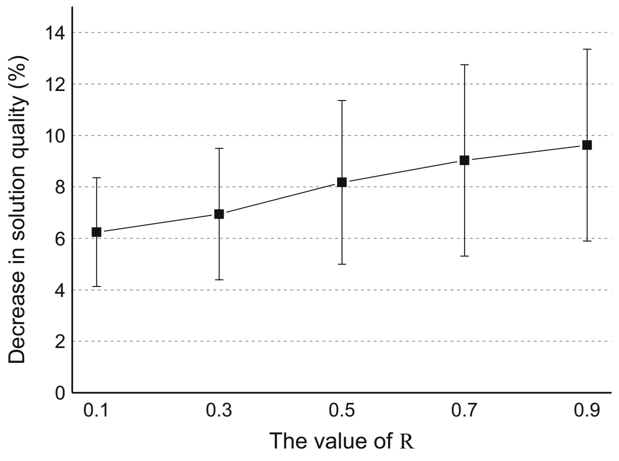


Fig. 13 The performance of ALNS/TP (no time slack) compared to the full ALNS/TPF algorithm on the OAS problem for different values of τ and R

quality. And among all the strategies, the time slack strategy contributes most performance. For all the three problems, the time slack strategy works better when there are more jobs. Because when there are more jobs, the superiority of postponing multiple jobs becomes obvious compared with Liu et al. (2017)'s strategy, which only moves two jobs. Besides, according to Fig. 13, for the OAS problem, the time slack strategy works better when τ is smaller and R is larger. When τ is smaller, the time window is longer and the time slack strategy can make more use of the long time window. When R is larger, the gap between the due time and end time of a time window grows. The due time slack strategy works better in this case. Similarly, according to Fig. 14, for the TDOPTW problem, the time slack strategy works better for longer scheduling horizon and larger variance in the lengths of time windows.

Performance of the best position selection strategy

Second, we study the best position selection strategy. The local optimality of this strategy can be proved in the premise that no look-ahead strategy is considered in this insertion algorithm, which will increase the complexity of the insertion algorithm. If we assume that when inserting a job, the following jobs are not considered, the optimal insertion position should be the one that increases the least setup time. This is because when we try to insert a job, the revenue and the processing time of the job are fixed. Since the total scheduling horizon is limited, the optimal insertion should be the one that inserts the job successfully (i.e., receives the revenue) as well as maximizes the remaining scheduling space for following jobs. The strategy guarantees that if a job can be inserted, the increased setup time is minimal. So this strategy finds the optimal insertion of a job.

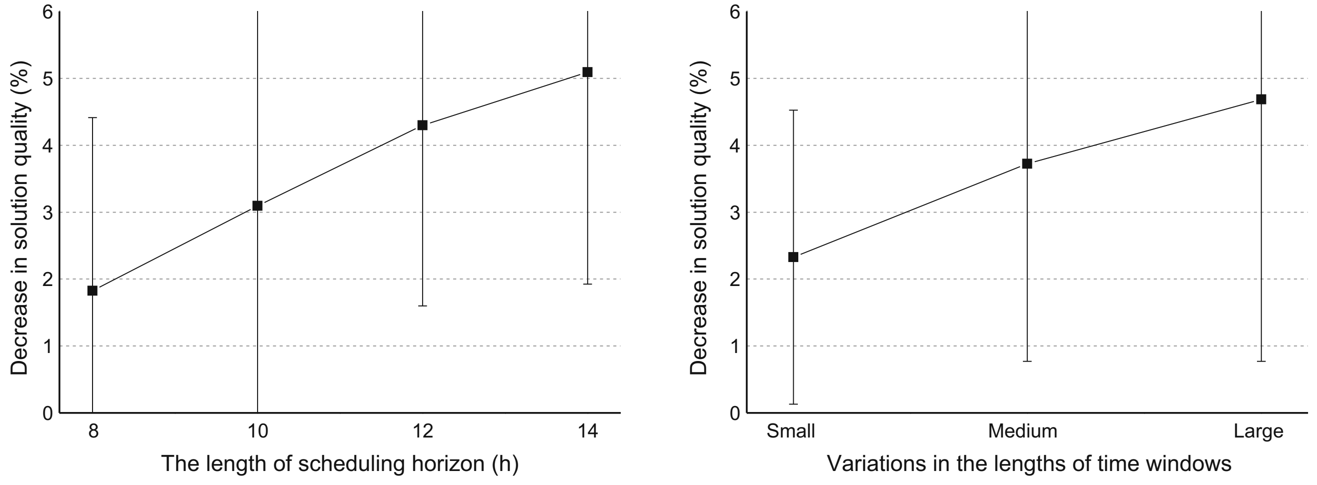


Fig. 14 The performance of ALNS/TP (no time slack) compared to the full ALNS/TPF algorithm on the TDOPTW problem

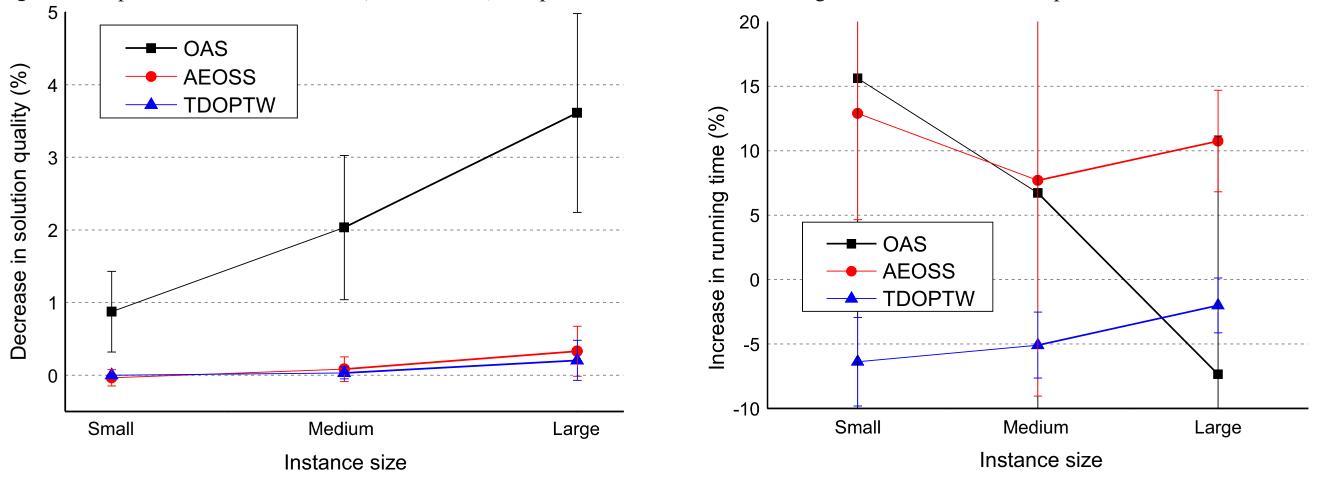


Fig. 15 The performance of ALNS/TP (no selection) compared to the full ALNS/TPF algorithm

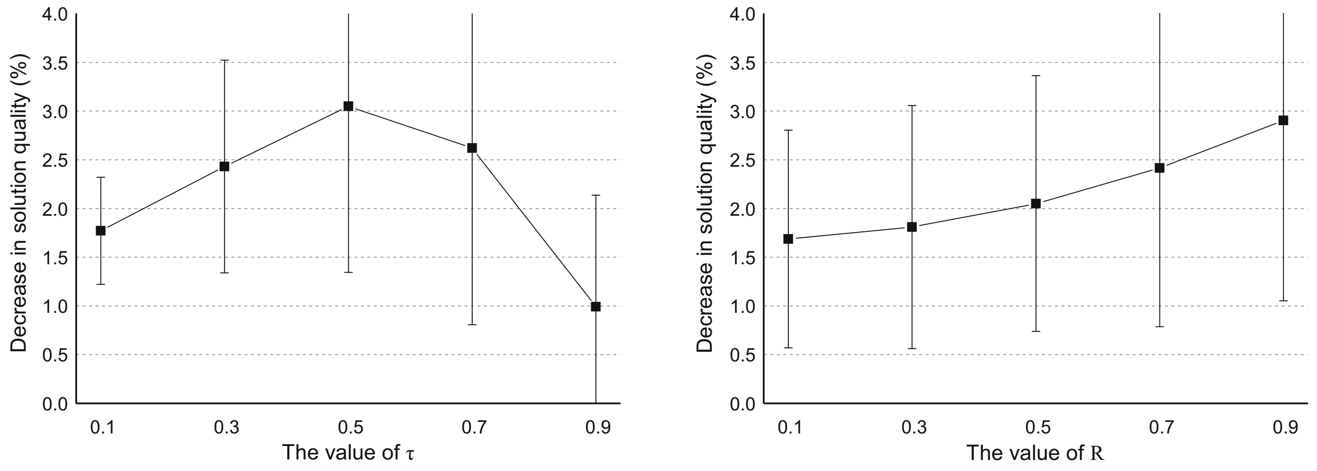


Fig. 16 The performance of ALNS/TP (no selection) compared to the full ALNS/TPF algorithm on the OAS problem for different values of τ and R

The algorithm without selection (ALNS/TP (no selection)) is compared with ALNS/TPF in Fig. 15. The strategy works significantly better on the OAS problem than it does on the other two domains. This is because for the AEOSS problem, the time windows of jobs are relatively short compared with the scheduling horizon. And for the TDOPTW problem, the number of jobs in the solution is small. Therefore for these two problems, when inserting a candidate job, the number of possible positions to insert the job is limited compared with the OAS problem. The selection of the best position does not improve the performance of the algorithm on these two domains much. This can also be proved by Fig. 16. For the OAS problem, the selection works better when τ is smaller and R is larger. When the time window is longer, the number of possible insertion positions also gets larger. In this case a sorting strategy to compare the insertion positions works well. When R is larger, the jobs in the current solutions have similar time windows. The candidate job to be inserted can neighbour more jobs in the solution, resulting in a larger number of possible insertion positions.

To sum up, from the algorithmic analysis of this section, we derive the following conclusions: (1) the insertion tabu and the removal tabu work better when the completion ratio is low, and the instant tabu works better when there are fewer jobs in the solution sequence; (2) our tight hybridization of ALNS and TS works better than the two-stage hybridization; (3) the multiple neighbourhood operators can be selected adaptively by the algorithm according to the properties of different problems; the randomization strategy performs well by improving the solution quality and reducing runtime; (4) the PSD works better when the instance grows in size, which proves that it helps to combine parts of different solutions, when the solution sequence gets long; (5) the best position selection contributes much to the solution quality, but also consumes more time; and (6) the time slack strategy works well in terms of solution quality and time complexity.

Comparison with state-of-the-art algorithms

In this section, we compare the performance of the full algorithm, ALNS/TPF, against state-of-the-art algorithms and solvers on each of the three domains.[4] Furthermore, we compare the proposed algorithm with IBM ILOG CP Optimizer (CPO) (Laborie et al. 2018) on a relaxed OAS problem. We do not compare the algorithm with CPO on the three original problems because CPO does not support the time-dependent setup constraints of the AEOSS problem and the TDOPTW problem and the special setup constraints (i.e., with the ‘max’ term) of the OAS problem well. More details about this can be found in “Comparison with CP optimizer” section.

Parameter setting

First, we provide the parameter setting of the algorithm for the three problems. The use of TS, the randomized operators, PSD, and FI is set according to the conclusions of the algorithmic analysis reported in “Algorithmic analysis” section. Other parameters such as the number of jobs to remove are set empirically through some exploratory experiments. Although these parameters are not optimal for all the instances, they provide relatively good solutions. The parameters are listed in Table 3. In Table 3, c_a is the coefficient of annealing to update the temperature. The temperature at the i^{th} iteration is $T_i = c_a T_{i-1}$. The three score increments are used for different performances of the selected operators: σ_3 is added to the score if a new best solution is found; σ_2 is added to the score if a new solution which is better than the current solution is found; σ_1 is added to the score if a new solution which is worse than the current solution is found, but it is accepted by the SA criterion.

Note that among all these parameters, only the maximum iteration and the use of the instant tabu are different on the three domains, which shows the robust performance of our algorithm on diverse problem instances.

The algorithm has two terminal conditions: when the maximum iteration is reached or when all the jobs are scheduled and get the full revenue (i.e., when the upper bound is reached). The maximum iteration is set so that the algorithm has similar runtime as its competing algorithms on the three domains. For the AEOSS problem, we set another terminal condition, when the solution is not improved for continuous 1000 iterations, because the algorithm converges fast on the AEOSS domain.

OAS problem

The ALNS/TPF algorithm is compared with state-of-the-art algorithms including DRGA (Nguyen et al. 2015), LOS (Nguyen 2016), ABC (Lin and Ying 2013), HSSGA (Chaurasia and Singh 2017), EG/G-LS (Chaurasia and Singh 2017), and ILS (Silva et al. 2018). A MILP formulation by the CPLEX solver has been tested by Cesaret et al. (2012), which shows bad performance for large instances with more than 15 instances. Therefore we do not compare ALNS/TPF against CPLEX in this section.

Our ALNS/TPF is run on Intel Core i5 3.20 GHz CPU with 8GB memory, using a single core. The results of other methods are from the respective articles, which were obtained using different machines with Intel Core i5, i7 and Xeon CPU, 3.00–3.40 GHz, 4–16 GB memory. Besides, unmen-

[4] The source code of our algorithm is available https://doi.org/10.4121/uuid:3a23b216-3762-4a61-ba2c-d3df6dc53268, and the datasets used in this paper are available https://doi.org/10.4121/uuid:1a4e5895-7dae-4b6a-9315-a9e8cb463d73.

Table 3 Parameter setting of the algorithm for the three problems

Parameter name	OAS	AEOSS	TDOPTW
Maximum iteration	$1000n$	10,000	50,000
Maximum iteration of no improvement	–	1000	–
Number of jobs to remove p_d	$0.1\|S\|$	$0.1\|S\|$	$0.1\|S\|$
Weight update parameter λ	0.5	0.5	0.5
Coefficient of annealing c_a	0.9975	0.9975	0.9975
Score increment σ_1	10	10	10
Score increment σ_2	20	20	20
Score increment σ_3	30	30	30
Removal tabu	Yes	Yes	Yes
Insertion tabu	Yes	Yes	Yes
Instant tabu	No	No	Yes
Randomized neighbourhood operators	Yes	Yes	Yes
Partial sequence dominance	Yes	Yes	Yes
Fast insertion algorithm	Yes	Yes	Yes

tioned details such as cache size can have a significant effect on the runtime. Hence we do not report detailed CPU time. On average, all the methods have comparable performance in terms of CPU time. According to the data reported in the references, ILS uses most time and ABC the least.

The solution quality of instances with 25–100 jobs are shown in Tables 4, 5 and 6.[5] Regarding the solution quality, we compare the gaps of these algorithms to the upper bounds by Cesaret et al. (2012). ALNS/TPF, TDRGA, LOS and ILS are run ten times on each of the instances. The average gaps of the ten runs for each instance are calculated first. Then for each group of instances with the same parameters, we calculate the minimum, average and maximum gaps of the ten instances in this group. The other three algorithms, ABC, HSSGA and EA/G-LS are run only once for each instance. Due to this, these methods can sometimes reach the optimal solution, resulting in a lower solution gap. DRGA and LOS only reported rounded-down integer values. But it is still obvious that ALNS/TPF produces the best solutions on nearly all the instances.

AEOSS problem

We compare the proposed ALNS/TPF with our previous algorithm called ALNS/TPI (He et al. 2018), the standard ALNS (Liu et al. 2017), the ILS algorithm (Peng et al. 2018), and an MIP model from He et al. (2018). We have obtained the source codes for the standard ALNS and the ILS algorithm from the corresponding authors. All algorithms are run on an Intel Core i5 3.20 GHz CPU, 8 GB memory, running Windows 7; only a single core is used. IBM ILOG CPLEX version 12.8 is used for MIP solving. A time limit of 3600 s is set for MIP solving, which uses more than 100x as much running time as ALNS/TPF does. The results for metaheuristics are the average of ten runs.

We compare the solution quality and the CPU time. The solution quality is the percentage of the total revenue of scheduled jobs (i.e., the objective value) divided by the total revenue of all the jobs. Table 7 shows the comparison of the five different algorithms. It shows that the CPU time of the ALNS/TPF increases slowly with the increasing number of jobs, and is about ten times faster than ILS, which is the fastest one among other algorithms. The solution quality is consistently higher than that of ILS, ALNS/TPI and ALNS. As expected, MIP by CPLEX can only find optimal solutions for small-size instances but performs badly when the instance size gets large. For the four small instances with optimal solutions by CPLEX, ALNS/TPF, ALNS/TPI and ILS also find the same optimal solution. Among all the methods, the standard ALNS performs the worst, consuming a long time to produce solutions with the lowest quality. Finally, Fig. 17 shows the anytime quality of different algorithms for the instance with 600 jobs distributed worldwide (CPLEX found no feasible solution within the time limit).

Then we study how the performance gaps between ALNS/TPF and other algorithms change with the increasing number of jobs. We evaluate the performance gap between an algorithm A and ALNS/TPF by the following formula:

$$Gap_A = \frac{SolutionQuality_{ALNS/TPF} - SolutionQuality_A}{SolutionQuality_{ALNS/TPF}} \tag{10}$$

[5] For the gaps of instances with 25 and 50 jobs reported by Silva et al. (2018), we identify some mistakes because the gaps in their table are smaller than the gaps between their tight upper bounds and the upper bounds by Cesaret et al. (2012). Therefore we decide not to include ILS in the comparison of instances with 25 and 50 jobs.

Table 4 Gaps (%) of various algorithms on the OAS instances with 25 jobs

τ	R	DRGA			LOS			ABC			HSSGA			EA/G-LS			ALNS/TPF		
		Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
0.10	0.10	2	3	4	1	2	3	1.45	2.7	3.85	1.09	2.08	3.16	1.09	2.16	3.16	**0.72**	**1.68**	**2.64**
	0.30	1	2	6	**0**	2	5	0.69	2.18	5.78	0.69	1.66	3.2	0.69	1.58	3.2	**0.34**	**1.31**	**2.78**
	0.50	1	1	3	**0**	1	3	**0**	1.38	2.45	**0**	0.9	1.9	**0**	0.91	1.77	**0**	**0.64**	**1.47**
	0.70	**0**	1	3	**0**	**0**	2	**0**	0.63	2.68	**0**	0.39	1.67	**0**	0.39	1.67	**0**	**0.31**	**1.6**
	0.90	**0**	1	2	**0**	**0**	2	**0**	0.35	2.09	**0**	0.35	2.09	**0**	0.35	2.09	**0**	**0.3**	**1.93**
0.30	0.10	2	3	5	1	3	4	1.21	3.11	5.24	1.21	2.48	4.55	1.21	2.44	4.55	**0.68**	**2.19**	**3.88**
	0.30	2	3	5	2	3	5	1.12	3.37	6.04	1.12	3.12	**4.62**	2.15	3.46	6.04	**1.11**	**2.69**	4.66
	0.50	2	2	**3**	**0**	2	**3**	1.12	2.4	5.24	1.12	**1.52**	3.5	2.15	1.85	3.5	**0.66**	1.53	**3.49**
	0.70	1	2	5	**0**	**1**	5	0.88	1.97	5.32	**0**	1.61	**4.61**	**0**	1.9	5.32	**0**	**1.5**	5.03
	0.90	**0**	1	3	**0**	1	**2**	**0**	1.27	3.38	**0**	1.05	2.82	**0**	1.1	2.82	**0**	**0.91**	**2.64**
0.50	0.10	2	4	7	**1**	4	**5**	2.8	4.88	6.76	**1.68**	4.17	6.76	**1.68**	4.11	6.76	**1.68**	**3.67**	**5.61**
	0.30	2	5	8	**1**	4	**7**	2.85	4.49	7.29	1.82	4.22	7.29	1.82	4.12	7.29	**1.81**	**3.87**	**7.12**
	0.50	1	5	**6**	1	**4**	**6**	1.94	4.48	6.02	**0.97**	4.28	6.08	**0.97**	4.25	6.08	**0.97**	**4.14**	**6.01**
	0.70	2	**4**	**7**	**0**	**4**	**7**	0.64	4.2	**7.17**	0.64	**4.16**	**7.17**	0.64	4.19	**7.17**	**0.63**	4.17	**7.17**
	0.90	1	3	7	1	3	**6**	1.03	3.37	7	1.03	3.05	7	1.03	3.13	6.79	**0.68**	**2.95**	**6.66**
0.70	0.10	1	8	16	**0**	**7**	15	0.93	7.72	15.88	0.93	7.43	**14.86**	0.93	7.53	16.22	**0.92**	**7.35**	**14.86**
	0.30	**5**	9	13	**5**	**8**	**12**	**5.05**	8.64	12.4	5.78	8.49	12.4	5.78	8.53	12.5	**5.05**	**8.32**	**12.39**
	0.50	6	**10**	**14**	**5**	**10**	**14**	**5.37**	10.45	14.03	**5.37**	10.05	14.03	**5.37**	10.17	14.03	**5.37**	**10.02**	**14.02**
	0.70	2	7	**12**	**1**	**6**	**12**	1.69	6.9	**12.08**	**1.58**	6.56	**12.08**	**1.58**	**6.47**	**12.08**	1.69	6.71	12.75
	0.90	1	**8**	14	**0**	**8**	**13**	0.01	8.12	13.64	0.01	8.12	13.64	0.01	8.27	13.64	**0**	**8.07**	**13.63**
0.90	0.10	**0**	1	5	**0**	1	5	**0**	0.59	4.93	**0**	**0.49**	4.93	**0**	**0.49**	4.93	**0**	**0.49**	**4.92**
	0.30	**0**	1	1	**0**	**0**	**0**	**0**	0.16	1.48	**0**	**0**	**0.01**	**0**	**0**	**0.01**	**0**	0.02	0.21
	0.50	**0**	**2**	**12**	**0**	**2**	**12**	**0**	2.58	**12.3**	**0**	**2.49**	**12.3**	**0**	**2.49**	**12.3**	**0**	2.51	**12.3**
	0.70	1	7	21	**0**	**6**	21	**0**	6.81	**20.99**	**0**	6.67	**20.99**	**0**	6.67	**20.99**	**0**	**6.65**	**20.99**
	0.90	1	6	20	**0**	6	**19**	**0**	6.09	19.33	**0**	**5.99**	19.1	**0**	**5.99**	19.1	**0**	**5.99**	**19.09**
Avg.		1	**3**	8	**0**	**3**	**7**	1.15	3.95	8.13	1	3.65	7.63	1.08	3.7	7.76	**0.89**	**3.51**	**7.51**

The best results among the algorithms compared are highlighted in bold

Table 5 Gaps (%) of various algorithms on the OAS instances with 50 jobs

τ	R	DRGA			LOS			ABC			HSSGA			EA/G-LS			ALNS/TPF		
		Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
0.10	0.10	1	2	3	1	2	3	1.2	1.89	2.81	0.51	1.18	1.69	0.51	1.08	1.47	**0.34**	**0.62**	**1.11**
	0.30	1	2	3	1	2	3	1.06	1.77	2.33	0.71	1.12	1.57	0.71	1.1	1.57	**0.45**	**0.8**	**1.24**
	0.50	1	1	2	**0**	1	2	0.34	0.96	2.03	**0**	0.43	0.84	**0**	0.45	0.95	**0**	**0.31**	**0.71**
	0.70	**0**	2	**16**	**0**	2	**16**	**0**	1.9	16.39	**0**	1.73	16.39	**0**	1.66	16.39	**0**	**1.63**	**16.38**
	0.90	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0.30	0.10	2	3	4	1	2	3	1.63	2.59	3.77	1.02	1.8	2.79	1.22	1.93	2.86	**0.6**	**1.36**	**2.36**
	0.30	2	3	4	2	3	4	1.94	2.98	4.49	1.55	2.06	2.66	1.54	2.18	3.05	**1.13**	**1.59**	**2.4**
	0.50	1	2	4	1	2	**3**	1.94	1.88	3.88	1.55	1.51	3.88	1.54	1.53	3.68	**0**	**1.25**	**3.36**
	0.70	**0**	1	2	**0**	1	2	**0**	0.75	1.56	**0**	0.39	1.03	**0**	0.35	1.04	**0**	**0.12**	**0.35**
	0.90	**0**	1	2	**0**	**0**	**1**	**0**	0.5	1.83	**0**	0.29	1.21	**0**	0.23	1.02	**0**	**0.16**	**1.01**
0.50	0.10	3	4	5	1	3	4	1.71	3.07	4.17	1.11	2.2	3.2	1.11	2.23	3.38	**0.81**	**1.71**	**2.36**
	0.30	3	5	7	2	4	6	3.08	4.45	6.11	2.26	3.43	5.15	2.26	3.53	5.73	**1.78**	**3.04**	**5.05**
	0.50	2	4	8	1	3	7	2.07	3.78	6.82	1.03	3.04	6.33	1.03	3.17	7.14	**0.97**	**2.79**	**5.99**
	0.70	1	3	6	**0**	2	4	0.76	2.28	4.24	0.36	2	3.87	0.38	2.03	4.24	**0.17**	**1.67**	**3.86**
	0.90	0	2	5	0	**1**	**4**	3.61	1.71	4.04	0	1.54	4.31	−4.02	1.57	4.04	**−4.41**	**1.24**	**4.03**
0.70	0.10	4	5	7	2	4	5	3.16	4.57	5.79	2.42	4.17	5.2	2.42	4.18	5.39	**1.74**	**3.62**	**4.58**
	0.30	4	7	10	3	5	9	3.51	5.74	9.36	3.01	5.02	8.3	3.17	5.05	8.3	**2.57**	**4.52**	**8.16**
	0.50	6	8	12	5	**6**	11	5.3	7.09	11.39	5.08	6.71	11.03	5.17	6.7	11.03	**4.57**	**6.16**	**10.32**
	0.70	3	8	15	2	7	**14**	1.89	7.56	15.26	1.72	7.18	15.38	**1.37**	7.09	**14.31**	1.39	**6.88**	14.76
	0.90	4	9	14	3	**7**	**13**	3.69	8.02	13.67	**2.91**	7.69	13.38	3.3	7.62	**13**	2.92	**7.51**	**13**
0.90	0.10	8	12	19	7	11	**16**	7.3	11.33	**16.77**	**6.55**	10.97	**16.77**	7.3	11.15	**16.77**	6.57	**10.88**	**16.77**
	0.30	**9**	14	18	**9**	**13**	**17**	**9.28**	13.77	17.8	**9.28**	13.49	17.6	**9.28**	13.47	**17.4**	**9.28**	**13.34**	17.58
	0.50	4	13	19	**3**	**12**	**16**	**3.28**	12.77	17.29	**3.28**	12.33	16.88	3.46	12.45	16.73	**3.28**	**12.25**	**16.51**
	0.70	7	13	18	6	**11**	**17**	5.81	11.53	17.85	**5.62**	11.48	17.85	**5.62**	11.39	17.85	**5.62**	**11.26**	**17.8**
	0.90	10	13	17	10	**12**	16	10.02	12.38	16.27	**9.57**	12.37	**15.92**	9.58	**12.3**	16.09	9.61	12.4	16.06
Avg.		3	5	9	2	5	8	2.9	5.01	8.24	2.38	4.57	7.73	2.28	4.58	7.74	**1.97**	**4.28**	**7.43**

The best results among the algorithms compared are highlighted in bold

Table 6 Gaps (%) of various algorithms on the OAS instances with 100 jobs

τ	R	DRGA			LOS			ABC			HSSGA			EA/G-LS			ILS			ALNS/TPF		
		Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
0.10	0.10	1	1	2	2	2	3	1.11	1.75	2.21	0.4	0.77	1.32	0.72	0.94	1.22	0.74	0.95	1.35	**0.28**	**0.44**	**0.64**
	0.30	1	1	2	1	2	3	0.9	1.35	1.71	0.45	0.71	0.98	0.44	0.74	1.06	0.44	0.74	1.09	**0.25**	**0.49**	**0.76**
	0.50	1	1	1	1	1	2	0.36	0.7	1.23	**0**	0.19	0.38	**0**	0.2	0.37	0.2	0.37	0.58	**0**	**0.07**	**0.18**
	0.70	**0**	1	1	**0**	**0**	1	**0**	0.09	0.35	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.04	0.12	**0**	**0**	**0**
	0.90	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.01	0.1	**0**	**0**	**0**
0.30	0.10	2	3	4	2	2	3	1.27	2.11	2.71	0.5	1.18	1.71	0.69	1.25	1.71	1	1.4	1.82	**0.35**	**0.79**	**1.37**
	0.30	1	3	4	1	3	4	1.4	2.12	3.97	0.66	1.27	2.64	0.66	1.41	2.64	0.73	1.38	2.67	**0.52**	**1.05**	**2.33**
	0.50	2	2	3	1	2	3	1.4	1.67	2.45	0.66	1.07	**1.52**	0.66	1.11	1.69	0.73	1.17	1.65	**0.54**	**0.95**	1.75
	0.70	1	1	1	**0**	1	2	**0**	0.63	1.16	**0**	0.35	0.85	**0**	0.24	0.63	0.27	0.44	0.74	**0**	**0.18**	**0.55**
	0.90	**0**	1	1	**0**	**0**	1	**0**	0.26	0.82	**0**	0.1	0.56	**0**	0.14	0.47	**0**	0.25	0.73	**0**	**0.02**	**0.15**
0.50	0.10	3	5	7	2	4	5	2.5	3.37	4.33	1.22	2.09	2.99	1.69	2.31	3.27	1.46	2.26	3.11	**0.72**	**1.49**	**2.47**
	0.30	4	4	6	2	3	5	2.46	3.01	4	1.54	2.29	2.95	1.67	2.43	2.96	1.72	2.32	3.08	**1.31**	**1.94**	**2.44**
	0.50	3	4	6	2	3	4	2.26	3.13	4.21	1.22	2.36	3.35	1.48	2.51	3.61	1.58	2.4	3.36	**1**	**2.07**	**3.05**
	0.70	1	3	4	**0**	2	3	0.77	1.82	2.99	0.51	1.49	2.33	0.51	1.32	2.24	0.49	1.61	2.8	**0.21**	**1.16**	**2.07**
	0.90	1	2	4	**0**	1	4	0.53	1.42	3.13	0.18	1.04	**2.15**	0.18	0.9	**2.15**	0.39	1.16	2.76	**0.01**	**0.67**	2.25
0.70	0.10	4	6	8	3	4	7	3.2	3.99	4.84	2.09	2.84	3.53	2.27	2.98	3.82	2.28	3.13	3.77	**1.45**	**2.18**	**2.91**
	0.30	4	6	9	2	5	7	2.37	4.72	7.16	1.71	3.88	6.07	1.67	4.09	6.26	1.89	3.86	5.82	**1.14**	**3.31**	**5.49**
	0.50	4	7	15	2	5	12	3.27	5.27	11.94	2.25	4.62	10.95	2.62	4.69	10.95	2.63	4.25	**8.69**[a]	**1.76**	**3.94**	10.31
	0.70	5	8	12	2	5	8	2.77	5.37	7.37	2.18	5.16	7.78	2.77	5.39	7.65	2.66	6.17	9.32	**1.56**	**4.42**	**6.67**
	0.90	4	7	10	3	5	8	3.57	5.9	8.33	3.73	5.48	7.49	3.15	5.34	6.88	3.92	6.6	8.27	**2.8**	**4.58**	**5.95**
0.90	0.10	6	9	12	4	6	9	4.7	6.62	9.44	4.57	6.27	9.06	4.75	6.45	9.34	4.4	7.02	9.07	**3.46**	**5.4**	**8.44**
	0.30	8	12	16	6	9	12	5.39	9.4	11.86	5.11	9.03	11.25	5.39	9.09	12.35	8.91	11.83	13.59	**4.98**	**8.47**	**11.11**
	0.50	11	14	19	8	11	16	8.82	11.88	17.25	8.6	11.26	16.29	8.28	11.2	**15.84**	10.12	14.06	18.4	**7.64**	**10.69**	15.91
	0.70	10	14	16	8	11	14	7.96	11.19	12.85	8.07	11.23	13.07	8.05	11.07	13.06	9.67	12.75	15.95	**7.14**	**10.54**	**12.69**
	0.90	8	13	19	4	11	18	4.11	11.31	16.83	3.57	11.07	16.6	4.07	11.02	16.73	7.21	13.23	18.43	**3.19**	**10.57**	**16.24**
Avg.		3	5	7	2	4	6	2.44	3.96	5.73	1.97	3.43	5.03	2.07	3.47	5.08	2.54	3.98	5.49	**1.61**	**3.01**	**4.62**

For this value reported by Silva et al. (2018), we find it smaller than the gaps between their tight upper bounds and the upper bounds by Cesaret et al. (2012)

The best results among the algorithms compared are highlighted in bold

Table 7 The results of various algorithms on the AEOSS problem

Dis	*n*	Solution quality (%)													Time(s)				
		Cplex	ALNS			ALNS/TPI			ILS			ALNS/TPF			Cplex	ALNS	ALNS/TPI	ILS	ALNS/TPF
			Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max					
Area	50	**85.77**	76.99	79.37	83.26	84.1	84.9	84.94	**85.77**	**85.77**	**85.77**	**85.77**	**85.77**	**85.77**	71.51	8.45	3.86	0.77	**0.09**
Area	75	78.94	72.51	74.49	75.44	78.95	79.23	79.24	81.87	82.34	**82.75**	**82.75**	**82.75**	**82.75**	3600	16.22	7.47	1.41	**0.19**
Area	100	43.79	65.46	66.74	67.95	69.07	69.98	70.65	75.17	75.71	76.07	**76.52**	**76.52**	**76.52**	3600	25.23	11.42	2.19	**0.28**
Area	125	49.82	59.17	60.58	61.69	62.59	63.64	64.57	69.06	69.24	69.78	**69.42**	**70**	**70.32**	3600	32.63	11.87	3.11	**0.35**
Area	150	1.45	50.44	51.64	52.76	55.09	56.02	57.7	62.5	63.05	63.37	**63.95**	**64.27**	**64.39**	3600	46.97	13.22	4.63	**0.45**
Area	175	–	44.84	46.36	48.4	52.7	53.37	54.05	59.95	60.45	60.93	**61.55**	**61.73**	**62.04**	–	67.44	15.17	6.1	**0.61**
Area	200	–	39.44	40.25	41.06	48.06	49.33	50.43	56.36	56.83	57.44	**58.08**	**58.58**	**58.73**	–	94.98	16.96	7.8	**0.83**
Area	225	–	37.07	37.99	38.78	44.77	45.58	46.39	53.52	53.93	54.47	**55.42**	**55.79**	**56.08**	–	121.01	17.93	9.31	**0.84**
Area	250	–	34.04	34.61	35.15	43.74	44.87	46.13	51.66	51.93	52.26	**54.21**	**54.41**	**54.72**	–	151.38	19.47	11.43	**0.9**
Area	275	–	31.82	32.24	32.68	40.89	41.85	43.39	47.77	48.24	48.87	**50.27**	**50.59**	**50.82**	–	162.73	20.94	13.38	**0.95**
Area	300	–	28.94	29.35	29.88	38.49	39.75	40.88	45.3	45.88	46.31	**47.97**	**48.29**	**48.63**	–	201.5	22.57	15.19	**1.03**
Area	325	–	29.16	29.57	30.15	37	37.96	38.91	43.52	43.96	44.44	**46.08**	**46.62**	**46.87**	–	233.95	24.24	17.27	**1.34**
Area	350	–	27.11	27.57	28.03	35.07	35.66	36.41	41	41.32	41.62	**43.64**	**44.14**	**44.43**	-	269.16	25.65	19.72	**1.61**
Area	375	–	27.19	27.66	28.15	33.09	34.23	35.07	38.65	39.1	39.39	**41.77**	**42.04**	**42.28**	–	321.31	26.78	22.25	**2**
Area	400	–	25.93	26.37	26.79	31.32	32	33.03	36.77	37.2	37.78	**39.43**	**39.94**	**40.29**	–	389.55	27.78	24.05	**1.84**
World	50	**100**	99.57	99.72	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	1.21	5.34	0.03	0.45	< **0.01**
World	100	**99.78**	**99.78**	**99.78**	**99.78**	**99.78**	**99.78**	**99.78**	**99.78**	**99.78**	**99.78**	**99.78**	**99.78**	**99.78**	17.39	22.29	7.96	1.29	**0.13**
World	150	**99.85**	98.96	99.13	99.4	99.7	99.7	99.7	**99.85**	**99.85**	**99.85**	**99.85**	**99.85**	**99.85**	342.67	40.9	15.2	2.6	**0.23**
World	200	97.09	96.86	96.96	97.2	98.09	98.19	98.43	**99.55**	**99.55**	**99.55**	**99.55**	**99.55**	**99.55**	3600	77	28.46	4.59	**0.33**
World	250	4.68	94.77	94.89	95.05	96.94	97.05	97.12	**98.2**	**98.2**	**98.2**	**98.2**	**98.2**	**98.2**	3600	119.82	42.88	6.83	**0.48**
World	300	0.53	92.33	93.19	94.13	94.88	95.38	95.71	**97.07**	**97.07**	**97.07**	**97.07**	**97.07**	**97.07**	3600	174.61	61.99	9.74	**0.72**
World	350	–	89.49	90.13	91.55	94.52	94.79	95.23	96.84	96.9	96.91	**96.97**	**96.97**	**96.97**	–	254.77	79.99	13.73	**0.84**
World	400	–	85.45	86.15	87.12	92.5	92.86	93.23	95.11	95.2	95.34	**95.61**	**95.61**	**95.61**	–	373.74	100.12	17.87	**1.18**
World	450	–	81.64	83.04	84.12	91.31	91.73	92.18	93.44	93.53	93.64	**94.17**	**94.24**	**94.27**	–	470.95	121.16	22.82	**1.94**
World	500	–	78.5	80.2	82.11	89.73	90.11	90.47	91.69	91.86	91.99	**92.65**	**92.73**	**92.78**	–	626.95	134.52	28.6	**2.19**
World	550	–	75.21	75.96	77.23	88.23	88.67	89.04	90.78	90.93	91.14	**92.16**	**92.35**	**92.48**	–	754.37	142.44	38.03	**1.78**
World	600	–	71.01	72.35	73.55	86.19	87.21	87.69	89.24	89.43	89.68	**90.89**	**91.06**	**91.18**	–	955.71	153.27	39.71	**2.58**
	Avg	–	63.47	64.31	65.24	69.88	70.51	71.13	74.09	74.34	74.61	**75.32**	**75.51**	**75.64**	–	222.92	42.72	12.77	**0.95**

We compare the solution quality and the CPU time. The solution quality is the percentage of the total revenue of scheduled jobs (i.e., the objective value) divided by the total revenue of all the jobs. "Dis" is short for the distribution type

The best results among the algorithms compared are highlighted in bold

From Fig. 18 we can find that gaps between the performance of ALNS/TPF and those of other algorithms tend to grow when there are more jobs and when jobs are distributed densely (i.e., when the conflict among jobs is higher). The performance of the standard ALNS decreases quickly when the problem grows in size and our proposed ALNS/TPF performs well for large and difficult instances.

TDOPTW problem

The ALNS/TPF algorithm is compared with the ACS algorithm by Verbeeck et al. (2017). A MILP formulation by the CPLEX solver has been tested by Verbeeck et al. (2017), which shows bad performance for large instances with more than 20 instances. Therefore we do not compare ALNS/TPF against CPLEX in this section.

Our ALNS/TPF is run on Intel Core i5-3470 3.20 GHz CPU with 8GB memory, using a single core. For the ACS method, we present the results from the respective article. Since ACS was run on a high performance computing system (48 Intel Xeon processors and 384 GB memory), the runtime of the two algorithms cannot be compared directly. We do not report detailed CPU time here.

The solution quality of this problem is evaluated by the total revenue of the solution (i.e., the value of the objective function). The total revenue of instances with 25–100 jobs is shown in Table 8. ALNS/TPF produces the best solutions on all the instances. Besides, ALNS/TPF performs much better than ACS when the instances grow in size (i.e., more jobs and longer scheduling horizon).

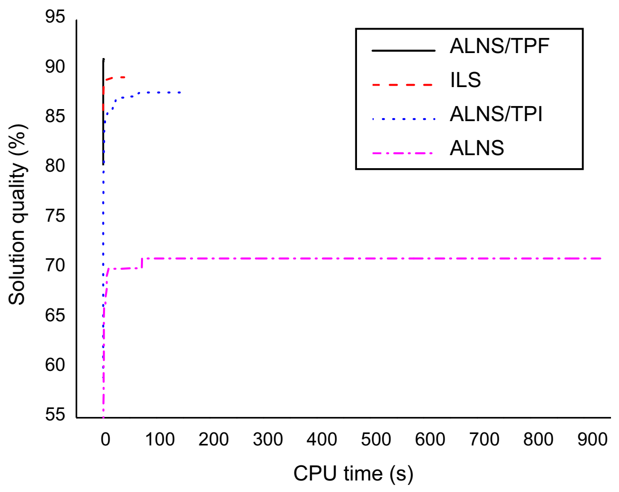
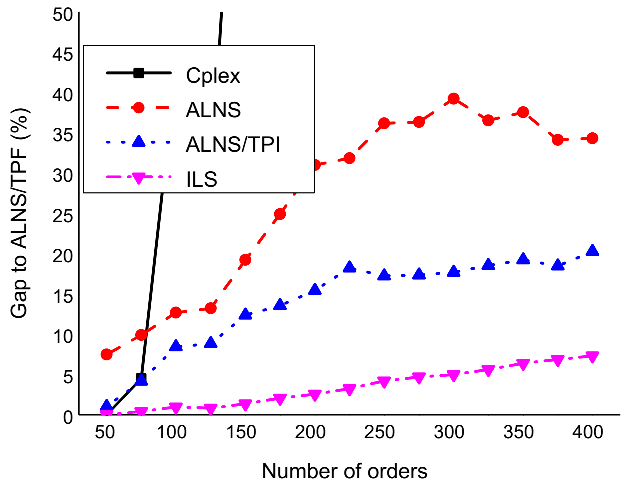


Fig. 17 Anytime quality of different algorithms for the instance with 600 jobs distributed worldwide

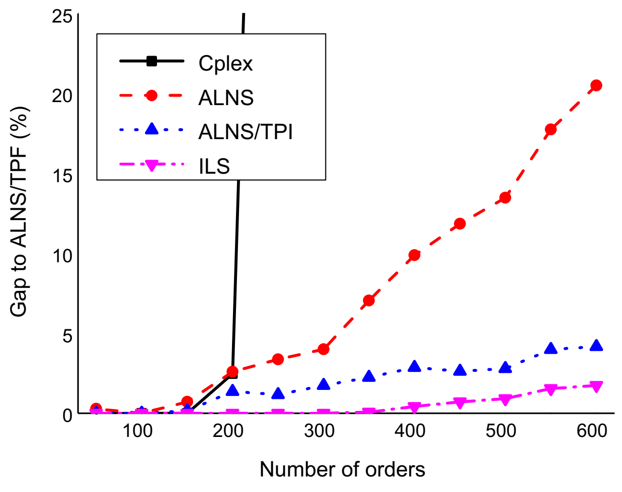
Comparison with CP Optimizer

Finally, we compare ALNS/TPF with the state-of-the-art commercial optimizer for scheduling problems, the IBM ILOG CP Optimizer (CPO), which is widely used in scheduling problems and shown to be very effective for this class of problems (Laborie et al. 2018).

CPO has a global constraint propagator $NoOverlap$ for setup constraints, which is very efficient (Laborie et al. 2018). However, $NoOverlap$ does not support time-dependent constraints of the AEOSS problem and the TDOPTW problem and the ‘max’ term in the setup constraints of the OAS problem mentioned in “Problem datasets” section. It is possible to build a constraint programming model that reasons only locally on direct neighbours of jobs. However, such a model turned out to be too slow to be acceptable. According



(a) Area distribution



(b) Worldwide distribution

Fig. 18 The gaps of different algorithms to ALNS/TPF on the AEOSS problem. Cplex can only find the optimal solution for four instances within the 3600 s time limit

Table 8 Total revenue of ACS and ALNS/TPF on the TDOPTW problem

Name	n	t_{max} (h)	Time windows	ACS			ALNS/TPF		
			S, M, L	Min	Avg	Max	Min	Avg	Max
20.1.1	20	8	S	**159**	**159**	**159**	**159**	**159**	**159**
20.1.2	20	8	M	**173**	**173**	**173**	**173**	**173**	**173**
20.1.3	20	8	L	**183**	**183**	**183**	**183**	**183**	**183**
20.2.1	20	10	S	**188**	**188**	**188**	**188**	**188**	**188**
20.2.2	20	10	M	**201**	**201**	**201**	**201**	**201**	**201**
20.2.3	20	10	L	**195**	**195**	**195**	**195**	**195**	**195**
20.3.1	20	12	S	**277**	**277**	**277**	**277**	**277**	**277**
20.3.2	20	12	M	**245**	**245**	**245**	**245**	**245**	**245**
20.3.3	20	12	L	**259**	**259**	**259**	**259**	**259**	**259**
20.4.1	20	14	S	**274**	**274**	**274**	**274**	**274**	**274**
20.4.2	20	14	M	**275**	**275**	**275**	**275**	**275**	**275**
20.4.3	20	14	L	**268**	**268**	**268**	**268**	**268**	**268**
50.1.1	50	8	S	**288**	**288**	**288**	**288**	**288**	**288**
50.1.2	50	8	M	**274**	**274**	**274**	**274**	**274**	**274**
50.1.3	50	8	L	**289**	**289**	**289**	**289**	**289**	**289**
50.2.1	50	10	S	**298**	**298**	**298**	**298**	**298**	**298**
50.2.2	50	10	M	**310**	**310**	**310**	**310**	**310**	**310**
50.2.3	50	10	L	**340**	**340**	**340**	**340**	**340**	**340**
50.3.1	50	12	S	339	339	339	**346**	**346**	**346**
50.3.2	50	12	M	**404**	**404**	**404**	**404**	**404**	**404**
50.3.3	50	12	L	**366**	**366**	**366**	**366**	**366**	**366**
50.4.1	50	14	S	471	476.6	**478**	**478**	**478**	**478**
50.4.2	50	14	M	435	439.8	**441**	**441**	**441**	**441**
50.4.3	50	14	L	**450**	**450**	**450**	**450**	**450**	**450**
100.1.1	100	8	S	**275**	**275**	**275**	**275**	**275**	**275**
100.1.2	100	8	M	**278**	**278**	**278**	**278**	**278**	**278**
100.1.3	100	8	L	**343**	**343**	**343**	**343**	**343**	**343**
100.2.1	100	10	S	**351**	**351.2**	**352**	**351**	**351.2**	**352**
100.2.2	100	10	M	366	366.6	**367**	**367**	**367**	**367**
100.2.3	100	10	L	**370**	**370**	**370**	**370**	**370**	**370**
100.3.1	100	12	S	435	436	**437**	**437**	**437**	**437**
100.3.2	100	12	M	444	446.6	**454**	**449**	**453.5**	**454**
100.3.3	100	12	L	466	467	468	**470**	**470**	**470**
100.4.1	100	14	S	478	480	**484**	**482**	**483.8**	**484**
100.4.2	100	14	M	491	494.6	**497**	**495**	**496.8**	**497**
100.4.3	100	14	L	519	526.8	538	**538**	**539.6**	**540**
			Avg.	327.1	327.9	328.8	**328.7**	**329**	**329.1**

The best results among the algorithms compared are highlighted in bold

to the experiments with time limit of ten minutes, for the AEOSS problem, it only finds feasible solutions for 5 out of 27 instances tested; for the OAS problem, it only finds feasible solutions for 20 out of 75 instances tested; for the TDOPTW problem, it finds no feasible solution for all the instances within the time limit. The solution quality is also poor compared with the solutions found by the metaheuristics. According to Aguiar Melgarejo (2016), a CP model with such constraints reasoning locally on direct neighbours is around 100 times slower than the model with the global constraint propagator. Clearly, CPO is not suitable for the three problem domains studied in this paper.

To compare ALNS/TPF against CPO with global constraint $NoOverlap$, we relaxed the setup constraint of the OAS problem as $p_i + d_i + s_{ij} \leq p_j$. Note that we do not do this on the other two problems because relaxing the time-dependent setup constraints will make them too different from the original ones. We compare the total revenue (i.e., the

Table 9 Comparison between ALNS/TPF and CPO on relaxed OAS problem

$n = 100$	R	Total revenue		CPU time (s)	
τ		ALNS/TPF	CPO	ALNS/TPF	CPO
0.1	0.1	**1076.10**	1075.00	**27.22**	600.00
	0.3	**979.10**	978.00	**28.79**	600.00
	0.5	**1169.00**	**1169.00**	0.72	**0.09**
	0.7	**1039.00**	**1039.00**	**0.02**	0.09
	0.9	**1163.00**	**1163.00**	**0.02**	0.09
0.3	0.1	**1018.40**	1017.00	**25.83**	600.00
	0.3	**1045.00**	1041.00	**26.89**	600.00
	0.5	**1030.00**	**1030.00**	**27.95**	600.00
	0.7	1184.20	**1185.00**	**18.56**	18.83
	0.9	**1094.00**	**1094.00**	20.07	**1.63**
0.5	0.1	**1217.40**	1210.00	**22.56**	600.00
	0.3	**989.80**	988.00	**25.54**	600.00
	0.5	**1043.50**	1042.00	**27.78**	600.00
	0.7	1030.41	**1032.00**	**28.90**	600.00
	0.9	**1094.59**	1094.00	**30.39**	600.00
0.7	0.1	**1039.70**	1039.00	**20.09**	600.00
	0.3	**1126.60**	1124.00	**22.27**	600.00
	0.5	**985.90**	981.00	**22.96**	600.00
	0.7	**1020.06**	1016.00	**21.86**	600.00
	0.9	**1019.48**	1014.00	**31.77**	600.00
0.9	0.1	**1027.10**	1027.00	**13.60**	600.00
	0.3	975.10	**976.00**	**13.69**	600.00
	0.5	**897.21**	889.00	**16.25**	600.00
	0.7	**983.56**	**983.56**	**17.64**	600.00
	0.9	**928.57**	927.24	**19.10**	600.00
Avg.		**1047.07**	1045.35	**20.42**	480.83

The best results among the algorithms compared are highlighted in bold

value of the objective function) and CPU time of ALNS/TPF and CPO on the relaxed problem, with time limit of ten minutes for each instance. Other settings are as in "OAS problem" section. In this experiment, we only run the algorithms on the first instance in each group of the OAS instances with 100 jobs and the same τ and R, because there are 10 instances in each group, and running the algorithms on all the instances takes too long time. The results of ALNS/TPF and CPO are shown in Table 9. The values of ALNS/TPF are the average of ten runs. It is clear that ALNS/TPF produces better solutions with less time compared with CPO.

Conclusions

The application of artificial intelligence techniques in the intelligent manufacturing industry offers a crucial opportunity to improve productivity and profit (Rao et al. 1999). This article studied an important class of over-subscribed scheduling problems in the intelligent manufacturing industry, which is characterised by time-dependency and/or sequence-dependency with time windows. We developed a novel hybridization of adaptive large neighbourhood search (ALNS) and tabu search (TS). We further introduced randomized generic neighbourhood operators, a partial sequence dominance heuristic, and a fast insertion strategy to the ALNS-TS hybridization. A detailed analysis of the algorithmic features provided evidence that: (1) the insertion tabu and the removal tabu work better when the completion ratio is low, and the instant tabu works better when there are fewer jobs in the solution sequence; (2) the randomization strategy in the neighbourhood operators works well in terms of the solution quality and the running time; (3) the partial sequence dominance heuristic performs better when the problem instance grows in size, indicating that it helps to combine parts of different solutions, when the solution sequence gets long; and (4) the fast insertion strategy contributes most to the performance, but also consumes the most time compared with other features.

An extensive empirical study on three domains demonstrated that, compared with the state-of-the-art approaches, our proposed ALNS/TPF produces solutions with higher quality in less time. The proposed algorithm is an intermediate approach between general methods and problem-specific methods, which exhibits better efficiency in solving this class of scheduling problems and can be generalized to different problem domains easily. Our work also proves that tight ALNS and TS hybridization is an efficient method for this class of scheduling problems. We believe generalizing this algorithm to other similar scheduling problems such as the multi-machine scheduling problem and the vehicle routing problem is relatively straightforward, since these novel techniques are applicable on such problems with similar sequencing and selecting properties.

In this article, we reported the identified correlations between algorithmic features and problem properties. However, selecting the optimal algorithmic features for new problems remains difficult. A research challenge remaining for further work is to let the algorithm learn this automatically. We believe the combination of online machine learning and optimization is a promising research direction towards this goal. The algorithm could learn such correlations during the optimization process and uses this knowledge to tune itself towards different problem domains, so that it can be applied to a set of problems without careful customization. By publishing the source code of our algorithm and providing all problem instances, we aim to facilitate such further study of the use of artificial intelligence techniques for manufacturing, as well as the use of this algorithm for other real-world problem domains.

Acknowledgements We gratefully thank Dr. Yuri Laio T. V. Silva and Dr. Anand Subramanian for providing the tight upper bound of each instance of the OAS problem. Additionally, we thank the three anonymous reviewers for their valuable comments and suggestions. Two preliminary versions of parts of this work appeared at the ICAPS 2018 Scheduling and Planning Applications Workshop (He et al. 2018) and ICAPS 2019 conference (He et al. 2019). This work was supported by the China Scholarship Council for Lei He's 2-year research visit to Delft University of Technology, the Netherlands (Grant No. 201703170269), and a China Hunan Postgraduate Research Innovating Project (Grant No. CX2018B020).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Appendix A: Fast insertion algorithm

The fast insertion algorithm first evaluates the feasibility and the cost of all the positions rapidly by a concept called *time slack*. Then the best position is selected to insert the job. The insertion algorithm is used in the repairing process when we insert jobs back to the solution. The detailed process of the fast insertion algorithm is shown in Algorithm 3.

Time slack and due time slack We set all the jobs to start as early as possible. Therefore when inserting one job into the current solution at a position, it is possible to create more space for the candidate job by postponing some jobs in the solution. In order to determine how much one job can be postponed, we adopt the *time slack* idea from Verbeeck et al. (2017). We further propose the *due time slack* heuristic for problems like the OAS problem from Oğuz et al. (2010) with tardiness penalty. The due time is a time point in the time window of a job. If a job starts after its due time, it receives some penalty on its revenue.

The time slack is defined as the maximum amount of time a job can be postponed before the solution becomes infeasible. The time slack of each job depends on the latest start time of its succeeding job. Thus it is calculated from the last job to the first one in a back-propagation manner. Let o_i be the i^{th} job in the current solution, o_{i+1} be its immediate successor

```
Algorithm 3: Fast insertion algorithm
Input: Current solution $S$, Destroyed solution $S_D$, candidate job $o_c$
Output: The solution $S_D$
1  function FastInsertion($S$, $S_D$, $o_c$)
2      for each scheduled job $o_i$ in $S_D$ do
3          if $e_c > b_i$ then
4              $t_1 \leftarrow$ Calculate the start time of $o_c$ if it is inserted after $o_i$
5              $t_{Temp} \leftarrow$ Calculate the temporary start time of $o_{i+1}$ after $o_c$
6              $t_2 \leftarrow t_{Temp} - p_{i+1}$
7              if $t_1 > e_c \vee t_2 >$ time slack of $o_{i+1}$ then
8                  continue
9              else
10                 if $t_1 \leq \bar{e}_c \wedge t_2 \leq$ due time slack of $o_{i+1}$ then
11                     $Position_i.SetupIncrease \leftarrow s_{ic} + s_{c(i+1)} - s_{i(i+1)}$
12                     Add $Position_i$ into position list $PL1$
13                 else
14                     $Position_i.Fitness \leftarrow$ Calculate the total fitness if $o_c$ is inserted
15                     if $Position_i.Fitness > f(S_D)$ then
16                         Add $Position_i$ into position list $PL2$
17     if $PL1 \neq \emptyset$ then
18         $BestPosition \leftarrow$ Select the position increasing minimum setup time
19         Insert $o_c$ into $S_D$ at $BestPosition$
20         Update start times and time slacks
21     else
22         if $PL2 \neq \emptyset$ then
23             $BestPosition \leftarrow$ Select the position increasing maximum fitness
24             Insert $o_c$ into $S_D$ at $BestPosition$
25             Update start times and time slacks
26     return $S_D$
```

and $|S|$ be the number of jobs in the current solution S. The latest start time of o_i is calculated by:

$$p_i^{Late} = \begin{cases} \max\{\min\{p_{i+1}^{Late} - s_{i(i+1)} - d_i, e_i\}, b_i\} & \text{if } 1 \leq i < |S| \\ e_i & \text{if } i = |S| \end{cases} \tag{11}$$

The latest start time of the last job in the solution is the latest start time defined by its time window. Then the time slack of o_i can be calculated by $p_i^{Late} - p_i$.

The due time slack is the maximum amount of time a job can be postponed without adding any penalty to any job. Similarly, to calculate the due time slack of a job, the latest start time of the job without receiving any penalty should be calculated. Let $b_i < \bar{e}_i < e_i$ be a time point called the due time in the time window of o_i, after which if the job is started, it receives some penalty on its revenue. The latest start time of the job without receiving any penalty is

$$p_i^{DueLate} = \begin{cases} \max\{\min\{p_{i+1}^{DueLate} - s_{i(i+1)} - d_i, \bar{e_i}\}, b_i\} & \text{if } 1 \leq i < |S| \\ \bar{e_i} & \text{if } i = |S| \end{cases} \tag{12}$$

Then the due time slack of o_i can be calculated by $p_i^{DueLate} - p_i$.

These heuristics facilitate determining the feasibility and the cost of one insertion only by comparing the time needed with the corresponding slack.

Best position selection For every candidate job, we calculate all possible insertion positions by comparing its time window with the current solution. For each possible solution, we do the following evaluation: Suppose we are evaluating the position between jobs o_i and o_{i+1} for the candidate job o_c, we calculate the start time of o_c if it is inserted after o_i (denoted as t_1) and the time o_{i+1} needed to be postponed (denoted as t_2). If t_1 is bigger than the latest start time of o_c or t_2 is bigger than the time slack of o_{i+1}, the position is given up; if t_1 is smaller than the due time of o_c and t_2 is smaller than the due time slack (note that the due time slack is always smaller than the time slack), we calculate the increase of setup time if inserting o_c, which is $s_{ic} + s_{c(i+1)} - s_{i(i+1)}$ and add the position to candidate position list 1, PL_1; otherwise (i.e., if t_1 is larger than the due time of o_c or t_2 is larger than the due time slack), we calculate the total fitness of the solution if we insert the job at this position and if it increases the fitness, we add the position to the candidate position list 2, PL_2. Finally, if PL_1 is not empty, the position with the smallest value of the increase of setup time in PL_1 is selected to insert the job; otherwise, the position with the highest total fitness in PL_2 is selected to insert the job. If both PL_1 and PL_2 are empty, the candidate job is given up.

We select the position according to the above strategy because when PL_1 is not empty, the candidate job can be inserted without receiving any penalty, which means the total fitness can be increased with the revenue of the candidate job. In this case we select the position increasing the minimum setup time. The rationale is that it is better to use the time more for processing jobs instead of setting up. If PL_1 is empty, some jobs will receive a penalty. In this case we have to compute the fitness to find the best insertion position.

When a job is inserted, the start times of all its succeeding jobs are updated until one whose start time does not change. The time slacks of all the jobs before the candidate job and the jobs whose start time is changed are also updated.

References

Abbaspour, R. A., & Samadzadegan, F. (2011). Time-dependent personal tour planning and scheduling in metropolises. *Expert Systems with Applications*, *38*(10), 12439–12452.

Aguiar-Melgarejo, P. (2016). A constraint programming approach for the time dependent traveling salesman problem. *Ph.D. thesis*, INSA Lyon.

Akturk, M. S., & KiliÇ, K. (1999). Generating short-term observation schedules for space mission projects. *Journal of Intelligent Manufacturing*, *10*(5), 387–404.

Augenstein, S., Estanislao, A., Guere, E., & Blaes, S. (2016). Optimal scheduling of a constellation of earth-imaging satellites, for maximal data throughput and efficient human management. In: *Proceedings of the 26th international conference on automated planning and scheduling (ICAPS 2016)* (pp. 345–352).

Bianchessi, N., Cordeau, J. F., Desrosiers, J., Laporte, G., & Raymond, V. (2007). A heuristic for the multi-satellite, multi-orbit and multi-user management of Earth observation satellites. *European Journal of Operational Research*, *177*(2), 750–762.

Cesaret, B., Oğuz, C., & Salman, F. S. (2012). A tabu search algorithm for order acceptance and scheduling. *Computers and Operations Research*, *39*(6), 1197–1205.

Chaurasia, S. N., & Kim, J. H. (2019). An artificial bee colony based hyper-heuristic for the single machine order acceptance and scheduling problem. In *Decision science in action* (pp. 51–63). Springer, New York.

Chaurasia, S. N., & Singh, A. (2017). Hybrid evolutionary approaches for the single machine order acceptance and scheduling problem. *Applied Soft Computing*, *52*, 725–747.

Chen, C., Yang, Z., Tan, Y., & He, R. (2014). Diversity controlling genetic algorithm for order acceptance and scheduling problem. *Mathematical Problems in Engineering*, *2014*, 1–11.

Cordeau, J. F., & Laporte, G. (2005). Maximizing the value of an Earth observation satellite orbit. *Journal of the Operational Research Society*, *56*(8), 962–968.

Cordeau, J. F., Laporte, G., & Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, *52*(8), 928–936.

Demir, E., Bektaş, T., & Laporte, G. (2012). An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research*, *223*(2), 346–359.

Dong, W. C., Lee, Y. H., Lee, T. Y., & Gen, M. (2014). An adaptive genetic algorithm for the time dependent inventory routing problem. *Journal of Intelligent Manufacturing*, *25*(5), 1025–1042.

Duan, C., Chao, D., Gharaei, A., Wu, J., & Wang, B. (2018). Selective maintenance scheduling under stochastic maintenance quality with multiple maintenance actions. *International Journal of Production Research*, *2*, 1–19.

Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., & Linaza, M. T. (2013). Integrating public transportation in personalised electronic tourist guides. *Computers and Operations Research*, *40*(3), 758–774.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, *13*(5), 533–549.

Gunawan, A., Lau, H. C., & Vansteenwegen, P. (2016). Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, *255*(2), 315–332.

He, L., De Weerdt, M., & Yorke-Smith, N. (2019). Tabu-based large neighbourhood search for time/sequence-dependent scheduling problems with time windows. In *Proceedings of 29th international conference on automated planning and scheduling (ICAPS'19)* (pp. 186–194). Berkeley, CA.

He, L., De Weerdt, M., Yorke-Smith, N., Liu, X., & Chen, Y. (2018). Tabu-based large neighbourhood search for time-dependent multi-orbit agile satellite scheduling. In *Proceedings of the ICAPS'18 scheduling and planning applications workshop* (pp. 45–52).

He, L., Guijt, A., De Weerdt, M., et al. (2019). Order acceptance and scheduling with sequence-dependent setup times: a new memetic algorithm and benchmark of the state of the art[J]. *Computers & Industrial Engineering*, *138*, 106102. https://doi.org/10.1016/j.cie.2019.106102.

Jacobs, F. R., Berry, W. L., Whybark, D. C., & Vollmann, T. E. (2010). *Manufacturing planning and control for supply chain management* (6th ed.). New York: McGraw-Hill.

Laborie, P., Rogerie, J., Shaw, P., & Vilím, P. (2018). IBM ILOG CP optimizer for scheduling. *Constraints*, *23*(2), 210–250.

Lemaître, M., Verfaillie, G., Jouhaud, F., Lachiver, J. M., & Bataille, N. (2002). Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology*, *6*(5), 367–381.

Lin, S. W., & Ying, K. (2013). Increasing the total net revenue for single machine order acceptance and scheduling problems using an artificial bee colony algorithm. *Journal of the Operational Research Society*, *64*(2), 293–311.

Liu, X., Laporte, G., Chen, Y., & He, R. (2017). An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time. *Computers and Operations Research*, *86*, 41–53.

Mirsanei, H. S., Zandieh, M., Moayed, M. J., & Khabbazi, M. R. (2011). A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, *22*(6), 965–978.

Nguyen, S. (2016). A learning and optimizing system for order acceptance and scheduling. *The International Journal of Advanced Manufacturing Technology*, *86*(5–8), 2021–2036.

Nguyen, S., Zhang, M., & Tan, K. C. (2015). A dispatching rule based genetic algorithm for order acceptance and scheduling. In *Proceedings of the 16th annual conference on genetic and evolutionary computation (GECCO 2015)* (pp. 433–440). ACM.

Oğuz, C., Salman, F. S., Yalçın, Z. B., et al. (2010). Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics*, *125*(1), 200–211.

Peng, G., Vansteenwegen, P., Liu, X., Xing, L., & Kong, X. (2018). An iterated local search algorithm for agile earth observation satellite scheduling problem. In *Proceedings of the 15th conference on space operations (SpaceOps 2018)* (p. 2311).

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers and Operations Research*, *34*(8), 2403–2435.

Poggi, M., Viana, H., & Uchoa, E. (2010). The team orienteering problem: Formulations and branch-cut and price. In *10th Workshop on algorithmic approaches for transportation modelling, optimization, and systems (ATMOS'10)* (pp. 142–155).

Prins, C., Prodhon, C., Ruiz, A., Soriano, P., & Wolfler Calvo, R. (2007). Solving the capacitated location-routing problem by a cooperative lagrangean relaxation-granular tabu search heuristic. *Transportation Science*, *41*(4), 470–483.

Rao, S. S., Nahm, A., Shi, Z., Deng, X., & Syamil, A. (1999). Artificial intelligence and expert systems applications in new product development: A survey. *Journal of Intelligent Manufacturing*, *10*(3–4), 231–244.

Rebai, M., Kacem, I., & Adjallah, K. H. (2012). Earliness–tardiness minimization on a single machine to schedule preventive maintenance tasks: metaheuristic and exact methods. *Journal of Intelligent Manufacturing*, *23*(4), 1207–1224.

Rogers, M. F., Howe, A. E., & Whitley, D. (2006). Looking for shortcuts: Infeasible search analysis for oversubscribed scheduling problems. In *Proceeding of the 16th international conference on automated planning and scheduling (ICAPS 2006)* (pp. 314–323).

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, *40*(4), 455–472.

Silva, Y. L. T., Subramanian, A., & Pessoa, A. A. (2018). Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times. *Computers and Operations Research*, *90*, 142–160.

Slotnick, S. A. (2011). Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research*, *212*(1), 1–11.

Thomas, C., & Schaus, P. (2018). Revisiting the self-adaptive large neighborhood search. In *Proceeding of the 15th international conference on the integration of constraint programming, artificial intelligence, and operations research (CPAIOR 2018)* (pp. 557–566).

Verbeeck, C., Vansteenwegen, P., & Aghezzaf, E. H. (2017). The time-dependent orienteering problem with time windows: A fast ant colony system. *Annals of Operations Research*, *254*(1–2), 481–505.

Wang, B., Guan, Z., Ullah, S., Xu, X., & He, Z. (2017). Simultaneous order scheduling and mixed-model sequencing in assemble-to-order production environment: A multi-objective hybrid artificial bee colony algorithm. *Journal of Intelligent Manufacturing*, *28*(2), 419–436.

Žulj, I., Kramer, S., & Schneider, M. (2018). A hybrid of adaptive large neighborhood search and tabu search for the order-batching problem. *European Journal of Operational Research*, *264*(2), 653–664.