Tabu-Based Large Neighbourhood Search for Time/Sequence-Dependent Scheduling Problems with Time Windows

Paper #74

Abstract

An important class of scheduling problems is characterised by time-dependency and/or sequence-dependency with time windows. We introduce and analyze four algorithmic ideas for this class: a novel hybridization of adaptive large neighbourhood search (ALNS) and tabu search (TS), randomized generic neighbourhood operators, a partial sequence dominance heuristic, and a fast insertion strategy. An evaluation of the resulting hybrid algorithm on two domains, a real-world multi-orbit agile Earth observation satellite scheduling problem, and an order acceptance and scheduling problem, shows that it robustly outperforms a mixed integer programming method, a two-stage hybridization of ALNS and TS, and recent state-of-the-art metaheuristic methods.

Introduction

An over-subscribed scheduling problem, where the capacity cannot meet the demand, consists of simultaneously selecting a subset of orders 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), the order acceptance and scheduling problem (Oğuz et al. 2010), and the orienteering problem (Verbeeck, Vansteenwegen, and Aghezzaf 2017). Many real-world instances in this class have time/sequence-dependent setup time and time windows: the setup time between every two orders depends on the specific pair of orders or their scheduled start times, and the scheduled start time of each order must be in its time window.

The adaptive large neighbourhood search (ALNS) algorithm and tabu search (TS) algorithm show promising performance for such problems (Liu et al. 2017; Pisinger and Ropke 2007; Cesaret, Oğuz, and Salman 2012). ALNS was first proposed by Ropke and Pisinger (2006) for pickup and delivery problem with time windows. It provides a flexible framework in which a portfolio of operators can be defined according to the problem characteristics. Thanks to the adaptive mechanism for the weights of multiple operators, it can provide robust solutions for instances with different characteristics. TS was first proposed by Glover (1986). In TS, recently visited solutions are stored in a tabu list to prevent short-term cycling.

Žulj, Kramer, and Schneider (2018) proposed the first hybridization of ALNS and TS and found that a simple hybridization of these two metaheuristics outperformed both the standalone methods. 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.

The major contributions of this paper are as follows:

1. In contrast to a two-stage hybridization, we propose a tight hybridization of ALNS and TS. Our novel hybrid ALNS–TS approach provides results with higher quality and robustness and consumes less time compared with state of the art. The tabu mechanism helps the ALNS to avoid searching recently visited solutions.
2. We observe there exists a correlation between the tabu and randomness types and the completion ratio of problem instances, which helps to tune the algorithm for over-subscribed problems.
3. We introduce a partial sequence dominance heuristic, which greatly improves the performance of ALNS, especially when the problem instances grow in size.
4. We develop a fast insertion algorithm consisting of insertion position ordering and time slack strategies. The method finds the optimal insertion for each order and rapidly determines insertion feasibility and cost.

Altogether, our hybrid algorithm, called ALNS/TPF, exhibits robust performance across a range of instances of over-subscribed scheduling problems with time/sequence-dependent setup time and time windows.

Background

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

Mathematical formulation

Consider a set of orders $O = \{o_1, ..., o_n\}$ that can be potentially scheduled. The sequence of orders is not fixed. Each

order 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 decision variable representing whether order o_i is selected and p_i be a decision variable representing the start time of o_i. The problem can be formulated as a mixed integer programming (MIP) model:

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

subject to

$$\begin{gathered} p_i + d_i + s_{ij} \leq p_j \\ \forall i, j \in \{1, ..., n\} \quad \text{if} \quad x_i = 1, x_j = 1, p_i < p_j \end{gathered} \tag{2}$$

$$b_i \leq p_i \leq e_i \quad \forall i \in \{1, ..., n\} \quad \text{if} \quad x_i = 1 \tag{3}$$

$$x_i \in \{0, 1\} \quad \forall i \in \{1, ..., n\} \tag{4}$$

The objective function (1) maximizes the total revenue of scheduled orders. Constraints (2) restrict the time between every two orders should be long enough for the setup, where s_{ij} is the setup time between orders o_i and o_j. The value of s_{ij} depends on i and j for the sequence-dependent setup time case and depends on p_i and p_j for the time-dependent setup time case. Constraints (3) and (4) define the domains of the decision variables p_i and x_i respectively.

Domain instances

Due to the large number of problem variants and solution approaches, the reader is referred to (Slotnick 2011; Gunawan, Lau, and Vansteenwegen 2016) for comprehensive surveys on this class of problems.

The Earth observation satellite scheduling (EOSS) problem is a typical representative of this problem class, where the satellite can only observe a subset of the user-specified orders in a limited time horizon and the transition time between two orders is time/sequence-dependent. Liu et al. (2017) and Peng et al. (2018) studied the time-dependency of the agile satellite observation scheduling problem. Liu et al. proposed an ALNS algorithm, where they also integrated ALNS with an insertion algorithm considering time-dependency by introducing forward/backward time slacks. Peng et al. proposed an iterated local search (ILS) algorithm. They further calculated the minimal transition time, the neighbours and earliest/latest start time of each order to accelerate the insertion.

The order acceptance and scheduling problem (OAS) is another important problem domain, arising for instance when a company does not have the capacity to meet demand. Oğuz et al. (2010) studied the OAS problem with sequence-dependent setup times and penalty of late completion. Compared with the time windows in the EOSS problem, those in the OAS are much longer. The problem was approached by TS (Cesaret, Oğuz, and Salman 2012), genetic algorithm (Nguyen, Zhang, and Tan 2015) and hyper-heuristic based methods (Nguyen 2016). Recently, Silva, Subramanian, and Pessoa (2018) proposed an iterated local search algorithm and used Lagrangian relaxation and column generation to find tight upper bounds of problem instances.

Despite all this work, there is no method capable of finding good solutions to diverse real life instances within the available solving time.

Algorithm 1 Overview of ALNS/TPF

```
Generate an initial solution S^I by greedy search[1];
Set S^I as the current and the best solution: S ← S^I, S* ← S^I;
repeat
    Choose destroy, repair operators D_i, R_i based on weights;
    S' ← R_i(D_i(S));
    Update tabu attributes of all removed and inserted orders;
    Produce compound solution S_c from S and S';
    if f(S_c) > f(S') then
        S' ← S_c;
    end if
    if SA accepts S' then
        S ← S';
    end if
    if f(S) > f(S*) then
        S* ← S;
    end if
    Update the weights of operators;
until Terminal condition is met;
return S*;
```

Standard ALNS framework

ALNS in particular is one of the most promising approaches. It starts from an initial solution usually generated by a simple heuristic, because it is less sensitive to the initial solution than general local search (Demir, Bektaş, and Laporte 2012). ALNS proceeds to generate new solutions through destroying and repairing. In the destroying process, some orders are removed from the current solution by removal operators. The unscheduled and removed orders are then inserted into the destroyed solution in the repairing process by insertion operators. There are multiple removal and insertion operators. At each iteration, a pair of removal and insertion operators is selected by a roulette wheel mechanism according to their weights. The weight of the operator w_i is updated adaptively 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 $\lambda \in [0, 1]$ is a reaction factor which controls how sensitive the weights are to changes in the performance of operators.

A simulated annealing (SA) criterion is used to control the acceptance of new solutions by a temperature parameter T. Let $f(S)$ and $f(S')$ be the reward of current solution S and new solution S' respectively. The new solution S' is accepted if $f(S') > f(S)$; otherwise, it is accepted with probability: $\rho = \exp\left(\frac{100}{T}\left(\frac{f(S')-f(S)}{f(S)}\right)\right)$.

ALNS/TPF: Tabu-based ALNS Algorithm

In this section, we introduce four new algorithmic features in our approach: 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.

[1]We sort the orders by an ascending order of begin times of their time windows and we attempt to start each order as early as possible under all the constraints.

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. As noted earlier, Žulj, Kramer, and Schneider (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. We declare a removal tabu attribute and an insertion tabu attribute for each order. Whenever an order o_i is inserted into the current solution, the removal of o_i is forbidden for the following θ iterations; whenever an order o_j is removed from the current solution, its reinsertion is forbidden for θ' iterations (Algorithm 1, Line 6). For the values of θ and θ', we follow Cordeau and Laporte (2005) and set both to $\sqrt{n/2}$, where n is the number of orders. We compare the two tabu types and the two ALNS–TS hybridizations in the experiments.

Randomized generic neighbourhood operators

In order to ensure the ALNS is suitable for a diverse range of problem instances with varying characteristics, we propose five generic removal operators and five generic insertion operators, and introduce a simple but effective randomization strategy to diverse the search.

The five removal operators are: min revenue (orders with lower revenue are removed first); min unit revenue (orders with lower unit revenue are removed first: the unit revenue is the order's revenue divided by its processing time); max setup time (orders with longer setup time are removed first); max opportunity (for the problems where orders have multiple time windows, orders with more time windows are removed first: the rationale of this operator is that these orders can be scheduled in other time windows easily); and max conflict (orders with higher conflict degree are removed first). The conflict degree of the time window tw_i is:

$$cd_i = \frac{\sum_{tw_j \in TW, i \neq j} TimeSpan(tw_i, tw_j)}{e_i + d_i - b_i} \qquad (5)$$

where TW is the set of all the time windows and the function $TimeSpan$ calculates the time span that two time windows overlap with each other.

The five insertion operators are: max revenue; max unit revenue; min setup time (due to the time/sequence-dependency, the accurate setup time cannot be calculated until the order is inserted in the solution; therefore for this operator, the average setup time of orders is calculated and used to rank the orders); min opportunity; and min conflict.

As mentioned above, the operators are selected adaptively by the roulette wheel mechanism according to their accumulated score in previous iterations. After selecting the removal/insertion operators, standard ALNS ranks the orders 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 orders are ranked in an ascending order of h, and the orders 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 orders 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.

Algorithm 2 Process of constructing a compound solution

1: **Input:** Current solution S, New solution S', the length (i.e., the number of orders) of partial sequences l;
2: Let S_c be the compound solution: $S_c \leftarrow \emptyset$;
3: Let S_p and S'_p be the partial sequence in the current and the new solution respectively: $S_p \leftarrow \emptyset$, $S'_p \leftarrow \emptyset$;
4: **for** $i \leftarrow 1, i \leq \#S, i{+}{+}$ **do**
5: Put the i^{th} order o_i of S in the partial sequence S_p;
6: **if** $\#S_p = l \vee i = \#S$ **then**
7: **for** $j \leftarrow 1, j \leq \#S', j{+}{+}$ **do**
8: **if** end time of o_j in S' $<$ end time of the last order in S_p **then**
9: Put o_j in the partial sequence S'_p;
10: **end if**
11: **if** $f(S_p) > f(S'_p)$ **then**
12: Add S_p into S_c;
13: **else**
14: Add S'_p into S_c;
15: **end if**
16: **end for**
17: **end if**
18: $S_p \leftarrow \emptyset$, $S'_p \leftarrow \emptyset$;
19: **end for**
20: **return** S_c;

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 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 its partial sequences. Inspired by genetic algorithms, we propose the partial sequence dominance (PSD) heuristic: when a 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 total revenue (i.e., objective function value) is stored in a temporary solution called the compound solution (Algorithm 1, line 7). The detailed process of constructing a compound solution from two solutions is shown in Algorithm 2.

A challenge for the PSD heuristic is that one order can appear in different partial sequences of the current solution and the new solution. Thus one order might be processed twice in the compound solution. To maintain the feasibility, all the repetitive orders in the compound solution are removed. If the repaired compound solution is better than the new solu-

Algorithm 3 Fast insertion algorithm

Input: Destroyed solution S_D, Set of unscheduled orders B;
Let S' be the repaired solution: $S' \leftarrow S_D$;
Sort the orders in B by the selected insertion operator;
for each candidate order o_c in B **do**
  for each scheduled order o in S_D **do**
    if $b_c <$ the end time of $o < e_c$ **then**
      Add the position after o in position list PL;
    end if
  end for
  Sort the positions in PL by the ascending setup time;
  for each position in PL **do**
    Let o_p and o_s be the preceding and succeeding orders of the position repectively;
    Let p_p and p_s be the current start time of o_p and o_s;
    Calculate earliest start time p_e of o_c according to p_p;
    Calculate earliest start time p'_e of o_s according to p_e;
    if $p'_e - p_s <$ time slack of o_s **then**
      Insert o_c at the current position in S';
      Update the start time of orders after o_c;
      Update the time slack of orders before the last order whose start time is changed;
      break
    end if
  end for
end for
return repaired solution S';

tion, it is accepted.

Fast insertion algorithm

Our last innovation is a fast insertion algorithm, which first evaluates and ranks all the possible insertion positions, by an insertion position ordering (IPO) heuristic. Then the feasibility and the cost of the positions are rapidly determined by a concept called *time slack*. The insertion algorithm is used in the repairing process when we insert orders back to the solution (Algorithm 1, Line 5). The detailed process of the fast insertion algorithm is shown in Algorithm 3.

In the IPO heuristic, for every candidate order, we calculate all possible insertion positions by comparing its time window with the current solution. Due to the time-dependency and sequence-dependency, the difference of setup times in different insertion positions can be large. We calculate the possible setup time for each position and insert the order into the positions following an ascending order of possible setup times. The rationale is that it is better to use time for processing instead of setup.

For the time-dependent setup time case, because we cannot know the start time until we insert the order into the solution sequence, we cannot know the exact setup time. Therefore, we use the time at the middle of the windows to compute an approximate setup time. This value is used to rank the possible positions. On a time-dependent Earth observation scheduling problem benchmark, the error of this approximation of the setup time is less than 15%.

We set all the orders to start as early as possible. Therefore when inserting one order into the current solution at the position selected by the IPO heuristic, it is possible to create more space for the candidate order by postponing some orders in the solution. In order to determine how much one order can be postponed, we adopt an idea from Verbeeck, Vansteenwegen, and Aghezzaf (2017) and propose the *time slack* and the *due time slack* heuristics. First, the time slack is defined as the maximum amount of time an order can be postponed before the solution becomes infeasible. The time slack of each order depends on the latest start time of its succeeding order. Thus it is calculated from the last order to the first one in a back-propagation manner. The due time slack is defined for the problem with soft time windows where the order receives some penalty because of ending after its due time. It is the maximum amount of time an order can be postponed without adding penalty to any order.

These heuristics facilitate determining the feasibility and the cost of one insertion only by comparing the time needed with the corresponding slack. Insertions with higher cost are considered later. When an order is inserted, the start times of all its succeeding orders are updated until one whose start time does not change.

Algorithmic Analysis

The proposed ALNS/TPF algorithm consists of four novel features, as just described. In order to understand how to effectively use the discussed algorithmic features, we first analyze them individually (both theoretically and empirically). Then in Table 1, for each new feature, we compare the algorithm without this feature against the full algorithm to understand its performance.

The benchmark instances we use are from Cesaret, Oğuz, and Salman (2012) due to their varying characteristics. We only test the larger instances with 25, 50 and 100 orders. Two main parameters were used to generate these instances. The first parameter, τ, influences the length of time windows: when τ is larger, the time windows are smaller; the second 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 parameters 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.

Tabu search and randomness

In the last section we proposed two types of tabu heuristics, the insertion tabu and the removal tabu. The first type is more common in over-subscribed problems (Bianchessi et al. 2007; Cordeau, Laporte, and Mercier 2001; Cordeau and Laporte 2005; Rogers, Howe, and Whitley 2006; Prins et al. 2007). The only removal tabu we found in the literature is from Rogers, Howe, and Whitley (2006). However, their strategy is for updating an infeasible solution by inserting orders first and then removing orders. 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).

We observe an interesting fact that for over-subscribed problems, the performance of the insertion/removal tabu and randomness correlates with the proportion of orders that can be fulfilled, which we call the completion ratio.

Table 1: The first two columns show the average solution quality and CPU time of the full algorithm, and then, for variants with each of the algorithmic features removed, the percentage of increase in gap (IG, lower is better) and increase in time (IT, lower is better) are given. The naming convention is that we include the first letters of the features that are activated. Each row reports the average value of all 250 instances with the same number of orders, and the average value of all 150 instances with the same value of τ and R.

Instances	ALNS/TPF		ALNS/PF		ALNS-TS		ALNS/TF		ALNS/TP (No IPO)		ALNS/TP (Liu et al.)	
	Quality/%	Time/s	IG/%	IT/%	IG/%	IT/%	IG/%	IT/%	IG/%	IT/%	IQ%	IT%
$n = 25$	3.62	1.43	3.78	7.85	6.56	4.02	0.30	-11.05	33.36	-21.28	33.92	19.12
$n = 50$	4.43	6.78	5.36	22.75	10.32	19.33	4.10	-8.40	46.48	-14.86	30.23	35.62
$n = 100$	3.25	36.90	9.50	35.28	12.74	29.56	17.88	-3.46	91.79	-9.98	41.35	44.13
$\tau = 0.1$	0.64	11.66	11.33	62.57	21.52	61.81	0.62	-4.45	240.96	39.08	133.47	96.20
$\tau = 0.3$	1.17	17.48	14.51	48.24	29.03	41.00	5.07	3.30	192.60	-4.91	110.19	58.24
$\tau = 0.5$	2.60	20.34	11.11	24.28	21.10	16.36	7.83	-3.63	111.90	-25.64	65.23	28.24
$\tau = 0.7$	6.12	15.82	5.53	16.76	8.03	10.75	6.86	-7.44	39.99	-27.89	26.94	19.70
$\tau = 0.9$	8.30	9.90	3.16	11.30	3.88	10.50	7.18	-15.17	15.02	-24.38	12.24	13.60
$R = 0.1$	3.05	15.32	6.10	9.21	10.62	3.60	5.14	-3.64	44.66	-27.26	55.21	59.74
$R = 0.3$	3.82	17.13	5.11	17.68	9.07	16.94	6.52	-0.33	45.25	-23.18	38.67	44.61
$R = 0.5$	4.09	18.32	4.87	25.85	9.06	24.70	5.96	-6.12	48.20	-22.52	32.85	30.36
$R = 0.7$	3.98	13.53	6.57	54.84	10.21	45.57	7.43	-7.10	62.91	6.57	27.12	38.07
$R = 0.9$	3.89	10.90	7.45	71.96	10.10	57.78	8.67	-6.07	73.03	27.94	23.75	37.62

To understand the correlation, we run experiments on the benchmark instances. The instances from Cesaret, Oğuz, and Salman (2012) have a relatively high completion ratio. Therefore we also generate three new sets of instances with varying completion ratios. The average results of the percentage of instances where the algorithm with a certain heuristic achieves the best solution are shown in Table 2.

According to these results, the insertion tabu works better when the completion ratio is lower, while the removal tabu works better when the completion ratio is higher. These results can be explained as follows. For insertion tabu, when the completion ratio is low, a large number of orders cannot be included in the solution. The insertion tabu which excludes some bad orders improves the search efficiency. However, when the completion ratio is high, only a small number of orders cannot be included in the solution. The insertion tabu reduces the search space too much and affects the solution quality. For the removal tabu, the effect is opposite. When the completion ratio is low, only a small number of orders can be scheduled, and the removal tabu which includes some orders in the solution reduces the number of solutions that can be explored. However, when the completion ratio is high, a large number of orders can be included in the solution. The insertion tabu which includes some good orders in the solution improves the search efficiency. Similar as the tabu types, the insertion randomness works better when a smaller proportion of orders can be scheduled, while the removal randomness works better when a larger proportion of orders can be scheduled.

Table 2: Percentage of instances where the algorithm with a certain heuristic achieves the best solution

Completion ratio	Tabu type		Randomness type	
	Insertion	Removal	Insertion	Removal
$< 50\%$	8.40	47.33	99.53	41.07
$> 50\%$	4.18	64.31	92.60	58.52

Another interesting observation is that the insertion randomness reduces the performance of the insertion tabu when the completion ratio is high. On average, when the completion ratio is higher than 75%, on 62.07% of the instances the algorithm with the insertion tabu achieves a lower solution quality because of the insertion randomness.

Since all the instances from Cesaret, Oğuz, and Salman (2012) have a relatively high completion ratio, the combination of removal tabu, removal randomness and insertion randomness without insertion tabu works best on average. We use this combination in the following experiments and we refer to it as 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. Table 1, column ALNS/PF, shows that without TS, all the gaps are increased. TS contributes to the solution quality, and it works better when the instance grows in size. It shows better performance when τ is smaller, i.e., when the time window gets longer and the feasible solution space gets larger. The TS strategy helps the algorithm to explore more solutions in the solution space, therefore it has better performance when the time window is longer. TS also reduces the CPU time much by forbidding useless removal of orders from the solution.

We further compare our tight hybridization with the two-stage hybridization of ALNS and TS (ALNS-TS). The full ALNS/TPF is run $1000n$ iterations, where n is the number of orders. In ALNS-TS, TS is run for $15n$ iterations after every $100n$ 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 $1000n$ neighbourhood moves in total. Recently visited solutions are inserted in a tabu list

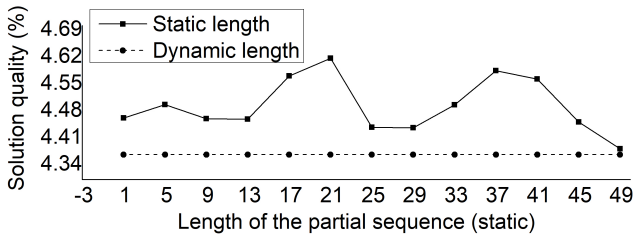

Figure 1: Effect of different partial sequence lengths. The solution quality refers to the average gap to the upper bounds by Cesaret, Oğuz, and Salman (2012).

for $\sqrt{n/2}$ iterations. Note that ANLS-TS might examine more neighbours than $1000n$ in order to find unvisited ones. From column ALNS-TS in the table, it is obvious that the two-stage strategy uses more time and produces worse solutions. The gap increases when the instance gets larger and τ is smaller. 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 this problem.

Partial sequence dominance

Next we study the PSD heuristic. Exploratory experiments have shown us that it is difficult to determine the length of the partial sequence a priori. If it is too long, the good quality of partial sequences can be neglected and PSD becomes useless; if it is too short, the setup time between the last order of the previous partial sequence and the first order of the following partial sequence may be too long so that the compound solution wastes much time, because the setup time between two partial sequences is not optimized by the algorithm. We propose a dynamic strategy to determine the length. It should be as short as possible as long as the increase brought by the setup time between partial sequences is smaller than the time saved by the PSD heuristic. Let l be the number of orders in the partial sequences as follows:

$$l \geq \frac{s_{avg} - s_{avg}^{cur}}{s_{avg}^{cur} - s_{avg}^{com} + d_{avg}^{cur} - d_{avg}^{com}} \tag{6}$$

where s_{avg} is the average setup of all the orders, s_{avg}^{cur} and s_{avg}^{com} are the average setup in the current solution and the compound solution respectively, and d_{avg}^{cur} and d_{avg}^{com} are the average processing time in the current solution and the compound solution respectively. In Eq. (6), the numerator is the loss by the random setup time between two partial sequences. The denominator is the gain of one order by the PSD heuristic.

First, to test the performance of the dynamic strategy, we compare it with different static lengths. The result is shown in Figure 1. While the dynamic length strategy does not dominate on every instance, on average the length given by Eq. (6) is better than any static lengths.

We then can test the performance of PSD by comparing the algorithm without PSD (ALNS/TF) with ALNS/TPF. According to Table 1, PSD does not show obvious improvements when the instance is small. However, when the instance grows in size, the improvement by PSD also grows significantly. PSD works well when the solution sequence gets long, because more partial sequences can be ignored in the long solution sequence. PSD also works better when τ and R are larger. Because when τ is larger, the time window is shorter, and when R is larger, the overlap degree of orders is smaller. If the time window is long and the overlap degree is high, one order can exist in different partial sequences in the current solution and the new solution respectively. Therefore there can be many repetitive orders in the compound solution, reducing its quality.

Fast insertion algorithm

The fast insertion algorithm contains two new ideas: the IPO heuristic and the time slack strategy. We study each in turn. The IPO strategy finds the best insertion for an order. The local optimality 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 an order, the following orders are not considered, the optimal insertion position should be the one that increases least setup time. This is because when we try to insert an order, the revenue and the processing time of the order are fixed. Since the total scheduling horizon is limited, the optimal insertion should be the one that inserts the order successfully (i.e., receives the revenue) as well as maximizes the remaining scheduling space for following orders. The IPO strategy guarantees that if an order can be inserted, the increased setup time is minimal. So IPO finds the optimal insertion of an order.

The algorithm without IPO (ALNS/TP (no IPO)) is compared with ALNS/TPF in Table 1. IPO works significantly better when τ is smaller. This is because 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. IPO also works better when R is larger. The reason is similar to above. When R is larger, the orders in the current solutions have similar time windows. The candidate order to be inserted can neighbour more orders in the solution, resulting in a larger number of possible insertion positions. An extra sorting process is needed for IPO, which increases the CPU time.

Second, 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 orders in the solution, while Liu et al.'s method only creates limited space by moving two orders.

In Table 1, ALNS/TPF with Liu et al.'s strategy is denoted as ALNS/TP (Liu et al.). It is obvious that our time slack strategy uses less time and has higher solution quality. The time slack strategy works better when τ and R are smaller. 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 smaller, the overlap of orders is larger, so when inserting an order, the number of orders that can be postponed gets larger. In this case, the time slack strategy works much better than the simple strategy of Liu et al. (2017), which can only move two orders.

To sum up, from the algorithmic analysis of this section, we derive the following conclusions: (1) the effectiveness of different tabu types and randomness correlates with the completion ratio: the insertion tabu and randomness work well when the completion ratio is low, while the removal tabu and randomness work well when the completion ratio is high; (2) our tight hybridization of ALNS and TS works better than their two-stage hybridization; (3) the dynamic PSD length works better than any static lengths, and 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; (4) IPO contributes most to the solution quality, but also consumes more time; and (5) the new time slack strategy works well in terms of solution quality and time complexity.

Comparison with State-of-the-Art Methods

In this section, our complete ALNS/TPF algorithm is compared with state-of-the-art methods on two different domains. We choose two representative problems, the agile Earth observation satellite scheduling problem (AEOSS) and the order acceptance and scheduling (OAS) problem.

AEOSS problem

We consider the AEOSS problem as in Liu et al. (2017). The transition time between two adjacent observations depends on the observing angles, which differ for different observation start times. Therefore it is not only sequence-dependent, but also time-dependent. The scheduling horizon is 24 hours, which means there are multiple time windows for each observation order. The generation of the problem instances follows the configuration from Liu et al. (2017). The orders 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 orders 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 orders contained in these instances changes from 50 to 600 with an increment of 50.

We compare the proposed ALNS/TPF with our previous algorithm called ALNS/TPI[2], the standard state-of-the-art ALNS (Liu et al. 2017), the ILS algorithm (Peng et al. 2018), and an MIP model. All algorithms are on an Intel Core i5 3.20GHz CPU, 8GB 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 3600s is set for MIP solving. The results for metaheuristics are the average of ten runs.

[2]The reference to this workshop paper is omitted for blind review. ALNS/TPI was developed specifically for the AEOSS.

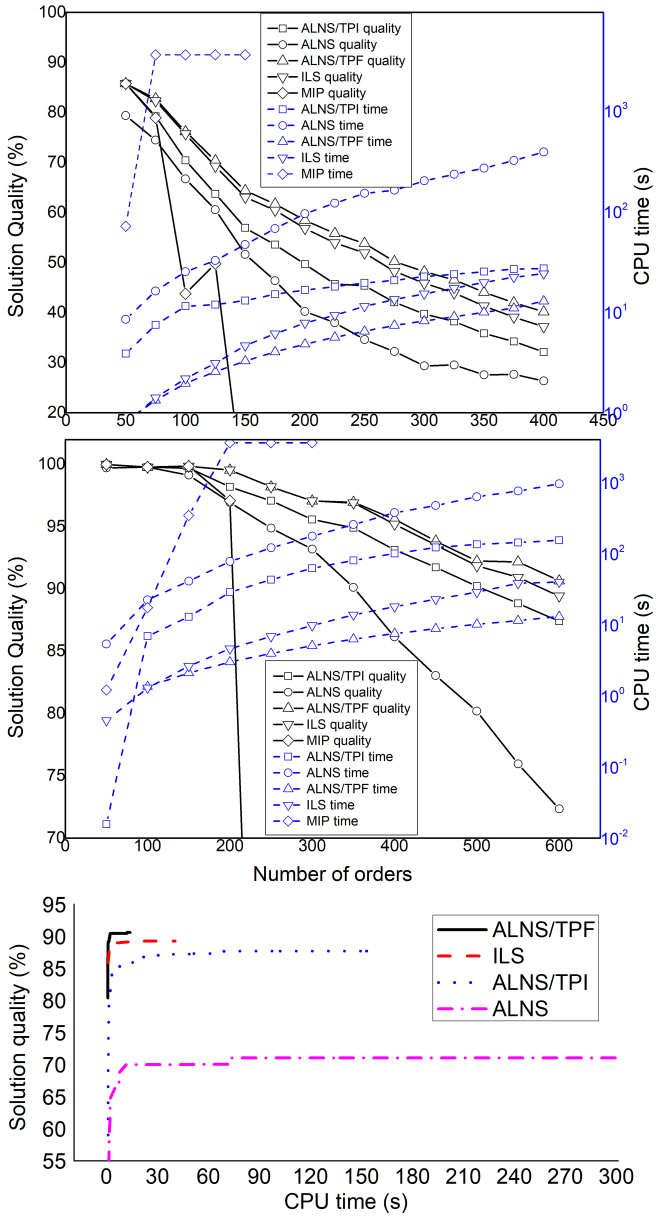

Figure 2: Comparison of algorithms on area distribution (top) and worldwide (middle) and the anytime quality of different algorithms (bottom) for the AEOSS problem

We compare the solution quality and the CPU time. The solution quality is the percentage of the total revenue of scheduled orders (i.e., the objective value) divided by the total revenue of all the orders. Figure 2 shows the comparison of the five different algorithms. In Figure 2 top (for Chinese area) and middle (for worldwide), black solid lines show the solution quality (left axis) and the blue dash lines show the CPU time (right axis, log scale). The CPU time refers to the time used by the MIP solver and the time corresponding to 10,000 iterations by the meta-heuristics, showing that the CPU time of the ALNS/TPF increases slowly with the increasing number of orders. The solution qual-

ity is higher than that of ILS, ALNS/TPI and ALNS. As expected, MIP can find optimal solutions for small-size instances but performs badly when the instance size gets large. For the three small instances with optimal solutions by MIP, 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. According to a paired t-test between the quality of ALNS/TPF and ILS, the P-value is 1.08×10^{-5}, indicating the improvement of the solution quality by ALNS/TPF is significant. Finally, Figure 2 bottom shows the anytime quality of different algorithms for the instance with 600 tasks distributed worldwide (the MIP solver found no feasible solution within the time limit).

OAS problem

We also consider the OAS problem as in Cesaret, Oğuz, and Salman (2012). In this problem, the setup time between two orders is sequence-dependent, and the time window has a due time: if an order o_i ends after its due time $\bar{d}_i$, it receives penalty $\omega_i T_i$ on its revenue, where ω_i is the penalty weight and T_i is the tardiness, $T_i = \max\{p_i + d_i - \bar{d}_i, 0\}$. We use the same benchmark instances as in the section with the initial algorithmic analysis reported earlier.

The ALNS/TPF algorithm is compared with ILS (Silva, Subramanian, and Pessoa 2018), TS (Cesaret, Oğuz, and Salman 2012), DRGA (Nguyen, Zhang, and Tan 2015), GA, HH, and LOS (Nguyen 2016). The MIP solver for OAS has been tested by Cesaret, Oğuz, and Salman (2012). Our ALNS/TPF is run on Intel Core i5 3.20GHz CPU with 8GB memory, using a single core. Since we do not have the source code of other algorithms, we compare our algorithm with the results published in the references. Hence due to the different machines used, 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 LOS the least.

Table 3 reports the gaps to the average upper bounds published by Cesaret, Oğuz, and Salman (2012). Regarding the gaps, TS, DRGA, GA, HH 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. According to a paired t-test between the quality of ALNS/TPF and ILS, the P-value is 2.69×10^{-4}. Therefore, the improvement by ALNS/TPF is significant. Additionally, we observe that ALNS/TPF can find much better solutions when τ and R are small.

Table 3: Results for the OAS problem, 100 orders

n=100		Gap(%)						
τ	R	TS	DRGA	GA	HH	LOS	ILS	ALNS/TPF
0.10	0.10	2	1	2	3	2	0.95	**0.53**
	0.30	2	1	2	3	2	0.74	**0.54**
	0.50	1	1	1	1	1	0.37	**0.07**
	0.70	**0**	1	**0**	**0**	**0**	0.04	**0.00**
	0.90	**0**	**0**	**0**	**0**	**0**	0.01	**0.00**
0.30	0.10	3	3	3	6	2	1.40	**0.88**
	0.30	3	3	2	5	3	1.38	**1.16**
	0.50	2	2	2	4	2	1.17	**0.96**
	0.70	2	1	1	2	1	0.44	**0.15**
	0.90	1	1	**0**	1	**0**	0.25	**0.01**
0.50	0.10	4	5	4	8	4	2.26	**1.83**
	0.30	4	4	4	7	3	2.32	**2.11**
	0.50	4	4	4	7	3	2.40	**2.33**
	0.70	3	3	2	5	2	1.61	**1.20**
	0.90	2	2	1	3	1	1.16	**0.77**
0.70	0.10	5	6	5	9	4	3.13	**2.40**
	0.30	7	6	5	9	5	3.86	**3.85**
	0.50	6	7	6	10	5	**4.25**	4.39
	0.70	7	8	6	9	**5**	6.17	**5.04**
	0.90	8	7	6	9	**5**	6.60	**5.40**
0.90	0.10	9	9	7	11	6	7.02	**5.47**
	0.30	15	12	10	14	9	11.83	**8.71**
	0.50	16	14	12	16	**11**	14.06	**11.07**
	0.70	16	14	12	16	**11**	12.75	**11.16**
	0.90	16	13	12	15	**11**	13.23	**11.23**
Avg.		6	5	4	7	4	3.98	**3.25**

Conclusion

We studied an important class of over-subscribed scheduling problems 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. Algorithmic analysis finds that: (1) there exists a correlation between the completion ratio and the tabu and randomness types: the insertion tabu and randomness work well when the completion ratio is low, while the removal tabu and randomness work well when the completion ratio is high; (2) 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; (3) the fast insertion strategy contributes most to the performance, but also consumes the most time compared with other features.

Extensive empirical results on two domains demonstrated that, compared with the state-of-the-art approaches, our proposed ALNS/TPF produces solutions with higher quality in less time. Our work proves that tight ALNS and TS hybridization is an efficient method for this class of scheduling problem.

Our next steps are to further evaluate the heuristics in this work, and to understand the effect of ALNS and TS hybridization and the new algorithmic features on other real-world problem domains in this class.

References

Augenstein, S.; Estanislao, A.; Guere, E.; and Blaes, S. 2016. Optimal scheduling of a constellation of earth-imaging satellites, for maximal data throughput and efficient human management. In *Proc. of the 26th International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 345–352.

Bianchessi, N.; Cordeau, J.-F.; Desrosiers, J.; Laporte, G.;

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

Cordeau, J.-F., and 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.; and 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.; and Laporte, G. 2012. An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research* 223(2):346–359.

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

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

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

Nguyen, S.; Zhang, M.; and 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)*, 433–440. ACM.

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.

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.; and Kong, X. 2018. An iterated local search algorithm for agile earth observation satellite scheduling problem. In *Proc. of the 15th Conference on Space Operations (SpaceOps 2018)*, 2311.

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

Prins, C.; Prodhon, C.; Ruiz, A.; Soriano, P.; and 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.

Rogers, M. F.; Howe, A. E.; and Whitley, D. 2006. Looking for shortcuts: Infeasible search analysis for oversubscribed scheduling problems. In *Proc. of the 16th International Conference on Automated Planning and Scheduling (ICAPS 2006)*, 314–323.

Ropke, S., and 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.; and Pessoa, A. A. 2018. Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times. *Computers & 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., and Schaus, P. 2018. Revisiting the self-adaptive large neighborhood search. In *Proc. of the 15th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2018)*, 557–566.

Verbeeck, C.; Vansteenwegen, P.; and 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.

Žulj, I.; Kramer, S.; and 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.