11-2019

Agile earth observation satellite scheduling: An orienteering problem with time-dependent profits and travel times

Guansheng PENG

Reginald DEWIL

Cédric VERBEECK

Aldy GUNAWAN
Singapore Management University, aldygunawan@smu.edu.sg

Lining XING

See next page for additional authors

Citation

PENG, Guansheng; DEWIL, Reginald; VERBEECK, Cédric; GUNAWAN, Aldy; XING, Lining; and VANSTEENWEGEN, Pieter. Agile earth observation satellite scheduling: An orienteering problem with time-dependent profits and travel times. (2019). *Computers and Operations Research*. 111, 84-98.
Available at: https://ink.library.smu.edu.sg/sis_research/4403

Published in Computers and Operations Research,
Vol 111, November 2019, Pages 84-98
DOI 10.1016/j.cor.2019.05.030

Agile earth observation satellite scheduling: An orienteering problem with time-dependent profits and travel times

Guansheng Peng[a,b], Reginald Dewil[c], Cédric Verbeeck[d], Aldy Gunawan[e], Lining Xing[a,*], Pieter Vansteenwegen[b]

[a] College of System Engineering, National University of Defense Technology, Changsha, 410073, China
[b] KU Leuven Mobility Research Center - CIB, Leuven, 3001, Belgium
[c] VU Amsterdam, School of Business and Economics, 1081 HV Amsterdam, the Netherlands
[d] EDHEC Business School, 24 Avenue Gustave Delory, 59057 Roubaix, France
[e] School of Information Systems, Singapore Management University, 178902 Singapore

ARTICLE INFO

Article history:
Received 14 November 2018
Revised 28 March 2019
Accepted 29 May 2019
Available online 15 June 2019

Keywords:
Agile satellite scheduling
Time-dependent transition time
Time-dependent profits
Iterated local search

ABSTRACT

The scheduling problem of an Agile Earth Observation Satellite is to schedule a subset of weighted observation tasks with each a specific "profit" in order to maximize the total collected profit, under its operational constraints. The "time-dependent transition time" and the "time-dependent profit" are two crucial features of this problem. The former relates to the fact that each pair of consecutive tasks requires a transition time to maneuver the look angle of the camera from the previous task to the next task. The latter follows from the fact that a different look angle of an observation leads to a different image quality, i.e., the collected profit. Since the specific look angle of a task depends on its observation start time, both the transition time and the profit are "time-dependent". We present a concept of "minimal transition time" to displace the transition time. On this basis, a bidirectional dynamic programming based iterated local search (BDP-ILS) algorithm is proposed, equipped with an insert procedure that avoids a full feasibility check. The bidirectional dynamic programming approach is integrated into the algorithm in order to efficiently evaluate a solution or an insert move when time-dependent profits are considered. Two types of experiments (with and without the time-dependent profits) are designed to evaluate the performance. The results without time-dependent profits show that our algorithm outperforms the state of the art in terms of solution quality and computational time. When time-dependent profits are considered, our BDP-ILS algorithm performs very well on smaller instances with a known optimal solution and on larger instances compared to four reference algorithms.

1. Introduction

The mission of an Earth Observation Satellite (EOS) is to acquire images of targets on the Earth surface, in response to observation requests. Each target is associated with a profit that can be collected once the target is successfully scheduled. The scheduling problem of an EOS is to select and schedule a subset of weighted imaging tasks under the operational constraints in order to maximize the collected profit. EOSs have been extensively employed in earth resources exploration, natural disaster surveillance, and military reconnaissance.

The Agile Earth Observation Satellite (AEOS) is a new generation of EOS, e.g., the well-known PLEIADES satellite in France. It can be mobile on three axes (roll, pitch and yaw), thus allowing maneuverability for image acquisitions as well as for transitions between observations. The exclusive mobility of pitching axes enables the agile satellite to observe a target before or after its upright pass (called the "nadir point"). As illustrated in Fig. 1, the satellite observes the target at three different observation start times, each with a different pitch angle during a specific period, called the Visible Time Window (VTW). An observation is defined as the satellite observing a target at a specific moment. The Observation Window (OW) is the time duration required for an observation. Due to the satellites' agility, the VTW is much longer than the OW for each observation, and the OW should be determined within the VTW. On the one hand, this potentially increases the effectiveness of the whole system, allowing the satellite to observe more targets in a

* Corresponding author.
E-mail address: milan.jirasek@fsv.cvut.cz (L. Xing).

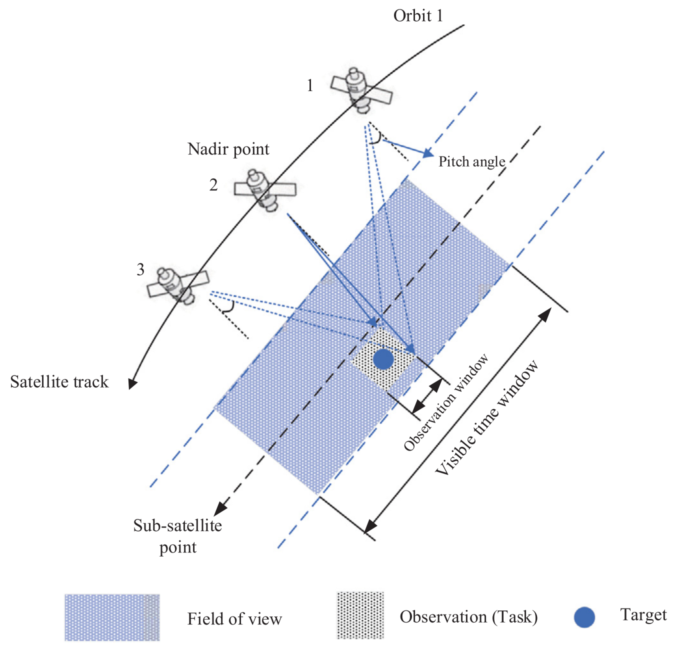


Fig. 1. An agile satellite images a target at different observation start times on orbit 1.

given period. On the other hand, the scheduling of an AEOS becomes more difficult since the search space is considerably larger.

Furthermore, "time-dependent transition time" and "time-dependent profits" are two crucial features in AEOS scheduling. Firstly, for each pair of consecutive observations, a transition time is required to change the look angles (roll, pitch, and yaw) of the satellite from the previous task to the next task. Therefore, the time between two consecutive observations should be longer than this transition time. The length of the transition time is determined by the angular changes on these three axes, but only the pitch angle depends on the observation start time, as can be seen in Fig. 1. Consequently, for each pair of consecutive observations, their transition time is time-dependent. Secondly, the collected profit also depends on the observation start time, because the images taken at different pitch angles have different image quality (profit). Undoubtedly, these two "time-dependent" features significantly increase the complexity of the scheduling.

Overall, this research solves the AEOS scheduling problem with time-dependent transition time and time-dependent profits. This problem can be modeled as the Orienteering Problem (OP) with time-dependent travel times, time-dependent profits and time windows, since only a subset of candidate vertices should be selected and sequenced, in order to maximize the total collected profit within a limited available time. Inspired by the existing methods for the OP (Vansteenwegen et al., 2009; Verbeeck et al., 2014), we present an efficient heuristic algorithm based on Iterated Local Search (ILS) and Bidirectional Dynamic Programming (BDP). The ILS combines a remove procedure and an insert procedure which avoids a full feasibility check of the transition time constraint. The BDP approach is integrated into the ILS in order to accurately and efficiently evaluate the solution. The first agile satellite of China, the AS-01 satellite, is considered in our work. Note that this agile satellite is not as agile as PLEIADES since its image is produced by the movement of the satellite on its track. Therefore, its look angles are fixed when imaging a target.

In the next section, a literature overview is presented and in Section 3, a rigorous problem description and a mathematical model are given. In Section 4 the heuristic algorithm is described in detail and in Section 5, the experimental results are presented. Section 6 provides our conclusions and further work.

2. Literature review

The scheduling of AEOS has been proven to be NP-hard (Lemaître et al., 2002). Very limited research has been conducted on the AEOS scheduling, probably due to the challenging complexity. Gabrel et al. (1997) study the scheduling problem for a semi-agile satellite which is weakly mobile on pitch axes and roll axes. Several algorithms are proposed for this problem, based on graph-theoretic concepts. Lemaître et al. (2002) provide a comprehensive description of the early research on the AEOS scheduling. Four algorithms (a greedy algorithm, a dynamic programming algorithm, a constraint programming algorithm and a local search algorithm) are proposed to solve a simplified AEOS scheduling problem. Cordeau and Laporte (2005) present a tabu search heuristic which is derived from the algorithm developed for the Vehicle Routing Problem with Time Windows (VRPTW). In order to obtain better solutions, they relax the time window constraint to allow a mixture of feasible and infeasible solutions during the search. Habet et al. (2010) study the AEOS scheduling problem with fixed transition times. They propose a tabu search algorithm based on consistent and saturated configurations to optimize a convex evaluation function. A secondary objective that minimizes the sum of the transition times is introduced. Pralet and Verfaillie (2013) consider the time-dependency of transition times and define a so-called Time-dependent Simple Temporal Network to model this constraint. Some techniques based on constraint propagation are proposed to solve the model. Unfortunately, the above mentioned techniques work only when a single orbit is considered. In practice, however, each target can be observed during several consecutive orbits for each day, and thus has more than one VTW. Therefore, our problem considers scheduling during multiple orbits, since this is more realistic for the daily management of AEOS.

A few papers consider scheduling satellites with multiple orbits. Tangpattanakul et al. (2015) investigate the AEOS scheduling problem with two objectives: maximizing the total profit and ensuring fairness among users by minimizing the maximum profit difference between users. In this study, as in ours, a single satellite is considered. Bianchessi and Righini (2008) consider multiple orbits and multiple satellites. They work on the scheduling problem of the COSMO-skyMed satellite constellation, where the acquisition and the download of satellite images are considered simultaneously. A constructive algorithm with look-ahead and back-tracking capabilities is developed in order to solve large-size instances in a short time. Both studies do not consider the time-dependency of transition times.

The only papers considering time-dependent transition times are (Liu et al., 2017) and (He et al., 2018). Liu et al. (2017) develop an Adaptive Large Neighborhood Search (ALNS) algorithm for a single agile satellite. He et al. (2018) extend this ALNS algorithm to schedule multiple satellites. In this ALNS, six removal operators and three insertion operators are designed for the search, and a fast insertion method is presented to confine the propagation of the transition time changes, based on the "time slack" of each selected task. According to the time slack, an extra task can easily be inserted by shifting its neighboring tasks earlier or later. However, this method ignores the fact that the other tasks could also be shifted, which may allow more insertions. Based on a modeling analysis of the time-dependent transition time, we specifically design an insertion procedure in our heuristic, inspired by the work on the Time-Dependent Orienteering Problem (TDOP) (Gunawan et al., 2014; Verbeeck et al., 2014). Furthermore, in order to build a linear mathematical model, Liu et al. (2017) simplify the transition time between two observations to a constant value. Still, this simplified model cannot be solved by the commercial solver CPLEX for instances involving more than 12 tasks. In this paper, we present a better mixed integer linear model in which the time-dependent transition times are considered and which can be solved for instances with up to 100 tasks.

Some variants of the AEOS scheduling problem are discussed in the literature. Liao and Yang (2007) developed an imaging order scheduler for the FORMOSAT-2 satellite considering weather uncertainty. Lagrangian relaxation and linear search techniques are proposed to solve the problem. Grasset-Bourdel et al. (2011) worked on the automatic planning of activities of a constellation of AEOSs. The scheduling of observation missions and data downloads are simultaneously considered in their model. These studies focused on some specific characteristics of the AEOS scheduling, which cannot be compared with our work.

In addition to the time-dependent transition time, the time-dependent profit is another crucial feature in our problem. In practice, the commercial value of a satellite image is significantly influenced by its image quality which depends on the observation start time. However, very few works on AEOS scheduling have considered the image quality and its time-dependency. Wolfe and Sorensen (2000) define the “window constrained packing problem” to model the AEOS scheduling problem, with a particular quality function associated with each VTW. It differs from our work by the fact that the image quality not only depends on the observation start time, but also depends on the observation duration. In other words, observations have no fixed duration, and preference is given to the observations with longer duration. Moreover, the transition time is not considered. Liu et al. (2017) have modeled it as a user-imposed constraint. They assessed the image quality of an observation on a ten-level scale over its VTW, and the profit of each observation can be awarded only if the minimum requirement is satisfied. However, in their model, the image quality is irrelevant to the scheduling, since the VTWs can be reduced beforehand to only the part that guarantees enough quality.

The “time-dependent profit” feature has been studied in the variants of other combinatorial optimization problem, such as Vehicle Routing Problem with Time-Dependent Rewards (VRP-TDR) (Yi, 2003), Orienteering Problem with Time-Dependent Rewards (OP-TDR) (Ekic et al., 2009; Erkut and Zhang, 1996). These problems arise from several real-life applications: blood transportation to Red Cross Centers (Yi, 2003), the repairing maintenance system (Afsar and Labadie, 2013) and the disaster relief chain (Ekici and Retharekar, 2013). Several exact and heuristic methods have been proposed to tackle the problem. However, in both of the problems, the profit of each vertex monotonously decreases over time, meaning that each visit is naturally scheduled as early as possible for a maximal collected profit. Our problem considers time-dependent profit with a non-monotonic function according to the practical need of satellite images. To the best of our knowledge, this feature has not been studied in the literature.

3. Problem description

In the daily management of an agile satellite, requests (targets with each a given geographic position and profit) from different users are collected. A single orbit is defined as the time interval that the satellite flies in the sunshine when circling the earth once (normally 45 min). The scheduling horizon (one day) is split into multiple orbits according to the prediction of the satellite orbit trajectory. Based on the visibility analysis, for each target, its VTWs and the look angles (roll, pitch and yaw) per second during its VTWs are calculated beforehand. Thus, each target is modeled by multiple tasks, one in each orbit when the target can be observed. Each task for a given target is defined by its VTW and its time dependent profit.

3.1. Assumptions

In practice, scheduling an AEOS is rather complicated due to the many constraints and user requirements. Therefore, a number of assumptions are made in order to simplify the problem, ignoring some non-significant constraints.

1) Only spot targets that can be observed in one pass are considered in our model. Polygon targets and stereo tasks are not considered in this study.

2) The limitation of energy on board is not taken into account. Due to the development of new technology, the solar panel can provide enough power for the satellite.

3) The scheduling of downloading images and the on-board memory constraint are not considered because they are not the focus of our study. More importantly, our work is developed based on the previous work by Liu et al. (2017). Despite the presence of the on-board memory constraint in their model, it is not considered in their algorithm and experiments. Therefore, in order to ensure a direct comparison, we ignore the limitation of the on-board memory, as well as the scheduling of downloading data.

3.2. Variables

As the input of the AEOS scheduling problem, a set of possible targets is required, denoted by $T = \{1, \ldots, N\}$, where N is the number of targets to be scheduled. For each target $I \in T$, we define:

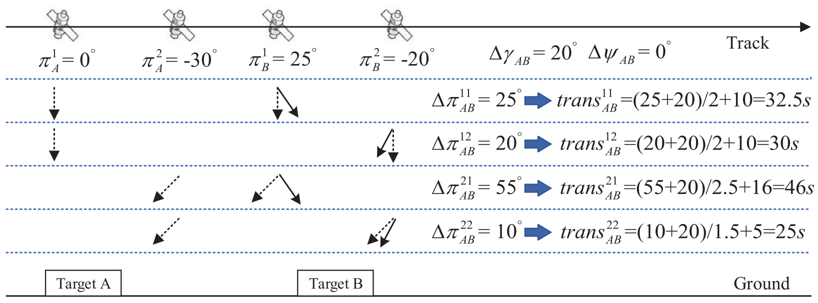
- P_I: the profit of target I;
- d_I: the duration of observing target I;
- b_I^k: the binary parameter which equals 1 only if target I has a visible time window during orbit k, otherwise $b_I^k = 0$;

Each target I can be divided into multiple tasks $i^k (k \in O)$, where O is the set of orbits and k is the orbit index. Each task i^k refers to a visible time window $[st_i^k, et_i^k]$, where st_i^k and et_i^k represent the start and the end time, respectively. When target I is scheduled on task i^k with its observation start time $h_i^k (st_i^k < h_i^k < et_i^k)$, an actual profit $p_i^k(h_i^k)$ is collected. The maximum profit $p_i^k(h_i^k)$ is equal to its corresponding target profit P_I. The duration d_i^k of task i^k is equal to the duration of its corresponding target I. For each pair of consecutive tasks i^k and j^k, a transition time $trans_{ij}^k(h_i^k, h_j^k)$ is defined, depending on their observation start times h_i^k and h_j^k.

To facilitate the discussion of the model, we drop the superscript k when discussing the scheduling during orbit k if this does not provoke ambiguity.

3.3. Minimal transition time

The time-dependent transition time is similar to the time-dependent travel time in the Time-Dependent Vehicle Routing Problem (TDVRP) (Malandraki and Daskin, 1992) or the Time-Dependent Orienteering Problem (TDOP) (Verbeeck et al., 2017). The major difference is that in the TDVRP and TDOP, the travel time only depends on the departure time of the previous vertex. However, the transition time in AEOS scheduling depends on both the observation start times of the two consecutive tasks. Fig. 2 gives an example of four different transitions between the observations of target A and target B. The four satellite icons represent four observations with different observation start times when the satellite is moving on its track. The former two are the observations of target A with their pitch angles π_A^1, π_A^2 (dotted arrow), and the latter two are the observations of target B with π_B^1, π_B^2 (solid arrow). The total change of the look angles are calculated by $\Delta g = |\Delta\gamma| + |\Delta\pi| + |\Delta\psi|$, where $\Delta\gamma$ and $\Delta\psi$ represent the change of the roll angle and the yaw angle. The roll angle only depends on the geographical locations of targets relative to the satellite track. We assume $\Delta\gamma$ between target A and B equals 20° and



$\pi_A^1 = 0^\circ$
$\pi_A^2 = -30^\circ$
$\pi_B^1 = 25^\circ$
$\pi_B^2 = -20^\circ$
$\Delta\gamma_{AB} = 20^\circ$ $\Delta\psi_{AB} = 0^\circ$
Track
$\Delta\pi_{AB}^{11} = 25^\circ$ $trans_{AB}^{11}$=(25+20)/2+10=32.5s
$\Delta\pi_{AB}^{12} = 20^\circ$ $trans_{AB}^{12}$=(20+20)/2+10=30s
$\Delta\pi_{AB}^{21} = 55^\circ$ $trans_{AB}^{21}$=(55+20)/2.5+16=46s
$\Delta\pi_{AB}^{22} = 10^\circ$ $trans_{AB}^{22}$=(10+20)/1.5+5=25s
Target A
Target B
Ground

Fig. 2. Four different pairs of observations for target A and target B.

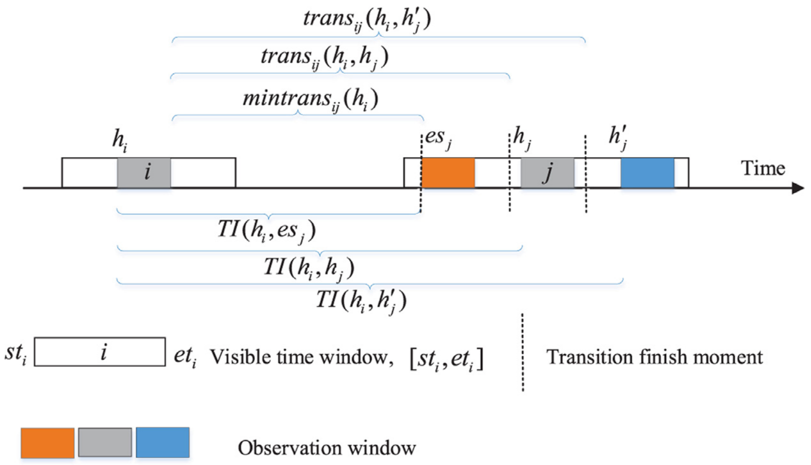


Fig. 3. The change of the transition time between task *i* and task *j* with different observation start times of task *j*.

is not time-dependent but constant. Since our AS-01 satellite is a semi-agile satellite where the image is produced by the movement of the satellite, $|\Delta\psi|$ can be approximated to zero. The transition time between two consecutive observations is calculated based on the following piecewise linear function:

$$trans_{ij} = \begin{cases} 11.66, & \Delta g \le 10 \\ 5 + \Delta g/v_1, & 10 < \Delta g \le 30 \\ 10 + \Delta g/v_2, & 30 < \Delta g \le 60 \\ 16 + \Delta g/v_3, & 60 < \Delta g \le 90 \\ 22 + \Delta g/v_4, & \Delta g > 90 \end{cases}, \tag{1}$$

where v_1, v_2, v_3, v_4 are four different angular transition velocities, which we consider as given, describing the mobility of the satellite. For our AS-01 satellite, the angular velocity values are $v_1 = 1.5^\circ/s, v_2 = 2^\circ/s, v_3 = 2.5^\circ/s, v_4 = 3^\circ/s$. As shown in Fig. 2, the larger the change of the pitch angle, the larger the transition time. Consequently, the transition time is determined by the observation start times of both tasks.

To simplify the transition time, we propose a concept of "minimal transition time". It represents the minimal required transition time that allows the observation of the next task to happen as early as possible when the ending time (or the starting time) of the previous task is given. This concept exploits the AEOS property that moving the camera is faster than moving the satellite, which is illustrated in recent work (Pralet and Verfaillie, 2013). To intuitively demonstrate it, Fig. 3 shows how the transition time changes with different observation start times of the next task *j* when observing two consecutive tasks *i* and *j*. The white rectangle represents a visible time window, and the colored rectangles represent the observation windows at different observation start times. We found that given an ending time of the previous task *i*, the later the observation of the next task *j* starts, the larger the waiting time is. The time interval $TI(h_i, h_j)$ is the period between the observation start times of the two tasks, and $trans_{ij}(h_i, h_j)$ is their transition time. The waiting time is equal to the time interval $TI(h_i, h_j)$ minus its transition time $trans_{ij}(h_i, h_j)$ and the duration d_i^k of task *i*. It can be described as the following formula:

$$\forall h_i \in [st_i, et_i], \forall h_j, h_j' \in [st_j, et_j], h_j \le h_j' \Rightarrow (h_j - trans_{ij}(h_i, h_j)) \le (h_j' - trans_{ij}(h_i, h_j')) \tag{2}$$

Based on this rule, the minimal transition time $mintrans_{ij}(h_i)$ can be defined as the transition time for which the waiting time is the smallest. The corresponding observation start time of the next task *j* is its earliest possible start time es_j, given that the observation of task *j* starts at h_j. Any observation that starts later than es_j always satisfies the transition time constraint. The calculation process of the earliest start time and the minimal transition time is defined as $EarliestStartTime_{ij}(h_i)$. The minimal transition time only depends on the observation start time of the previous task, the same as the time-dependent travel time in TDVRP and TDOP. Consequently, some existing methods for the TDVRP or TDOP can be utilized for reference (Gunawan et al., 2014; Verbeeck et al., 2017).

Similarly, the latest start time of the previous task can be calculated if the observation start time of the next task is given. The later the observation start time of the previous task is, the later the transition finishes. This rule can be expressed as follows:

$$\forall h_j \in [st_j, et_j], \forall h_i, h_i' \in [st_i, et_i], h_i \le h_i' \Rightarrow (h_i + trans_{ij}(h_i, h_j)) \le (h_i' + trans_{ij}(h_i', h_j)), \tag{3}$$

where h_i and h_i' represent two observation start times of the previous task *i*. The calculation of the latest start time is defined as $LatestStartTime_{ij}(h_j)$.

In this work, since the input data of look angles for each VTW is given in a table look-up fashion per second, the minimal transition times are also calculated in discrete form. In order to avoid duplicate calculations of *EarliestStartTime*() and *LatestStartTime*(), we pre-calculate the minimal transition times between each pair of candidate tasks for each second during their VTWs. This process is based on a dichotomy algorithm and will be introduced in Section 4.4.

3.4. Time-dependent profit

In practice, an AEOS should deliver high-quality satellite images in order to satisfy requirements for image recognition, target detection and so on. The best image quality is obtained at the nadir point where the satellite observes a target directly below and its pitch angle is equal to zero. The larger the absolute value of the pitch angle, the lower the image quality. The nadir point is normally in the middle of a complete VTW. Both edges of a VTW have the largest absolute value of the pitch angle and the lowest profit. Hence, the collected profit of a task depends on its observation start time. The "time-dependent profit" is a crucial property of our scheduling problem.

Since the image quality and the absolute value of the pitch angle are negatively correlated, the profit of a task executed at moment h_i is given by the equation below:

$$p_i(h_i) = P_I * (1 - \frac{|\pi(h_i)|}{90}),$$

where $\pi(h_i)$ is the pitch angle when scheduling task i at moment h_i, and P_I is its corresponding target profit. According to this equation, the whole target profit can be collected when the pitch angle of an observation is equal to 0, while only half of the target profit can be collected at the edge of the VTW. In order to address the time-dependent profit, a trade-off should be made between scheduling more tasks and obtaining better observation start times of the scheduled tasks.

3.5. Mixed integer programming model

In this section, we formulate the AEOS scheduling problem as a mixed integer programming (MIP) model. In this model, three sets of decision variables are defined:

x_{IJ}^k: binary variable equal to 1 if target I and J are scheduled during orbit k and J is scheduled immediately after I, and 0 otherwise. Two virtual targets S and E are added to the solution as the source target and end target, respectively.

y_I^k: binary variable equal to 1 if target I is scheduled during orbit k (with task i^k), and 0 otherwise.

h_i^k: the observation start time of task i^k.

Based on the statements and assumptions above, the mixed integer programming model can be formulated as follows:

$$\textit{Maximize} \sum_{I \in T} \sum_{k \in O} p_i^k(h_i^k). \tag{4}$$

$$\sum_{k \in O} y_I^k \le 1, \ \forall I \in T \tag{5}$$

$$\sum_{\substack{J \in T \cup \{E\} \\ J \ne I}} x_{IJ}^k = \sum_{\substack{J \in T \cup \{S\} \\ J \ne I}} x_{JI}^k = y_I^k, \ \forall I \in T, k \in O \tag{6}$$

$$\sum_{J \in T \cup \{E\}} x_{SJ}^k = 1, \ \forall k \in O \tag{7}$$

$$\sum_{J \in T \cup \{S\}} x_{JE}^k = 1, \ \forall k \in O \tag{8}$$

$$h_i^k + d_i^k + mintrans_{ij}^k(h_i^k) - h_j^k \le M(1 - x_{IJ}^k), \ \forall I, J \in T, k \in O \tag{9}$$

$$y_I^k \le b_I^k, \ \forall I \in T, k \in O \tag{10}$$

$$st_i^k \le h_i^k \le et_i^k, \ \forall I \in T, k \in O \tag{11}$$

$$p_i^k(h_i^k) \le P_I y_I^k, \ \forall I \in T, k \in O \tag{12}$$

$$x_{IJ}^k \in \{0,1\}, y_I^k \in \{0,1\}, \ \forall I, J \in T \cup \{S, E\}, \ k \in O \tag{13}$$

The objective function (4) maximizes the total selected profit. Constraints (5) state that each target I can be observed during at most one orbit. Constraints (6) are the flow balance constraints that ensure that the number of predecessors is equal to the number of successors for each task in the solution. Constraints (7) and (8) express that for each orbit, the task sequence starts at virtual target S and ends at virtual target E. Constraints (9) indicates that every two successive observations during the same orbit must satisfy the required transition time limitation. Constraints (10) enforce that each target can only be scheduled during an orbit where there is a VTW for the target. Constraint (11) and (12) restrict the start time of a task to its visible time window. The domains of decision variables are defined in constraints (13).

This mathematical model cannot be directly solved by CPLEX due to the non-linear transition time constraints (9) and the non-linear objective function (4). The transition time between each pair of tasks is calculated based on the data of look angles per second during their VTWs which are given in a table look-up fashion and cannot be directly passed to the solver. In order to solve this problem, we build a CPLEX model where each moment during a VTW is regarded as a vertex with each a specific profit. Each task is now divided into multiple vertices. For example, given a time step of 5 s for a VTW of 5 min results in 60 vertices. For each pair of tasks, directed arcs are constructed among all their vertices for which the transition time constraint is satisfied. So, between two tasks with a VTW of 5 min, up to (60*60=)3600 directed arcs can be constructed. Consequently, the scheduling during each orbit can be modeled by a weighted network model in which an optimal sequence of vertices with the highest collected profits should be found. Compared to the simplified MILP model built by Liu et al. (2017), our alternative MILP model retains the time-dependency for the transition time constraint, which is more practical. However, since each task is replaced by a high number of vertices, of which many are connected by an arc to many vertices of other tasks, the size of this network grows exponentially. Therefore, this alternative MILP model can only be solved to optimality for smaller instances, as will be illustrated in Section 5.3.

4. Bidirectional dynamic programming based iterated local search heuristic

Previous studies proved that the AEOS scheduling problem with time-dependent transition time is NP-hard, which means it is unlikely to find the optimal solution within polynomial time (Lemaître et al., 2002). Moreover, the "time-dependent profit" characteristic considerably enlarges the solution space since not only the number of scheduled tasks but also the exact times of performing these tasks determine the solution quality.

In this paper, a Bidirectional Dynamic Programming based Iterated Local Search (BDP-ILS) algorithm is developed to tackle the problem. The choice for this ILS framework is motivated by the fact that, generally, a very complex problem requires a fast and

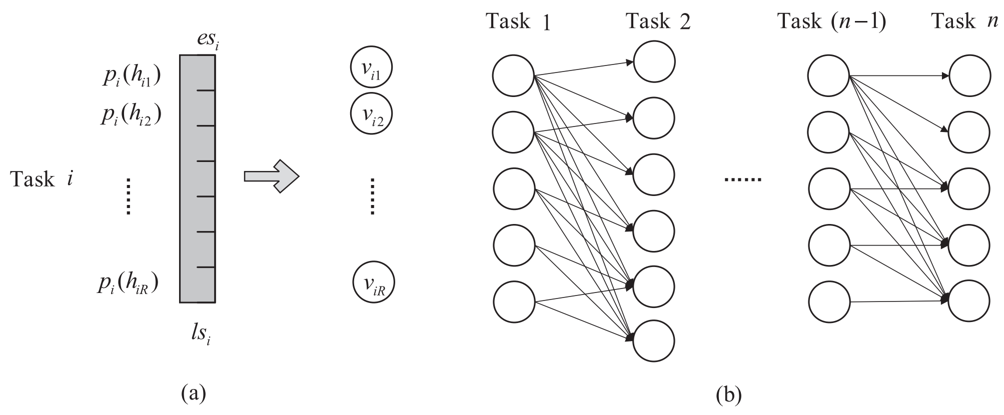


Fig. 4. Evaluation of a solution with time-dependent profits.

straightforward solution framework. Moreover, ILS has been implemented successfully before to deal with variants of the orienteering problem and problems with time windows (Lourenço et al., 2003; Stützle, 1999; Vansteenwegen et al., 2009). Our algorithm combines an insert procedure for intensification and a remove procedure for diversification. The insert procedure is designed based on several auxiliary features to address the time-dependent transition time and the visible time window constraints. A Bidirectional Dynamic Programming (BDP) approach is presented and incorporated into the insert procedure to efficiently evaluate each insert move.

The remainder of this section is organized as follows. First, the auxiliary features that need to be tracked during the search are defined in Section 4.1, and the auxiliary functions that calculate these features are presented in Section 4.2. The framework of BDP-ILS is explained in Section 4.3. A pre-processing method is described in Section 4.4. Then, Section 4.5 and Section 4.6 discuss the insert procedure and the remove procedure, respectively.

4.1. Auxiliary features

In order to solve this problem efficiently, we introduce four auxiliary features that should be kept track of during the search: the earliest start time, the latest start time, the forward accumulated profit and the backward accumulated profit. In our problem, a solution comprises several task sequences, each of which is the scheduled sequence of selected tasks during an orbit.

Firstly, for each task i in a sequence, we record its earliest start time es_i and its latest start time ls_i within the sequence when considering the visible time window constraint and the transition time constraint (defined in Section 3.3). In this way, a full feasibility check can be avoided for each possible insertion by only comparing the earliest start time and latest start time of the inserted task. Thus, the computational time can be drastically reduced. These two features can be directedly obtained based on the pre-calculation results of minimal transition times (see Section 4.4).

Secondly, for each possible observation start time h_i of task i during $[es_i, ls_i]$, we define its forward accumulated profit $p_i^{fa}(h_i)$ and its backward accumulated profit $p_i^{ba}(h_i)$. The forward accumulated profit $p_i^{fa}(h_i)$ represents the maximal collected profit from the beginning of the sequence up to and including task i at moment h_i (including the profit of task i). It can be expressed as

$$p_i^{fa}(h_i) = \max\{p_j^{fa}(h_j) + p_i(h_i),\ \forall\ h_j\ \mathit{satisfies\ transition\ constraint}\}, \tag{14}$$

where task t_j immediately precedes task i in the sequence, and h_j can be any possible observation start time that satisfies the transition time constraint while observing task i at h_i.

Similarly, the backward accumulated profit $p_i^{ba}(h_i)$ represents the maximal profit that can be collected in the sequence after observing task i at moment h_i. The backward accumulated profit is calculated by:

$$p_i^{ba}(h_i) = \max\{p_j^{ba}(h_j) + p_j(h_j),\ \forall\ h_j\ \mathit{satisfies\ transition\ constraint}\}, \tag{15}$$

where task t_j immediately succeeds task i in the sequence, and h_j is its any possible observation start time that satisfies the transition time constraint after observing task i at h_i.

The aim of tracking the forward and backward accumulated profits is to apply a dynamic programming approach to a solution. As a result of this method, the best observation start times of each selected task in the sequence can be determined in order to obtain a maximal collected profit. The functions that calculate these two auxiliary features are defined in the next subsection.

4.2. Auxiliary functions

In this section, we present two auxiliary functions to calculate the forward and backward accumulated profits efficiently: *ForwardRecursion*() and *BackwardRecursion*(). Since the profit of a task depends on its observation start time, an evaluation of a solution is required to obtain the best observation start time of each task for a maximal collected profit. Given the earliest start time and the latest start time of each task in the sequence, a Bidirectional Dynamic Programming (BDP) approach is introduced to optimize their observation start times and to evaluate the solution, as shown in Fig. 4.

In Fig. 4, the time span $[es_i, ls_i]$ of each task i is evenly discretized into a sequence of moments $\{h_{i1}, \ldots, h_{i2}, \ldots, h_{iR}\}$ by using a time step T_{step} that specifies a discrete time resolution. The profit of task i at moment h_{ir} is denoted by $p_i(h_{ir})$. Then the sequence of selected tasks can be expressed as a time labeled graph, in which each vertex v_{ir} represents an observation of task i at moment h_{ir}, and the directed edges represent the possible transitions between vertices, given the visible time window constraints. The problem is equivalent to finding a maximal profit path that starts from the first task and ends at the last task in the sequence. The BDP calculates the forward accumulated profit and the backward accumulated profit for each vertex in the graph by recursively using Eqs. (14) and (15) from two opposite directions.

Based on this approach, a full evaluation of a solution and a fast evaluation of an insertion can be processed. The full evaluation calculates these two auxiliary features for a sequence of tasks from scratch. In this way, at each task, the maximum sum of the forward and backward accumulated profits equal the maximum total profit that this sequence of tasks can feasibly achieve. The fast evaluation of an insertion only calculates these two features of the inserted task based on its neighboring tasks, and, consequently, can accurately calculate the impact on the total objective function value.

Algorithm 1 shows the pseudo-code of the full evaluation.

Algorithm 1 Bidirectional Dynamic Programming approach.

```
Input: $S_c$: Current solution;
Output: $P_S$: Total collected profit of solution $S_c$;
  for each orbit $k$ in solution $S_c$ do
    Find the task sequence $\{1, \ldots, n\}$ during orbit $k$;
    $i \leftarrow 1$;
    while $i\ != n$ do
      ForwardRecursion($i$, $(i+1)$);//calculate forward accumulated profit
      $i \leftarrow i+1$;
    end while
    while $i\ != 1$ do
      BackwardRecursion($(i-1)$, $i$);//calculate backward accumulated profit
      $i \leftarrow i-1$;
    end while
    for each task $i$ in the sequence $\{1, \ldots, n\}$ do
      Find the vertex $v_{ir^*}$ with the highest value of $p_i^{fa}(h_{ir^*}) + p_i^{ba}(h_{ir^*})$;
      $h_i \leftarrow h_{ir^*}$, $p_i \leftarrow p_i(h_{ir^*})$;//determine its observation start time and its collected profit
    end for
    Choose any task $i$ from $\{1, \ldots, n\}$;
    $p_{\{1,\ldots,n\}} \leftarrow p_i^{fa}(h_{ir^*}) + p_i^{ba}(h_{ir^*})$; get the total collected profit of task sequence $\{1, \ldots, n\}$
    $P_S \leftarrow P_S + p_{\{1,\ldots,n\}}$;
  end for
```

Briefly, for each orbit k of the current solution S_c, a task sequence $\{1^k, \ldots, n^k\}$ is used to find out its maximal profit $p_{\{1^k,\ldots,n^k\}}$, i.e., to optimize the observation start times and the collected profit for each task in the sequence. Firstly, we recursively calculate the forward accumulated profit of each vertex from the first task 1^k to the last task n^k. Secondly, the backward accumulated profit of each vertex can also be calculated from the end to the start. *ForwardRecursion(i, j)* and *BackwardRecursion(i, j)* are defined as the recursion processes from these two opposite directions, respectively (displayed in Algorithms 2 and 3). After the recursion steps, the best observation start time and the profit of each task are determined by the vertex with the highest sum of its forward accumulated profit and its backward accumulated profit. This sum is the total collected profit of the task sequence. The total collected profit of the current solution S_c is the sum of the total collected profits of all the sequences of tasks during the different orbits.

Algorithm 2 updates the forward accumulated profit from the previous task i to the next task j. As mentioned above, building an edge from a vertex v_{ir} of task i to a vertex $v_{jr'}$ of task j represents a possible transition starting from task i at moment h_{ir} and ending at task j at moment $h_{jr'}$. After the pre-calculation of minimal transition times, for each vertex v_{ir} of task i, the earliest vertex (moment) of task j can be directly obtained, denoted by a pointer $v_{ir}.ear$. Then, we use Eq. (14) to update the forward accumulated profit for each vertex of the next task. This process requires to traverse all the edges (possible transition) connecting the vertices of task i and task j. Hence, the time complexity is $O(R^2)$, where R is the number of vertices each task has on average. However, not all

Algorithm 2 ForwardRecursion(*i, j*).

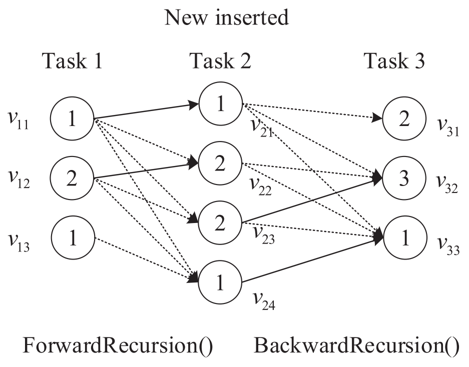
```
Input: $\{v_{i1}, \ldots, v_{iR}\}$: a vertex sequence of the previous task $i$;
  $\{v_{j1}, \ldots, v_{jR'}\}$: a vertex sequence of the next task $j$;
  Define an index variable $m \in \{1, \ldots, R\}$ for the vertices of task $i$ and initialize it with 1;
  $v_{im}.ear \leftarrow v_{j1}$;// Initialize the pointer of vertex $v_{im}$
  for each vertex $v_{ir}$ of task $i$ do
    if $p_i^{fa}(h_{ir}) >= p_i^{fa}(h_{im})$ then
      Find the earliest vertex of task $j$ that vertex $v_{ir}$ can reach, and denote it by $v_{ir}.ear$;
      for each vertex $v_{jr'}$ from $v_{im}.ear$ to vertex $v_{ir}.ear$ do
        $p_j^{fa}(h_{jr'}) \leftarrow p_i^{fa}(h_{im}) + p_j(h_{jr'})$;
        $m \leftarrow r$;
      end for
    else
      continue;
    end if
  end for
  for each vertex $v_{jr'}$ from $v_{im}.ear$ to the last vertex $v_{jR'}$ do
    $p_j^{fa}(h_{jr'}) \leftarrow p_i^{fa}(h_{im}) + p_j^{fa}(h_{jr'})$;
  end for
```

Algorithm 3 BackwardRecursion(*i, j*).

```
Input: $\{v_{i1}, \ldots, v_{iR}\}$: a vertex sequence of the previous task $i$;
  $\{v_{j1}, \ldots, v_{jR'}\}$: a vertex sequence of the next task $j$;
  for each vertex $v_{ir}$ of task $i$ do
    $p_i^{ba}(h_{ir}) \leftarrow p_i(h_{ir})$;//Initialize its backward accumulated profit to its profit
  end for
  Define an index variable $v_m \leftarrow R'$;
  $v_{jm}.lat \leftarrow v_{iR}$;
  for each vertex $v_{jr'}$ of task $j$ do
    if $p_j^{ba}(h_{jr'}) + p_j(h_{jr'}) >= p_j^{ba}(h_{jm}) + p_j(h_{jm})$ then
      Find the latest vertex of task $t_i$ that can access to vertex $v_{jr'}$, and denote it by $v_{jr'}.lat$;
      for each vertex $v_{ir}$ from $v_{jr'}.lat$ to vertex $v_{jm}.lat$ do
        $p_i^{ba}(h_{ir}) \leftarrow p_j^{ba}(h_{jm}) + p_j(h_{jm})$;
        $m \leftarrow r'$;
      end for
    else
      continue;
    end if
  end for
  for vertex $v_{ir}$ from the first vertex $v_{i1}$ to $v_{jm}.lat$ do
    $p_i^{ba}(h_{ir}) \leftarrow p_j^{ba}(h_{jm}) + p_j(h_{jm})$;
  end for
```

of the edges need to be visited. An improved recursive method is presented in Algorithm 2 to avoid unnecessary visits.

Specifically, we traverse all the vertices of task i from the earliest to the latest. We define a variable m as the subscript of the vertex that owns the maximal forward accumulated profit until now. If the current vertex v_{ir} has a higher forward accumulated profit than vertex v_{im}, all the vertices of task j that can be reached by v_{im} will be updated by summing up the profit of that vertex and the forward accumulated profit of v_{im}, except the ones that start later than vertex $v_{ir}.ear$, i.e., the earliest vertex that vertex v_{ir} can reach. Then, the variable m is also updated to r. Otherwise, if the forward accumulated profit of the current vertex is lower, the algorithm will skip to the next vertex. This improved recursive method ensures that each vertex of task j is visited only once during the recursion. Thus, the time complexity is improved to $O(R)$. Likewise,



Vertex	$p_2^{fa}(h_2)$	$p_2^{ba}(h_2)$	Sum
v_{21}	2	3	5
v_{22}	4	3	7
v_{23}	4	1	5
v_{24}	3	1	4

Fig. 5. An example of the fast evaluation.

Algorithm 3 updates the backward accumulated profit from the next task j to the previous task i. In order to explain this improved recursive method in detail, a simple example in Fig. 5 is discussed later.

In previous work with time-dependent profits, some heuristic algorithms are proposed to address the time-dependency (Afsar and Labadie, 2013; Ekici and Retharekar, 2013; Victoria et al., 2015). In their algorithms, when inserting an extra task, all the tasks after the insert position require an update of the observation start time and the collected profit. The insertion is considered to be acceptable only if the total collected profit increases. This method consumes much time due to a large number of unacceptable insert attempts. Moreover, in our problem, updating the observation start times of all other tasks means recalculating their transition times, which is also time-consuming.

Based on the forward recursion and the backward recursion, a fast evaluation method for an insertion can be implemented. Algorithm 4 shows the fast evaluation of an inserted task i between its preceding task j and its succeeding task k. We assume that the accumulated profits of task j and k in the solution are given, and it is feasible to insert i between j and k. For each vertex (moment) of the inserted task i during $[es_i, ls_i]$, its forward accumulated profit is calculated from task j, and its backward accumulated profit is calculated from task k. Afterward, the sum of the forward accumulated profit and the backward accumulated profit are calculated for each vertex. The highest value of the sum is equal to the total collected profit after the insertion. An insertion that increases the total collected profit will be implemented. Due to these accumulated profit functions, it is unnecessary to update the observation starts times of all other tasks when evaluating an insertion. This drastically reduces the computational time.

Algorithm 4 FastEvaluation(i, j, k).

```
Input: $i$: the inserted task;
    $j$: the preceding task of $i$;
    $k$: the succeeding task fo $i$;
Output: $p$: The total collected profit after the insertion of $i$;
    $ForwardRecursion(j, i)$;
    $BackwardRecursion(i, k)$;
    Find the vertex $v_{ir*}$ of task $i$ with the highest value of $p_i^{fa}(h_{ir*}) + p_i(h_{ir*})$;
    $p \leftarrow p_i^{fa}(h_{ir*}) + p_i(h_{ir*})$;
```

By way of illustration, a small example of the fast evaluation and the improved recursive method is discussed in Fig. 5. Task 2 is inserted between task 1 and task 3. The circles represent different observation start times of a task, and the number in the circle is its corresponding profit. The directed edges between every two consecutive tasks represent the possible transitions among the vertices. By using the improved version, only some of the edges need to be considered, indicated by solid arrows. The rest of the edges, indicated by dotted arrows, can be omitted. In this example, for each vertex of task 2, its forward accumulated profit is calculated from task 1 by *ForwardRecursion*(), and its backward accumulated profit is calculated from task 3 by *BackwardRecursion*().

For the forward recursion, since v_{12} has a larger profit than v_{11}, the forward accumulated profit of v_{21} is updated by $p_2^{fa}(h_{21}) = p_2^{fa}(h_{11}) + p_2(h_{21}) = 1 + 1 = 2$. Afterwards, since the profit of v_{13} is lower than that of v_{12}, there is no need to visit the edges connected to v_{13}. For the rest of the vertices of task 2, their forward accumulated profits are all updated based on the forward accumulated profit of vertex v_{12}. The backward accumulated profits of task 2 can also be calculated similarly, illustrated in *BackwardRecursion*(). Finally, the values of the forward and backward accumulated profits and their sums are displayed in the table. The total collected profit of this task sequence {1, 2, 3} is the maximum value of these sums ($p_{\{1,2,3\}} = 7$), where task 2 is inserted and observed at vertex v_{22}. The observation start times of other tasks can be easily determined by applying a backtracking method.

4.3. General outline

Algorithm 5 presents the framework of our BDP-ILS algorithm.

Algorithm 5 Bidirectional Dynamic Programming based Iterated Local Search.

```
PreProcessing();
$S_c \leftarrow \varnothing$, $S_b \leftarrow \varnothing$ ;// Current Solution $S_c$ and best-found solution $S_b$
$Iter \leftarrow 0$;// Count of iteration
while $Iter < IterationNum$ do
    while Can still insert task? do
        $S_c \leftarrow$InsertProcedure($S_c$);
    end while
    if $S_c$ is better than $S_b$ then
        $S_b \leftarrow S_c$;
    end if
    $Iter \leftarrow Iter + 1$;
    $S_c \leftarrow$ RemoveProcedure($S_c$);
    Full evaluation of $S_c$;
end while
return $S_b$;
```

The algorithm starts with a pre-processing procedure that identifies for each orbit which ordering of pairs of tasks is possible or not. This procedure prevents invalid insert attempts and reduces the computation time later in the process. Then, the algorithm performs a maximal number of iterations *IterationNum*. At each iteration, an insert procedure and a remove procedure are successively carried out. The insert procedure is iteratively executed until no extra tasks can be inserted. In the remove procedure, a certain number of consecutive tasks are removed from the current solution for each orbit. After that, a full evaluation is applied in order to obtain the collected profit. The newly generated solution can be accepted only if it is better than the best-found solution. This is the so-called Iterated Local Search with a random walk acceptance criterion (Lourenço et al., 2003). Notice that after the remove procedure, a full evaluation of the current solution is required in order to update the forward and backward accumulated profits for all remaining tasks.

4.4. Pre-processing

In order to maintain acceptable computational times on large instances, we present a pre-processing method. This method can be divided into two parts: the structure of neighboring tasks and the pre-calculation of transition times.

Firstly, inspired by a recent work (Verbeeck et al., 2017), we construct the neighboring relation among all the tasks in order to reduce the number of invalid insertion attempts during the search. For each task i on the same orbit, we define its preceding neighbor set V_i^p and its succeeding neighbor set V_i^s. A task j is a succeeding neighbor task of i only if it satisfies the following equation:

$$EarliestStartTime_{ij}(st_i) + d_i \le et_j - d_j,$$

where $EarliestStartTime_{ij}(st_i)$ calculates the earliest start time of task j based on the time window of task i. This equation indicates that it is possible for the satellite to observe task j after observing task i. Conceivably, the succeeding neighbor j of task i also means task i is the preceding neighbor of task j. An unscheduled task can be inserted between task i and task j, only if it is a succeeding neighbor of i and a preceding neighbor of j. Confining the inserted tasks can avoid a large number of invalid insert attempts and feasibility checks, and therefore significantly reduces the computational time.

Secondly, we pre-calculate the minimal transition times between each pair of tasks for each moment during their VTWs by using the calculations of $EarliestStartTime()$ and $LatestStartTime()$, as illustrated in Section 3.3. After that, once a task is scheduled at a certain moment, the earliest start time (latest start time) of its next (previous) task can be directly obtained without any calculation. We employ a dichotomy algorithm to accelerate these calculations. For simplicity, we only give the pseudo code of procedure $EarliestStartTime_{ij}(h_i)$ in Algorithm 6. This procedure calculates the earliest start time es_j of the next task j and its minimal transition time $mintrans_{ij}(h_i)$ after the previous task i is observed at moment h_i.

First of all, the start time st_j and the end time et_j of task j are respectively used to check the transition time constraint. If st_j satisfies the constraint, it is the earliest start time. If et_j is unacceptable, no feasible observation start time exists for task t_j. Otherwise, the algorithm tries to search the earliest start time during a specific time interval $[lb, ub]$ by repeatedly checking its midpoint and dividing the search interval. The lb and ub are initially assigned the values of st_j and et_j respectively and will be altered during the search. The search ends if the length of the interval is less than 2 s. Then the ub is recorded as the earliest start time, and its corresponding transition time is the minimal transition time $mintrans_{ij}(h_i)$.

Despite a considerable amount of memory usage, this pre-calculation method only costs a few seconds. After the minimal transition times for each pair of tasks at each moment are pre-calculated, the computational time needed to calculate the four

Algorithm 6 Procedure $EarliestStartTime_{ij}(h_i)$.

```
Calculate the transition time $trans_{ij}(h_i, st_j)$ and $trans_{ij}(h_i, et_j)$.
if $h_i + d_i + trans_{ij}(h_i, st_j) \le st_j$ then
    Return $es_j = st_j$ and $mintrans_{ij}(h_i) = trans_{ij}(h_i, st_j)$;
else
    if $h_i + d_i + trans_{ij}(h_i, et_j) > et_j$ then
        No feasible observation time for task $j$, Stop;
    else
        $lb \leftarrow st_j$; $ub \leftarrow et_j$;
        while true do
            $t \leftarrow lb + \frac{(ub-lb)}{2}$;
            Calculate the transition time $trans_{ij}(h_i, t)$;
            if $ub - lb < 2$ then
                Return $es_j = ub$, $mintrans_{ij}(h_i) = trans_{ij}(h_i, t)$;
            end if
            Calculate its transition finish time $F_{trans} = h_i + d_i + trans_{ij}(h_i, t)$;
            if $F_{trans} \le t$ then
                $ub \leftarrow t$;
            else
                $lb \leftarrow t$;
            end if
        end while
    end if
end if
```

auxiliary features (see Section 4.1) for each insertion is significantly reduced.

4.5. Insert procedure

The time-dependent transition time, the visible time window and the time-dependent profits increase the complexity and difficulty of an insertion procedure. On the one hand, when inserting an unscheduled task into a sequence, it should be verified that all the tasks scheduled after the insert position still satisfy the transition time and visible time window constraints. This full feasibility check would require much computational time. On the other hand, inserting an unscheduled task may decrease the total collected profit, since the observation start times and the collected profit of the tasks after the insert position may be changed. Therefore, the main contribution of our insert procedure can be summarized into two parts: an efficient feasibility check in order to handle the transition time constraint and a fast evaluation when considering time-dependent profits. The pseudo code of the insert procedure is presented in Algorithm 7.

As mentioned in Section 4.1, the earliest start time and the latest start time are recorded for each task in the solution. These two auxiliary features are inspired from an evaluation metric (called "maxshift") in recent work on TOPTW (Vansteenwegen et al., 2009) and TDOP (Verbeeck et al., 2014). The difference is that in our algorithm, the observation start time of each task in the current solution is only determined during the evaluation of an insertion. As shown in Fig. 6, the earliest start time can be iteratively calculated from the first task to the last task in the solution by using the auxiliary function $EarliestStartTime()$. The earliest start time of a task is calculated based on that of its previous task, and the earliest start time of the first task is equal to the start time of its visible time window. Likewise, the latest start time of each selected task can be calculated from the last task to the first task.

In Fig. 6, when trying to insert task i between task j and task $(j+1)$, task i should belong to the succeeding neighbor set V_j^s of task j and the preceding neighbor set V_{j+1}^p of task $(j+1)$. Otherwise, it will be discarded. This step avoids a large number

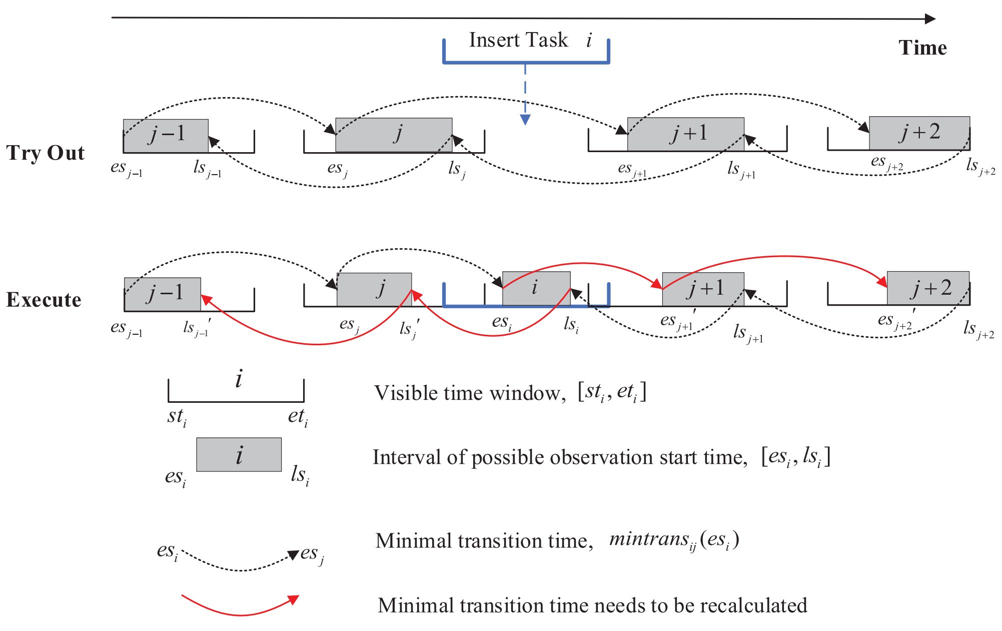


Fig. 6. An example of insert procedure.

Algorithm 7 InsertProcedure(S_c).

```
for each unscheduled target $I$ do
    Find all the tasks (VTWs) of target $I$;
    for each task $i$ of target $I$ do
        Find the task sequence $\{1, \dots, n\}$ in the current solution $S_c$ during the orbit of $i$.
        for each insert position $ip$ in $\{1, \dots, n\}$ do
            Assume position $ip$ is between task $j$ and $(j+1)$;
            if $i \in V_j^s$ and $i \in V_{j+1}^p$ then
                $es_i \leftarrow EarliestStartTime_{ji}(es_j)$;
                $ls_i \leftarrow LatestStartTime_{i(j+1)}(ls_{j+1})$;
                if $es_i <= ls_i$ then
                    Evaluate the insertion of $i$ by using $FastEvaluation(i, j, (j+1))$;
                    Store the insertion as a candidate insertion if the profit increases;
                end if
            end if
        end for
    end for
end for
for each orbit $k$ in the current solution do
    Execute the candidate insertion with the highest profit increase during orbit $k$;
    Remove the other candidate insertions which observe the same target;
    for each task after the inserted task in the current solution do
        Update its earliest start time and forward accumulated profit;
    end for
    for each task before the inserted task in the current solution do
        Update its latest start time and backward accumulated profit;
    end for
end for
```

of invalid insert attempts and reduces computational time. Afterwards, we calculate the earliest and latest start times of task i based on the earliest start time of task j and the latest start time of task $(j+1)$. This task can be inserted into the position only if its earliest start time is no later than its latest start time, i.e., $es_i <= ls_i$. Otherwise, inserting this task will make the solution infeasible due to the transition time and visible time window constraints. As a result of this insert mechanism, the computational time required to check these constraints for other tasks in the solution is significantly reduced.

After the feasibility check, a fast evaluation for an insertion based on the BDP is applied to the solution (see Algorithm 4). This method can efficiently evaluate an insertion without updating the observation start times of the other tasks in the solution. Only if the total collected profit increases, the insertion will be stored as a candidate insertion. After trying all the unscheduled tasks, the candidate insertion with the highest profit increase will be executed. The other candidate insertions that observe the same target will not be considered for the next insertion anymore, which ensures that only one task is scheduled for each target. After insertion, the tasks after the inserted task require an update of the earliest start time and the forward accumulated profit, and the task before the inserted task require an update of the latest start time and the backward accumulated profit. These four auxiliary features are updated according to the functions listed in Section 4.2.

4.6. Remove procedure

The remove procedure is used to escape local optima. In this procedure, one or more tasks are removed from the current solution for each orbit. A remove ratio α is used to indicate how many consecutive tasks to remove in the solution during each orbit. Namely, each time the algorithm removes $\lfloor \alpha * |n^k| \rfloor$ consecutive tasks for orbit k, where $|n^k|$ represents the number of scheduled tasks during orbit k in the current solution. The place where to start the removal process is randomly chosen from the task sequence for each orbit. If during removal the last scheduled task is reached, it continues after the first task in the current solution. The removal process is always accepted no matter how the solution changes. This is called the random walk acceptance criterion in the ILS procedure (Lourenço et al., 2003). After the removal,

Table 1
Results of the scheduling without time-dependent profits for the Chinese instances.

Scenario	N_t	P_t	ALNS				ILS				
($\|T\|$)			T_s	P_s	P_s/P_t	$CPU(s)$	T_s	P_s	P_s/P_t	$OutPerform(\%)$	$CPU(s)$
100_A	179	571	81.8	520.4	0.911	22.06	98.2	568.4	0.995	8.41	13.72
200_A	361	1066	88.8	635.2	0.603	89.69	151.0	894	0.849	24.58	69.20
300_A	523	1484	85.4	640.6	0.435	175.65	167.8	998.6	0.677	24.29	147.14
400_A	716	2075	94.4	654.2	0.317	303.85	188.0	1162.8	0.563	24.63	260.36
500_A	912	2718	91.0	709	0.264	420.61	193.4	1297.6	0.484	21.94	398.85
600_A	1091	3263	97.4	813.4	0.252	504.08	202.4	1399.8	0.433	18.15	478.56
Average					0.464				0.667	20.33	

Table 2
Results of the scheduling without time-dependent profits for the worldwide instances.

Scenario	N_t	P_t	ALNS				ILS				
($\|T\|$)			T_s	P_s	P_s/P_t	$CPU(s)$	T_s	P_s	P_s/P_t	$OutPerform(\%)$	$CPU(s)$
100_W	398	550	100.0	550	1.000	8.02	100.0	550	1.000	0.00	5.83
200_W	381	1004	193.0	1004	1.000	15.50	193.0	1004	1.000	0.00	10.49
300_W	615	1622	291.0	1622	1.000	31.61	291.0	1622	1.000	0.00	25.89
400_W	800	2263	387.4	2254.6	0.996	265.36	390.0	2263	1.000	0.37	45.77
500_W	985	2693	469.4	2660.2	0.988	398.91	480.0	2686	0.997	0.96	74.10
600_W	1056	3129	439.8	2783.8	0.890	621.47	579.0	3122	0.998	10.81	103.09
Average					0.979				0.999	2.02	

the earliest start times of the tasks succeeding the removed tasks should be updated, using the same method in the insert procedure. Similarly, for the tasks preceding the removed task, the latest start times should be updated.

5. Experimental results

Since we are the first to solve the AEOS, considering both time-dependent transition times and time-dependent profits, we cannot directly compare the performance of our heuristic with other approaches. Therefore, we first compare the performance of our algorithm with the ALNS developed in (Liu et al., 2017) while solving the AEOS without time-dependent profits (but with time-dependent transition times). In the second experiment, we evaluate the performance of our heuristic for the complete AEOS, including both time-dependent profits and transition times.

All the algorithms were implemented in C#, and the experiments were tested on a personal computer Intel Core i5 with 2.5 GHz processor and 8 GB Ram.

5.1. Test instances

As mentioned by Liu et al. (2017), different physical design and ability parameters lead to large differences between the AEOS in different countries regarding capability, constraints and management. No common benchmark instances are available for this AEOS scheduling problem.

However, since we consider the same agile satellite AS-01 as in Liu et al. (2017) and we want to compare the performance of our algorithm with theirs, we will use their test instances. The targets for observation are randomly generated with a uniform distribution in two regions: a Chinese area distribution (3°N-53°N and 74°E-133°E) and a worldwide distribution. The Chinese instances are much more difficult to solve than the worldwide instances, since the VTWs in the Chinese instances considerably overlap with each other, while in the worldwide instances, the VTWs rarely overlap. The invalid (parts of) VTWs in which the satellite flies in the dark are removed from the test instances. The profit and the observation duration of each target are uniformly generated in the range of [1,10] and [15,30] in seconds. The scheduling horizon is 24 h according to the rules of practical operations and the number of orbits $|O|$ is around 15. More details on the test instances and the basic parameters of the AS-01 satellite can be found in the paper of Liu et al. (2017).

5.2. Results without time-dependent profits

A crucial novelty of our algorithm is the way we handle the time-dependent transition times, as discussed in Section 4.4. In order to evaluate the effect of this novelty, we compare our ILS with the ALNS algorithm of Liu et al. (2017) without considering the time-dependent profits. In this case, the BDP is not necessary to evaluate each insertion, and the full evaluation of a solution after the remove procedure is removed.

Tables 1 and 2 compare the results of both algorithms on the Chinese and worldwide instances. All results are based on the average of 5 runs. In order to make a fair comparison, the maximum number of iterations of both algorithms is set to 5000. Other parameters of ALNS can be found in the paper of Liu et al. (2017). The scenarios are denoted by "$|T|$_M", where $|T|$ is the number of targets in this instance and "M" is the distribution mode, with "A" for the Chinese area distribution and "W" for the worldwide distribution. N_t and P_t respectively represent the number of valid tasks (i.e., VTWs in the sunshine) and the sum of all the targets' profits. The number of scheduled targets T_s, the collected profit P_s and the average computational time *CPU* (in seconds) are recorded. The column P_s/P_t is the ratio of the scheduled profits compared to the total target profit. The higher this P_s ratio, the better the result. The *OutPerform* column shows how many percents our ILS outperforms ALNS in terms of the scheduled profits compared to the total profit.

Table 1 shows the results for the Chinese instances. The scheduled profits of our ILS are on average 20.33% higher than those of the ALNS for these instances. When the number of targets increases, the number of scheduled targets remains stable in ALNS while this number keeps growing for our ILS. In the instances with more than 300 targets, the number of scheduled targets and the collected profit are almost twice those of ALNS. That this percentage in itself is rather low is not surprising since the targets in a specific area are distributed closely together and have overlapping VTWs. Due to the limited imaging time of the satellite, selecting targets is required. The results also show that the computational times of these two algorithms are similar. However, in most cases,

Table 3
Comparison of the optimal solution and the BDP-ILS solution with time-dependent profits for small instances.

Scenario	N_t	CPLEX		BDP-ILS		
($\|T\|$)		$P_{optimal}$	$CPU(s)$	P_s	$Gap(\%)$	$CPU(s)$
50_A	90	252.19	115.83	251.31	0.35	0.60
80_A	142	410.77	1304.05	405.70	1.23	0.88
100_A	179	494.93	2429.01	487.14	1.57	1.23
50_W	108	242.10	21.26	242.04	0.02	0.47
80_W	154	388.50	70.20	388.38	0.03	0.72
100_W	398	–	–	529.04	–	1.51

our ILS algorithm converges much faster than the ALNS, which means the computation time of our ILS could easily be reduced, without loss of quality, by reducing the number of iterations. The comparison of the different number of iterations is discussed in Section 5.4.

Table 2 shows that both of the ALNS and the ILS can schedule almost all of the targets in all the worldwide instances except for the 600-targets instance, which is very different from the results of the Chinese instances. This is because, in the worldwide instances, the targets are spread more evenly over all orbits, while in the Chinese instances the targets only appear in around half of the orbits (during daytime in China). Therefore, the satellite has sufficient scheduling time to image most of the targets in these instances. However, in the instance with 600 targets, the best-found solution of ALNS only has 439.8 targets and 89% of profits on average while ILS can still schedule 579 targets and 99.8% of profits. It proves the high-performance quality of our algorithm for large instances. When comparing the computational time, our ILS is many times faster than the ALNS algorithm. For instances with no more than 500 targets, our ILS only takes less than 100 s to obtain a satisfactory solution. The computational times of ILS increase much less with the size of the instances compared to the ALNS. However, if we compare the computational times of ILS in Tables 1 and 2, it seems that the complexity of solving this scheduling problem is more related to the degree of overlap between time windows (larger in the Chinese instances), rather than to the number of targets available. We conclude that our ILS is both faster and significantly more effective than the only previously published algorithm for solving the AEOS problem without time-dependent profits.

5.3. *Results with time-dependent profits*

In order to evaluate the performance in case of time-dependent profits, we carry out two experiments. The first experiment is a straightforward comparison of the results obtained by solving our MILP model (see Section 3.5) with the commercial solver CPLEX (version 12.8.0) and our BDP-ILS for small instances with 50, 80 or 100 targets. Unfortunately, instances with more than 50 targets could not be solved due to memory limitations. Since a complete VTW contains around 300 s, considering each second during the VTW seems to be unnecessary. Thus, we define a time step T_{step} to evenly discretize the VTWs and specify the resolution for these small instances. As can be expected, a large time step can lead to a loss of solution quality but reduces the computational time and memory usage. The effect of different values of the time step will be discussed in Section 5.4. Fig. 3 shows the results of CPLEX and our algorithm on small instances with $T_{step} = 5$ s. Since both CPLEX and the BDP-ILS require a pre-calculation of minimal transition times, the required computing time of the pre-processing is included in the computation time, as displayed in the column *CPU*(s). The column *Gap* represents the percentage gap between the collected profit of the optimal solution and the collected profit of our algorithm.

The results in Table 3 prove the validity of our MILP model and the excessive computational time despite an approximation of the original input data with T_{step}. The Chinese instances require a higher computational time than the worldwide instances for CPLEX with a similar number of tasks (VTWs). Furthermore, this table illustrates that our BDP-ILS algorithm obtains high-quality results and has very short computational times on these small instances. However, due to the memory limitations, larger instances could not be solved by CPLEX based on our MILP model.

The second experiment evaluates the performance of our algorithm on large instances. We introduce four algorithms to be compared with our BDP-ILS: the Bidirectional Dynamic Programming based Adaptive Large Neighborhood Search (BDP-ALNS), the “regular Iterated Local Search”, the Earliest Start based Iterated Local Search (ES-ILS), and Unidirectional Dynamic Programming based Iterated Local Search (UDP-ILS). The BDP-ALNS retains the framework of the ALNS by Liu et al. (2017) including the destroy operators and repair operators which insert unscheduled tasks according to three randomly selected strategies. But since the original ALNS does not consider time-dependent profits, we apply the BDP to the newly generated solution at each iteration of the ALNS. In this way, the observation start time of each selected task is optimized given the time-dependent profits. The last three algorithms are all based on our ILS framework including the insert procedure, the remove procedure, and the acceptance criterion. The difference is how the observation start time of each task in the solution is determined and how the solution quality is evaluated.

The “regular ILS” is the same as the ILS without considering the time-dependent profits, where only the earliest start time and the latest start time are recorded for each selected task. This method inclines to insert the task with the highest target profit in each insert procedure. The information of time-dependent profits is not utilized during the search. In order to make a fair comparison, after the search, the BDP is applied to the best-found solution to optimize the observation start time and obtain the highest possible profit.

As for the ES-ILS, in the insert procedure, the inserted task is always scheduled at its earliest start time and the profit at that moment is collected. The insertion is accepted only if the total collected profit is improved. This insertion mechanism is generally used in papers of the Orienteering Problem with Time-dependent Rewards (OP-TDR) (Ekici and Retharekar, 2013), where the profit of each vertex decreases over time. Scheduling the observations as early as possible is also frequently used in the satellite scheduling applications. For the same reason as with the “regular ILS”, the BDP is also applied to the best-found solution in the ES-ILS.

Unlike the BDP-ILS, the UDP-ILS only adopts the *ForwardRecursion*(), meaning that only the forward accumulated profit is recorded for each task at each moment. After each insertion, in order to obtain the maximal collected profit of a task sequence, it needs to update forward accumulated profit from the inserted task to the last task. The maximal forward accumulated profit of the last task is equal to the maximal collected profit of that sequence, and then the observation start time for each task is determined. It can be expected that the solution quality of the UDP-ILS and the BDP-ILS are similar since they are all based on the dynamic programming approach. However, when evaluating each insertion, the UDP needs to update the observation start times of all the tasks after the inserted task, while only the inserted task requires an update in BDP.

Table 4 compares the four reference algorithms and our BDP-ILS on the Chinese instances. Since CPLEX solutions are not considered in this experiment, the time step of these instances is set to 1 s. The maximal number of iterations of the four algorithms is set to 200 and the remove ratio α is set to 0.1. The columns are the same as used in Tables 1 and 2.

Table 4
Results of the scheduling with time-dependent profits for the Chinese instances.

Scenario ($\|T\|$)	N_t	P_t	BDP-ALNS P_s/P_t	BDP-ALNS $CPU(s)$	regular ILS P_s/P_t	regular ILS $CPU(s)$	ES-ILS P_s/P_t	ES-ILS $CPU(s)$	UDP-ILS P_s/P_t	UDP-ILS $CPU(s)$	BDP-ILS P_s/P_t	BDP-ILS $CPU(s)$
100_A	179	571	0.67	3.26	0.69	1.63	0.77	7.71	0.86	34.51	0.86	8.77
200_A	361	1053	0.41	5.14	0.54	6.43	0.62	41.29	0.71	186.83	0.71	33.13
300_A	523	1474	0.32	8.25	0.42	15.15	0.50	69.38	0.57	221.92	0.58	50.87
400_A	716	2065	0.25	9.01	0.36	31.88	0.43	120.02	0.50	453.31	0.51	100.91
500_A	912	2683	0.19	10.61	0.31	55.52	0.37	180.81	0.43	708.01	0.43	166.76
600_A	1091	3231	0.16	14.20	0.28	79.91	0.33	250.67	0.39	953.30	0.39	228.01

Table 5
Results of the scheduling with time-dependent profits for the worldwide instances.

Scenario ($\|T\|$)	N_t	P_t	BDP-ALNS P_s/P_t	BDP-ALNS $CPU(s)$	regular ILS P_s/P_t	regular ILS $CPU(s)$	ES-ILS P_s/P_t	ES-ILS $CPU(s)$	UDP-ILS P_s/P_t	UDP-ILS $CPU(s)$	BDP-ILS P_s/P_t	BDP-ILS $CPU(s)$
100_W	398	550	0.94	6.09	0.84	2.57	0.83	5.51	0.96	29.05	0.96	11.15
200_W	381	1004	0.88	10.89	0.82	2.48	0.80	6.58	0.97	43.96	0.97	16.04
300_W	615	1622	0.87	18.01	0.78	5.54	0.80	17.81	0.96	126.40	0.96	32.29
400_W	800	2263	0.87	27.02	0.77	9.42	0.79	31.55	0.95	220.57	0.95	49.65
500_W	985	2693	0.82	33.46	0.73	14.55	0.79	55.09	0.95	398.21	0.94	74.97
600_W	1056	3129	0.72	34.46	0.70	17.46	0.76	78.83	0.91	558.68	0.91	89.90

When comparing the profit, the UDP-ILS and the BDP-ILS perform noticeably better than the other three algorithms. It proves that when the time-dependent profits are considered, the observation start times of each task should be optimized during the search. The dynamic programming approach can effectively optimize the observation start times and thus guide the search. The ALNS of Liu et al. (2017), extended with the BDP to optimise the observation start times, is clearly outperformed by the other algorithms. There are two reasons for this: the local search operators of ALNS are not designed well to allow more candidate tasks to be inserted and these operators cannot efficiently evaluate the insertions in case of time-dependent profits. The results of ES-ILS demonstrates that scheduling the tasks at their earliest start moments is useless in the case of time-dependent profits with a non-monotonic distribution. For the computational time, our BDP-ILS is many times faster than the UDP-ILS, since only the inserted task requires an update of the forward and backward accumulated profits, benefiting from the bidirectional recursion mechanism. While in UDP-ILS, the forward accumulated profits of all the tasks following the inserted task should be updated in order to obtain the maximal collected profit after the insertion.

Table 5 presents the comparison results for the worldwide instances. The same conclusion can be drawn for these results, illustrating that the UDP-ILS and our BDP-ILS still outperform the other algorithms regarding solution quality. Unlike the results on the Chinese instances, the BDP-ALNS performs slightly better than the regular ILS and the ES-ILS for most of the worldwide instances. This is because in the worldwide instances, most of candidate tasks can be selected by both the ALNS and ILS. However, the BDP-ALNS optimizes the observation start times at each iteration while the regular ILS and the ES-ILS only apply the BDP at the end of the algorithm, which results in the difference. Even so, the BDP-ALNS is still outperformed by the UDP-ILS and the BDP-ILS, since the insert operator of the latter ones can efficiently distinguish which insertions can improve the solution. For these instances, more than 90% of the profits are collected in the UDP-ILS or BDP-ILS solution, since VTWs in the worldwide instances rarely overlap each other, and thus most of tasks are scheduled at the nadir point (the midpoint of a VTW). For the worldwide instance with 500 targets, the profits of UDP-ILS are slightly larger than those of BDP-ILS. However, a more evident difference is that the BDP-ILS runs several times faster than the UDP-ILS for the worldwide instances.

5.4. Sensitivity analysis

In this section, the impact of the design parameters of the BDP-ILS and the instances is discussed. The first parameter that needs separate attention is the time step T_{step} used in the instances. It is interesting to know how much the solution quality and the computational time will change for both our MILP and our algorithm when setting different values of T_{step}. Table 6 shows the solution quality of T_{step} with 1, 5 or 10 s on small instances. It can be expected that a larger T_{step} leads to a reduction of the computational time and memory usage. We found that CPLEX cannot solve these instances with $T_{step} = 1$ s, and even the 100-target worldwide instance with $T_{step} = 5$ s. Thus, only the gap between the optimal solution and the BDP-ILS solution with $T_{step} = 5$ *or* 10 s are presented in the table. The results show that the gap between the optimal solution and our algorithm is quite small for these small instances, while a very short computational time is required for our algorithm. As T_{step} increases from 5 s to 10 s, the solution quality slightly decreases, which infers a conclusion that for the instances with $T_{step} = 1$ s, our algorithm can still obtain a high-quality solution, even though the optimal solution is unknown. A second conclusion is that the loss of solution quality for the worldwide instances are particularly smaller as T_{step} increases, compared to the Chinese instances. Therefore, it is reasonable and effective to reduce the computational time for the worldwide instances by setting a larger T_{step} without losing much solution quality.

Now we will analyze the parameters of our algorithm: the maximal number of iterations *IterationNum* and the remove ratio α. Table 7 presents the comparative results of these parameters tested on both the Chinese instance and the worldwide instance with 200 targets. Results for other numbers of targets are similar. All the results are based on five independent runs of the algorithm. The first row summarises the performance of our algorithm with the basic setting of these parameters. The column P_s is the gap of the collected profit compared to the results in the first row, displayed as a percentage. The first parameter *IterationNum* is altered from 200 to 100 or 400. Unsurprisingly, the solutions keep improving and the computational time increases rapidly when the number of iterations is increased. However, as the number is increased from 200 to 400, the difference in solution quality remains small. It indicates that our algorithm can converge to a satisfactory solution within 200 iterations. The comparison of different α values shows that for that Chinese instance, a lower remove ratio ($\alpha = 0.05$) leads to

Table 6
Effect of time step T_{step} for the CPLEX and the BDP-ILS on small instances.

Scenario	CPLEX				BDP-ILS							
	$T_{step} = 5$		$T_{step} = 10$		$T_{step} = 1$		$T_{step} = 5$			$T_{step} = 10$		
($\|T\|$)	P_s	$CPU(s)$	P_s	$CPU(s)$	P_s	$CPU(s)$	P_s	$Gap(\%)$	$CPU(s)$	P_s	$Gap(\%)$	$CPU(s)$
50_A	252.19	115.83	250.13	19.77	248.20	1.84	246.53	2.25	0.42	244.43	2.28	0.26
100_A	494.93	2429.01	488.99	388.31	486.86	4.61	484.47	2.11	1.00	476.83	2.49	0.63
50_W	242.10	21.26	241.25	5.11	242.97	1.21	241.27	0.34	0.34	240.60	0.27	0.23
100_W	–	–	527.58	1112.28	529.18	4.90	527.84	–	1.06	523.42	0.79	0.70

Table 7
Sensitivity analysis of the design decisions.

Testing Parameter	Parameter Value		Chinese instance		Worldwide instance	
	IterationNum	α	P_s	$CPU(s)$	P_s	$CPU(s)$
basic setting	200	0.1	747.65	32.32	969.57	16.20
IterationNum	100	0.1	745.74	20.19	969.10	9.27
	400	0.1	750.81	64.49	970.22	31.83
α	200	0.05	752.63	25.44	968.68	15.78
	200	0.2	739.85	51.18	968.71	22.81

a better solution, while for the worldwide instance, the algorithm performs better with $\alpha = 0.1$. The results suggest that changing the value of α influences the quality of the results, however, determining the best value for α depends on the specific instance considered.

6. Conclusions and further work

In this paper, we study the AEOS scheduling problem with time-dependent transition times and time-dependent profits. This problem arises from the fact that the observation start time of an observation influences its image quality and the transition time. This problem is formulated as a mixed integer programming model. A Bidirectional Dynamic Programming based Iterated Local Search algorithm (BDP-ILS) is developed to solve the problem.

In order to handle the time-dependent transition times, we present a concept of “minimal transition time” and specifically design a fast insert procedure to avoid a full feasibility check of each insert attempt. We compare our algorithm with the state-of-the-art ALNS algorithm for the same scheduling problem but without time-dependent profits. The results for one set of benchmark instances show that our algorithm performs on average 20.33% better than the state-of-the-art algorithm while the computation times are similar. For the other set of available benchmark instances, our algorithm performs slightly better in terms of solution quality, but the computational time is much shorter.

When considering time-dependent profits, a bidirectional dynamic programming approach is proposed and incorporated into the ILS heuristic in order to efficiently and accurately evaluate the solution during the search. The results of our algorithm are compared with the optimal results found by solving the MIP model presented in Section 3.5 for a set of small instances. The results prove that our algorithm can find a high-quality solution within a short computational time. Four reference algorithms are presented to evaluate the performance of our BDP-ILS algorithm for large instances. The results illustrate that the dynamic programming approach that optimizes the observation start times for each insert move can effectively improve the solution when time-dependent profits are considered. Furthermore, our BDP-ILS requires very small computational time due to the bidirectional recursion mechanism.

Further research could focus on developing algorithms for multiple satellites scheduling where the size of instances are typically huge, and a workload-balancing between different satellites should be enforced. Also the scheduling of downloading images should be considered during satellite scheduling in the future. Moreover, this work can be extended to AEOS scheduling problems with uncertainty in which the presence of clouds over targets reduces the images quality, and thus the success probability of observations. Therefore, according to the prediction of clouds, a different observation start time leads to a different success probability. In order to improve the expected total profit, each target can be observed more than once. To the best of our knowledge, this problem has not yet been studied. Furthermore, our Bidirectional Dynamic Programming approach for the time-dependent profits can be extended to other combinational optimization problems such as the vehicle routing problem or the job shop scheduling problem.

Acknowlegements

This work was supported by the National Natural Science Foundation of China (No. 71501180, 71801218,U1501254 and 61773120) and the China Scholarship Council. The authors would like to thank Xiaolu Liu for providing the code of her algorithm and useful comments.

References

Afsar, H.M., Labadie, N., 2013. Team orienteering problem with decreasing profits. Electron. Notes Discrete Math. 41, 285–293.
Bianchessi, N., Righini, G., 2008. Planning and scheduling algorithms for the COSMO-SkyMed constellation. Aerosp. Sci. Technol. 12 (7), 535–544.
Cordeau, J.-F., Laporte, G., 2005. Maximizing the value of an earth observation satellite orbit. J. Oper. Res. Soc. 56 (8), 962–968.
Ekic, A., Keskinocak, P., Koenig, S., 2009. Multi-robot routing with linear decreasing rewards over time. In: 2009 IEEE International Conference on Robotics and Automation, pp. 958–963.
Ekici, A., Retharekar, A., 2013. Multiple agents maximum collection problem with time dependent rewards. Comput. Ind. Eng. 64 (4), 1009–1018.
Erkut, E., Zhang, J., 1996. The maximum collection problem with time-dependent rewards. Nav. Res. Logist. 43 (5), 749–763.
Gabrel, V., Moulet, A., Murat, C., Paschos, V.T., 1997. A new single model and derived algorithms for the satellite shot planning problem using graph theory concepts. Ann. Oper. Res. 69 (0), 115–134.
Grasset-Bourdel, R., Verfaillie, G., Flipo, A., 2011. Building a really executable plan for a constellation of agile earth observation satellites. IWPSS-International Workshop on Planning & Scheduling for Space. Darmstadt
Gunawan, A., Lau, H., Yuan, Z., Fügenschuh, A., 2014. A mathematical model and metaheuristics for Time Dependent Orienteering Problem. Angewandte Mathematik und Optimierung Schriftenreihe. Helmut-Schmidt-Univ., Univ. der Bundeswehr Hamburg.
Habet, D., Vasquez, M., Vimont, Y., 2010. Bounding the optimum for the problem of scheduling the photographs of an agile earth observing satellite. Comput. Optim. Appl. 47 (2), 307–333.

He, L., Liu, X., Laporte, G., Chen, Y., Chen, Y., 2018. An improved adaptive large neighborhood search algorithm for multiple agile satellites scheduling. Comput. Oper. Res. 100, 12–25.

Lemaître, M., Verfaillie, G., Jouhaud, F., Lachiver, J.-M., Bataille, N., 2002. Selecting and scheduling observations of agile satellites. Aerosp. Sci. Technol. 6 (5), 367–381.

Liao, D., Yang, Y., 2007. Imaging order scheduling of an earth observation satellite. IEEE Trans. Syst. Man Cybern.Part C 37 (5), 794–802.

Liu, X., Laporte, G., Chen, Y., He, R., 2017. An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time. Comput. Oper. Res. 86, 41–53.

Lourenço, H.R., Martin, O.C., Stützle, T., 2003. Iterated Local Search. Springer US, Boston, MA, pp. 320–353.

Malandraki, C., Daskin, M.S., 1992. Time dependent vehicle routing problems: formulations, properties and heuristic algorithms. Transp. Sci. 26 (3), 185–200.

Pralet, C., Verfaillie, G., 2013. Timedependent simple temporal networks: properties and algorithms. RAIRO 47 (2), 173198.

Stützle, T., 1999. Local search algorithms for combinatorial problems - analysis, improvements, and new applications. DISKI.

Tangpattanakul, P., Jozefowiez, N., Lopez, P., 2015. A multi-objective local search heuristic for scheduling earth observations taken by an agile satellite. Eur. J. Oper. Res. 245 (2), 542–554.

Vansteenwegen, P., Souffriau, W., Berghe, G.V., Oudheusden, D.V., 2009. Iterated local search for the team orienteering problem with time windows. Comput. Oper. Res. 36 (12), 3281–3290.

Verbeeck, C., Sörensen, K., Aghezzaf, E.-H., Vansteenwegen, P., 2014. A fast solution method for the time-dependent orienteering problem. Eur. J. Oper. Res. 236 (2), 419–432.

Verbeeck, C., Vansteenwegen, P., Aghezzaf, E.-H., 2017. The time-dependent orienteering problem with time windows: a fast ant colony system. Ann. Oper. Res. 254 (1), 481–505.

Victoria, J.F., Afsar, H.M., Prins, C., 2015. Vehicle routing problem with time-dependent demand in humanitarian logistics. In: 2015 International Conference on Industrial Engineering and Systems Management (IESM), pp. 686–694.

Wolfe, W.J., Sorensen, S.E., 2000. Three scheduling algorithms applied to the earth observing systems domain. Manage. Sci. 46 (1), 148–166.

Yi, J., 2003. Vehicle routing with time windows and time-dependent rewards: a problem from the american red cross. Manuf. Serv. Oper. Manage. 5 (1), 74–77.