8-2020

An exact algorithm for Agile Earth Observation Satellite Scheduling with time-dependent profits

Guansheng PENG

Guopeng SONG

Lining XING

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

Pieter VANSTEENWEGEN

An Exact Algorithm for Agile Earth Observation Satellite Scheduling with Time-Dependent Profits

Guansheng Peng[ab], Guopeng Song[ac], Lining Xing[a], AldyGunawan[d], PieterVansteenwegen[b]

[a] *College of Systems Engineering, National University of Defense Technology, Changsha, 410073, China*
[b] *KU Leuven Mobility Research Center - CIB, Leuven, 3001, Belgium*
[c] *KU Leuven ORSTAT, Leuven, 3001, Belgium*
[d] *School of Information Systems, Singapore Management University, Singapore 178902*

Published in Computers & Operations Research, August 2020, Volume 120, Article No. 104946
https://doi.org/10.1016/j.cor.2020.104946

Abstract

The scheduling of an Agile Earth Observation Satellite (AEOS) consists of selecting and scheduling a subset of possible targets for observation in order to maximize the collected profit related to the images while satisfying its operational constraints. In this problem, a set of candidate targets for observation is given, each with a time-dependent profit and a visible time window. The exact profit of a target depends on the start time of its observation, reaching its maximum at the midpoint of its visible time window. This time dependency stems from the fact that the image quality is determined by the look angle between the satellite and the target to be observed. We present an exact algorithm for the single-orbit scheduling for an AEOS considering the time-dependent profits. The algorithm is called Adaptive-directional Dynamic Programming with Decremental State Space Relaxation (ADP-DSSR). This algorithm is based on the dynamic programming approach for the Orienteering Problem with Time Windows (OPTW). Several algorithmic improvements are proposed to address the time-dependent profits. The proposed algorithm is evaluated based on extensive computational tests. The experimental results show that the algorithmic improvements significantly reduce the required computational time. The comparison between the proposed exact algorithm and a state-of-the-art heuristic illustrates that our algorithm can find the optimal solutions for sufficiently large instances within limited computational time. The results also show that our algorithm is capable of efficiently solving benchmark OPTW instances.

Keywords
Agile satellite scheduling, Time-dependent profits, Dynamic programming, Decremental state space relaxation

1. Introduction

The Agile Earth Observation Satellite (AEOS) belongs to a new generation of imaging platforms to acquire images of targets on the Earth surface in response to observation requests, playing an increasingly important role in resource exploration, disaster alerts, climate change analysis, and other applications Liu et al. (2017). The scheduling of an AEOS corresponds to scheduling a set of weighted observations with the objective of maximizing the total collected profit associated with each observation, while satisfying the operational constraints.

The AEOS can be mobile on three axes (roll, pitch and yaw), thus allowing maneuverability for image acquisitions and transitions between every two consecutive observations. The maneuverability of the roll angle allows the observations of targets located at two sides of the sub-satellite point in a certain range. The mobility of the pitch angle enables the satellite to observe targets before or after the upright pass (called the “nadir point”), as can be seen in Figure 1. The agility associates each target with a long time window, called the Visible Time Window (VTW). Note that observing a target at different moments during its VTW corresponds to different pitch angles, and observing different targets corresponds to different roll angles. For example, in Figure 1, observation 1 and 2 have different pitch angles towards target 1, and observing target 1 and 2 at their own nadir points corresponds to different roll angles. For each pair of consecutive observations, a transition time is required to finish the maneuverability process. The length of the transition time is proportional to the angular changes on the three axes, and therefore the transition time depends on the start times of these two consecutive observations. The regular AEOS scheduling consists of the selection and sequencing of possible targets to be observed, satisfying the VTW constraints and the time-dependent transition time constraints.

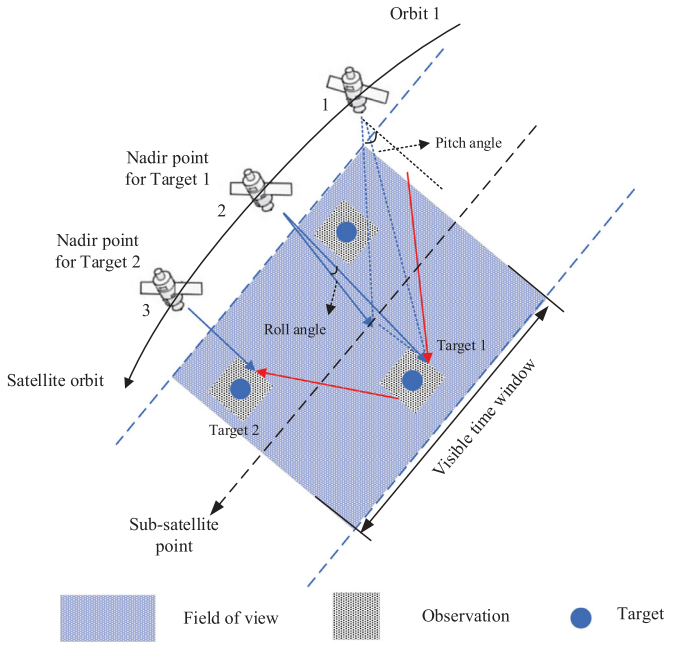


Fig. 1. An agile satellite images a target at different start times on orbit 1.

In this paper, we consider an additional time-dependency feature for the AEOS scheduling, which is the time-dependent profits. It is due to the fact that a satellite image acquired at the nadir point has a higher image quality than those taken at the side of the VTW. The time-dependent profits significantly increase the solution space since the exact start time of each observation becomes important. This makes the problem much more difficult to solve. The AEOS scheduling can be classified into two categories: the single-orbit scheduling and the multiple-orbit scheduling problems. An orbit is defined as the time interval that the satellite flies in the sunshine while circling the earth once. Thus, a satellite can orbit the Earth many times during a long-term period and thereby each target may have multiple VTWs during multiple orbits. Scheduling multiple orbits directly is needed in practice, but a very complex problem due to the huge solution space. A simplification can be made by using a two-stage model: firstly, the targets are allocated to different orbits; secondly, the single-orbit scheduling is conducted for each orbit. Some previous work He et al. (2018); Xiaolu et al. (2014) proposed to use this two-stage model in order to address multiple-orbit or multiple satellite scheduling, but none of them can solve the problem exactly or even provide the optimality gap. In order to obtain optimal solutions (or provide an optimality gap), a branch and price framework coupled with a column generation technique could be applied, using a similar idea of the two-stage model. The implementation of this exact framework requires an efficient and exact algorithm for solving its pricing subproblem, namely the single-orbit scheduling. Thus, an exact and efficient method for the single-orbit scheduling is crucial for multiple-orbit scheduling. Additionally, the algorithm for the single-orbit scheduling can also be applied in the scheduling with meteorological uncertainty where multiple observations for each target are allowed Wang et al. (2019). Given these crucial methodological and practical values, this work focuses on developing an efficient and exact algorithm for the single-orbit scheduling.

The main contributions of this paper are threefold: (1) we present the first exact algorithm for the single-orbit AEOS scheduling with time-dependent profits based on a Dynamic Programming (DP) approach; (2) several algorithmic improvements for the DP are proposed to address the time-dependent profits and reduce the computational time; (3) extensive experimental results prove the effectiveness of the proposed algorithm, and offer benchmark instances and solutions for future researchers.

The remainder of this paper is structured as follows. In Section 2, we provide an extensive literature review. Section 3 presents the problem description and mathematical formulation. In Section 4, a novel labeling algorithm based on Dynamic Programming (DP) and Decremental State Space Relaxation (DSSR) is proposed to tackle this problem. Section 5 gives the experimental results, and concluding remarks are provided in Section 6.

2. Literature Review

Due to the computational difficulty of the AEOS scheduling, most of the previous works focuses on developing heuristics for this problem, but very few of them compare the performance of their algorithms with each other. Differences in physical design and ability parameters lead to large differences between the AEOS instances in different countries regarding capability, constraints and management. Furthermore, plenty of constraints and features in the AEOS scheduling make the different models difficult to compare. For instance, some research may ignore the time dependency of the transition time by using a fixed length of time Bianchessi et al. (2007); Wang et al. (2011); Wei-Cheng Lin et al. (2003), and some work uses a so-called semi-agile satellite whose look angles remain fixed during image acquisition Gabrel et al. (1997).

A variety of meta-heuristics and heuristics has been proposed for various AEOS scheduling problems. Lemaître et al. Lemaître et al. (2002) design four heuristic algorithms including a greedy algorithm, a local search algorithm, a dynamic programming algorithm and a constraint programming approach. Note that the latter two algorithms only solve a simplified version of the problem with fixed transition times. Some meta-heuristic methods are proposed, including tabu search Bianchessi et al. (2007); Cordeau and Laporte (2005); Habet et al. (2010); Lin et al. (2003), simulated annealing Dilkina and Havens (2005); Li et al. (2007), genetic algorithms Li et al. (2007); Wolfe and Sorensen (2000), a priority-based constructive algorithm Wang et al. (2011); Wolfe and Sorensen (2000); Xu et al. (2016) and an adaptive large neighborhood search algorithm He et al. (2018); Liu et al. (2017).

Until now, very few exact algorithms have been proposed in the literature for these kinds of problems. Bianchessi et al. Bianchessi et al. (2007) present a column generation algorithm to solve a linear relaxation of the problem and give the upper bound. However, the authors do not mention how to model the time-dependent transition time in the pricing problem. Wang et al. Wang et al. (2011) propose a mixed integer programming (MIP) model, where each VTW is split into only three fixed observations with each a specific pitch angle. As a result of this oversimplified approximation, the transition times can be pre-computed and thus the solution space is significantly reduced. This model is then solved by CPLEX, but only for very small instances. Chen et al. Chen et al. (2019) build a MIP model for multiple satellite scheduling. In their model, the transition times are displaced by their upper bounds, i.e., the maximum possible transition times. Thus, the time-dependency of transition times is ignored. In this work, the time-dependent transition times are taken into account in our proposed exact algorithm.

Moreover, very limited attention has been paid to the AEOS scheduling with time-dependent profits. The time-dependent profits feature arises from the fact that the profit of an observation, i.e., the image quality, highly depends on the start time. Wolfe and Sorensen Wolfe and Sorensen (2000) associate each

VTW with a particular quality function, where the duration of an observation is not fixed and can influence its profit. Liu et al. Liu et al. (2017) model the image quality on a ten-level scale over its VTW. The profit of each observation is fixed but can only be awarded when it satisfies the minimum quality requirement. This minimum quality requirement, however, is irrelevant to the scheduling, since the VTWs can be reduced beforehand to the part that guarantees enough quality. Peng et al. Peng et al. (2019) model the time-dependent profits in a form where for each observation, the maximal profit is collected in the middle of its VTW, and half of the maximal profit is collected at the edge. They propose an iterative local search heuristic coupled with a bidirectional dynamic programming approach to address the problem. However, since no exact algorithms have been presented for this problem before, no guarantee for their solution quality can be provided. In the current paper we develop an exact optimization technique to find the optimal solutions for the scheduling problem with time-dependent profits. To evaluate the performance, we utilize the same time-dependent profit model and benchmark instances proposed in Peng et al. (2019) (the instances are available at https://www.mech.kuleuven.be/en/cib/op/opmainpage#section-32). It should be noted that we focus on single-orbit scheduling, while the work in Peng et al. (2019) addresses multiple orbits.

In the Vehicle Routing Problem with Time-Dependent Rewards Yi (2003) and the Orienteering Problem with Time-Dependent Rewards Ekic et al. (2009); Erkut and Zhang (1996), the profit of each vertex monotonously decreases over time, which arises from some real-life applications. For example, the blood transportation problem searches the best route to visit blood collection points while keeping the collected blood as fresh as possible. In this case, the vertex can be naturally scheduled as early as possible. This monotonous model also corresponds to a special case of "soft time windows" where a penalty for a late visit is imposed, and visits during a certain time window have no impact on the collected profit. However, our work considers a non-monotonic profit function according to the practical need for satellite images. To the best of our knowledge, no exact techniques have been proposed to address the time-dependent profits with non-monotonic profit function.

When the time-dependent profits are not considered, the problem considered is very similar to that of the Orienteering Problem with Time Windows (OPTW) Duque et al. (2015); Righini and Salani (2009), where an observation of a satellite can be regarded as a visit to a vertex and the transition time corresponds to the travel time between each pair of vertices. The objective of the OPTW is to maximize the collected profits associated with the visited vertices, while satisfying the time window constraints and the elementary path constraint (i.e., each vertex can be visited at most once). It can also be formulated as a special case of an Elementary Shortest Path Problem with Resource Constraints (ESPPRC) where the travel time is regarded as a resource consumption. The ESPPRC is a general and fundamental NP-hard network optimization problem, often encountered as a subproblem of more complicated routing problems such as the Vehicle Routing Problem (VRP) and the Team Orienteering Problem (TOP). Several exact techniques have been proposed to solve these problems to optimality. Given the similarities between the OPTW and the ESPPRC, we exploit the same exact methodology as in Righini and Salani (2009) to solve the AEOS scheduling problem without time-dependent profits. Several algorithmic improvements are presented to address the time-dependent profits, and tested on the AEOS instances developed in Peng et al. (2019). Based on the similarities with the OPTW and given the availability of experimental results for exact techniques on the OPTW, these instances and results will also be used to evaluate the performance of our exact algorithm, when it is applied to instances without time-dependent profits.

3. Problem description

As the inputs of the scheduling, a set of possible targets T with each a given geographic position and a maximum profit, based on requests by different users. A visibility analysis based on the geographic position and the satellite's track is pre-processed to generate a set of VTWs for the possible targets. During each orbit, each visible target i corresponds to a VTW, denoted by $[st_i, et_i]$ where st_i and et_i are the window start time and the window end time. The visibility analysis provides for each VTW the required look angles (roll, pitch and yaw angles) for observation per second. The roll angle describes the rotation of the camera (or the satellite) towards the associated target around the satellite's centerline while the pitch angle controls the nose of the satellite to move up or down, as can be seen in Figure 1. The yaw angle determines the circular (clockwise or counter clockwise) movement around the axis vertical to the Earth surface. The duration of observing target i is also given by users, denoted by d_i. Notations of this study are summarized in Table 1.

The objective of this problem is to maximize the total profit collected by observing a subset of the possible targets. The higher the image quality, the higher profit the observation has. In practice, 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. In general, the image quality is negatively correlated to the absolute value of the pitch angle. Figure 2(a) illustrates that the pitch angle nonlinearly decreases over time during a VTW. Thus, for a complete VTW, the maximum profit of the corresponding target is collected at the midpoint and the least profit is collected at the edge. We use the same profit function as proposed in Peng et al. (2019). We define a discrete decision variable $t_i (t_i \in \{st_i, st_i + 1, ..., et_i\})$ to represent the start time of target i. Then the exact profit of a target i scheduled at moment t_i is calculated as follows:

$$p_i(t_i) = P_i \cdot (1 - \frac{|\pi(t_i)|}{90}), \tag{1}$$

where $\pi(t_i)$ is the pitch angle at moment t_i, and P_i is the maximal profit of target i that can be collected. Here we assume the maximal pitch angle of the satellite used is 45°. Figure 2(b) shows the ratio of the exact profits over the maximum profit for different moments. According to this, observations with the worst image quality at the edge of the VTW can still have half of the maximal profit. Note that this accumulated profit function is a non-linear function and cannot be explicitly expressed since the input data of the pitch angle is given in a discrete form (per second) and it also changes nonlinearly, as shown in Figure 2(a).

This problem mainly considers two specific constraints: first, each observation of a target should be scheduled during its VTW (called visible time window constraint); second, for each pair of consecutive observations, a transition time is required to maneuver the look angle of the camera from the previous observation to the next observation. However, this transition time cannot be easily pre-computed, since its duration depends on the total change of the look angles between these two consecutive observations. The calculation equation of the transition time is given as follows Peng et al. (2019):

$$trans(\Delta g) = \begin{cases} 11.66, & \Delta g \le 10 \\ 5 + \Delta g/v_1, & 10 < \Delta g \le 30 \\ 10 + \Delta g/v_2, & 30 < \Delta g \le 60 \\ 16 + \Delta g/v_3, & 60 < \Delta g \le 90 \\ 22 + \Delta g/v_4, & \Delta g > 90 \end{cases}, \tag{2}$$

where v_1, v_2, v_3, v_4 are four different angular transition velocities, which are given as the parameters of the satellite. For the satellite we consider, the angular velocity values are $v_1 = 1.5°/s, v_2 = 2°/s, v_3 = 2.5°/s, v_4 = 3°/s$. Δg represents the total change of the

Table 1
Notations

T	set of possible targets
i, j	target index, $i, j \in T \cup \{s, e\}$, in which s, e are virtual source target and sink target
$[st_i, et_i]$	visible time window for target i
P_i	maximum profit of target i
t_i	start time of the observation for target i, decision variable
$p_i(t_i)$	exact profit collected at moment t_i for target i
d_i	duration of observing target i
γ, π, ψ	roll, pitch and yaw angles
Δg	total change of the look angles for a transition of the camera
$trans(\Delta g)$	transition time with respect to the total change of the look angles
$mintrans_{ij}(t_i)$	minimal transition time between the observations of target i and j while observing target i at moment t_i

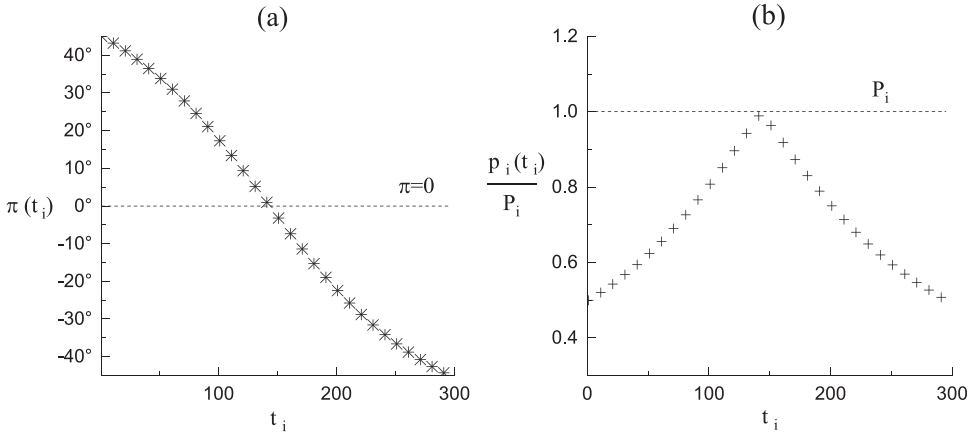


Fig. 2. The change of pitch angle (a) and the profit function (b) for a VTW.

look angles between two consecutive observations and calculated by $\Delta g = |\Delta\gamma| + |\Delta\pi| + |\Delta\psi|$, where $\Delta\gamma$, $\Delta\pi$ and $\Delta\psi$ are the change of the roll angle, the pitch angle and the yaw angle. The roll angle only depends on the geographical locations of targets relative to the satellite track and thereby the change of the roll angle between each pair of targets is fixed and can be pre-computed. The agile satellite used in this study is a semi-agile satellite which has no mobility on the yaw axes. Thus, the yaw angle is fixed to zero. As discussed above, the pitch angle nonlinearly decreases during a VTW (see Figure 2(a)), which indicates that the transition time between two consecutive observations depends on their start times. Peng et al. Peng et al. (2019) illustrate this time-dependency in detail. They present the "minimal transition time" (denoted by $mintrans_{ij}(t_i)$) to replace the actual transition time, only depending on the start time of the previous observation. The minimal transition time for each pair of VTWs at each possible start time can be efficiently pre-computed based on a dichotomy algorithm Peng et al. (2019). Another issue is that either the transition time or the minimal transition time follow the First-In-First-Out (FIFO) principle, regardless their time-dependency. It means that the later the previous observation takes place, the later the next observation can start. Furthermore, the triangle inequality is satisfied in this study, indicating that executing an additional observation between two observations consumes more transition time in total.

3.1. Assumptions

Since the AEOS scheduling problem has lots of constraints that could be considered, two assumptions are made to ignore some non-significant issues:

1) The satellite has sufficient on-board power and memory for each orbit.
2) The targets considered are all spot targets which can be observed in one pass. This assumes that larger Polygon targets are transferred into several independent spot targets with each a duration.

3.2. Integer Linear Programming Model

Considering the time-dependent profits, two decisions should be made in the model: first, the selection and sequence of the possible targets; second, the start times of all the scheduled targets. Since the profit function of each target is given in a discrete form (per second), we define a binary variable y_{it} $(t \in \{st_i, st_i + 1, ..., et_i\})$ to indicate at which moment the observation of target i starts. When target i is scheduled at moment t, then $y_{it} = 1$; otherwise, $y_{it} = 0$. Thus, the decision variable t_i can be written as $t_i = \sum_{t=st_i}^{t=et_i} t \cdot y_{it}$ with $\sum_{t=st_i}^{t=et_i} y_{it} = 1$. The profit function of target i can be expressed as $\sum_{t=st_i}^{t=et_i} p_{it} \cdot y_{it}$, where parameter p_{it} equals to $p_i(t)$ $(t \in \{st_i, st_i + 1, ..., et_i\})$. In addition, we define another decision variable x_{ij} which takes the value one when target j is observed immediately after target i and zero otherwise. The integer linear formulation of the single-orbit AEOS scheduling can be expressed as follows:

$$\text{Maximize} \sum_{i \in V} \sum_{t=st_i}^{t=et_i} p_{it} \cdot y_{it} \tag{3}$$

$$\sum_{\substack{j \in V \cup e \\ j \neq i}} x_{ij} = \sum_{\substack{j \in V \cup s \\ j \neq i}} x_{ji} = \sum_{t=st_i}^{t=et_i} y_{it}, \quad \forall i \in V \tag{4}$$

$$\sum_{j \in V} x_{sj} = 1 \tag{5}$$

$$\sum_{j \in V} x_{je} = 1 \tag{6}$$

$$\sum_{t=st_i}^{t=et_i} t \cdot y_{it} + d_i + mintrans_{ij}(\sum_{t=st_i}^{t=et_i} t \cdot y_{it}) - \sum_{t=st_j}^{t=et_j} t \cdot y_{jt} \leq M(1 - x_{ij}), \ \forall i, j \in V \quad (7)$$

$$\sum_{t=st_i}^{t=et_i} y_{it} \leq 1, \ \forall i \in V \quad (8)$$

$$x_{ij} \in \{0, 1\}, y_{it} \in \{0, 1\}, \ \forall i, j \in V \cup \{s, e\}. \quad (9)$$

The objective function (3) aims to maximize the total profit of the scheduled targets. Constraints (4) are the flow balance constraints and the elementary path constraints, and build the connection between variable x_{ij} and y_{it}, meaning that once a target is selected for observation, its start time should be determined. Constraints (5) and (6) express that the solution starts from the virtual source target s and ends at the virtual sink target e. Constraints (7) and (8) impose the transition time constraints and the visible time window constraints. Constraints (8) stipulate that at most one start time should be determined for each target. The domains of decision variables are defined in constraints (9).

4. Adaptive-directional Dynamic Programming with Decremental State Space Relaxation

When the time-dependent profits are not considered, our problem can be formulated as the OPTW or the ESPPRC which have been identified to be NP-hard in the strong sense Dror (1994). The time-dependent profits create an additional difficulty since the start times of observations can also influence the objective value. Therefore, our AEOS scheduling problem with time-dependent profits is also strongly NP-hard.

Due to the similarity between our satellite scheduling problem and the ESPPRC or the OPTW, some existing exact solution techniques for the ESPPRC and the OPTW can also be applied to our problem. The most commonly used exact method is Dynamic Programming (DP), based on the work by Desrochers et al. Desrochers et al. (1990). The DP, using a labeling algorithm, builds new paths (encoded by "labels") starting from the virtual source vertex, by extending paths one-by-one into all feasible directions. A dominance rule is applied for each pair of labels associated with the same vertex to fathom unpromising labels that cannot lead to the optimal solution. The elementary path constraint is imposed by adding a dummy vertex resource to each vertex which can be consumed by the visitation of labels. However, the introduction of dummy vertex resources increases the dimension of the state-space, which results in the low efficiency of DP.

The DP technique for the OPTW is improved by the work of Righini and Salani Righini and Salani (2009). They propose a Bidirectional Dynamic Programming with Decremental State Space Relaxation (BDP-DSSR) to tackle the OPTW efficiently. The labels are extended simultaneously in both the forward and backward direction, and stopped at the "half-way" of the path. This corresponds to half of the latest allowed arrival time to the depot. Afterwards, the forward and backward sub-paths are matched to produce complete paths, guaranteeing their feasibility. To speed up the search, a so-called Decremental State Space Relaxation (DSSR) technique is employed to repeatedly solve a relaxation of the OPTW through DP, ignoring the elementary path constraint. In particular, a "critical vertex set" is introduced and augmented at each iteration with the vertices visited more than once in the optimal solution of the relaxed problem. Multiple visits are forbidden on these critical vertices at the current iteration. The procedure terminates when the optimal path of the relaxed problem turns out to be elementary.

In this paper, we present an Adaptive-directional Dynamic Programming with Decremental State Space Relaxation (ADP-DSSR) algorithm to tackle our satellite scheduling problem, including the time-dependent profits. The ADP-DSSR algorithm is derived from the BDP-DSSR, but several improvements are made in order to address the time-dependent profits feature and accelerate the search. The four improvements are partial dominance, merging labels, detour pruning and the adaptive-directional extension. The adaptive-directional extension means that both the forward and backward extensions of labels are considered, but the direction processed is adaptively determined according to the intermediate results.

Hereafter we give the details on the extension rule, the dominance test and the matching procedure. Then, the DSSR technique is illustrated. Afterward, the four improvements are discussed in detail.

4.1. Dynamic Programming

The Bidirectional DP algorithm consists of three parts: the definition of states, labels and their extension rules, the dominance rule and the matching procedure.

4.1.1. States, labels and extension

The single-orbit scheduling problem without time-dependent profits can be regarded as solving an OPTW in an undirected graph $G = (V, E)$ with a set of vertices V and a set of edges E, where each target with its VTW corresponds to a vertex with its time window and the minimal transition time corresponds to the travel time. Two virtual targets are introduced as source vertex s and sink vertex e. Due to the time dependency of the minimal transition time, the travel time for each pair of vertices depends on the start time of the previous vertex. This time-dependent travel time has been defined in a variant of the OPTW, namely the Time-Dependent Orienteering Problem with Time Windows (TD-OPTW). However, since the FIFO principle and the triangle inequality are satisfied for this time-dependency, solving this TD-OPTW is equivalent to solving an OPTW while using the DP method except for one difference: when extending labels (paths) between the same pair of vertices with different leaving times, the travel times are different. To avoid the confusion, we adopt the definitions and concepts of the DP methology for the OPTW in the remainder, which can be referred in Righini and Salani (2009).

For each vertex $i \in V$, we associate a group of forward labels and backward labels. A forward label L represents a path starting from source vertex s and ending at the current associated vertex. A backward label represents a path starting from the sink vertex e and ending at its associated vertex. A forward label can be expressed as a tuple $L = (R_L, i, es_L, P_L(t), path(L))$ where R_L represents a binary vector to indicate which vertices cannot be visited anymore, i is the associated vertex, es_L is the earliest possible start time of L on vertex i, $P_L(t)$ represents its accumulated profit function, and $path(L)$ is the path corresponding to label L, composed by a sequence of visited vertices. The backward label has the same components as the forward one except the earliest start time es_L. It uses ls_L to indicate its latest possible start time on its associated vertex. The DP algorithm extends the forward labels from source vertex s to its successors and the backward labels from sink vertex e to its predecessors. In what follows, we explain the components in the tuple and their updating rules for a forward extension from label L associated with vertex i to label L' associated with vertex j.

Considering the time-dependent profits, for a given path, choosing different start times of the visited vertices may correspond to different collected profits. Thus, we define a group of states

for each label. Each state represents the path of the associated label, arriving at a possible start time and collecting a maximal accumulated profit along the path up to that start time. A state ω associated with L can be indicated by a tuple $\omega = (R_\omega, i, t(\omega), p_\omega, path(\omega))$ where $t(\omega)$ represents the start time of the state, R_ω is same to R_L and p_ω is equal to $P_L(t(\omega))$ indicating the accumulated profit of the state, and $path(\omega)$ is the corresponding path. In our algorithm, the states are recorded by a discrete "accumulated profit function" $P_L(t)$, where $t \in \{es_L, es_L + 1, ..., et_i\}$ for a forward label and $t \in \{st_i, st_i + 1, ..., ls_L\}$ for a backward label. Each state corresponds to a certain data point in the accumulated profit function. The accumulated profit describes the trade-off between the collected profits and the consumed time, using a series of data points (also called "trade-off" curve). A similar application of this "trade-off" curve in the DP for the Time-Dependent Vehicle Routing Problem with Time Windows (TD-VRPTW) is the ready time function which describes the time consumption of the associated path with respect to the departure time at the depot Dabia et al. (2013).

For a forward extension from label L on vertex i to label L' to vertex j, the accumulated profit function is updated according to the formula

$$P_{L'}(t_j) = max_{t_i}\{P_L(t_i) |\ EarliestStartTime_{ij}(t_i) \le t_j,\ es_L \le t_i \le et_i\} + p_j(t_j), \tag{10}$$

where $EarliestStartTime_{ij}(t_i)$ represents the earliest start time on vertex j when leaving vertex i at moment t_i. The calculation of the earliest start times between each pair of vertices can be preprocessed, referring to the work by Peng et al. Peng et al. (2019). This extension is feasible only if $EarliestStartTime_{ij}(es_L) \le et_j$. Then the earliest start time $es_{L'}$ of label L' is updated to $EarliestStartTime_{ij}(es_L)$. An improved update method of the accumulated profit function is presented by Peng et al. Peng et al. (2019). According to this update equation, the accumulated profit function of a forward label may not be a non-decreasing function. However, the states with lower profits and more consumed time are dominated by the states of the same label with the higher (or same) profits and less consumed time. Hence, these dominated states are replaced by the dominating ones, resulting in a non-decreasing accumulated profit function. Once a state is chosen for a given label, the start times of other visited vertices can be easily determined by applying a backtracking algorithm.

To avoid a cyclic visit, a dummy vertex resource is associated with each vertex i in the graph: each vertex only has one unit of the resource and it is consumed when the vertex is visited. The consumption of the dummy vertex resources is indicated by a vector S (visitation resources vector) with $|V|$ entries initialized at 0. Note that S does not keep any information about the order of the visited vertices. With this definition of vector S, a label cannot dominate another label if the label has visited more vertices. Feillet et al. Feillet et al. (2004) present an "unreachable vertex vector" to replace the visitation vector. Vertices that either has been visited or cannot be visited due to resource limitations are identified to be unreachable. In our problem, a vertex j is unreachable from a label $L = (R_L, i, es_L, P_L(t))$ when $path(L)$ includes vertex j or when its time window would be violated by any visit starting from L, that is $es_L + d_i + mintrans_{ij}(es_L) > et_j$. If label L cannot reach vertex j directly, no other paths extended from $path(L)$ can reach vertex j since the triangle inequality holds for the time-dependent transition time. We denote the unreachable vertex set by a binary vector R with $|V|$ entries. The update rule of vector R for a forward extension from L to L' is

$$R^k_{L'} = \begin{cases} R^k_L + 1, & k = j \\ R^k_L, & k \ne j \end{cases} \tag{11}$$

Afterwards, for each unvisited vertex, we check whether the current label can visit that vertex and update the corresponding element in vector $R_{L'}$.

The extension rules for backward labels are symmetrical to what has been presented before. We define the time window $[st^{bw}_i, et^{bw}_i]$ as the backward time window of vertex i: it is obtained by adding the duration of service d_i to the forward time window $[st_i, et_i]$. The backward extension is feasible only if $LatestStartTime_{ij}(ls_L) \ge st^{bw}_j$ where ls_L is the latest start time of L and $LatestStartTime_{ij}(ls_L)$ calculates the latest start time on vertex j when leaving vertex i at moment ls_L.

To illustrate how the extension rules work, a small example of the forward extension with 3 vertices and 2 labels is presented in Figure 3. The number in the circle is the vertex index and the number on the line is its travel time. The orange directed lines represent label L_1 starting from vertex 1 and reaching vertex 3 through vertex 2. The blue directed line represents label L_2 reaching vertex 3 directly from vertex 1. The time windows and the exact profits during the VTWs of these three vertices are shown in the table (a). The accumulated profit functions of the two labels are reported in table (b). Each cell in table (b) represents a state of its corresponding label. When generating label L_2 from vertex 1 to vertex 3, the accumulated profit of label L_2 at $t_3 = 5$ is calculated by $P_{L_2}(5) = max\{p_1(1), p_1(2)\} + p_3(5) = 2 + 3 = 5$, where L_2 arrives at $t_3 = 5$ only when it visits vertex 1 with $t_1 \le 2$. Likewise, the accumulated profit of label L_2 at $t_3 = 7$ can be calculated: $P_{L_2}(7) = max\{p_1(1), p_1(2), p_1(3), p_1(4)\} + p_3(7) = 3 + 1 = 4$. Since it can be dominated by its previous states, its accumulated profit is replaced by 5. When we compare label L_1 and L_2, the states of these two labels with the same value of start time are compared. Dominance rules are applied for this comparison and will be discussed in the next subsection. If the rules are satisfied, the dominated states will be discarded and the non-dominated states are stored. In this example, the numbers in table (b) displayed in bold correspond to the non-dominated states.

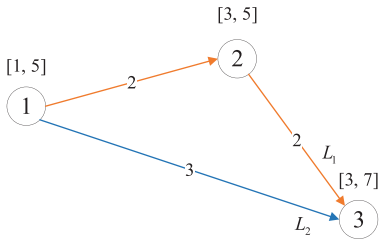
4.1.2. Dominance rules

Dominance tests are performed to reduce the number of states and labels generated by extending paths to other vertices. It is executed between a newly generated label and each stored label associated with the same vertex. The labels that cannot lead to the optimal path will be dominated and removed during the search. In our case, for a given label, different start times (states) correspond to different profits. Thus, the dominance test will delete parts of the accumulated profit function (a group of states) according to the dominance rules. If all the states of a label are deleted, the label will be removed from the label list to be extended.

We assume that label L_1 and label L_2 associated with the same vertex i are compared for dominance testing. Let $\omega_{L_1} = (R_{L_1}, i, P_{L_1}(t_1), t_1)$ be a state of label L_1 at moment t_1 and $\omega_{L_2} = (R_{L_2}, i, P_{L_2}(t_2), t_2)$ be a state of label L_2 at moment t_2. The former state dominates the latter state only if

$$\begin{cases} R_{L_1} \le R_{L_2}, \\ P_{L_1}(t_1) \ge P_{L_2}(t_2), \\ t_1 \le t_2 \end{cases} \tag{12}$$

and at least one of the inequalities is strict. Due to the first rule (called "unreachable vector condition"), this dominance test needs to enumerate and compare each element of the unreachable vector R, which consumes a large computational time. An efficient improvement is adding a variable $q_L = \sum_{k=1}^{|V|} R^k_L$ to record the number of unreachable vertices. If $q_{L_1} \ge q_{L_2}$, label L_1 is impossible to dominate label L_2. Once the dominance test succeeds, the dominated states are discarded and the corresponding data points in the accumulated profit function are removed. If there exists some states surviving the test, the newly generated label will be stored into the



(a)

Vertex	$[st_i, et_i]$	$p_i(t_i)$				
1	[1, 5]	1	2	3	2	1
2	[3, 5]	2	3	2	--	--
3	[3, 7]	1	2	3	2	1

(b)

Label	$[es_L, et_3]$	$P_L(t_3)$			
		t=4	t=5	t=6	t=7
L_1 (1->2->3)	[5, 7]	--	**6**	**6**	**6**
L_2 (1->3)	[4, 7]	**3**	5	5	5

Fig. 3. An illustration for states, labels and their forward extensions.

unextended label list. Meanwhile, the unextended labels associated with the corresponding vertex will be checked by the new label to delete their dominated states in the similar way.

4.1.3. Matching procedure

In this procedure, forward and backward labels are matched together to yield a complete path from vertex 0 to vertex $|V|+1$. Let $L_f = (R_{L_f}, i, es_{L_f}, P_{L_f}(t))$ be a forward label and $L_b = (R_{L_b}, i, ls_{L_b}, P_{L_b}(t))$ be a backward label. Both these two labels are associated with vertex i. The matching operation can be accepted only if it satisfies the following feasibility conditions:

$$\begin{cases} R^k_{L_f} + R^k_{L_b} \le 1, \ \forall\ k \in V \\ es_{L_f} + d_i \le ls_{L_b} \end{cases} \tag{13}$$

If feasible, the maximal collected profit of the newly matched path can be calculated by:

$$P_{max} = \max\{P_{L_f}(t) + P_{L_b}(t) - p_i(t),\ es_{L_f} \le t \le ls_{L_b}\} \tag{14}$$

After matching all the forward and backward labels, the optimal solution is the feasible and complete path with the highest collected profit. This process requires to enumerate all the combinations of the forward and backward labels. Note that if we only performs the forward (backward) extension, only the sink (source) vertex needs to be checked for the matching procedure.

4.2. Decremental State Space Relaxation

The Decremental State Space Relaxation (DSSR) is proposed by Righini and Salani Righini and Salani (2008, 2009), with the idea of iteratively reducing the relaxation of the state space of the problem. Boland et al. Boland et al. (2006) present a similar idea, but refer to it as the State Space Augmenting algorithm. The basic idea of this technique is to solve a relaxation of the primal problem with DP where the elementary path constraints are ignored for all the vertices. Then this relaxation is iteratively tightened by imposing the elementary path constraints on a subset of vertices step by step. To be specific, we define Θ as the critical vertex set in which vertices can be visited at most once and are considered in the unreachable vector condition in the DP. Θ is initialized to be empty, allowing multiple visits to all the vertices. If the optimal solution of this relaxation is elementary, then it is also the optimal solution for the original problem; otherwise, some vertices that appear more than once in the solution should be inserted into Θ to reduce the relaxation. This procedure is repeated until the obtained solution is elementary.

Righini and Salani Righini and Salani (2008) present three different strategies of inserting vertices into Θ and find that there is no clear dominance between these strategies. Our preliminary experiments found that the MO-ALL strategy, which inserts all the duplicate vertices appearing in the optimal solution, is most effective, since it can greatly reduce the number of iterations with only adding very few "unnecessary" vertices into Θ. Righini and Salani also present some initialization heuristics for the critical vertex set Θ in order to reduce the number of iterations and the computational time. However, these heuristics do not perform very well in our instances, probably because of some special characteristics of our instances such as the time-dependent transition time and the highly overlapping time windows. Therefore, we do not use any initialization heuristic in our algorithm.

4.3. Algorithmic improvements

In this section, we introduce four algorithmic improvements to accelerate the solution procedure.

4.3.1. Partial Dominance rule

A full and strict dominance rule is introduced in Section 4.2, where the dominance test can be passed only if all the conditions are satisfied. For instance, when checking if label L_2 is dominated by label L_1 associated with vertex i, the condition $R_{L_1} \le R_{L_2}$ should be reached; otherwise, the states of L_1 can not be dominated even though the other two conditions are reached. The unreachable vector is associated with its label and does not depend on the start times (or the states) of that label. This full dominance rule is commonly used in some approaches proposed for variants of routing problems such as the TD-VRPTW Dabia et al. (2013) and the Vehicle Routing Problem with Soft Time Windows (TD-VRPSTW) Liberatore et al. (2011). However, even if the unreachable vector condition is not satisfied, some states can still possibly be compared.

Suppose that there are two labels L_1 and L_2 associated with vertex i. L_1 has visited vertex k but L_2 has not, and thus their unreachable vectors satisfy $R^k_{L_1} > R^k_{L_2}$. We further assume that for any $m \in V\backslash\{k\}$, $R^m_{L_1} \le R^m_{L_2}$. When checking if L_2 is dominated by L_1, the full dominance test fails because $R_{L_1} \le R_{L_2}$ is not satisfied. However, label L_1 and L_2 can still be compared for the states with their start times satisfying $t \ge UT^k_i$, where UT^k_i corresponds to the earliest time when vertex i cannot visit vertex k due to the time window constraints. With this dominance rule, the points corresponding to the dominated states in the accumulated profit function will be removed and will not be considered in further steps. We refer to this dominance rule as the Partial Dominance Rule.

To efficiently implement this idea, we define a vector UT_i with N entries for each vertex i. For any moment $t \ge UT^k_i$, vertex i cannot reach vertex k due to the time windows constraints. These vectors can be pre-calculated before running the algorithm in order to reduce the computational time. In the dominance test of L_2 with L_1, we only check the states whose start times are later than moment $\widetilde{UT}(L_1, L_2) = \max\{UT^k_i |\ R^k_{L_1} > R^k_{L_2}, \forall k \in N\}$. While $R_{L_1} \le R_{L_2}$ is satisfied, $\widetilde{UT}(L_1, L_2)$ equals to es_{L_2}, meaning that all the states in L_2 can be checked by L_1. By doing so, some states or labels which

cannot lead to an optimal path will be pruned and the total number of generated labels in DP can be significantly reduced.

4.3.2. Merging Labels

In general, the effectiveness of the DP algorithm heavily depends on the number of states generated. In addition to fathoming labels with the dominance test, another approach to reduce this number in our algorithm is to merge labels whose unreachable vectors have the same values. Specifically, we assume that state ω_1 belongs to label L_1 and state ω_2 belongs to label L_2, and $R_{L_1} = R_{L_2}$. If ω_1 and ω_2 have the same start time, one of them will be removed due to the dominance test; otherwise, these two states can be stored in the same label even though they correspond to different paths. By grouping all the states from L_1 and L_2 in the same label (either L_1 or L_2), another label will no longer need to be extended. This merging process requires to insert all the data points in the accumulated profit function from one label to the function in another label. In this way, the total number of the states and labels generated will be significantly reduced. For each combination of unreachable vector R with respect to a certain vertex, at most one label is recorded in memory. It should be noted that after using this procedure, each label may corresponds to multiple labels since the states of the combined label come from different initial labels.

When combining the merging operation with the partial dominance rule, we merge label L_2 into label L_1 only if $es_{L_2} \geq \widetilde{UT}(L_1, L_2)$. This is because the states of L_2 with $t < \widetilde{UT}(L_1, L_2)$ cannot be compared, limited by the unreachable vector condition in the dominance rule. The merging operation is processed for each unextended label with the current label after applying the partial dominance test.

4.3.3. Detour pruning strategy

In a recent work Duque et al. (2015), a pruning strategy is proposed to accelerate the DP in a so-called "pulse algorithm" (similar to branch-and-bound) for solving the OPTW. The main idea is that when directly extending a label L from vertex i to j satisfying $es_L \leq st_j$, if there exists a feasible detour $path(L) \cup \{k\} \cup \{j\}$ and its arrival time to vertex j is still less than st_j, then this direct extension to vertex j can be ignored and vertex k is called a detour vertex for arc (i, j). This is because at least one path to vertex j can dominate the label directly extended from label L, which is the path $path(L) \cup \{k\} \cup \{j\}$. We first apply this strategy to the labeling algorithm with the DSSR technique and illustrate its adaption on the forward extension in the reminder of this section. Note that in our problem, this pruning strategy is available only when the profits collected at different moments for each VTW are all positive.

To check the feasibility of detours, for each arc (i, j) and for each detour vertex k, we define the latest start time for a label L associated with vertex i to reach vertex j through detour vertex k, denoted by $LT^k_{(i,j)}$. This value can be calculated in the preprocessing phase in order to avoid duplicate calculations during the search. Despite the presence of the time-dependent transition time, the triangle inequality in our problem is still satisfied Peng et al. (2019). Hence, if label L starts early than $LT^k_{(i,j)}$, detour $path(L) \cup \{k\} \cup \{j\}$ is feasible. Algorithm 1 demonstrates how to calculate the value $LT^k_{(i,j)}$ for the forward extension.

In Algorithm 1, $LatestStartTime_{jk}(st_j)$ calculates the latest start time of vertex k if vertex j is visited at its window start time st_j (see the preprocessing procedure in Peng et al. (2019)). Since our algorithm considers two directions of extending labels, the detour pruning can also be used in the backward extension. In this case, a variable $ET^k_{(i,j)}$ is defined to record the earliest start time of vertex

Algorithm 1 Pre-calculation of detours in the forward extension

```
for all i ∈ V do
  for all j ∈ V do
    for all k ∈ V\{s, e} do
      LT^k_(i,j) ← ∞;
      if LatestStartTime_jk(st_j) ≥ st_k then
        t_temp = LatestStartTime_jk(st_j);
        if LatestStartTime_ki(t_temp) ≥ st_i then
          LT^k_(i,j) ← LatestStartTime_ki(t_temp);
        end if
      end if
    end for
  end for
end for
```

i to reach vertex j through the detour vertex k. The calculation of $ET^k_{(i,j)}$ is symmetrical to $LT^k_{(i,j)}$.

In the pulse algorithm from Duque et al. (2015), every time an unvisited vertex is added to the current path, other unvisited vertices will be checked for the detour pruning. This process requires to traverse all the unvisited vertices until a feasible detour vertex is found. It is worth mentioning that compared with the pulse algorithm, the labeling algorithm usually visits vertices (or arcs) much more times, which implies that the adaption of the detour pruning in our algorithm may consume large computational time. We improve this pruning strategy by ordering the detour vertices k for each arc (i, j) in the forward extension by non-increasing $LT^k_{(i,j)}$. This ordering operation can also be pre-processed. The algorithm only needs to check the feasibility of the first detour in the ordered list for each pruning. If the first detour vertex k_1 satisfies $es_L \leq LT^{k_1}_{(i,j)}$, then the states of label L with their start times smaller than $LT^{k_1}_{(i,j)}$ can be pruned. Otherwise, no states can be pruned by any other detour vertices, since vertex k_1 has the largest value of $LT^{k_1}_{(i,j)}$. Obviously, this process performs more efficiently than traversing all the detour vertices until a feasible one is found, as presented in Duque et al. (2015).

Another adaption of this pruning strategy is to combine it with the DSSR technique. Since the DSSR allows multiple visits for any vertices not belonging to Θ, any vertex $k \in V \backslash (\{i, j\} \cup \Theta)$ can be checked as a detour vertex for arc (i, j) no matter it has been visited or not. It is important to note that if detour vertex k is a critical vertex, vertex k cannot be checked even though it has not been visited, due to the time-dependent profits feature. The following proposition supports this conclusion.

Proposition 4.1. *When extending a label L from i to j with the time-dependent profits considered, for any detour vertex k, the detour pruning can be successfully applied to the extension only if vertex k is not a critical vertex (called "non-critical condition") and satisfies the feasibility condition, i.e.,* $es_L \leq LT^k_{(i,j)}$.

Proof. In Figure 4, given a critical vertex set Θ, let $path^*$ be the optimal path (solid lines) at the current DSSR iteration, consisting of a sequence of states (blue rectangle), each indicating a visited vertex at its corresponding moment. Suppose $path^*$ contains a sub-path $(\omega^0_i - \omega^0_j - \omega^0_k)$ where i, j and k are three visited vertices. First, we assume that vertex k is a critical vertex and satisfies $t(\omega^0_i) \leq LT^k_{(i,j)}$, where $t(\omega^0_i)$ represents the start time of state ω^0_i. If the extension of ω^0_i from vertex i to j is pruned by detour $path(\omega^0_i) \cup \{k\} \cup \{j\}$, then it is impossible to obtain sub-path $(\omega^0_i - \omega^0_j - \omega^0_k)$, as well as the optimal path $path^*$. Second, we assume that vertex k is not a critical vertex. If $t(\omega^0_i) \leq LT^k_{(i,j)}$ is

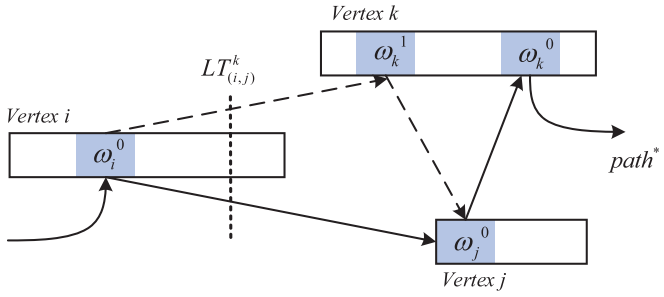


Fig. 4. An example of incorrect detour pruning.

satisfied, there must exists another sub-path $(\omega_i^0 - \omega_k^1 - \omega_j^0 - \omega_k^0)$ dominating sub-path $(\omega_i^0 - \omega_j^0 - \omega_k^0)$, where $t(\omega_k^1) < t(\omega_k^0)$. Then, a new path which has a higher profit than $path^*$ can be obtained by replacing sub-path $(\omega_i^0 - \omega_j^0 - \omega_k^0)$ by $(\omega_i^0 - \omega_k^1 - \omega_j^0 - \omega_k^0)$ in $path^*$. This is a contradiction to the hypothesis of the optimal path and thus the vertex k should not satisfy the feasibility condition. Thus, the detour pruning can work only if the non-critical and the feasibility conditions are satisfied simultaneously □

In Figure 4, if the non-critical and the feasibility conditions are satisfied, and sub-path $(\omega_i^0 - \omega_k^1 - \omega_j^0 - \omega_k^0)$ appears in the optimal path at the current iteration, vertex k will become a critical vertex at the next iteration. It is noteworthy that if the time-dependent profits are not taken into account, this non-critical condition can be omitted. This is because, in Figure 4, any paths extended from $(\omega_i^0 - \omega_j^0)$ can be dominated by the paths extended from sub-path $(\omega_i^0 - \omega_k^1 - \omega_j^0)$, since these two sub-paths have the same collected profit, and the visits starting from ω_j^0 to any other vertices always arrives earlier than those starting from ω_k^0, due to the triangle inequality considered in this work.

Algorithm 2 shows how we apply the detour pruning strategy

Algorithm 2 *DetourPruning*(L, j)

```
for all k ∈ V do
  if k ∈ Θ then
    continue;
  end if
  if es_L ≤ LT^k_(i,j) then
    return LT^k_(i,j);
  else
    break;
  end if
end for
return et_i;
```

with the DSSR technique when extending label L associated with vertex i and vertex j to be extended. We check the feasiblity of the first vertex not belonging to Θ in the ordered detour list. If it is feasible, the algorithm will return $LT_{(i,j)}$ and only the states that start later than $LT_{(i,j)}$ need to be extended to vertex j. Otherwise, the window end time of vertex i will be returned, meaning that all states should be considered in the extension. If all the states start earlier than $LT_{(i,j)}$, the extension from label L to vertex j will be discarded.

4.3.4. Adaptive-directional extension

Previous studies have shown that the Bidirectional DP algorithm with an effective bounding criterion is usually superior to its mono-directional counterpart. This is because in general, the

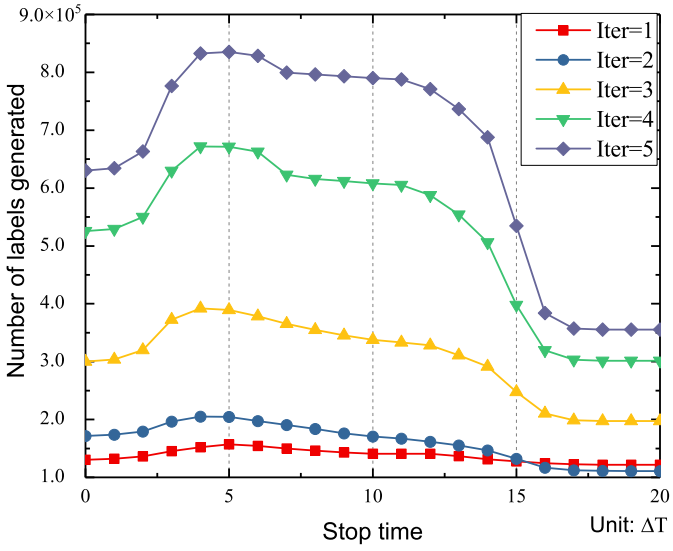


Fig. 5. An example instance showing the total number of labels generated versus the stop times at different iterations of DSSR.

number of labels generated grows exponentially with the number of arcs visited, which intuitively implies that exploiting two smaller sets of space states is faster than exploiting a large set of the whole space states. Tilk et al. Tilk et al. (2017) claim that the bidirectional search works particularly well if the sizes of the forward and backward labels are similar. In order to balance the forward and backward workload, they present a dynamic half-way point as the stop time of the forward and backward extensions. However, our experimental results show that the bidirectional labeling usually spends more time than the mono-directional ones for our problem, as can be seen in Figure 5. This figure shows how the total number of labels (sum of forward and backward ones) changes with the stop time varied from zero to $20\Delta T$ for a certain instance, where $20\Delta T$ equals to the maximum window end time among all the vertices. If the stop time is set to zero or $20\Delta T$, a mono-directional labeling is applied; otherwise, a bidirectional search is performed. The different colored curves account for the fluctuation at different iterations of the DSSR. Clearly, the bidirectional labeling creates more labels than the forward or backward one, possibly due to the unavoidable duplication of paths in the bidirectional search. The 'trade-off" curve, i.e., the accumulated profit function, which requires to keep a large number of non-dominated states for both directions of labels within their time windows, may account for the duplication. A similar conclusion can be found in Desaulniers et al. (2016). Another experiment (not shown here) demonstrates that the duplication also becomes worse when the length of the time windows or the degree of overlap increases.

Other interesting preliminary results of this example show that the forward labeling runs faster than the backward labeling, and the difference becomes much larger when the number of DSSR iterations increases. The reason can be twofold: firstly, the asymmetric instances data, e.g., time windows being unevenly spread, give rise to the unbalanced forward and backward extensions; secondly, the presence of critical vertices dramatically increases the number of non-dominated labels, which exacerbates the imbalance. The results reveal a regularity that if most of the critical vertices are distributed closely to the source vertex and greatly overlap with each other, the number of non-dominated labels increases rapidly when extending labels from the part of critical vertices to other vertices.

From our analysis, a mono-directional labeling algorithm is preferable for our problem. However, it is unclear which direction of the labeling extension is the most efficient. This can be strongly instance-dependent. One possible approach is to determine the di-

rection according to the distribution structure of the time windows and the current states of the critical vertex set for each iteration. Hereby we present an Adaptive-directional Dynamic Programming (ADP) algorithm to address this imbalance issue. Algorithm 3 explains how we determine the efficient direction, given a group of vertices and a critical vertex set Θ.

Algorithm 3 *GetExtensionDirection*(Θ)

```
if Θ = ∅ then
    return Bidirection;
end if
ETAmount ← 0, LTAmount ← 0;
for all k ∈ Θ do
    for all i ∈ V do
        ETAmount ← ETAmount + max{min{et_i, et_k} − st_i, 0};
        LTAmount ← LTAmount + max{et_i − max{st_i, st_k}, 0};
    end for
end for
if ETAmount ≤ LTAmount then
    return Backward;
else
    return Forward;
end if
```

In Algorithm 3, *ETAmount* counts, for all the time windows, the amount of time before the start time of each critical vertex at the current iteration. Likewise, *LTAmount* counts the amount of time which starts later than the end time of each critical vertex. Afterwards, these two statistics are compared in order to determine which direction will be used at that iteration. The setting of the algorithm is based on the idea that the labeling extension between the critical vertices may produce many non-dominated labels, and the more vertices that can be extended by these labels, the more the total number of labels at the end. At the first iteration when no critical vertices exist, a bidirectional extension is performed.

4.4. Adaptive-directional dynamic programming with decremental state space relaxation

Algorithm 4 outlines the framework of our ADP-DSSR algorithm.

Algorithm 4 Adaptive-directional DP with DSSR.

```
Θ ← ∅;
while true do
    Direction ← GetExtensionDirection(Θ);
    if Direction = Forward then
        S* ← ForwardDP();
    else if Direction = Backward then
        S* ← BackwardDP();
    else
        S_f ← ForwardDP();
        S_b ← BackwardDP();
        S* ← Matching(S_f, S_b);
    end if
    if S* is non-elementary then
        Θ ← Θ ∪ Θ_new;
    else
        The optimal solution S* is obtained;
        Return;
    end if
end while
```

At first, the critical vertex set Θ is initialized to be empty. Then, the algorithm enters a WHILE loop and its condition equals to true. At each iteration, a *GetExtensionDirection*(Θ) procedure is executed to determine which direction should be used to extend labels with respect to the current critical vertex set Θ. If Θ is empty, the bidirectional DP is performed and a matching procedure is required to connect the forward labels $\overline{S}_f$ and the backward labels $\overline{S}_b$; otherwise, a forward or a backward extension is processed. Afterwards, the solution S^* with the highest collected profit among all the generated paths is obtained. If S^* is non-elementary, the set of duplicate vertices Θ_{new} is added into Θ, and the algorithm goes to the next iteration. Otherwise, S^* is the optimal elementary solution and the algorithm terminates.

5. Computational results

In this section, the performance of the proposed algorithm will be discussed. We are the first to solve the AEOS scheduling problem to optimality, hence, no existing algorithm or known optimal solutions can be compared with. Therefore, we design three experiments to evaluate the performance: the first experiment evaluates the four algorithmic improvements by comparing different settings where these improvements are included or not; secondly, we compare the performance of our ADP-DSSR algorithm with the performance of a state-of-the-art heuristic Peng et al. (2019); lastly, our algorithm is tested on OPTW benchmark instances, and compared with two reference exact algorithms in the literature Duque et al. (2015); Righini and Salani (2009) when the time-dependent profits are not taken into account. We coded our algorithm in C#, using Visual Studio 2010, and performed our experiments on a laptop with a 2.5 GHz Intel Core i5-7300HQ and 8 GB RAM.

5.1. Evaluating the four algorithmic improvements

We test our ADP-DSSR algorithm on the Chinese instances proposed by Liu et al. Liu et al. (2017). Targets for observation are randomly generated with a uniform distribution in the Chinese area (3°N-53°N and 74°E-133°E). A visibility analysis is processed to generate the visible time windows and the look angles per second for each target. The time step is defined to evenly discretize the time windows in order to simplify the calculation. The maximum profit and the service time of each target are uniformly generated respectively from [1,10] and [15,30] in seconds. The scheduling horizon is 24 hours, meaning that the satellite can pass by a certain target multiple times during different orbits. Thus, for an instance with a given number of targets (called "multi-orbit instance"), the scheduling horizon is divided in multiple orbits, each called a "single-orbit instance". Our algorithm is designed to solve these single-orbit instances. Each single-orbit instance has a specific number of targets to be scheduled. The larger the number of targets in a single-orbit instance, the higher overlap between their VTWs. The time-dependent profits of each target are distributed in its visible time window following the model proposed by Peng et al. Peng et al. (2019) and explained in Section 3. More details on the test instances and the satellite parameters can be found in the paper of Liu et al. Liu et al. (2017). The instances are available at https://www.mech.kuleuven.be/en/cib/op#section-32.

To evaluate the performance of our ADP-DSSR algorithm, two comparative experiments are carried out. The first experiment compares the proposed algorithm with three variants to investigate the impact of the proposed algorithmic improvements. These variants are generated by respectively removing the partial dominance rule, the merging process of labels and the detour pruning strategy. The comparative results for the multi-orbit instances from 300 to 600 targets with the time step equals to five seconds are presented in Table 2. Each row presents the average results over

Table 2
Comparative results of the algorithms without the proposed improvements.

Instance Name	VTWs	ADP-DSSR	No Partial Dominance		No Merging Labels		No Detour Pruning	
		CPU time(s)	CPU time(s)	T_{dif}	CPU time(s)	T_{dif}	CPU time(s)	T_{dif}
300_A	65.38	1.98	2.89	-0.60	13.35	-4.97	2.56	-0.47
400_A	89.50	1.52	2.62	-0.88	11.95	-7.63	2.36	-0.41
500_A	115.00	2.33	3.39	-0.58	21.55	-8.42	3.58	-0.39
600_A	137.38	2.83	5.36	-0.72	30.35	-10.78	4.24	-0.37
Average				-0.70		-7.87		-0.41

eight single-orbit instances from one multi-orbit instance. The individual results, for each single-orbit instance are available here: https://www.mech.kuleuven.be/en/cib/op#section-32. The column VTWs represents the average number of targets visible during a single orbit. The CPU time (in seconds) of each algorithm is shown, followed by the difference ratio compared with the complete ADP-DSSR algorithm. This ratio is calculated by:

$$T_{dif} = \frac{time_{prop} - time_{var}}{time_{prop}}, \tag{15}$$

where $time_{prop}$ represents the CPU time of the proposed ADP-DSSR algorithm and $time_{var}$ is the CPU time of the corresponding variant of the algorithm. The negative value of this ratio means the proposed algorithm is faster, while the positive value means the variant algorithm is faster.

The results show that the complete version of the proposed algorithm is the fastest among these algorithms for all the instances, implying that these three improvements presented are all very effective. The most crucial improvement is the merging of the labels, which reduces the computational time almost eight times on average. The reason is that considering a trade-off curve, the labeling algorithm usually generates a large number of labels that are mergeable but cannot dominate each other since the states of these labels have different start times. Compared with the full dominance rule, our proposed partial dominance rule considerably speeds up the search, because there exists a large number of unpromising states which can be detected and removed by the partial dominance test. Even though the detour pruning is originally developed as a component of the pulse algorithm, the results prove that it can still work very well when adapting it in the labeling algorithm by sorting the detour vertices.

In the second experiment, we compare the DP algorithms with different directions: the proposed adaptive-directional DP, the forward DP, the backward DP and the bidirectional DP where its stop time equals half of the scheduling horizon. All algorithms consider the DSSR technique and the three improvements mentioned above. As can be seen in Table 2, most of the instances presented there can be solved within three seconds by our algorithm. To obtain a clear difference for different directions, we use instances with a higher density of targets generated in only a quarter part of China (28°N-53°N and 103°E-133°E). Furthermore, the time step is set to one second. Table 3 reports the results on the high-density instances with 500 and 600 targets. Each row presents the CPU time for each single-orbit instance. The single-orbit instance name is denoted by "$|V|\_|K|$", where $|V|$ is the number of targets in the corresponding multi-orbit instance and $|K|$ is the number of VTWs during that orbit.

As discussed in Section 4.3.4, the bidirectional DP performs worst in this comparison due to the presence of a large number of duplicated paths produced. The results show a large difference between the forward DP and the backward DP in terms of the CPU time for some instances. For example, the backward DP can solve the instance "500_117" in 22.31 seconds but the forward one needs 279.09 seconds, while solving the instance "600_62" with the backward DP takes 232.80 seconds but the forward DP only takes 18.03 seconds. This implies that choosing a wrong direction for the extension may lead to a very high computational time. Moreover, from the comparative results, there is no clear dominance between these two approaches, and thus the efficient direction depends on the instance tested. We observe that the computational time of our adaptive-directional DP is typically not less than but close to the best of the forward DP and the backward DP for most of the instances. Specifically, the forward DP is on average 45% slower than our algorithm for the 500-target instances and 56% slower for the 600-target instances. Similarly, for the backward DP, the number reaches 32% for the 500-target instances and 58% for the 600-target instances. In a few cases, the adaptive-directional DP performs worse than both the forward and backward DP (instance "500_391", "600_71" and "600_144"), but the differences are rather limited. From these comparison, it is clear that the adaptive-directional DP can be regarded as a conservative but efficient choice to run the DP approach for our problem. It certainly avoids the very long computation times that sometimes appear with the forward and backward DP. We further compare the results of our algorithm with the best and worst CPU time among the forward and backward DP for each individual instance. The average results (not displayed in Table 3) show that our algorithm is only 15% slower than when the best direction would be known beforehand for each individual instance, but it is 120% faster than when we select the opposite direction.

5.2. Comparison with a state-of-the-art heuristic

In previous studies, only one heuristic approach has been proposed to tackle the AEOS scheduling with time-dependent profits, called the Bidirectional Dynamic Programming based Iterated Local Search (BDP-ILS) algorithm Peng et al. (2019). We compare the performance of our exact approach with the performance of this heuristic in terms of the solution quality and the CPU time. Note that the BDP-ILS is originally developed to solve multiple-orbit instances, but we apply this algorithm here on single-orbit instances. The single-orbit instances used in this comparison are based on multiple-orbit instances with 300, 400, 500 and 600 targets. For each multi-orbit instance, the four single-orbit instances with the most visible targets are solved. The results are reported in Table 4. The time step is set to five seconds. We keep the best settings of the parameters for the heuristic in Peng et al. (2019), i.e., the remove ratio is 0.1 and the iteration number is 200. The last column presents the gap between the best known solution of the heuristic and the optimal value obtained by our algorithm. A detailed solution for each instance is available here: https://www.mech.kuleuven.be/en/cib/op#section-32. Both these two algorithms are run on the same computer as mentioned before.

From Table 4, we can observe that our exact algorithm is almost as fast as the state-of-the-art heuristic for most of the instances, while this heuristic only obtains the optimal solution of one instance (instance "300_102"). For some instances, our exact algorithm is even slightly faster than the heuristic. For instances "300_100", "400_156" and "500_186" and "600_209", our exact algorithm has a shorter computational time than the heuristic while

Table 3
Comparative results of different direction of labeling extension.

Instance Name	Adaptive-directional DP	Forward DP		Backward DP		Bidirectional DP	
	CPU time(s)	CPU time(s)	T_{dif}	CPU time(s)	T_{dif}	CPU time(s)	T_{dif}
500_38	2.17	7.39	-2.41	1.76	0.19	3.18	-0.47
500_391	149.35	135.02	0.10	139.04	0.07	243.32	-0.63
500_60	64.66	13.96	0.78	163.98	-1.54	49.73	0.23
500_117	25.62	279.09	-9.89	22.31	0.13	440.84	-16.21
500_385	133.50	108.46	0.19	168.63	-0.26	261.29	-0.96
Average	75.06	108.79	-0.45	99.14	-0.32	199.67	-1.66
600_71	12.18	10.76	0.12	5.67	0.53	12.65	-0.04
600_476	686.17	1464.24	-1.13	709.65	-0.03	2244.74	-2.27
600_62	29.33	18.03	0.39	232.80	-6.94	139.44	-3.75
600_144	141.15	57.35	0.59	136.50	0.03	208.94	-0.48
600_441	251.82	193.06	0.23	682.44	-1.71	968.23	-2.85
Average	224.13	348.69	-0.56	353.41	-0.58	714.80	-2.19

Table 4
Comparative results of the proposed exact algorithm with a state-of-the-art heuristic.

Instance Name	Exact algorithm (ADP-DSSR)		Heuristic (BDP-ILS)		Gap (%)
	Optimal value	CPU time(s)	Best solution value	CPU time(s)	
300_100	157.47	0.91	151.75	1.32	3.63
300_102	179.18	2.47	179.18	1.52	0.00
300_106	165.59	1.83	165.39	1.36	0.12
300_120	181.53	1.79	177.92	1.56	1.99
400_136	193.81	3.19	188.79	2.27	2.59
400_143	198.95	1.59	193.27	2.68	2.85
400_152	195.98	2.48	195.07	2.52	0.46
400_156	194.01	1.46	185.28	2.97	4.50
500_183	206.22	4.54	201.87	3.85	2.11
500_186	191.97	2.66	184.50	4.13	3.89
500_191	220.61	4.00	219.05	4.29	0.71
500_197	224.34	4.51	222.20	4.36	0.95
600_209	217.64	1.42	209.24	4.35	3.86
600_222	227.24	4.12	222.64	4.54	2.02
600_229	227.63	5.57	223.76	5.88	1.70
600_236	213.83	8.55	210.42	5.80	1.59

the gaps of these instances exceed 3%. It is also noteworthy that for the multi-orbit instance with 600 targets, our algorithm only consumes less than 10 seconds for each single orbit, which proves the high performance of the developed algorithm on large-scale instances.

This state-of-the-art heuristic has been compared with using the CPLEX solver directly on the mathematical formulation, for small instances in Peng et al. (2019). The results show that the heuristic clearly outperforms CPLEX, with high-quality results and much smaller computation times. Since our exact algorithm outperforms this heuristic, we do not include a comparison with the CPLEX solver in this work.

5.3. Results for OPTW

For the OPTW, there are two existing exact algorithms in the state of the art: the Bidirectional DP with DSSR (BDP-DSSR) Righini and Salani (2009) and the pulse algorithm Duque et al. (2015). These two algorithms are tested on two classes of instances derived from the well-known Solomon's data-set of VRPTW instances Solomon (1987) and from the Cordeau et al. Cordeau et al. (1997) for the Multi-Depot Periodic Vehicle Routing Problem (MDPVRP). Solomon's instances contains 29 instances with 100 vertices classified into three categories: the R-instances where vertices are randomly located, the C-instances where vertices are clustered and the RC-instances where some vertices are located randomly and others are clustered. The number of vertices in Cordeau's instances varies from 48 to 288 vertices. These instances are considered to be harder than those of Solomon due to their larger time windows. These two sets of instances are available at https://www.mech.kuleuven.be/en/cib/op.

For these instances without time-dependent profits, the adaptive-directional extension and the merging process is no longer efficient. Thus, we use the same BDP-DSSR framework as in Righini and Salani (2009) while still maintaining the detour pruning strategy. Besides, at each iteration of DSSR, we delete the non-dominated labels which visit the current critical vertices more than once but preserve the rest of the non-dominated labels to the next iteration. These preserved labels can be used to dominate inferior labels in the next run of the DP and thereby reduce the number of unpromising labels generated. To avoid confusion, we call our algorithm for the OPTW the "improved BDP-DSSR" and the algorithm in Righini and Salani (2009) is called the "original BDP-DSSR". To ensure fair comparisons due to different experimental environments, a LINPACK benchmark is used to scale the computational time of the pulse algorithm to compare with the original BDP-DSSR Duque et al. (2015). However, we found that the coefficients used to scale the time of our hardware towards the ones used for the original BDP-DSSR and the pulse algorithm are inconsistent with the coefficient of 4.3 used to compare these two reference algorithms in Duque et al. (2015), probably due to the parallel search used in the pulse algorithm. To avoid the inconsistency, we keep the comparative results in Duque et al. (2015) but we rerun the code of the pulse algorithm provided by the authors. After that,

Table 5
Computational times (in seconds) of the proposed algorithms and the state-of-the-art for the OPTW over Solomon's instances.

Instance	Optimal value	Improved BDP-DSSR (s)	Original BDP-DSSR (s)	T_{dif}(Original) (-)	Pulse (s)	T_{dif}(Pulse) (-)
C101	320	0.00	0.06	-49.85	0.00	-2.39
C102	360	0.48	3.81	-6.90	0.54	-0.12
C103	400	60.28	1081.04	-16.93	8.94	0.85
C104	420	44.31	1856.39	-40.89	8.94	0.80
C105	340	0.33	0.12	0.63	0.07	0.79
C106	340	0.18	0.14	0.22	0.07	0.61
C107	370	0.12	0.2	-0.68	0.07	0.41
C108	370	0.74	1.43	-0.94	0.14	0.81
C109	380	1.88	10.57	-4.64	0.34	0.82
R101	198	0.05	0.03	0.39	0.07	-0.43
R102	286	9.61	233.2	-23.26	7.51	0.22
R103	293	130.21	5498.81	-41.23	31.29	0.76
R104	303	1826.33	> 7200	-2.94	80.04	0.96
R105	247	0.44	0.23	0.48	0.07	0.84
R106	293	14.93	334.49	-21.41	12.66	0.15
R107	299	150.12	2979.94	-18.85	35.49	0.76
R108	308	547.00	> 7200	-12.16	75.64	0.86
R109	277	2.15	3.09	-0.44	0.2	0.91
R110	284	3.47	30.83	-7.89	0.75	0.78
R111	297	46.62	1408.8	-29.22	14.56	0.69
R112	298	87.44	2508.17	-27.68	9.41	0.89
RC101	219	0.05	0.23	-3.37	0.07	-0.33
RC102	266	0.57	6.11	-9.74	0.13	0.77
RC103	266	6.50	88.12	-12.56	0.47	0.93
RC104	301	5.47	264.84	-47.38	1.56	0.72
RC105	244	0.64	2.86	-3.48	0.07	0.89
RC106	252	1.08	2.08	-0.92	0.13	0.88
RC107	277	3.18	49.19	-14.48	0.41	0.87
RC108	298	2.39	68.95	-27.85	0.81	0.66
Average				-14.62		0.53

Table 6
Computational times (in seconds) of the proposed algorithms and the state-of-the-art for the OPTW over Cordeau's instances.

Instance	Optimal value	Improved BDP-DSSR (s)	Original BDP-DSSR (s)	T_{dif}(Original) (-)	Pulse (s)	T_{dif}(Pulse) (-)
pr01_48	308	0.17	1.19	-6.09	0.14	0.17
pr02_96	404	3.46	37.52	-9.85	1.22	0.65
pr03_144	394	7.63	151.73	-18.89	2.30	0.70
pr04_192	489	24.68	648.82	-25.29	4.06	0.84
pr05_240	595	128.00	6815.82	-52.25	36.57	0.71
pr06_288	591	171.02	> 7200	-41.10	78.23	0.54
pr07_72	298	0.42	3.65	-7.77	0.27	0.35
pr08_144	463	6.65	90.71	-12.63	1.69	0.75
pr09_216	493	127.39	3270.88	-24.68	81.01	0.36
pr10_288	594	106.88	> 7200	-66.37	64.26	0.40
Average				-26.49		0.55

we scale the computational time of our algorithm according to the ratio of the results obtained by running the pulse algorithm in a different hardware environment.

Table 5 and 6 report on the experimental comparison between the original BDP-DSSR, our improved BDP-DSSR and the pulse algorithm. The first two columns report the instance name and the optimal value; column 3 presents the scaled computational time of the improved BDP-DSSR; column 4 and column 6 respectively show the (scaled) CPU times of the original BDP-DSSR and the pulse algorithm in seconds as reported in Duque et al. (2015), followed by their difference ratios of the computational time compared with the improved BDP-DSSR. The average value of these ratios are given at the end of the table.

Results in Table 5 and 6 show that our improved BDP-DSSR outperforms the original BDP-DSSR while the pulse algorithm is still the fastest algorithm. For Solomon's instances, the difference ratio between our algorithm and the orginal algorithm is -14.62. The results on Cordeau's instances exhibit a larger difference with a difference ratio of -26.49. These numbers show that compared with the original BDP-DSSR, our algorithm can solve the OPTW instances more efficiently, despite that the pulse algorithm still performs the best among these algorithms. It should be noted that the main purpose of this comparison is not to compete with the pulse algorithm in terms of the computational time, but rather to show that our algorithm has sufficient capability to also solve the OPTW, without time-dependent profits, in an efficient way. On the other hand, the pulse algorithm cannot be easily applied for the time-dependent case. For instance, the pulse algorithm uses a so-called "soft dominance test". It swaps two chosen visited vertices (except the source vertex and the last visited vertex) in a path, and the path is dominated and discarded if after the swapping operation a new path with less consumed time is found. The test is available only when swapping the visited vertices will not change the total profits. However, when time-dependent profits would be consid-

ered, the ordering of the visited vertices will greatly influence the total profit.

6. Conclusions

In this paper, we have presented an exact optimization algorithm for the single-orbit scheduling problem of an Agile Earth Observation Satellite (AEOS) considering time-dependent profits. The algorithm is developed based on the bidirectional dynamic programming with decremental state space relaxation which is originally designed for the ESPPRC or the OPTW. To address the time-dependent profits feature, a "trade-off" curve, namely the accumulated profit function, is associated with each path to be extended. We have described the corresponding dominance rule, the decremental state space relaxation and other algorithmic details. In addition, we have proposed four algorithmic improvements to speed up the algorithm, including the partial dominance rule, the process of the merging labels, the detour pruning strategy and the adaptive-direction extension. The experimental results of our real-world instances prove that each improvement can effectively reduce the computational time for our instances. Furthermore, we compared the performance of our exact algorithm and a state-of-the-art heuristic. The results showed that the computation times of these two algorithms are similar while the heuristic algorithm fails to find the optimal solutions for most of the instances. Since there is no previous work solving the AEOS scheduling problem to optimality, we also tested our algorithm on the OPTW instances without considering the time-dependent profits and compared it with two algorithms in the state of the art. Our algorithm performs better than the original BDP-DSSR algorithm but slightly worse than the state-of-the-art pulse algorithm. However, we conclude that our algorithm is capable of solving the OPTW efficiently and we argue that the pulse algorithm cannot be easily adapted for addressing time-dependent profits due to its algorithmic mechanism.

Further research could focus on the AEOS scheduling problem with multiple orbits. This rather interesting extension of the problem allows optimizing the schedule on multiple orbits, satisfying the practical needs. A column generation technique can be utilized to solve this problem while its pricing problem can be regarded as the single-orbit scheduling problem and this can be solved efficiently by our ADP-DSSR problem. To the best of our knowledge, this problem has not yet been addressed with an exact solution technique. In addition to the time-dependent profits, the time-dependent energy resource can also be considered in the model since the transition of look angles requires the consumption of the on-board energy. In this case, the trade-off between the resources and profits should be made and the design of the exact solution will become more complicated.

CRediT authorship contribution statement

Guansheng Peng: Conceptualization, Writing - original draft, Data curation, Visualization. **Guopeng Song:** Writing - review & editing, Visualization. **Lining Xing:** Conceptualization, Supervision, Visualization. **Aldy Gunawan:** Writing - review & editing, Visualization. **Pieter Vansteenwegen:** Conceptualization, Writing - original draft, Supervision, Visualization.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (No. 61873328, 61525304, 71801218, 71501180, U1501254 and 61773120) and China Scholarship Council (201703170228). It is also supported by the Foundation for the Author of National Excellent Doctoral Dissertation of China (2014-92), and the Innovation Team of Guangdong Provincial Department of Education (2018KCXTD031). The authors thank Professor Andrés L. Medaglia for sharing the code of the pulse algorithm, and two anonymous referees for their valuable suggestions and corrections.

References

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

Boland, N., Dethridge, J., Dumitrescu, I., 2006. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. Operations Research Letters 34 (1), 58–68.

Chen, X., Reinelt, G., Dai, G., Spitz, A., 2019. A mixed integer linear programming model for multi-satellite scheduling. European Journal of Operational Research 275 (2), 694–707.

Cordeau, J.-F., Gendreau, M., Laporte, G., 1997. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 30 (2), 105–119.

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

Dabia, S., Ropke, S., Van Woensel, T., de Kok, T., 2013. Branch and price for the time-dependent vehicle routing problem with time windows. Transportation Science 47 (3), 380–396.

Desaulniers, G., Errico, F., Irnich, S., Schneider, M., 2016. Exact algorithms for electric vehicle-routing problems with time windows. Operations Research 64 (6), 1388–1405.

Desrochers, M., Desrosiers, J., Solomon, M.M., 1990. A new optimization algorithm for the vehicle routing problem with time windows. Operations Research 40, 342–354.

Dilkina, B., Havens, B., 2005. Agile Satellite Scheduling via Permutation Search with Constraint Propagation. Technical Report. Actenum Corporation.

Dror, M., 1994. Note on the complexity of the shortest path models for column generation in VRPTW. Operations Research 42 (5), 977–978.

Duque, D., Lozano, L., Medaglia, A.L., 2015. Solving the orienteering problem with time windows via the pulse framework. Computers & Operations Research 54, 168–176.

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. Kobe

Erkut, E., Zhang, J., 1996. The maximum collection problem with time-dependent rewards. Naval Research Logistics 43 (5), 749–763.

Feillet, D., Dejax, P., Gendreau, M., Gueguen, C., 2004. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. Networks 44 (3), 216–229.

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. Annals of Operations Research 69, 115–134.

Habet, D., Vasquez, M., Vimont, Y., 2010. Bounding the optimum for the problem of scheduling the photographs of an agile earth observing satellite. Computational Optimization and Applications 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. Computers & Operations Research 100, 12–25.

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

Li, Y., Xu, M., Wang, R., 2007. Scheduling observations of agile satellites with combined genetic algorithm. In: Third International Conference on Natural Computation (ICNC 2007), 3, pp. 29–33. Haikou

Liberatore, F., Righini, G., Salani, M., 2011. A column generation algorithm for the vehicle routing problem with soft time windows. 4OR 9, 49–82.

Lin, W.-C., Liu, C.-Y., Liao, D.-Y., Lee, Y.-Y., 2003. Daily imaging scheduling of an earth observation satellite. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 35, 213–223.

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

Peng, G., Dewil, R., Verbeeck, C., Gunawan, A., Xing, L., Vansteenwegen, P., 2019. Agile earth observation satellite scheduling: An orienteering problem with time-dependent profits and travel times. Computers & Operations Research 111, 84–98.

Righini, G., Salani, M., 2008. New dynamic programming algorithms for the resource constrained elementary shortest path problem. Networks 51 (3), 155–170.

Righini, G., Salani, M., 2009. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Computers & Operations Research 36, 1191–1203.

Solomon, M.M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35 (2), 254–265.

Tilk, C., Rothenbcher, A.-K., Gschwind, T., Irnich, S., 2017. Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. European Journal of Operational Research 261 (2), 530–539.

Wang, J., Demeulemeester, E., Hu, X., Qiu, D., Liu, J., 2019. Exact and heuristic scheduling algorithms for multiple earth observation satellites under uncertainties of clouds. IEEE Systems Journal 13 (3), 3556–3567.

Wang, P., Reinelt, G., Gao, P., Tan, Y., 2011. A model, a heuristic and a decision support system to solve the scheduling problem of an earth observing satellite constellation. Computers & Industrial Engineering 61 (2), 322–335.

Wei-Cheng Lin, Chung-Yang Liu, Da-Yin Liao, Yung-Yao Lee, 2003. Daily imaging scheduling of an earth observation satellite. In: SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483), 2, pp. 1886–1891vol.2.

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

Xiaolu, L., Baocun, B., Yingwu, C., Feng, Y., 2014. Multi satellites scheduling algorithm based on task merging mechanism. Applied Mathematics and Computation 230, 687–700.

Xu, R., Chen, H., Liang, X., Wang, H., 2016. Priority-based constructive algorithms for scheduling agile earth observation satellites with total priority maximization. Expert Systems with Applications 51, 195–206.

Yi, J., 2003. Vehicle routing with time windows and time-dependent rewards: A problem from the american red cross. Manufacturing & Service Operations Management 5 (1), 74–77.