Satellite downlink scheduling problem: A case study

Daniel Karapetyan\textsuperscript{1,2,*}, Snezana Mitrovic Minic\textsuperscript{1,3}, Krishna T. Malladi\textsuperscript{1}, Abraham P. Punnen\textsuperscript{1}

Abstract

The synthetic aperture radar (SAR) technology enables satellites to efficiently acquire high quality images of the Earth surface. This generates significant communication traffic from the satellite to the ground stations, and, thus, image downlinking often becomes the bottleneck in the efficiency of the whole system. In this paper we address the downlink scheduling problem for Canada’s Earth observing SAR satellite, RADARSAT-2. Being an applied problem, downlink scheduling is characterised with a number of constraints that make it difficult not only to optimise the schedule but even to produce a feasible solution. We propose a fast schedule generation procedure that abstracts the problem specific constraints and provides a simple interface to optimisation algorithms. By comparing empirically several standard meta-heuristics applied to the problem, we select the most suitable one and show that it is clearly superior to the approach currently in use.

Keywords: Satellite, Scheduling, Optimisation, Meta-Heuristics.

1. Introduction

Efficient scheduling of image acquisition and image downlinking plays a vital role in satellite mission planning. These operations are often interlinked and solved using scheduling heuristics. Most of the literature on satellite mission planning (image acquisition and downlinking) is divided into two categories: optical satellites \cite{25, 27} and Synthetic Aperture Radar (SAR) satellites \cite{4, 5, 7, 10}.

This paper deals with the downlink scheduling portion of the mission planning operations of Canada’s Earth observing SAR satellite, RADARSAT-2. We

\textsuperscript{*}This research work was supported by an NSERC CRD grant awarded to Abraham P. Punnen with the MDA Systems Ltd. as the collaborating industrial partner.

\textsuperscript{*}Corresponding author

\textsuperscript{1}Department of Mathematics, Simon Fraser University, Surrey, BC V3T 0A3, Canada

\textsuperscript{2}ASAP Research Group, School of Computer Science, University of Nottingham, Nottingham NG8 1BB, UK

\textsuperscript{3}MDA Systems Ltd., Richmond, BC V6V 2J3, Canada
assume that image acquisition schedule is given and not to be changed. This is consistent with existing policies and practices.

Recently, there is a steady increase in demand for RADARSAT-2 imagery. Thus any improvements in image downlink operations would improve the efficiency of the RADARSAT-2 mission and this is the primary motivation behind this study. Currently used downlink scheduling process exploits a greedy-like algorithm [16] followed by human intervention whenever necessary. We explored local search heuristics and metaheuristics to improve the efficiency of the downlink scheduling. Our experimental study on real-world problem instances has shown that the proposed techniques significantly improved downlink throughput and schedule quality.

The satellite image downlink scheduling problem and its variations have been studied by many authors. Some of these works were focused on case studies for specific satellites [10, 18] whereas others are more general purpose in nature [3, 14, 8, 11, 13, 24, 26, 27, 28]. Literature from machine scheduling [6, 23] and resource-constrained project scheduling [17] are also relevant in solving the satellite image downlink scheduling problem (SIDSP). However, each mission planning problem has its own restrictions and properties that can be exploited. Special care is needed to make sure that such restrictions are handled adequately which sometimes changes the inherent combinatorial structure of the problem significantly. Thus, investigating case studies of special SIDSPs are interesting and relevant as established in this study, although existing literature on the SIDSP considerably influenced our work.

The paper is organised as follows. Section 2 describes the real-world problem, and Section 3 introduces its mathematical model as well as various notations and definitions. Sections 4 and 5 deal with our heuristic algorithms. Data analysis and test instances are reported in Section 6 followed by computational results in Section 7 and concluding remarks in Section 8.

2. The RADARSAT-2 SIDSP

We consider the problem arising in satellite industry that deals with scheduling downlinks of images assuming that the schedule for image acquisition is already generated. We study RADARSAT-2 satellite that orbits the Earth to acquire images that can be downlinked to a set $G$ of stationary ground stations for further processing. Since the problem has a lot in common with scheduling problems and, in particular, with the Resource Constrained Project Scheduling Problem, our notations are close to the ones used in the scheduling literature.

Let $V$ be a set of $n$ downlink requests to be scheduled within the planning horizon of 24 hours. For each request $j \in V$, release time $r_j$, deadline $d_j$, downlink duration $p_j$, priority $w_j$ and ground station $g_j \in G$ are prescribed. The interval $[r_j, d_j]$ is called the time window of request $j$. Downlink requests are classified as regular and urgent. The downlink of urgent requests has to start as close as possible to their release times. However, downlinks of regular requests are more flexible and are primarily governed by their priorities. Urgent requests have absolute priority dominance over the regular (non-urgent) requests, i.e.,
any (small) improvement in downlinking of urgent requests is preferred over large improvements in downlinking of regular requests. Finally, some requests have to be downlinked to two ground stations and we call them dual requests. Each dual request is represented by two requests $i, j \in V$, and we are given a set of pairs $(i, j)$ of dual requests.

A downlink activity can be carried out only when the satellite is passing over a ground station. This time interval is called visibility mask of the station. The RADARSAT-2 has two antennas for downlinking, and it can work in half-power and full-power setting. When working in half-power setting, the two antennas can work separately and they can independently downlink two different images to one or two different ground stations simultaneously. In full-power setting, the satellite can process only one downlink at a time.

Each ground station $g \in G$ has one or two channels for receiving the downlinked images. Ground stations can also be classified based on their transmission power. Let $G_1$ be the set of ground stations that are in half-power setting and $G_2$ be the set of ground stations that are in full-power setting. Then $G = G_1 \cup G_2$ and $G_1 \cap G_2 = \emptyset$. When two images are downlinked one after another to stations with the same power setting, there must be a gap (set up time) $\delta$ time units between the two downlinks. When two images are downlinked consecutively to two stations with different power settings, the required gap between the downlinks is $\Delta > \delta$.

The visibility masks of a ground station $g \in G$ can be represented by a collection $M_g$ of non-overlapping time intervals called the normal visibility masks. Certain downlink requests require better reliability and they have to be downlinked within high reliability visibility mask $M^1_g$. Each high reliability visibility mask $m^1 \in M^1_g$ is a sub-interval of some normal visibility mask $m \in M_g$.

3. Mathematical Model

Note that SIDSP deals with the problem of finding an image downlink schedule so that a utility function is maximized. The utility function considers the number of downlinks scheduled, their priority values, and the difference between the downlink start time and the start time of the request time window (tardiness). Rejection of requests is allowed, and a rejected request is referred to as unscheduled.

A solution $S$ to SIDSP — a schedule — is a set $S \subseteq V$ of scheduled requests and associated downlink start times $S_j$ for each $j \in S$. Our model for the SIDSP

---

4Due to large customer demand for satellite imagery, the downlink scheduling problem is usually oversubscribed. Thus, if request rejection is not allowed, the downlink scheduling problem would often be infeasible. Furthermore, the request rejection assumption allows the scheduling algorithm to choose the requests that maximise the resource utilisation.
is to:

\[
\text{Maximize } f(S) = \sum_{j \in S} w_j \left( 1 - \alpha \cdot \frac{S_j - r_j}{d_j - p_j - r_j} \right)
\]

subject to \( S \in F \),

where \( 0 \leq \alpha \leq 1 \) is a parameter reflecting the importance of tardiness minimisation (\( \alpha = 0 \) disables tardiness minimisation while \( \alpha = 1 \) means that scheduling a request to the end of its time window is as bad as rejecting it), and the collection \( F \) contains all schedules \( S \) satisfying the following constraints:

1. No downlink activity happens outside the planning horizon.
2. A downlink cannot start earlier than the release time of the corresponding request and must be finished by its deadline: \( r_j \leq S_j \leq d_j - p_j \) for each \( j \in S \).
3. Once a downlink starts, it cannot be preempted, i.e., an image cannot be split into several fragments.
4. Each urgent request has to be scheduled at the earliest possible time in its time window even at the cost of delaying or cancelling some regular downlinks.
5. For each dual request \((i, j) \in D, i \in S \) if and only if \( j \in S \).
6. There must be a gap of at least \( \delta \) units between two consecutive downlinks under the same power setting.
7. There must be a gap of at least \( \Delta \geq \delta \) time units between two consecutive downlinks under different power setting.
8. If the satellite is in the full-power setting, then only one antenna can work and the other has to be idle. For both of the satellite antennas to be working independently, the satellite has to be in the half-power setting. The full-power setting is used if and only if the satellite transmits data to a full-power ground station.

We assume the following:

1. The satellite is in the half-power setting at the beginning of the planning horizon and that it has to be in the half-power setting at the end as well.
2. Downlinking (in half-power setting) can start right from the beginning of the planning horizon, i.e., it is guaranteed that no half-power downlink activity happened within \( \delta \) time units before the beginning of the planning horizon.
We tackle constraint (6) by the following pre-processing procedure: Add $\delta$ to $p_j$ and $d_j$ for all requests $j \in V$. Also, add $\delta$ to the upper bounds of each of the time intervals in normal and high reliability visibility masks for all $g \in G$. Extend the planning horizon by $\delta$. Further, replace $\Delta$ with $\Delta - \delta$.

The SIDSP is NP-hard since several NP-hard machine scheduling problems are special cases of it. Consider, for example, the parallel machine scheduling problem with two machines representing satellite antennas.

Heuristic algorithms for the scheduling problems of this class usually exploit the so-called serial scheduling scheme, in which the solver searches in the space of job sequences while a polynomial time schedule generator converts the job sequences into schedules. The schedule generator is a greedy algorithm scheduling the jobs (requests in our work) in the given order, choosing the earliest available position for each of them. An important property of the schedule generator is that it always generates active schedules, i.e., schedules such that none of the unscheduled jobs can be added to it and no scheduled job can be advanced without delaying some other job [19]. Given that our objective function is regular (i.e., delaying or deleting a downlink cannot improve the solution if no other changes are introduced), there exists a sequence of requests generating an optimal schedule [12].

Moreover, for any active schedule, there exists a sequence generating that schedule (indeed, it is enough to sort the jobs in ascending order by their start times). This implies that the schedule generator can produce a worst possible active schedule. For example, it can produce a schedule $S$ of objective $f(S) = 0$ by scheduling requests of zero priority and leaving the requests with non-zero priorities unscheduled.

With the serial scheduling scheme in mind, the SIDSP can be viewed as a problem of optimising a permutation of the downlink requests for which a schedule will be generated in the following manner: From the order defined by the permutation, schedule each request to the earliest available time considering the constraints related to the satellite antennas, ground station channels, visibility masks and dual requests. Let $\varphi$ be the set of all permutations of elements of $V$. For any $\pi \in \varphi$, let $S_\pi$ be the corresponding schedule generated by the schedule generator algorithm (see Section 5). Thus, to solve SIDSP, we solve the following Downlink Request Permutation Problem (DRPP):

Maximize $\phi(\pi) = f(S_\pi)$
subject to $\pi \in \varphi$.

Note that to evaluate the solution quality of a permutation we need to generate the schedule from that permutation, i.e., to apply the schedule generator algorithm described in Section 5.

4. Solution Approach

Instead of dealing with urgent requests separately in the objective function or in some other way, we used a two phase solution approach. The first phase
schedules all urgent requests, and the second phase schedules all regular requests using the remaining resources. Hence, our approach respects the requirement to give the urgent requests ultimate priority over the regular requests. A high-level description of the algorithm is given as follows:

**Phase 1 — Urgent requests:** Schedule all urgent requests using heuristic $H$.

**Phase 2 — Regular requests:** Fix the urgent requests scheduled in Phase 1, update resource availability accordingly and schedule regular requests using heuristic $H$.

In the following sections we describe several heuristic algorithms for DRPP that can be sued as $H$. We primarily focused on standard algorithmic paradigms since one of the objectives was to propose algorithms that are easy to understand and implement. In particular, we limited our experiments to the Greedy Randomised Adaptive Search Procedure (GRASP), Ejection Chain, Simulated Annealing and Tabu Search algorithms.

4.1. Construction Heuristic

One of the main components of our heuristics is a construction algorithm, that generates an initial schedule from the list of sorted requests in $V$ (the so-called priority-rule based scheduling method known to be efficient for quick generation of reasonably good solutions [12]). We considered the following sorting criteria and tie breakers:

1. Priority $w_j$;
2. Time window duration $|d_j - r_j - p_j|$;
3. Downlink duration $p_j$.

After extensive computational experiments using various combinations of the listed criteria, we selected sorting the elements of $V$ by $w_j$ in the descending order, ties broken by $|d_j - r_j - p_j|$ in the ascending order, and remaining ties are broken by $p_j$ in the ascending order.

4.2. GRASP

The Greedy Randomised Adaptive Search Procedure is a simple meta-heuristic often applied to scheduling problems [22]. GRASP repeatedly generates solutions with a randomised greedy constructor followed by a local search phase. Due to the randomness of the greedy procedure, GRASP is likely to produce new solutions on every iteration. The best out of all the produced solutions is selected in the end. For details, see Algorithm 1.

\footnote{Without rounding these values, further tie breakers would not affect the solution as the time window durations for all the requests are likely to be different.}
Algorithm 1 GRASP Algorithm

**Input:** Given time

**Output:** A permutation optimised with respect to \( \phi(\pi) \)

while given time has not elapsed do
  \( \pi' \leftarrow \text{GreedyRandomisedConstructor()} \)
  \( \pi' \leftarrow \text{LocalSearch}(\pi') \)
  if \( \phi(\pi') > \phi(\pi) \) then
    \( \pi \leftarrow \pi' \)
  end if
end while

return \( \pi \)

As a randomised greedy constructor \( \text{GreedyRandomisedConstructor}() \), we use a modification of the construction procedure described in Section 4.1. In particular, on every iteration of building \( \pi \), \( \text{GreedyRandomisedConstructor}() \) orders all the remaining downlink requests as described in Section 4.1 and selects one of the first ten candidates randomly.

Our local search \( \text{LocalSearch}(\pi) \) explores a swap neighbourhood. The swap neighbourhood \( N_{\text{swap}}(\pi) \) consists of all the solutions that can be obtained from the permutation \( \pi \) by swapping two of its elements. The size of the neighbourhood is \( |N_{\text{swap}}(\pi)| = \frac{n(n-1)}{2} \), and, hence, it would take \( O(n^5) \) time to explore it (we will show in Section 5 that the complexity of the schedule generator algorithm is \( O(n^3) \)). With such a high complexity of the local search, GRASP is likely to perform only few iterations, which is not enough to exploit the strength of the meta-heuristic. To speed up the local search phase, we decided to explore the neighbourhood in a random order and terminate the search when a prescribed time limit is reached. For details, see Algorithm 2.

Algorithm 2 GRASP local search \( \text{LocalSearch}(\pi) \)

**Input:** \( \pi \) — initial permutation; the time given for one run of the local search

**Output:** Improved permutation with respect to \( \phi(\pi) \)

while given time did not elapse do
  Select randomly \( i \neq j \in \pi \)
  \( \pi' \leftarrow \text{swap } i \text{ and } j \text{ in } \pi \)
  if \( \phi(\pi') > \phi(\pi) \) then
    \( \pi \leftarrow \pi' \)
  end if
end while

return \( \pi \)

4.3. Ejection Chain Algorithm

Ejection chain methods [15] have commonly been used in developing Very Large Scale Neighbourhood (VLSN) search algorithms [1, 2] for solving com-
plex combinatorial optimisation problems. For example, the well known Lin-
Kernighan heuristic — an efficient heuristic for solving the travelling salesman
problem  — is an ejection chain algorithm. We use the idea of the ejection
chains to develop a simple and effective heuristic to solve DRPP.

The data structure used in our ejection chain algorithm is a pair \((\pi, h)\), where
\(\pi\) is a permutation of all the requests in \(V\) and \(h \in \{1, 2, \ldots, n\}\) is a position
in this permutation called a hole. In order to calculate the objective value \(\phi(\pi, h)\),
copy all the permutation \(\pi\) skipping the \(h\)th element and feed this copy to the
schedule generator algorithm (Section 5) to obtain the schedule and calculate
its objective.

The basic move in our ejection chain algorithm is swapping the \(i\)th element
of \(\pi\) with the ‘hole’, for some \(i \neq h \in \{1, 2, \ldots, n\}\). There are \(n - 1\) options
for this move, and we select the best one with respect to \(\phi(\pi, h)\). In addition
to calculating \(\phi(\pi, h)\), we also calculate \(\phi(\pi)\) on every iteration to keep track
of the best ‘full’ solution found. A version of this ejection chain algorithm is
presented in a preliminary report on this problem [18]. For a formal description
of our ejection chain algorithm see Algorithms 3 and 4.

### Algorithm 3 Ejection Chain Algorithm

| Input: \(\pi\) — permutation; depth — maximum ejection chain length |
| Output: Improved permutation with respect to \(\phi(\pi)\) |

\[
\begin{align*}
\sigma & \leftarrow \pi \\
c & \leftarrow n; \{c \text{ counts non-improving iterations}\} \\
h & \leftarrow 1 \\
\text{while } c > 0 \text{ do} \\
\quad \text{if } \text{improvement}(\pi, h, \sigma, D) = 1 \text{ then} \\
\qquad c & \leftarrow n \\
\qquad \sigma & \leftarrow \pi \text{ as changed by } \text{improvement}(\pi, h, \sigma, D) \\
\quad \text{else} \\
\qquad c & \leftarrow c - 1 \\
\quad \text{end if} \\
\quad \text{Next hole position: If } h = n, \text{ then } h \leftarrow 1, \text{ else } h \leftarrow h + 1 \\
\text{end while} \\
\text{return } \sigma
\end{align*}
\]

4.4. Simulated Annealing

Simulated Annealing (SA) is a stochastic (global) optimisation technique
widely used in the literature for solving various optimisation problems. SA
algorithm is similar to the randomised local search with the exception that the
worsening moves can also be accepted. We implemented the standard SA based
on the \(N_{\text{swap}}(\pi)\) neighbourhood, see Algorithm 5.

We use a simple variation of the Simulated Annealing meta-heuristic based
on the swap neighbourhood. In each iteration, we swap two randomly selected
elements in \(\pi\). If the obtained solution is better than \(\pi\), we replace \(\pi\) with
Algorithm 4 Recursive Build of the Ejection Chain: \textit{improvement}(\pi, h, \sigma, d)

\textbf{Input:} \pi — permutation; \ h — hole position; \ \sigma — the best solution found so far; \ d — the remaining search depth

\textbf{Output:} \ 1 — improving ejection chain exists; \ 0 — otherwise

\begin{enumerate}
\item \textbf{if} \ d = 0 \textbf{then}
\item \textbf{return} \ 0
\item \textbf{end if}
\item \phi_p \leftarrow \phi(\pi, h)
\item \ j \leftarrow 0
\item \textbf{for} \ i \leftarrow 1, 2, \ldots, h - 1, h + 1, h + 2, \ldots, n \textbf{do}
\item \pi' \leftarrow \text{swap } \pi(h) \text{ and } \pi(i) \text{ in } \pi
\item \textbf{if} \ \phi(\pi') > \phi(\sigma) \text{ then}
\item \sigma \leftarrow \pi'
\item \textbf{return} \ 1
\item \textbf{end if}
\item \textbf{end if}
\item \textbf{end for}
\item \textbf{if} \ j > 0 \textbf{then}
\item \pi' \leftarrow \text{swap } \pi(h) \text{ and } \pi(j) \text{ in } \pi
\item \textbf{if} \ \textit{improvement}(\pi', j, \sigma, d - 1) = 1 \textbf{then}
\item \textbf{return} \ 1
\item \textbf{end if}
\item \textbf{end if}
\item \textbf{return} \ 0
\end{enumerate}

that solution. Otherwise, the probability of accepting the solution is \( e^{\frac{\phi(\pi') - \phi(\pi)}{T}} \), where \( T \) is the current temperature. In each iteration, the temperature decreases linearly from a given initial value \( T_0 \) to 0.

4.5. Tabu Search

The Tabu Search (TS) meta-heuristic is a neighbourhood-based search methodology with a tabu list mechanism for escaping local maxima. By storing certain features of the recent solutions in a \textit{tabu list}, TS avoids re-exploring previously visited areas of the search space, which, in turn, allows the algorithm to accept worsening solutions when it is at a local maximum. A high-level description of the TS procedure is given in Algorithm 6.

The efficiency of the TS algorithm significantly depends on the features to be kept in the tabu list. For problems with permutation-based solution representation, it is a common practice to use pairs of recently modified elements and their positions as such features. Any solution that has the saved elements at
Algorithm 5 Simulated Annealing improvement heuristic

**Input:** \( \pi \) — permutation; \( T_0 \) — the initial temperature; given time  
**Output:** Improved permutation with respect to \( \phi(\pi) \)  

\[
\text{while given time has not elapsed do}
\]

Select \( i \neq j \in \pi \) randomly  
\( \pi' \leftarrow \text{swap } i \text{ and } j \text{ in } \pi \)  
if \( \phi(\pi') > \phi(\pi) \) then  
\( \pi \leftarrow \pi' \)  
else  
\( T \leftarrow T_0 \cdot \frac{\text{"remaining time"}}{\text{given time}} \)  
\( p \leftarrow e^{\frac{\phi(\pi') - \phi(\pi)}{T_0 \text{ given time}}} \)  
\( r \leftarrow \text{random number uniformly distributed in } [0, 1] \)  
if \( r < p \) then  
\( \pi \leftarrow \pi' \)  
end if  
end if

\[\text{return } \pi\]

exactly the same positions is excluded from exploration in the next few iterations (tabu tenure) of the search.

However, our experiments have shown that such a TS implementation performs poorly on DRPP. For the explanation, observe that one SIDS P solution can be represented by many distinct DRPP solutions. For instance, if a request \( j \) is scheduled at its release time \( r_j \), advancing \( j \) in the permutation \( \pi \) does not change the resulting schedule \( S_\pi \). Hence, simple constraints on the permutation \( \pi \) do no guarantee that the SIDSP solution \( S_\pi \) is excluded from the search, which might affect the ability of the TS to escape the local maximum. In other words, tabu lists based on request positions work well in the space of permutations but fail in the space of schedules.

To make sure that the search does not return to the recently explored region of SIDSP solutions, we save features of schedules (rather than permutations) in the tabu list. Each element of our tabu list includes the objective value and the average request tardiness \( t(S) = \frac{1}{|S|} \sum_{j \in S} S_j - r_j \) of a recently explored solution \( S = S_\pi \). If both the objective value and the average tardiness of a new solution \( S' \) are close to the ones in the list, such a solution is excluded from the search. More formally, with respect to a tabu list element \((f, t)\), a solution \( S' \) is a tabu if (a) \( \frac{|f(S') - f|}{f} \leq \epsilon \) and (b) \( \frac{|t(S') - t|}{t} \leq \epsilon \), where \( 0 < \epsilon \ll 1 \) is a tolerance parameter of the algorithm.

5. Schedule Generator Algorithm

This section describes the greedy algorithm we use to generate a schedule from a given ordered subset of requests \( V \). Let us start by introducing some
Algorithm 6 Tabu Search improvement heuristic

**Input:** $\pi$ — initial permutation; $L$ — the tabu list length; given time

**Output:** Improved permutation with respect to $\phi(\pi)$

$\pi_{\text{cur}} \leftarrow \pi \{\text{We will be exploring the neighbourhood of } \pi_{\text{cur}}\}$

$\pi^* \leftarrow \pi \{\pi^* \text{ is the best solution found in the neighbourhood of } \pi_{\text{cur}}\}$

**while** given time has not elapsed **do**

**for all** $\{i, j\} \subset \pi_{\text{cur}}$ **do**

$\pi' \leftarrow$ swap $i$ and $j$ in $\pi_{\text{cur}}$

if $\phi(\pi') > \phi(\pi)$ **then**

$\pi \leftarrow \pi'$ \{Record as the best found solution\}

$\pi^* \leftarrow \pi'$ \{Update $\pi^*$ ignoring the possible tabu\}

else if $\pi'$ is not tabu and $\phi(\pi') > \phi(\pi^*)$ **then**

$\pi^* \leftarrow \pi'$

**end if**

**end for**

Add a tabu for solution $\pi^*$ to the tabu list

If the tabu list length exceeds $L$, remove the oldest list element

$\pi_{\text{cur}} \leftarrow \pi^*$ \{Move to the best found solution\}

**end while**

**return** $\pi$

terminology and notations to simplify the discussion.

5.1. Interval Sets

Let $I$ be a finite set of non-intersecting intervals. We call such a structure interval set. Since the elements of $I$ are non-intersecting intervals, $I$ can also be viewed as an ordered set with the natural order produced by the position of these intervals on the real line. Thus, $I$ is represented as $I = \{(\ell_1, u_1], (\ell_2, u_2], \ldots, (\ell_v, u_v]\}$, where $\ell_1 < u_1 < \ell_2 < u_2 < \cdots < \ell_v < u_v$ and $v = |I|$. We refer to the $k$th interval in $I$ as $I_k$.

Consider an arbitrary interval $[a, b]$ and an interval set $I = \{(\ell_1, u_1], (\ell_2, u_2], \ldots, (\ell_v, u_v]\}$. Let us introduce sets of intervals $B$, $C$ and $D$ as follows:

\[
B = \{[(\ell_k, u_k]) \in I : \ell_k < b \text{ and } u_k > a\},
\]

\[
C = \begin{cases} 
\{[\ell, a]\} & \text{if } \exists [\ell_k, u_k] \in I \text{ such that } \ell_k < a < u_k \\
\emptyset & \text{otherwise, and}
\end{cases}
\]

\[
D = \begin{cases} 
\{[b, u_k]\} & \text{if } \exists [\ell_k, u_k] \in I \text{ such that } \ell_k < b < u_k \\
\emptyset & \text{otherwise.}
\end{cases}
\]

Now, subtracting interval $[a, b]$ from $I$ generates a set $I \ominus [a, b]$ of non-intersecting intervals given by

\[
I \ominus [a, b] = (I \setminus B) \cup C \cup D.
\]

The notation $I \ominus [a, b]$ is read as $I$ minus interval $[a, b]$. (see Figure 1)
5.2. Implementation of the Schedule Generator

Let \( V^* \) be an ordered subset of \( V \). Given \( V^* \), we now present a greedy algorithm to schedule requests in \( V^* \) following the order prescribed in \( V^* \). The algorithm maintains several indicator interval sets representing channel availability at ground stations and antenna availability on the satellite. After scheduling a request, the algorithm updates these indicator sets.

Let \( A \) be an interval set that represents the time intervals when both satellite antennas are available. Let \( H \) and \( F \) be interval sets indicating the time intervals when a half-power and full-power downlinks can happen, respectively. Let \( Q_g \) and \( Q_{g1} \) be interval sets indicating the availability of the ground station \( g \in G \) in normal and high reliability visibility, respectively. Note that we do not need the second pair of indicator sets for a two channel ground station. Indeed, the number of channels in this case is not limiting since the number of simultaneous downlinks is constrained by the number of antennas. Hence, resource availability of the two channel ground stations does not need to be tracked.

The greedy algorithm works as follows:

1. Let \( V^{**} \leftarrow V^* \).

2. Initialize the sets \( A \), \( H \) and \( F \) to the planning horizon \( \{[0, 24 \text{ hours}]\} \). Initialize the set \( Q_g \) to the normal visibility mask of \( g \) and \( Q_{g1} \) to the high reliability visibility mask of \( g \) for every ground station \( g \in G \). Note that for every \([\ell_k, u_k] \in Q_g \) there is an interval \([\ell_l, u_l] \in Q_g \) such that \( \ell_l < \ell_k < u_k < u_l \).

3. Select the first request \( j \in V^{**} \).

4. Suppose that \( j \) is to be downlinked to a half-power station \( g \in G_1 \). Let \( Q \leftarrow Q_g \) if \( j \) requires normal reliability and \( Q \leftarrow Q_{g1} \) otherwise. Choose
the smallest $k \in \{1, 2, \ldots, |H|\}$ and $l \in \{1, 2, \ldots, |Q|\}$ such that $|H_l \cap Q_k \cap [r_j, d_j]| \geq p_j$. If no such $k$ and $l$ exist, leave the request $j$ unscheduled and proceed to Step 3. Otherwise schedule $j$: assign $S \leftarrow S \cup \{j\}$ and $S_j \leftarrow x$, where $H_k \cap Q_l \cap [r_j, d_j] = [x, y]$.

After scheduling $j$, update the indicator sets as follows. Let $X = [S_j, S_j + p_j]$. Apply $X \leftarrow X \cap [\ell_k, u_k]$ for every $[\ell_k, u_k] \in A$. Then $t \in X$ iff $t \in [S_j, S_j + p_j]$ and exactly one antenna was available at time $t$ before scheduling $j$. Apply $H \leftarrow H \cap [\ell_k, u_k]$ for every $[\ell_k, u_k] \in X$ to reflect that the time intervals $X$ are no longer available to half-power downlinks. Also apply $A \leftarrow A \cap [S_j, S_j + p_j]$ and $F \leftarrow F \cap [S_j - \Delta, S_j + p_j + \Delta]$. Finally, if $g$ is a one channel ground station, update $Q_g \leftarrow Q_g \cap [S_j, S_j + p_j]$ and $Q_g^1 \leftarrow Q_g^1 \cap [S_j, S_j + p_j]$ (recall that two channel ground stations are never limiting the number of simultaneous downlinks).

5. Suppose that $j$ is to be downlinked to a full-power station $g \in G_2$. Let $Q \leftarrow Q_g$ if $j$ requires normal reliability and $Q \leftarrow Q_g^1$ otherwise. Choose the smallest $k \in \{1, 2, \ldots, |F|\}$ and $l \in \{1, 2, \ldots, |Q|\}$ such that $|F_k \cap Q_l \cap [r_j, d_j| \geq p_j$. If no such $k$ and $l$ exist, leave the request $j$ unscheduled and proceed to Step 3. Otherwise schedule $j$: assign $S \leftarrow S \cup \{j\}$ and $S_j \leftarrow x$, where $H_k \cap Q_l \cap [r_j, d_j] = [x, y]$.

After scheduling $j$, update the indicator sets as follows: $F \leftarrow F \cap [S_j, S_j + p_j]$ and $H \leftarrow H \cap [S_j - \Delta, S_j + p_j + \Delta]$. Note that it is not necessary to update the indicator set $A$ since no downlink can happen if neither $H$ nor $F$ is available.

6. Update $V^{**}$ to $V^{**} \setminus \{j\}$ and, if $|V^{**}| > 0$, proceed to Step 3.

7. If either $i \in S$ and $j \notin S$, or $i \notin S$ and $j \in S$ for some pair $(i, j)$ of dual requests, remove both $i$ and $j$ from $V^*$ and proceed to Step 2.

8. Assign the downlinks to the particular ground station channels and satellite antennas.

Our updating scheme of the interval sets $A$, $H$ and $F$ ensures that no antenna conflict arises and all the downlinks happen within the planning horizon. By subtracting $[S_j - \Delta, S_j + p_j + \Delta]$ from $F$ for every half-power downlink request $j$, we guarantee that no full-power downlink can happen within $\Delta$ units of the downlink $j$. Similarly, no half-power downlink can happen within $\Delta$ units of a full-power downlink $j$. Also, the updating of the indicator sets $Q_g$ and $Q_g^1$

---

6In fact, it is enough to roll back the generation process until none of the dual requests $(i, j)$ is scheduled, exclude $i$ and $j$ from $V^*$ and proceed as normally. While being potentially beneficial for the performance of the generator, this heuristic would not improve the worst time complexity of the algorithm, though it would complicate the implementation (note that rolling back would require some form of restoration of the indicator sets). Considering that, in our instances, the number of dual requests was low, we decided to restart the generation procedure for each dual request violation.
guarantees that no channel conflicts occur and the downlinks obey visibility mask constraints. The pre-processing of data assures that there is a gap of at least $\delta$ units between two consecutive downlinks. Finally, dual requests constraint is satisfied due to Step 7 of the algorithm. Thus, the schedule generated is feasible.

Let us now analyse the complexity of the algorithm. The primary operations in each iteration are: (1) to find the smallest $k$ and $l$ to satisfy certain condition, and (2) to update the indicator sets $A$, $H$, $F$, $Q_g$ and $Q_g^1$. Note that the size of the indicator set $Q_g$ (and $Q_g^1$) for some $g \in G$ may increase but it is bounded by $O(n_g)$, where $n_g$ is the number of requests to be scheduled to the station $g$. Similarly, the sizes of the indicator sets $A$, $H$ and $F$ are limited by $O(n)$. Thus, operation (1) for these sets can be performed in $O(n)$ time by simultaneous scanning of the sets. Operation (2) can also be performed in $O(n)$ time. Indeed, we only need to update a fixed number of indicator sets, and updating each of them takes $O(n)$ time (for a downlink to a half-power setting ground station, manipulation with $X$ also requires only $O(n)$ time). The number of iterations is at most $O(n^2)$ and, thus, the complexity of the algorithm is $O(n^3)$.

6. Real-World Problem Instances

The algorithms presented in this paper have been tested on real data — the RADARSAT-2 problem instances: (1) 10 low-density instances (LD1—LD10) collected in Autumn 2011, each containing approximately 100 requests per planning horizon (24 hours); and (2) 10 high-density instances (HD1—HD10) collected in August 2011, each containing approximately 300 requests. There are at most ten ground stations involved in each instance. Each ground station is visible between 4 to 10 times from the satellite during the planning horizon, depending on the ground station location. Around 70% of the ground stations have one channel and 30% have two channels.

For the purpose of our experimental study, we were provided with the real downlink schedules implemented for each of the low- and high-density instances. The process currently in use for satellite mission planning includes two phases: (1) construction of the schedules with a priority rule-based algorithm and (2) human intervention. The system operator modifies the machine-generated solutions with the aim of scheduling some additional downlink requests and satisfying additional considerations known to the operator at that time. Such a solution may be imprecise. For example, a human operator may sometimes use his/her judgement to schedule a downlink request even if a downlink goes beyond the prescribed visibility mask by a very small amount of time. Such a solution would be infeasible as per our model as we use crisp visibility mask boundaries, as per satellite mission planning requirements.

---

Note that each of the $H \leftarrow H \ominus [\ell_k, u_k]$ for $[\ell_k, u_k] \in X$ operations needs only $O(1)$ time since all of these operations affect only one interval in $H$.

The details of that algorithm are unavailable to us.
We call such real schedules human-rescheduled (H-R) and in the following section we compare them against our algorithms.

7. Experimental Study

We implemented our algorithms in C++ and tested them on a PC with Intel Core i7-3820 CPU (3.60 GHz). The low-density and high-density instances discussed in Section 6 were used in this experimental study.

Following an empirical parameter tuning procedure, we set the time given to each local search run within GRASP to 1 second, the EC search depth to 10, the simulated annealing initial temperature $T_0$ to 0.001, the tabu list length $L$ to 4, and the tabu tolerance parameter $\epsilon$ to 0.01.

7.1. Computational Results

In this section we compare the performance of GRASP (named GR. in the tables below), Ejection Chain (EC), Simulated Annealing (SA) and Tabu Search (TS) algorithms (see Section 4) to the H-R schedules (see Section 6). The computational results for low-density and high-density instances are reported in Tables 1 and 2, respectively. In our experiments, all our algorithms were given equal time for fair comparison. Since EC is our only algorithm that does not have an explicit setting for the running time, we gave each of GRASP, SA and TS algorithms as much time as EC needed to terminate for each particular instance.

The columns of Tables 1 and 2 are as follows (from left to right): the instance name, the number of downlink requests $|V|$, the number of urgent requests, the time given to each of our algorithms, the number of unscheduled (urgent/total) requests for each of the algorithms, the average tardiness of the urgent requests for each of the algorithms, and the overall average tardiness for each of the algorithms. The tardiness of a request $j$ is measured as $S_j - \tau_j$, where $\tau_j$ is the earliest time $j$ can be downlinked subject to no other downlinks are scheduled.

It follows from the results of our computational experiments that the low-density instances are relatively easy to solve. Observe that each of our algorithms (GRASP, EC, SA and TS) scheduled all the requests and achieved 0.0 seconds urgent request tardiness for each instance. In terms of the overall average tardiness, the SA algorithm is the winner.

The high-density instances are much harder to solve. Each of the algorithms left several requests unscheduled for each of the instances. In terms of urgent requests, all our algorithms performed very similarly. In terms of overall performance, SA and EC were the leaders, followed by GRASP.

Our algorithms clearly outperformed the H-R solutions. For the low-density instances, the H-R solutions left, on average, one unscheduled request while all of our algorithms managed to schedule all the requests. For the high-density instances, the H-R solutions left, on average, 98.2 requests unscheduled, which is more than twice compared to any of our algorithms. Our best algorithms also significantly decreased the average tardiness (both overall and for urgent requests) compared to the H-R solutions.
| Inst. | | Urg. Time | | H-R | GR. | EC | SA | TS | | H-R | GR. | EC | SA | TS | H-R | GR. | EC | SA | TS |
|-------|-------|----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| LD1   | 110   | 31      | 1.8 | 0 / 1 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 762 | 133 | 135 | 135 | 271 |
| LD2   | 108   | 34      | 2.6 | 0 / 1 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 0.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1001 | 234 | 229 | 271 | 393 |
| LD3   | 115   | 29      | 2.8 | 0 / 1 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 0.7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 881 | 111 | 94 | 149 | 159 |
| LD4   | 105   | 25      | 2.2 | 0 / 1 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 0.7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1613 | 459 | 513 | 292 | 680 |
| LD5   | 104   | 32      | 1.5 | 0 / 2 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 701 | 184 | 184 | 184 | 186 |
| LD6   | 105   | 33      | 0.7 | 0 / 1 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 1.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 202 | 119 | 119 | 119 | 135 |
| LD7   | 90    | 30      | 1.0 | 0 / 1 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 3.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 379 | 80 | 79 | 80 | 97 |
| LD8   | 108   | 31      | 1.8 | 0 / 1 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 4.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 234 | 105 | 102 | 98 | 118 |
| LD9   | 110   | 28      | 1.5 | 0 / 1 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 11.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 975 | 238 | 237 | 238 | 278 |
| LD10  | 134   | 27      | 4.9 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 564 | 64 | 65 | 66 | 199 |
| Average | 108.9 | 30.0 | 2.1 | 0 / 1 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 2.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 731.1 | 172.8 | 175.9 | 157.7 | 251.6 |

Table 1: Comparison of the heuristic algorithms for the low-density instances.

| Inst. | | Urg. Time | | H-R | GR. | EC | SA | TS | | H-R | GR. | EC | SA | TS | H-R | GR. | EC | SA | TS |
|-------|-------|----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| HD1   | 211   | 34      | 36.4 | 0 / 53 | 0 / 19 | 0 / 0 | 0 / 18 | 0 / 2 | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2013 | 629 | 457 | 422 | 630 |
| HD2   | 301   | 35      | 155.6 | 0 / 123 | 0 / 66 | 0 / 67 | 0 / 64 | 0 / 65 | 0.6 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3788 | 3123 | 2725 | 3258 | 5446 |
| HD3   | 324   | 42      | 538.2 | 0 / 120 | 0 / 51 | 0 / 48 | 0 / 47 | 0 / 50 | 0.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2329 | 2148 | 2248 | 2184 | 2532 |
| HD4   | 294   | 41      | 337.5 | 2 / 116 | 0 / 56 | 0 / 52 | 0 / 53 | 0 / 60 | 0.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2025 | 968 | 1248 | 1261 | 1606 |
| HD5   | 260   | 36      | 161.9 | 0 / 85 | 0 / 49 | 0 / 49 | 0 / 49 | 0 / 48 | 0.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2621 | 1095 | 1021 | 1024 | 1623 |
| HD6   | 356   | 46      | 375.8 | 2 / 130 | 1 / 61 | 1 / 58 | 1 / 70 | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2555 | 2454 | 1563 | 1809 | 4751 |
| HD7   | 259   | 38      | 113.1 | 3 / 87 | 0 / 31 | 0 / 30 | 0 / 30 | 0 / 30 | 92.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2448 | 2025 | 1867 | 1924 | 2557 |
| HD8   | 304   | 54      | 218.0 | 1 / 98 | 0 / 36 | 0 / 33 | 0 / 33 | 0 / 36 | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2230 | 2005 | 1338 | 1116 | 2348 |
| HD9   | 278   | 39      | 163.0 | 3 / 90 | 0 / 37 | 0 / 36 | 0 / 37 | 0 / 37 | 0.5 | 0.6 | 0.5 | 0.5 | 0.5 | 0.5 | 2463 | 1087 | 904 | 808 | 2139 |
| HD10  | 230   | 25      | 153.7 | 0 / 80 | 0 / 28 | 0 / 28 | 0 / 28 | 0 / 30 | 0.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2880 | 2068 | 2096 | 2075 | 2201 |
| Average | 281.7 | 39.0 | 223.3 | 1.1 / 98.2 | 0.1 / 43.8 | 0.1 / 42.3 | 0.1 / 41.7 | 0.1 / 45.0 | 9.7 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 2535.1 | 1760.3 | 1546.6 | 1587.9 | 2583.2 |

Table 2: Comparison of the heuristic algorithms for the high-density instances.
It is worth noting that our construction heuristic described in Section 4.1 also outperformed the H-R solutions. For example, for the high-density instances, it, on average, left only 56.9 requests unscheduled. Compare it to the 98.2 unscheduled requests in the H-R solutions. However, each of our meta-heuristics significantly improved the results of the construction heuristic, leaving only 41–45 requests unscheduled.

7.2. Behaviour of the Algorithms

In this section we test the performance of our algorithms given more time. The GRASP, SA and TS algorithms have explicit parameters to adjust their running time. The EC algorithm has only one parameter, the search depth, that might affect the running time of EC. In this experiment, we ran GRASP, SA and TS given 4, 8, 16, . . . , 8192 seconds and EC with depth = 2, 5, 10, 20, 50. The results for each algorithm and each setting were averaged over all the high-density instances.

The results of the experiment are reported in Figure 2. An important observation is that neither the running time nor the solution quality of the EC algorithm notably depend on the value of parameter depth. The other algorithms’ quality significantly improves when given more time. TS shows relatively poor performance when given little time which, however, rapidly improves with the
increase of the running time. GRASP and SA are less sensitive to the given time. The winning algorithm is clearly SA: it outperforms other algorithms on the whole range of given times, producing reasonable solutions in just a few seconds and being able to gradually improve the solution quality.

7.3. Parameter $\alpha$

Recall that the objective function for the SIDSP uses a parameter $\alpha$ that represents the importance of tardiness minimisation compared to the importance of scheduling as many requests as possible. The value of $\alpha$ is irrelevant as long as all the requests are scheduled (like in our solutions of the low-density instances). However, if the problem is over-subscribed, the parameter $\alpha$ controls the trade-off between the number of scheduled requests and their tardiness. Figure 3 shows how the performance of the SA algorithm depends on $\alpha$.

![Figure 3: Demonstration of how the value of $\alpha$ influences the trade-off between average tardiness and the number of unscheduled requests. The number on the right of each point is the value of $\alpha$. Each point was obtained by averaging the results of 30-second runs of the SA algorithm for 10 different seed values and all the high-density instances.](image)

As expected, increasing the value of $\alpha$ puts emphasis on tardiness minimisation at the cost of scheduling less requests. Nevertheless, the range of solutions produced for different values of $\alpha$ is relatively small. Hence, the results obtained in this paper for $\alpha = 0.5$ are expected to hold for different values of $\alpha$.

It is also interesting to note that the solutions obtained for $\alpha = 0$ are, on average, dominated by the solutions obtained for $\alpha = 0.01$. We link it to the
changing fitness landscape. Observe that, for $\alpha = 0$, the objective value of a solution depends only on the set $S$ but not the times $S_j$. Hence, many distinct solutions sharing the same set $S$ cannot be distinguished by the solver. That creates the so-called high neutrality of the fitness landscape that is known to reduce the performance of optimisation heuristics \[21\].

8. Conclusions

In this paper, we formalised the satellite downlink scheduling problem and proposed a flexible solution approach separating the details of the problem-specific constraints from the optimisation mechanism. That significantly simplified the design and implementation of optimisation meta-heuristics. We tested several standard search techniques, including GRASP, Ejection Chain, Simulated Annealing and Tabu Search, and chose Simulated Annealing as the most efficient algorithm.

Our heuristic achieved very promising results comparing to the solutions obtained by the currently implemented method. For situations and problem instances where downlink scheduling was oversubscribed, the number of unsuccessful downlink requests was reduced twice. Although many of unsuccessful downlinks could have been 'background' acquisitions, the increase in the downlink throughput was significant and could be used for generating additional imagery products, increasing the number of customers, and thus increasing the company’s profit. Moreover, our algorithms produced solutions with lower, on average, total tardiness.

This research shows how a complicated model can be combined with modern optimisation heuristic methods in such a way that the resulting system is easy to use and maintain. The proposed approach gives an opportunity to handle other variations of the problem as well as potential additional constraints without re-implementing the optimisation part, which is a vital requirement of industrial systems. Hence, our study provides an excellent example of the use of simple management science ideas to enhance the operational efficiency and cost savings of an important real life optimisation problem.

References

[1] R.K. Ahuja, O. Ergun, J.B. Orlin, and A.P. Punnen, A survey of very large scale neighborhood search techniques. *Discrete Applied Mathematics* 123 (2002) 75–102.

[2] R.K. Ahuja, O. Ergun, J.B. Orlin, and A.P. Punnen, Very Large Scale Neighborhood Search: Theory, Algorithms and Applications, *Approximation Algorithms and Metaheuristics*, T. Gonzalez (ed), CRC Press, 2007.

[3] D. Gamboa, C. Osterman, C. Rego and F. Glover, An Experimental Evaluation of Ejection Chain Algorithms for the Traveling Salesman Problem. Technical report, School of Business Administration, University of Mississippi, 2006.
[4] L. Barbulescu, J.P. Watson, L.D. Whitley, and A.E. Howe, Scheduling space-ground communications for the Air Force Satellite Control Network. *Journal of Scheduling* 7 (2004) 7–34.

[5] L. Barbulescu, L. D. Whitley, and A. E. Howe, Leap before you look: An effective strategy in an oversubscribed scheduling problem, 2004.

[6] J. F. Bard and S. Rojanasoonthon, A branch and bound algorithm for parallel machine scheduling with time windows and job priorities. *Naval Research Logistics* 53 (2006) 24–44.

[7] G. Beaumet, G. Verfaillie, and M.-C. Charmeau, Feasibility of autonomous decision making on board an agile earth-observing satellite. *Computational Intelligence* 27 (2011) 123–139.

[8] T. Benoist and B. Rottenbourde, Upper bounds for revenue maximization in a satellite scheduling problem. *4OR* 2 (2004) 235–249.

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

[10] N. Bianchessi and G. Righini, Planning and scheduling algorithms for the COSMO-SkyMed constellation. *Aerospace Science and Technology* 12 (2008) 535–544.

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

[12] E.L. Demeulemeester and W. S. Herroelen, *Project Scheduling: A Research Handbook*, Kluwer Academic Publishers (2002).

[13] V. Gabrel and C. Murat, Mathematical programming for Earth observation satellite mission planning. In *Operations research in space and air*, Boston: Kluwer Academic (Chap. 7) 2003.

[14] A. Globus, J. Crawford, J. Lohn, and A. Pryor, A comparison of techniques for scheduling earth observing satellites. In *Proceedings of the sixteenth innovative applications of Artificial Intelligence Conference* (IAAA-04), San Jose, California, 2004.

[15] F. Glover, Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems. *Discrete Applied Mathematics* 65 (1996) 223–253.

[16] G. Gutin and D. Karapetyan, Greedy Like Algorithms for the Traveling Salesman and Multidimensional Assignment Problems. In *Advances in Greedy Algorithms* (Chap. 16), Witold Bednorz (editor) (2008) 291–304.
[17] S. Hartmann and D. Briskorn, A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* 207 (2010), 1–14.

[18] D. Karapetyan, S. Mitrovic-Minic, K.T. Malladi and A.P. Punnen, The satellite downlink scheduling problem: A case study of RADARSAT-2.. To appear in *Optimum Decision Making: Case studies of realistic applications*, K.G. Murty (editor), Springer, 2014.

[19] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research* 90(2) (1996), 320–333.

[20] F. Marinelli, S. Nocella, F. Rossi and S. Smriglio, A Lagrangian heuristic for satellite range scheduling with resource constraints. *Computers & Operations Research* 38 (11) (2011) 1572–1583.

[21] M. Marmion, C. Dhaenens, L. Jourdan, A. Liefooghe, and S. Verel, On the Neutrality of Flowshop Scheduling Fitness Landscapes, in *Proceedings of the 5th International Conference on Learning and Intelligent Optimization* (LION’05), Rome, Italy (2011), 238–252.

[22] R. Martí, J.M. Moreno-Vega, A. Duarte, Advanced Multi-start Methods, in *Handbook of Metaheuristics* 146, International Series in Operations Research & Management Science, Springer (2010) 265–281.

[23] S. Rojanasoonthon and J. Bard, A GRASP for parallel machine scheduling with time windows. *INFORMS Journal on Computing* 17 (2005) 32–51.

[24] B. Sun, W. Wang, X. Xie, and Q. Qin, Satellite mission scheduling based on genetic algorithm. *Kybernetes* 39 (2010) 1255–1261.

[25] M. Vasquez and J.-K. Hao, A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Computational Optimization and Applications* 7 (2003) 87–103.

[26] G. Verfaillie, C. Pralet, and M. Lemaitre, How to model planning and scheduling problems using constraint networks on timelines. *Knowledge Engineering Review* 25 (2010) 319–336.

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

[28] N. Zufferey, P. Amstutz, and P. Giaccari, Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling* 11 (2008) 263–277.