Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 854–862
computers & operations research
www.elsevier.com/locate/cor

Solving the swath segment selection problem through Lagrangean relaxation

Roberto Cordone, Federico Gandellini, Giovanni Righini*

Università degli Studi di Milano, Dipartimento di Tecnologie dell'Informazione, Via Bramante 65, 26013 Crema, Italy

Available online 6 June 2006

Abstract

The swath segment selection problem (*SSSP*) is an $\mathcal{NP}$-hard combinatorial optimization problem arising in the context of planning and scheduling satellite operations. It was defined by Muraoka et al. [ASTER observation scheduling algorithm. In: Proceedings of SpaceOps 1998, Tokyo, Japan, 1998] and Knight and Smith [Optimal nadir observation scheduling. In: Proceedings of the fourth international workshop on planning and scheduling for space (IWPSS 2004), Darmstadt, Germany, 2004], who respectively proposed a greedy algorithm, named ASTER, and a branch-and-bound algorithm based on a network flow relaxation. Here we tackle the problem with more advanced mathematical programming tools: using a Lagrangean relaxation, coupled with a Lagrangean heuristic and subgradient optimization, we solve in one hour instances with up to 500 000 swath segments within 0.4% of the optimum. The algorithm also proves experimentally superior to commercial *MIP* solvers in computing heuristic solutions.

Keywords: Earth observation satellites; Lagrangean relaxation; Combinatorial optimization

1. Introduction

The development of aerospatial programs for the observation of the Earth has opened an interesting field of application for OR techniques. A typical optimization problem arising in this context is the satellite scheduling problem [1,2], which has been tackled with a number of different heuristic algorithms, see for instance [3–7].

Satellites for Earth observation can be equipped with many different instruments and may have different degrees of freedom in their movements. In this paper we consider those endowed with an instrument pointing in a fixed *Nadir* direction, that is downwards, and following quasi-polar orbits. During each semi-orbit, from pole to pole, these satellites observe a stripe of the Earth surface, named *swath*. When a target is too large to be observed in one shot, it is decomposed into pieces, belonging to different swaths and it is observed during different transits of the satellite. Different swaths can intersect each other, when they cover the same part of the Earth surface.

Following the problem definition given by Knight and Smith [2], we assume that a set of swaths and a set of target areas are given. Due to the quasi-polar orbit, swaths are partitioned into two subsets, corresponding to descending semi-orbits (from North-East to South-West) and ascending semi-orbits (from South-East to North-West). The combination of the movement of the satellite and the rotation of the Earth implies that descending and ascending swaths intersect, whereas two swaths in the same direction overlap only near the poles, as shown in Fig. 1.

* Corresponding author. Tel.: +39 0373 898060; fax: +39 0737 898010.
E-mail address: righini@dti.unimi.it (G. Righini).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.04.005

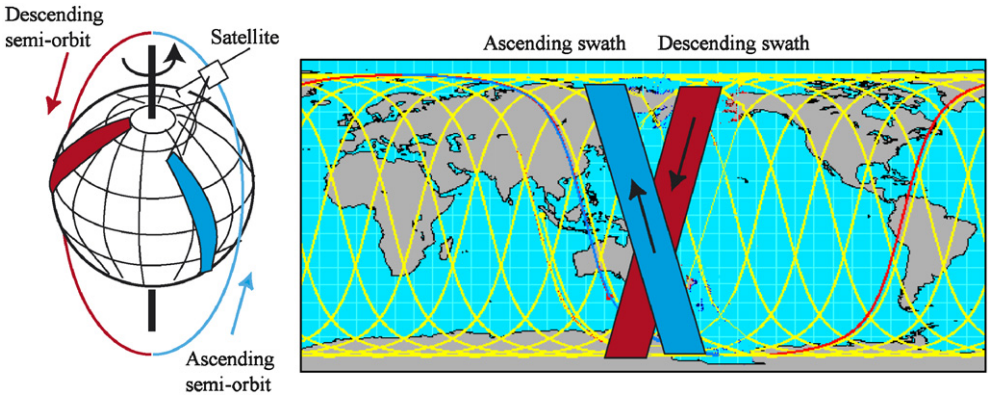

Fig. 1. A satellite in quasi-polar orbit, an ascending and a descending swath.

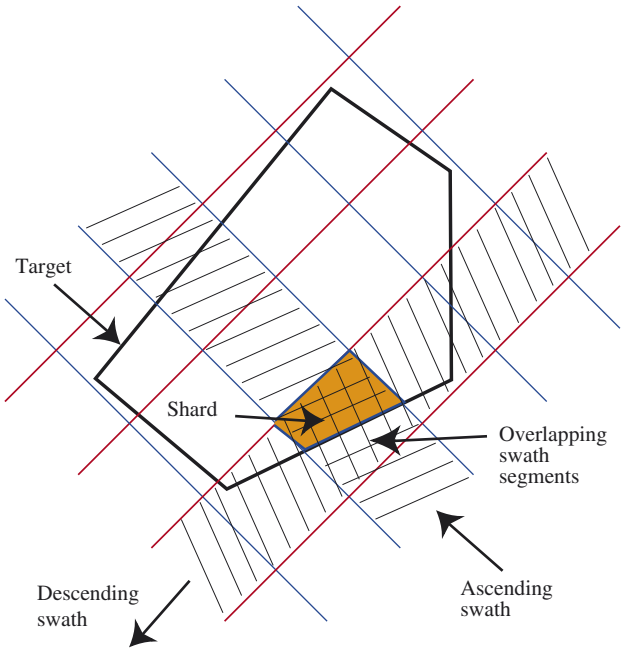

Fig. 2. An ascending and a descending *swath* (dashed) intersect, identifying two overlapping *swath segments* (squared); a *target* area (in bold) intersects the two segments in a *shard* (filled).

A swath is composed of a sequence of *swath segments*, corresponding to its intersections with the swaths in the opposite direction, and each portion of a target included in a segment is named *shard*. Therefore, a shard always belongs to an ascending and a descending segment, whereas a segment either includes a shard or none (see Fig. 2).

In our problem a given reward is associated with the observation of each shard. The amount of memory required to store an image on board, however, may be different if the shard is observed during an ascending or a descending transit. This is because the height and width of the two corresponding segments with opposite directions exchange and, while the height of the image stored equals the height of the shard, its width is fixed. Therefore, memory consumption is associated with segments instead of shards (see Fig. 3).

Since the images obtained are temporarily stored in a memory device on board and later downlinked to ground stations, memory is a critical resource. We assume that a satellite can downlink all stored data between two consecutive swaths, so that the memory capacity constraint can be enforced on each swath independently of the others.

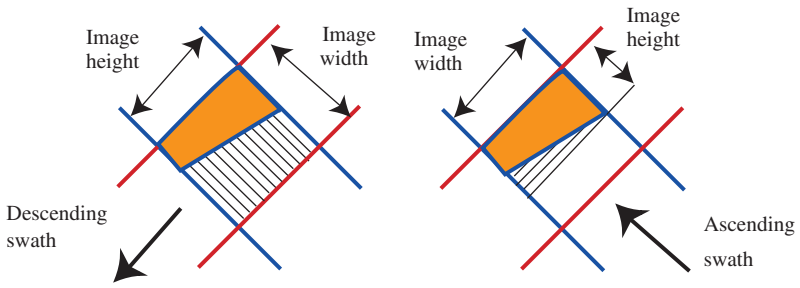

Fig. 3. Since the width of the image stored is fixed, observing the shard during the descending transit (on the left) implies acquiring a large uninteresting area (marked in dashed lines). Observing it during the ascending transit (on the right) strongly reduces the height, and therefore the area acquired.

The swath segment selection problem (*SSSP*) consists in selecting the shards to be observed and the appropriate corresponding segment to observe each of them (whether ascending or descending), in order to maximize the overall reward of the images collected, complying with the memory capacity constraints.

The *SSSP* is different from the already mentioned satellite scheduling problem, where requests can be issued dynamically and other operational constraints should be taken into account. However in dynamic environments it is common to build observation plans by first scheduling long term and high priority observations and then adding to the plan further observations in order to exploit the residual available resources. Therefore the observation of a large target area requires the solution of an optimization problem in which all the detailed operational constraints are not taken into account because they are not binding, but rather they are represented by a unique surrogate capacity constraint to be satisfied every time the satellite flies over the target area. Hence in the framework of satellite scheduling the *SSSP* considered here well represents the first "high level" phase to be followed by more detailed "low level" planning decisions.

At the best of our knowledge the only algorithmic solutions to the *SSSP* available in the literature are the *ASTER* algorithm due to Muraoka [1] and a branch-and-bound algorithm due to Knight and Smith [2]. The former is a greedy constructive deterministic algorithm: it is very fast but the quality of the solutions produced is largely suboptimal. The branch-and-bound algorithm of Knight and Smith consistently obtains solutions whose reward is twice as large. This last algorithm is a depth-first-search branch-and-bound, where the dual (upper) bound is computed by solving to optimality a relaxation of the *SSSP*, transformed into a max-flow problem on a special layered graph. This reformulation has two major drawbacks: first, it assumes that the rewards scale linearly with the size of the corresponding shards, which is not always the case, as the authors admit; second, the upper bound provided by the network flow reformulation is loose, because it corresponds to a linear relaxation of the *SSSP*.

In this paper we give a mathematical model of the *SSSP* and we show that it is equivalent to a set of binary knapsack problems, one for each swath, coupled by constraints due to the common shards. This suggests to apply Lagrangean relaxation to the coupling constraints in order to achieve an upper bound. The Lagrangean subproblem does not possess the integrality property, because it decomposes into a number of independent binary knapsack subproblems, and therefore the upper bound dominates that provided by the linear programming relaxation. Moreover a Lagrangean heuristic easily provides feasible solutions and corresponding lower bounds during the search. We show computational results on random instances with a number of segments ranging from 200 to 500 000, with different reward, memory consumption and downlink capacity. We compare the performances of our algorithm with those of *ILOG CPLEX* 8.0; the results prove the superiority of our approach with respect to the quality of both upper and lower bounds and to the computational time required.

In Section 2 we outline the mathematical formulation of the *SSSP* and we discuss its complexity; in Section 3 we describe our algorithm; in Section 4 we present our experimental results.

2. Formulation and Lagrangean relaxation

Knight and Smith [2] compared their branch-and-bound algorithm against an IP solver, without explicitly defining the formulation used. Hereafter, we present a mathematical model of the *SSSP*.

We are given a set $\mathscr{S}$ of swaths, a set $\mathscr{H}$ of shards and a set $\mathscr{N}_i$ of observable swath segments in each swath $i \in \mathscr{S}$. Let $\mathscr{N} = \cup_{i\in\mathscr{S}} \mathscr{N}_i$ be the set of all segments. For each shard $k \in \mathscr{H}$, we indicate by $\mathscr{G}_k$ the subset of segments, that is observation opportunities, covering the shard. We are given a reward r_k for each shard $k \in \mathscr{H}$, a memory consumption a_j for each segment $j \in \mathscr{N}$ and a capacity b_i for each swath $i \in \mathscr{S}$. For each segment $j \in \mathscr{N}$ we introduce a binary variable x_j, which is equal to 1 if an observation is done along the segment and to 0 otherwise. In case the instrument observation capability depends on the illumination conditions, we may take this into account by setting to zero the x_j variables corresponding to unfeasible observations.

Two swaths have either zero or one shard in common, and $\mathscr{G}_k$ contains exactly two segments for each shard $k \in \mathscr{H}$ (one ascending and one descending). Our model is more general than the one considered by Knight and Smith [2], where all swaths have the same capacity b and the memory consumption a_j is the same for all segments covering the same shard.

$$SSSP: \max \quad z = \sum_{k\in\mathscr{H}} \left(r_k \sum_{j\in\mathscr{G}_k} x_j \right) \tag{1}$$

$$\text{s.t.} \quad \sum_{j\in\mathscr{N}_i} a_j x_j \leqslant b_i \quad \forall i \in \mathscr{S}, \tag{2}$$

$$\sum_{j\in\mathscr{G}_k} x_j \leqslant 1 \quad \forall k \in \mathscr{H}, \tag{3}$$

$$x_j \in \{0,1\} \quad \forall j \in \mathscr{N}. \tag{4}$$

The objective function (1) asks for the maximization of the overall reward given by the shards observed; constraints (2) impose the capacity restriction on the observations in each swath $i \in \mathscr{S}$; constraints (3) forbid multiple observations of a same shard $k \in \mathscr{H}$ in different swaths. The *SSSP* is obviously $\mathscr{NP}$-hard because it contains the binary knapsack problem as a special case.

A Lagrangean relaxation of the *SSSP* is the following:

$$LR(\lambda): \max \quad z^{\mathrm{LR}}(\lambda) = \sum_{k\in\mathscr{H}} \left(r_k \sum_{j\in\mathscr{G}_k} x_j \right) + \sum_{k\in\mathscr{H}} \lambda_k \left(1 - \sum_{j\in\mathscr{G}_k} x_j \right) \tag{5}$$

$$\text{s.t.} \quad \sum_{j\in\mathscr{N}_i} a_j x_j \leqslant b_i \quad \forall i \in \mathscr{S}, \tag{6}$$

$$x_j \in \{0,1\} \quad \forall j \in \mathscr{N}, \tag{7}$$

where λ_k is the Lagrangean multiplier associated with shard $k \in \mathscr{H}$. Since each segment $j \in \mathscr{N}$ belongs to a single swath $i \in \mathscr{S}$, problem $LR(\lambda)$ decomposes into the following $|\mathscr{S}|$ independent binary knapsack subproblems:

$$LR_i(\lambda): \max \quad z_i^{\mathrm{LR}}(\lambda) = \sum_{j\in\mathscr{N}_i} (r_{k_j} - \lambda_{k_j}) x_j \tag{8}$$

$$\text{s.t.} \quad \sum_{j\in\mathscr{N}_i} a_j x_j \leqslant b_i, \tag{9}$$

$$x_j \in \{0,1\} \quad \forall j \in \mathscr{N}_i, \tag{10}$$

where k_j is the index of the unique shard covered by segment j and $z^{\mathrm{LR}}(\lambda) = \sum_{i\in\mathscr{S}} z_i^{\mathrm{LR}}(\lambda) + \sum_{k\in\mathscr{H}} \lambda_k$.

The binary knapsack problem is $\mathscr{NP}$-hard (see [8] for a classical reference) and hence it does not have the integrality property. Therefore the upper bound provided by the optimal solution of *LR* dominates the upper bound given by the linear relaxation of *SSSP*. Hence this approach allows to make the best of the existing optimization algorithms for the binary knapsack problem to achieve a good upper bound for the *SSSP*.

3. The algorithm

Our algorithm is a branch-and-bound relying on the Lagrangean relaxation outlined in the previous section. At each node of the search tree we iteratively solve problem $LR(\lambda)$ with different values of λ, updated by subgradient

optimization. During the process, heuristic solutions are computed from the solutions of the current Lagrangean subproblem. The best Lagrangean multipliers found during the process also guide the branching strategy. In this section we describe the components of our algorithm.

Upper bounds. For each choice of λ we solve $|\mathscr{S}|$ binary knapsack problem instances to optimality with a modified version of the MINKNAP algorithm [9]; the Lagrangean subproblems $LR_i(\lambda)$, in fact, have real-valued rewards, while MINKNAP deals with knapsack instances whose rewards have integer values. To overcome this limitation, we employed the MINKNAP code generalized by Ceselli [10].

For any $\lambda \geqslant 0$, the optimal solution $z^{\mathrm{LR}}(\lambda)$ provides an upper bound to the *SSSP*. To achieve the tightest possible upper bound, one has to solve the Lagrangean dual problem

$$\min_{\lambda \geqslant 0} \left\{ \max_{x} \{ z^{\mathrm{LR}}(\lambda) \} \right\}$$

which can be done via subgradient optimization. The updated rule used by the subgradient optimization algorithm [11] is

$$\lambda_k^{(t+1)} = \min \left\{ \max \left\{ \lambda_k^{(t)} + \pi^{(t)} s_k(\lambda^{(t)}) \frac{|z(\lambda^{(t)}) - LB|}{\|s_k(\lambda^{(t)})\|^2}, 0 \right\}, r_k \right\}, \tag{11}$$

where $s_k(\lambda^{(t)}) = \sum_{j \in \mathscr{G}_k} x_j - 1$ is the amount by which constraint (3) is violated for shard $k \in \mathscr{H}$ at iteration t, LB is the best incumbent lower bound, that is the value of the best known feasible solution found so far, and $\pi^{(t)}$ is a step coefficient decreasing with t. Besides being constrained to be non-negative, the Lagrangean multipliers are also constrained to be not greater than the corresponding rewards r_k, because larger values would yield dominated relaxations.

The subgradient optimization algorithm stops after a predefined number of iterations or when $\lfloor z^{\mathrm{LR}}(\lambda^{(t)}) \rfloor \leqslant LB$, which means that LB cannot be improved since the rewards are integer. The value $\min_t \{ \lfloor z^{\mathrm{LR}}(\lambda^{(t)}) \rfloor \}$, that is the minimum upper bound found by the subgradient optimization algorithm, is retained as the final upper bound for the current node of the search tree.

In our experimental campaign, the subgradient optimization subroutine has been run for 1000 iterations at the root node, and for 100 iterations at the other nodes; the starting value for all multipliers has been set to zero, while the initial step coefficient was set to $\pi^{(0)} = 2.0$ and it was multiplied by 0.99 at each iteration.

Lower bounds. At each subgradient iteration we also compute feasible solutions to the *SSSP*. Since constraints (3) have been relaxed, the current optimal solution of the Lagrangean relaxation may correspond to observe some shards more than once. Heuristic feasible solutions can be obtained as follows: all binary variables corresponding to the shards observed exactly once in the current optimal solution of $LR(\lambda)$ are fixed to their current values; then a binary knapsack problem instance $LR_i(\lambda)$ is solved for each swath $i \in \mathscr{S}$, where some of the variables have been fixed as above. To avoid observing a shard more than once, the knapsack instances are solved sequentially in a suitable order, so that if a segment covering a shard is selected, then the other segment covering the same shard is forbidden.

The solution obviously depends on the order in which the swaths are considered. For the sake of simplicity, we have considered only two orders: either the knapsack problem instances corresponding to ascending swaths are solved before those corresponding to descending swaths or the other way around. Therefore we obtain two heuristic solutions corresponding to these two orders and we check whether any of the two improves the current best incumbent solution, whose value is LB.

In our experimental campaign, the two heuristic solutions are computed every time the current upper bound improves.

Branching strategy. The branching operation in the branch-and-bound algorithm of Knight and Smith [2] generates as many children nodes as the number of segments in the problem. This produces a search tree of very large breadth. To avoid this drawback, we use ternary branching; given the optimal solution of the Lagrangean relaxation corresponding to the best upper bound at the father node, we select one of the shards, say $\overline{k}$, and we generate three subproblems: in the first one, shard $\overline{k}$ is observed in the ascending swath; in the second one, shard $\overline{k}$ is observed in the descending swath; in the third one, shard $\overline{k}$ is not observed. This implies fixing two binary variables in each of the three children nodes.

The choice of the branching shard $\overline{k}$ is based on the Lagrangean solution and on the Lagrangean rewards $(r_k - \lambda_k)$ corresponding to the best known multipliers. If any of the shards is observed more than once in the Lagrangean

solution, we choose the one with the largest Lagrangean reward. If none is, we choose the unobserved shard with the largest Lagrangean reward. If all shards are observed exactly once, then the Lagrangean solution is feasible and it is also optimal for its node, since complementary slackness conditions hold; in this case no branching is required.

Search strategy. The branching tree is visited according to a *best-first* search strategy: the node with the largest upper bound is considered first.

4. Experimental results

The algorithm proposed was implemented in *C* language and tested on a 1.6 GHz Pentium PC with 512 MB RAM and Linux operating system.

Data-sets. Since the test data of Knight and Smith [2] had not been made available to us, we generated random test instances in the following way. We considered a square target region covered by $S/2$ ascending and $S/2$ descending orthogonal swaths. Parallel swaths do not overlap, whereas each ascending swath intersects each descending swath on a square region. Therefore there are $S^2/4$ shards and $S^2/2$ segments. We generated problem instances of increasing size, ranging from $S/2 = 10$ to 100 by steps of 10 and from $S/2 = 100$ to 500 by steps of 100. The number of swath segments, therefore, ranges from 200 to 500 000 and the total number of size classes is 14.

The rewards associated to the shards have been randomly generated as integer numbers with uniform probability distribution in two different ranges: [1, 100] and [51, 100]. The memory consumption associated to the two swath segments covering a same shard can be either the same for both segments or independently generated. In either cases, it is represented by one or two random integers in [1, 100] or in [51, 100]. The combinations of these three parameters yield $2 \times 2 \times 2 = 8$ classes of instances.

The algorithm can manage any value for the capacity b_i of each swath $i \in \mathcal{S}$. However our experiments proved that the value of the capacity is not a relevant feature of the instances. Therefore for the sake of simplicity we assigned an identical capacity b to all swaths. This capacity is defined as a fraction α of the total potential memory consumption along the swath with the lowest demand: $b = \alpha \min_{i \in \mathcal{S}} \{\sum_{j \in \mathcal{N}_i} a_j\}$. The scaling parameter α, which modulates the tightness of the capacity constraint, assumes three different values in our tests: 20%, 30% and 40%. Hence we have three classes of instances according to the capacity.

Since we have one instance for each combination of size (14 classes), rewards and consumptions (8 classes) and capacity (3 classes), our test set is made of $14 \times 8 \times 3 = 336$ instances.

Comparison with CPLEX. All computational results are presented by comparing the performance of the algorithm here proposed and that of ILOG CPLEX 8.0 applied to formulation (1)–(4). To enhance the performance of CPLEX, the following parameter setting has been used; the absolute gap below which a solution is declared optimal has been set to 1.0 (because all data are integer), a built-in heuristic rounding procedure has been called at the root node and all general purpose cuts have been enabled both at the root node and in the branching nodes. This implies that, though the linear relaxation of formulation (1)–(4) is dominated by the Lagrangean relaxation, the gap obtained by CPLEX at the root node using valid inequalities can be lower than the gap obtained by our algorithm and this is actually the case for some of the instances. The linear relaxation is solved by the dual simplex algorithm, in order to exploit the fact that the formulation has less constraints than variables, and in order to obtain a valid upper bound at any moment during the computation. It must be noticed that this parameter setting is relevant, since the standard one yields significantly worse performances.

Table 1 presents the computational results on the 336 instances in one hour of computation. Each row reports average results on the 12 instances having the same size and the same relation between ascending and descending memory consumption; the upper half of the table refers to the instances in which the two memory consumptions are independently generated, while the lower half refers to the instances in which they are identical. The first column reports the size of the instances, the next four columns report the results of our algorithm, namely the average best known solution (z^*), the average upper bound (*Bound*), the average primal-dual gap (Δ) and the number of branching nodes (*#B*). The primal-dual gap is defined as $(Bound - z^*)/z^*$. The last four columns report the same data for CPLEX. The best values in each row are bolded.

The first remark which can be done is that the *SSSP* appears to be a rather hard problem, since the size of the instances which can always be solved exactly is $S/2 = 10$. However, the hardness of the problem strongly depends on the structure of the instance, since we observed solved instances with size up to $S/2 = 70$. The percentage gap is most

Table 1
Comparison between the SSSP solver and CPLEX in one hour of computation (average results for each size)

Size	SSSP solver				CPLEX			
	z^*	Bound	Δ (%)	#B	z^*	Bound	Δ (%)	#B
10	**4881**	**4881**	**0.00**	16	**4881**	**4881**	**0.00**	57
20	**20 633**	**20 655**	**0.09**	4856	20 620	20 660	0.16	18 809
30	**47 122**	**47 193**	**0.13**	4021	47 043	47 221	0.33	26 318
40	**84 617**	**84 792**	**0.19**	2326	84 433	84 847	0.50	38 894
50	**131 768**	**131 999**	**0.17**	1699	131 452	132 088	0.50	41 515
60	**191 308**	**191 673**	**0.18**	1320	190 916	191 786	0.48	39 195
70	**259 871**	**260 339**	**0.18**	1047	259 445	260 506	0.44	38 182
80	**342 310**	**342 958**	**0.19**	715	341 983	343 113	0.36	30 053
90	**433 966**	**434 735**	**0.18**	579	433 504	434 924	0.34	23 587
100	**534 394**	**535 321**	**0.18**	521	534 004	535 514	0.30	16 572
200	**2 155 259**	2 158 306	**0.15**	107	2 138 639	**2 158 208**	0.17	1469
300	**4 870 804**	4 877 656	0.15	47	4 870 298	**4 876 518**	**0.14**	1208
400	8 650 279	8 663 104	0.16	29	**8 650 545**	**8 660 235**	**0.12**	2308
500	13 530 228	13 550 970	0.17	17	**13 531 935**	**13 546 659**	**0.12**	926
10	**4453**	**4453**	**0.00**	330	4450	4454	0.06	42 078
20	**18 564**	**18 829**	**1.34**	7897	18 485	18 837	1.81	230 689
30	**42 394**	42 880	**1.17**	3955	42 173	**42 865**	1.71	190 907
40	**74 954**	75 554	**0.83**	2558	74 615	**75 523**	1.27	148 539
50	**118 315**	119 046	**0.64**	1827	117 882	**119 000**	1.01	142 465
60	**170 160**	170 957	**0.49**	1401	169 580	**170 903**	0.83	148 372
70	**232 296**	233 172	**0.39**	1121	231 741	**233 109**	0.64	118 733
80	**305 003**	305 981	**0.33**	915	304 354	**305 900**	0.54	80 899
90	**385 219**	386 281	**0.28**	752	384 537	**386 166**	0.45	51 861
100	**475 897**	477 050	**0.25**	626	475 173	**476 871**	0.39	28 222
200	**1 914 481**	1 917 779	**0.18**	176	1 911 904	**1 916 168**	0.24	207
300	**4 314 081**	4 321 814	**0.19**	73	4 305 733	**4 316 465**	0.28	0
400	**7 702 877**	7 717 979	**0.20**	41	7 687 658	**7 706 078**	0.26	0
500	**12 030 572**	12 055 997	**0.22**	23	12 007 501	**12 034 732**	0.24	0

of the time quite small and it tends to decrease as the size increases; all instances with $S/2 \geqslant 100$ have a gap smaller than 0.4%. The instances with identical memory consumptions during ascending and descending transits appear to be harder for both algorithms, though the effect is more marked for our solver than for CPLEX.

Our algorithm provides better heuristic results on almost all size classes. When considering the detailed results (not presented here for obvious reasons of space limit), our algorithm beats CPLEX in 288 cases out of 336, solving to optimality 56 instances against 49 solved by CPLEX. As far as the upper bound is concerned, our algorithm performs better than CPLEX when the memory consumptions during ascending and descending transits are independent, while it performs worse when they are identical. In detail, the upper bound provided by our algorithm is the best 139 times, while CPLEX wins 197 times. Finally, our primal-dual gap is smaller 249 times over 336. In practical cases, a mixed situation is to be expected; the shards inside the target region are likely to have identical consumption in both directions, whereas the shards on the borders of the target region are likely to have different consumptions.

The comparable quality of the linear relaxation upper bound and the Lagrangean relaxation upper bound is entirely due to the generation of all additional general-purpose cuts available in CPLEX, which substantially improve the linear relaxation. Preliminary experiments performed with CPLEX standard parameter setting, in which the cuts to be generated are chosen automatically, showed that CPLEX consistently provided worse upper bounds, as well as worse heuristic solutions.

The number of branching nodes generated by our algorithm is from one to three orders of magnitude inferior to that generated by CPLEX. Therefore, our algorithm is far more efficient from the point of view of memory requirements. For large instances, however, CPLEX does not generate branching nodes, since the time limit of one hour is insufficient to solve to optimality the linear relaxation of the problem.

Table 2
Comparison between the *SSSP* solver and CPLEX in one hour of computation (average results for each reward and memory class)

Memory	Reward	Independent a_j		Identical a_j	
		SSSP solver (%)	CPLEX (%)	*SSSP* solver (%)	CPLEX (%)
[1, 100]	[1, 100]	0.10	0.14	0.51	0.65
[1, 100]	[51, 100]	0.12	0.20	0.47	0.53
[51, 100]	[1, 100]	0.25	0.40	0.53	0.79
[51, 100]	[51, 100]	0.12	0.39	0.35	1.81

Table 3
Comparison between the *SSSP* solver and CPLEX in one hour of computation (average results for each capacity class)

Downlink capacity (%)	Independent a_j		Identical a_j	
	SSSP solver (%)	CPLEX (%)	*SSSP* solver (%)	CPLEX (%)
20	0.09	0.31	0.48	0.82
30	0.19	0.31	0.46	0.70
40	0.18	0.23	0.46	0.56

Dependency on the data. Table 2 shows the results obtained by our solver and by CPLEX from a different point of view; here they are averaged with respect to the memory and reward ranges, indicated in the first column. The following two columns report (respectively, for our solver and for CPLEX) the percentage gap on the instances in which the memory consumption during ascending and descending transits are independently generated. The last two columns provide the same results for the instances with identical memory consumption. The purpose is to evaluate how the performance of our algorithm depends on the structure of the specific instance class considered.

Intuitively we expected instances with rewards and memory consumptions in [51, 100] to be harder to solve, as it should be more difficult to distinguish between good and bad choices. For the same reason, instances with an identical memory consumption in the ascending and descending swath segments were expected to be harder. These expectations were confirmed, at least in part, by the outcome of the experiments. The comparison between the left and the right part of the table shows that the instances with identical consumptions in both directions are actually harder. The instances whose rewards and memory consumptions have been both generated in [51, 100] are harder for CPLEX, but this not so evident for our algorithm; while narrowing the memory consumption range seems to make the instances slightly harder, narrowing the reward range often makes them easier.

Finally, Table 3 concerns the same instances, but the results are averaged with respect to each value of the capacity parameter (first column). The instances with a loose capacity constraint are harder for our algorithm when the memory consumption in the two directions are independent (left part of the table), whereas no clear dependence is observed in the opposite case. CPLEX, on the contrary, seems to become more effective as the constraint is relaxed.

5. Conclusions

This paper presents an exact algorithm for the *SSSP*, a combinatorial optimization problem with relevant applications in planning and scheduling of Nadir observations. We have described a branch-and-bound algorithm based on the Lagrangean relaxation of the constraints that forbid multiple observations of the same shard, enriched with a Lagrangean heuristic. The experimental comparison proves the effectiveness of this approach: though only small instances can be solved to optimality, the algorithm outperforms the state-of-the-art *MIP* solver ILOG CPLEX 8.0 with respect to the quality of the heuristic solutions provided and the primal-dual gap obtained in the same computing time. The upper bounds provided by our algorithm are dominated by the ones provided by CPLEX when all general-purpose cuts are added to the initial formulation, whereas the opposite occurs when using CPLEX's standard parameter setting. The experimental results also prove the hardness of the problem; in spite of the impossibility to solve medium-sized

instances to optimality, our algorithm yields in one hour primal-dual gaps smaller than 0.4% on benchmarks instances with up to 500 000 swath segments.

The rewards associated to the shards can be figures of merit conventionally set by the users. However, they may as well represent the price at which the acquired images are sold to the end users. Owing to the very long operational life of Earth observation satellite systems, even a very small improvement in these gains may result into a significant additional revenue.

References

[1] Muraoka H, Cohen RH, Ohno T, Doi N. ASTER observation scheduling algorithm. In: Proceedings of SpaceOps 1998, Tokyo, Japan. 1998.
[2] Knight R, Smith B. Optimal nadir observation scheduling. In: Proceedings of the fourth international workshop on planning and scheduling for space (IWPSS 2004), Darmstadt, Germany. 2004.
[3] Wolfe WJ, Sorensen SE. Three scheduling algorithms applied to the earth observing systems domain. Management Science 2000;46(1): 148–68.
[4] Vasquez M, Hao JK. A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. Journal of Computational Optimization and Applications 2001;20:137–57.
[5] Vasquez M, Hao JK. Uppers bounds for the SPOT 5 daily photograph scheduling problem. Journal of Combinatorial Optimization 2003;7(1): 87–103.
[6] Lemaître M, Verfaillie G, Jouhaud F, Lachiver JM, Bataille N. Selecting and scheduling observations of agile satellites. Aerospace Science and Technology 2002;6:367–81.
[7] Bianchessi N, Piuri V, Righini G, Roveri AZM, Laneve G. An optimization approach to the planning of earth observing satellites. In: Proceedings of the fourth international workshop on planning and scheduling for space (IWPSS), Darmstadt. 2004.
[8] Martello S, Toth P. Knapsack problems: algorithms and computer implementations. New York: Wiley; 1990.
[9] Pisinger D. A minimal algorithm for the 0–1 knapsack problem. Operations Research 1997;45:758–67.
[10] Ceselli A. Two optimization algorithms for the capacitated p-median problem. 4OR 2003;1(4):319–40.
[11] Held M, Wolfe P, Crowder HP. Validation of subgradient optimization. Mathematical Programming 1974;6:62–88.