Journal of Combinatorial Optimization, 7, 87–103, 2003

Upper Bounds for the SPOT 5 Daily Photograph Scheduling Problem

MICHEL VASQUEZ vasquez@site-eerie.ema.fr
LGI2P-EMA, Parc Scientifique G. Besse, 30035 Nîmes Cedex 1, France

JIN-KAO HAO hao@info.univ-angers.fr
LERIA Université d'Angers, 2 bd Lavoisier, 49045 Angers Cedex 1, France

Received November 2, 2000; Revised March 30, 2001; Accepted October 11, 2001

Abstract. This paper introduces tight upper bounds for the daily photograph scheduling problem of earth observation satellites. These bounds, which were unavailable until now, allow us to assess the quality of the heuristic solutions obtained previously. These bounds are obtained with a partition-based approach following the "divide and pas conquer" principle. Dynamic programming and tabu search are conjointly used in this approach. We present also simplex-based linear programming relaxation and a relaxed knapsack approach for the problem.

Keywords: bounds, partition, constraint relaxation, tabu search, branch & bound

1. Introduction

The daily photograph scheduling problem (DPSP) is one of the key applications for an earth observation satellite such as Spot 5. The main purpose of the DPSP is to schedule a subset of photographs from a set of candidate photographs which will be effectively taken by the satellite. The resulting subset of photographs must satisfy a large number of imperative constraints of different types and at the same time maximize a given profit function. The profit function reflects several criteria such as client importance, demand urgency, meteorological forecasts and so on. The constraints include both physical constraints such as the recording capacity on board of the satellite and logic constraints such as non overlapping trials and meeting the minimal transition time between two successive trials on the same camera. This constrained maximization problem is also important and interesting from a complexity point of view. Indeed, it can be formulated as a generalized knapsack problem, which is known to be NP-hard.

So far, several methods have been proposed to tackle this problem. In Verfaillie et al. (1996), the authors introduced an exact algorithm called Pseudo Dynamic Search, which embodies a Branch & Bound technique within an iterative optimization process. This approach has been used to solve 20 benchmark instances presented in Bensana et al. (1999). For the 13 simple single-orbit instances, the approach managed to solve all of them to optimality. On the contrary, this method was unable to solve 6 of 7 multi-orbit instances within reasonable time. Indeed, the multi-orbit case is much more difficult than the single-orbit case due to the much larger size of the instances and the presence of a knapsack constraint.

Several heuristic approaches have also been developed for the DPSP. These heuristics include, among others, greedy procedures, algorithms based on simulated annealing and tabu search (Bensana et al., 1996; Vasquez and Hao, 2001). In particular, in Vasquez and Hao (2001), the authors presented an advanced tabu algorithm which finds the optimal solution for the single orbit instances and improves on the previously best known results for the multi-orbit instances. Notice that these heuristic solutions gives lower bounds for this maximization problem. Upper bounds are needed to assess the quality of heuristic solutions.

In Bensana et al. (1996), the authors mentioned linear programming (LP) relaxation for this problem, but they did not report computational results. In Gabrel (1999), the author experimented this approach using CPLEX 4.1 and reported continuous optimal values (COV) for the single-orbit instances. However, these upper bounds are largely above the discrete optimal values (DOV): the ratio (COV-DOV)/COV goes from 28% to 74%! No result was reported for the much larger multi-orbit instances. The author also tried a new integer linear programming formulation of the problem based on decomposition and column generation techniques (Gabrel, 1999). Using CPLEX 4.1, together with a column generation procedure, this approach produced better upper bounds for the single-orbit instances, reducing the above (COV-DOV)/COV ratio to the range of 0 to 20%. Once again however, no result was obtained for the multi-orbit instances, due to the huge number of columns that have to be managed during the resolution. Until now, there is no upper bound available for the multi-orbit instances of the daily photograph scheduling problem.

The study presented in this paper tries to fill this important gap. To achieve this goal, we experiment first with the most evident approach based on linear programming relaxation. We continue our investigation with dynamic programming applied to the problem where logic constraints are relaxed. As we see later, these approaches are unable to find tight upper bounds. We devise finaly a partition-based approach which gives much better bounds.

The paper is organized as follows. The DPSP problem and its "logic constrained" knapsack formulation are introduced in Section 2. The benchmark instances of the DPSP together with their best known results are presented in Section 3. In Sections 4 and 5 are presented LP relaxation approach and logic constraint relaxation approach. The partition-based approach is presented in Section 6. Upper bounds obtained with this approach are shown in Section 7. Conclusions are given in the last section.

2. SPOT photograph daily scheduling problem (DPSP)

We give now a brief review of the problem as well as a "logic-constrained" knapsack formulation presented in Vasquez and Hao (2001).

2.1. Problem definition

We are given the following problem components.

- A set $P = \{p_1, p_2, \ldots, p_n\}$ of candidate photographs, mono or stereo, which can be scheduled to be taken on the "next day" under appropriate conditions of the satellite trajectory and oblique viewing capability.

- A "profit" associated with each photograph p_i, which is the result of the aggregation of several criteria such as client importance, demand urgency, meteorological forecasts and so on.
- A "size" associated with each photograph p_i, which represents the amount of memory required to record p_i when it is taken.
- A set of possibilities associated with each photograph p_i in P corresponding to the different ways to take p_i: (1) for a mono p_i, there are three possibilities because a mono photograph can be taken by any of the three cameras (front, middle and rear) on the satellite and (2) for a stereo p_i, there is one single possibility because a stereo photograph requires simultaneously the front and the rear camera.
- A set of imperative constraints:

  1. *Binary constraint*: for some couples (photo, camera), it is forbidden to schedule simultaneously p_1 on the camera x and p_2 on y. This constraint is the first type of logical constraint.
  2. *Ternary constraint*: for some couples (photo, camera), it is forbidden to schedule simultaneously p_1 on the camera x, p_2 on y and p_3 on z. This constraint is the second type of logical constraint.
  3. *Capacity (or knapsack) constraint*: the sum of the sizes of the scheduled photos cannot exceed the recording capacity on board.

The DPSP is then to find a subset P' of P which maximizes the sum of the profits of the photographs in P' and satisfies all the logical and knapsack constraints. In practice, both the number of photographs in P and the number of logical constraints may be quite large (more than one thousand for P and tens of thousands for logical constraints).

2.2. *A "logic-constrained" knapsack formulation*

A first integer formulation consists in considering a photo as an integer variable and representing the three cameras by 1, 2 and 3. Thus, a variable for a mono photo has a value domain of $\{0, 1, 2, 3\}$ where the value 0 means the photo is not scheduled. Similarly, a variable for a stereo photo has a value domain of $\{0, 13\}$ where the value 13 represents the simultaneous use of the cameras 1 and 3. Constraints and objective function may be deduced accordingly. More details about this formulation is given in Bensana et al. (1996).

Another formulation is based on binary variables, each representing a couple (photo camera): $x_i = (p, c)$ takes the value 1 if the photo p is taken by the camera c, 0 otherwise. A schedule corresponds to the non-zero components of a binary vector. This formulation requires the introduction of a supplementary constraint, called *consistency constraint*, stating that a photo will not be scheduled more than once.

We give now the formal definition of this formulation (Vasquez and Hao, 2001). Let us define the following notations:

- m_1: the cardinality of the set P of candidate photos;
- $\mathrm{D}(p)$: the value domain of the photo p, $\mathrm{D}(p) = \{1, 2, 3\}$ if p is a mono photo, $\mathrm{D}(p) = \{13\}$ otherwise;

- $n = \sum_{p=1}^{p=m_1} |\mathrm{D}(p)|$: the number of binary variables of the problem;
- m_1 vectors $A_1, \ldots, A_{m_1} \in \{0, 1\}^n$: the non-zero components of each A_p correspond to all the possible couples (p, c) for the photo p with c being a camera;
- m_2 vectors $A_{m_1+1}, \ldots, A_{m_1+m_2} \in \{0, 1\}^n$: for each A_i, $m_1 + 1 \leq i \leq m_1 + m_2$, its 2 non-zero components correspond to the 2 couples (p_1, c_1), (p_2, c_2) involved in the $(i - m_1)^{th}$ binary constraint;
- m_3 vectors $A_{m_1+m_2+1}, \ldots, A_{m_1+m_2+m_3} \in \{0, 1\}^n$: for each A_i, $m_1 + m_2 + 1 \leq i \leq m_1 + m_2 + m_3$, its 2 non-zero components correspond to the couples (p_1, c_1), (p_2, c_2), (p_3, c_3) involved in the $(i - (m_1 + m_2))$th ternary constraint;
- $m = m_1 + m_2 + m_3 + 1$: the total number of constraints;
- $A_m \in \mathbb{N}^n$: the memory size of each couple (p, c), i.e. the memory required to record p;
- $A \in \mathbb{N}^{m \times n}$: the matrix composed of m vectors A_1, ..., A_m;
- $b \in \mathbb{N}^m$:
  - $\forall i \in [1, m_1 + m_2]\ b_i = 1$: consistency constraint and binary logical constraints;
  - $\forall i \in [m_1 + m_2 + 1, m_1 + m_2 + m_3]\ b_i = 2$: ternary constraints;
  - b_m: recording capacity or knapsack constraint.
- $c \in \mathbb{N}^n$: the vector of profits associated to each couple (photo camera).

$A_1, \ldots, A_{m-1}$ represent logical constraints. A_m represents the knapsack constraint. The parameters n and m characterize the dimension of a given instance of the problem in this formulation: n is the number of binary variables and m the total number of constraints to be satisfied.

Using these notations, the initial daily photograph scheduling problem can be formulated as follows:

$$\text{DPSP01} \begin{cases} \text{maximize} & z = c \cdot x \\ \text{subject to} & A \cdot x \leq b \text{ and } x \in \{0, 1\}^n \end{cases}$$

We have thus a logic-constrained 0-1 knapsack problem, which is a special case of the general multidimensionnel knapsack problem (MKP01). Let us notice that in practice, the number of constraints in this formulation may be very important, up to several tens of thousands compared with only several tens constraints for well-known MKP01 benchmark instances.

3. Benchmark instances and their lower bounds

In Bensana et al. (1999), the authors presented a set of 20 benchmark instances for the DPSP.[1] These instances belong to two series of different complexities. The first one has 13 instances corresponding to the simpler single-orbit case where the recording capacity (knapsack) constraint is absent. The second series includes 7 instances corresponding to the multi-orbit case where the recording capacity (knapsack) constraint is present. All the single-orbit instances have been solved to optimality. No optimal solution is known yet for 6 of the 7 multi-orbit instances.

3.1. Single-orbit instances

Table 1 summarizes the main characteristics of the 13 instances of the simple-orbit series. The first column gives the label of each instance, the second column the number of candidate photos in set P, the third column the number of binary variables. The next three columns indicate the number of binary and ternary constraints. The column z^* indicates the optimal (maximal) values (given in italic numbers) for the objective function. These values have been obtained thanks to an exact algorithm called Pseudo Dynamic Search (Verfaillie et al., 1996). The last column $\mathbf{1} \cdot x^*$ indicates the range for the number of taken photos in solutions found by the tabu search algorithms of Vasquez and Hao (2001).

3.2. Multi-orbit instances

Table 2 shows the main characteristics of the 7 instances of the multi-orbit case. The first six columns have the same meanings as in Table 1. The column b_m defines the maximal

Table 1. Benchmark instances for the simple-orbit case.

Pb.	Photos	n	m_1	m_2	m_3	z^*	$\mathbf{1} \cdot x^*$
54	67	125	67	389	23	*70*	45
29	82	120	82	610	0	*12032*	34
42	190	304	190	1762	64	*108067*	80
28	230	346	230	6302	590	*56053*	46–47
5	309	809	309	13982	367	*115*	93–96
404	100	158	100	919	18	*49*	31–33
408	200	328	200	2560	389	*3082*	60–63
412	300	544	300	6585	389	*16102*	77–79
11^2	364	692	364	9456	4719	*22120*	95–98
503	143	259	143	705	86	*9096*	69–70
505	240	448	240	2666	526	*13100*	82–85
507	311	573	311	5545	2293	*15137*	89–92
509	348	652	348	7968	3927	*19125*	93–96

Table 2. Benchmark instances for the multi-orbit case.

Pb.	Photos	n	m_1	m_2	m_3	b_m	z^*	$\mathbf{1} \cdot x^*$
1401	488	914	488	11893	2913	200	176056	146
1403	665	1317	665	14997	3874	200	176140	216
1405	855	1815	855	24366	4700	200	176179	249
1021^3	1057	2355	1057	30058	5875	200	176245	309
1502	209	413	209	296	29	200	*61158*	166
1504	605	1253	605	5106	882	200	124243	276
1506	940	2060	940	19033	4775	200	168247	307

recording capacity for the knapsack constraint. The column z^* indicates either the optimal value of the objectif fonction (given in italic numbers) or the lower bound for each instance. Notice that in addition to the presence of knapsack constraint, these instances are much larger than those of the single-orbit case.

Except the smallest instance 1502, neither optimal solution nor upper bound is known yet for these instances. The best known results have been obtained by the tabu search algorithm of Vasquez and Hao (2001), giving lower bounds for these instances.

4. Linear programming relaxation

One basic and general approach for generating upper bounds is Linear Programming (LP) relaxation. In LP relaxation we relax the integrality constraint of the initial problem and solve the relaxed problem using a standard method like simplex. Given the fact that there is no known result of LP relaxation for the muilti-orbit case of the general DPSP01. We decided to study first this basic approach.

Let us define the relaxed problem (denoted by DPSP in contrast to DPSP01) as follows:

$$\text{DPSP} \begin{cases} \text{maximize} & z = c \cdot x \\ \text{subject to} & A \cdot x \leq b \text{ and } x \in [0, 1]^n \end{cases}$$

We solved exactly this relaxed DPSP using the simplex method. Given the huge size of some multi-orbit instances, we had to implement a compact version of the simplex method using the code proposed in Press et al. (1992). Our implementation allowed us to deal with the most important instances which require up to 600 MB RAM.

We applied this approach to the instances of the single-orbit and multi-orbit cases. Upper bounds are reported in Tables 3 and 4.

4.1. Upper bounds for single-orbit instances

From Table 3, we remark that the upper bounds $\bar{z}$ are very far from the optimal values z^* of Table 1. The last column of Table 3 quantifies the gap between optimal continuous and binary values of the objectif function by the relative deviation $(\bar{z} - z^*)/\bar{z}$. The gap is in general more than 40% and goes beyong 50% for two instances. The gap may also be observed by comparing the columns $\mathbf{1} \cdot \bar{x}$ of Table 3 and $\mathbf{1} \cdot x^*$ of Table 1.

4.2. Upper bounds for multi-orbit instances

From Table 4, we observe that the upper bounds for the multi-orbit instances are in general of better quality than for the single-orbit ones. Given the fact that the values z^* of Table 2 are lower bounds obtained by tabu search, the stability of the relative deviation $(\bar{z} - z^*)/\bar{z}$ constitutes a good indicator concerning the quality of the solutions of tabu search. Notice, however, the absolute values of $\bar{z}$ and $\mathbf{1} \cdot \bar{x}$ remain far away from those of z^* and $\mathbf{1} \cdot x^*$.

Table 3. Upper bounds with LP relaxation: Single-orbit instances.

Pb.	n	m	Pivots	$\bar{z}$	$\mathbf{1} \cdot \bar{x}$	$(\bar{z} - z^*)/\bar{z}$ (%)
54	125	479	136	83.00	55.00	15.66
29	120	692	140	13057.00	54.50	7.85
42	304	2016	460	190567.50	125.00	43.29
28	346	7122	599	221090.50	147.50	74.65
5	809	14658	1625	315.00	281.25	63.49
404	158	1037	181	96.00	64.50	48.96
408	328	3149	436	5188.00	133.00	40.59
412	544	7274	801	31323.50	212.50	48.59
11	692	14539	1108	40416.00	266.00	45.27
503	259	934	292	12637.50	101.75	28.02
505	448	3432	564	22236.00	173.50	41.09
507	573	8149	833	27361.50	222.50	44.68
509	652	12243	1005	36394.00	252.00	47.45

Table 4. Upper bounds with LP relaxation: Multi-orbit instances.

Pb.	n	m	Pivots	$\bar{z}$	$\mathbf{1} \cdot \bar{x}$	$A_m.\bar{x}$	$(\bar{z} - z^*)/\bar{z}$ (%)
1401	914	15295	3261	300000.00	151.00	200.00	41.31
1403	1317	19537	3828	300149.00	283.75	200.00	41.32
1405	1815	29922	4342	300207.00	344.25	200.00	41.31
1021	2355	36991	5122	300385.00	507.75	200.00	41.33
1502	413	535	313	64160.50	169.75	147.50	4.68
1504	1253	6594	1368	191279.00	352.25	200.00	35.05
1506	2060	24749	3682	276863.00	492.25	200.00	39.23

It is generally believed that LP relaxation is not very powerful for generating good upper bounds. The above results are consistent with this fact.

5. Relaxing logic constraints

Given that the DPSP01 is defined by a knapsack constraint and a large number of logical constraints, we experimented a second approach based on the relaxation of these logical constraints. In this relaxed problem, the integrality constraint over the variables x_i is maintained. Such a relaxation leads to the following binary knapsack problem (KP01):

$$\text{KP01} \begin{cases} \text{maximize} & z = c \cdot x \\ \text{subject to} & A_m \cdot x \leq b_m \text{ and } x \in \{0, 1\}^n \end{cases}$$

Table 5. Maxima of KP01.

Pb.	$\tilde{z}$	Pb.	$\tilde{z}$
1401	332000	1502	89196
1403	332182	1504	245276
1405	332255	1506	294440
1021	332462		

For this relaxed problem KP01, there are several exact algorithms for solving instances of medium size ($n \leq 200$) (Fayard and Plateau, 1982; Nauss, 1976). When the values of the profit vector c are loosely correlated to the coefficients of the knapsack constraint A_m, there is an algorithm for solving instances of very large size ($n \leq 100{,}000$) (Martello and Toth, 1990).

There exists also a pseudo polynomial algorithm for solving KP01 using dynamic programming (Bellman and Stuart, 1962; Gondran and Minoux, 1985; Smith, 1990). This algorithm has a space and time complexity $O(n \times (b_m + 1))$. In our case, $b_m = 200$ for all the multi-orbit instances, this algorithm is quite interesting for the problem. We have implemented this algorithm and the results for the multi-orbit instances are given in Table 5.

From Table 5, one observes that bounds $\tilde{z}$ are worse that those of $\bar{z}$ obtained with LP relaxation. This is not really surprising given the large number of relaxed logical constraints. However, compared with LP relaxation, the running time of this approach is much faster: on a PIII500 MHZ, less than 2 seconds are necessary for the dynamic programming algorithm, 10 minutes to three days for the simplex algorithm.

In summary, neither LP relaxation nor relaxing logical constraints is able to produce good upper bounds for the benchmark instances. In the next section, we present a specialized approach which will give us upper bounds of high quality.

6. A partition based approach

6.1. Rationale

The partition-based approach follows the well-known "divide and conquer" principle. The rationale of this approach is the following.

We have exhaustive search algorithms which are able to solve exactly small problems, but have difficulties to tackle larger ones. We divide thus a large DPSP problem into several smaller sub-problems and solve these smaller problems exactly. Using such a strategy, we relax naturally the constraints which link some sub-problems. The sum of the optimal values of all the sub-problems must be greater or equal to the optimum of the initial problem, leading to an upper bound. This argument may be formally stated as follows.

- Let X be the set of all the couples (p, c) (variables) of a DPSP01 instance;
- Let $\mathring{z}$ be the optimal value of DPSP01;
- Let $\{X_l\}_{1 \leq l \leq k}$ a *partition* of X composed of k classes;

- Define now k sub-problems DPSP01(X_l) for $1 \leq l \leq k$, each DPSP01(X_l) being defined by the set of variables X_l and the constraints containing only these variables;
- Let $\{\mathring{z}_l\}_{1 \leq l \leq k}$ the set of optimal values of the above k sub-problems DPSP01(X_l).

Here constraints connecting two or more sub-problems are ignored, thus the set of sub-problems DPSP01(X_l), $1 \leq l \leq k$ constitutes a relaxation of the initial problem DPSP01. Thus, we have the following relation:

$$\hat{z} = \sum_{l=1}^{k} \mathring{z}_l \geq \mathring{z}$$

That is, the value $\hat{z}$ gives an upper bound of the initial DPSP01(X) instance.

Our partition-based approach may be summarized as follows.

1. Generate a partition $\{X_l\}_{1 \leq l \leq k}$ of the variable set X of the given DPSP01 instance.
2. Solve *exactly* each of k sub-problems DPSP01(X_l) ($1 \leq l \leq k$) to get the set of optimal values $\{\mathring{z}_l\}_{1 \leq l \leq k}$.
3. Sum up the values of the set $\{\mathring{z}_l\}_{1 \leq l \leq k}$.

In what follows, we describe first the exact algorithm for solving the sub-problems and then the heuristics for partitioning an initial problem.

6.2. *An iterative enumerative algorithm*

To solve exactly each DPSP01(X_l), we developed an iterative backtracking algorithm. This algorithm is inspired of the Russian Doll Search (Bensana et al., 1996; Verfaillie et al., 1996), which is itself a nested Branch & Bound algorithm. For each set of variables X_l $1 \leq l \leq k$, the algorithm solves successively X_l nested sub-problems including only 1 variable, then 2 variables...then $|X_l|$ variables (see Algorithm 1). The exact value obtained at a step is used at the next step in order to prune the search tree.

This algorithm uses a static order of the variables in $X_l = \{x_1 \cdots x_{|X_l|}\}$. Each entry i of the table $zmax$ records the optimal value of the sub-problem DPSP01($\{x_{i+1} \cdots x_{|X_l|}\}$).

Algorithm 1: SOLVE(X_l)

```
begin
    i ← |X_l|
    zmax[i] ← 0
    i ← i − 1
    while i ≥ 0 do
        zmax[i] ← ENUMERATE(X_l, i)
        i ← i − 1
    ž_l ← zmax[0]
end
```

Algorithm 2: ENUMERATE(X_l, i)

```
begin
    j, z, z* ← i, 0, −1
    go_forward
        if j = |X_l| then
[1]         if z > z* then z* ← z
            backtrack
        else
            j ← j + 1
            x_j ← 1
            z ← z + l_j
            if there is constraint violation then backtrack
            go_forward
    backtrack
        if x_j = 0 then
            j ← j − 1
            if j = i then return z*
            backtrack
        else
            x_j ← 0
            z ← z − l_j
[2]         if z + zmax[j] ≤ z* then backtrack
            go_forward
end
```

The most important element of this nested search procedure is ENUMERATE(X_l, i) (see Algorithm 2). This algorithm realises a deep-first search, *backtracks* only on the last $|X_l| - i$ elements of the set X_l. Line (2), together with the update of line (1) allows a better pruning of the search tree than a standard depth first Branch & Bound search.

Not surprisingly, this algorithm has an exponential time complexity in the worse case with respect to the number of the variables of the problem. However, the algorithm is able to solve problems of reasonable size. Next section describes a way allowing us to partition a large problem into such solvable (sub-)problems.

6.3. Heuristics for partitioning

6.3.1. An equal size partition. In order to partition a given DPSP01, one must answer two questions. First, what is the appropriate number k of the sub-problems? Second, for a fixed k, how should the initial problem be partitioned. Intuitively, both points would have important influence on the quality of the bounds. These issues are further studied in the next section. In fact, we will see that an optimal solution for these two points may be as hard as solving the initial problem. For the moment, we limit ourselves to a trivial partition in order to get a primary idea about the merit of this partition-based approach for upper bounds.

For a fixed number k, the n variables of the problem are divided into k classes of equal size $\lfloor n/k \rfloor$ following their order in n: the first $\lfloor n/k \rfloor$ variables of n go to the class X_1, the next $\lfloor n/k \rfloor$ variables to the class X_2 and so on.[4]

Table 6. Trivial partitions for the instance 1401.

k	*#RC*	$\hat{z}$	k	*#RC*	$\hat{z}$
20	8635	263069	15	7728	247070
19	8459	252070	14	7242	240064
18	8321	258066	13	6973	245062
17	8039	254067	12	6648	242064
16	7865	247067	11	6513	243063

Table 6 shows the results for the instance 1401 ($n = 914$) using this partition strategy with k fixed at 10 different values from 20 to 11. The column *#RC* indicates the number of relaxed constraints (for a total of 15294 constraints): a relaxed constraint is one whose variables belong to more than one variable class (except the knapsack constraint which is not relaxed). Comparing Tables 6 and 4, we observe that these upper bounds are better than those of LP relaxation. However, these bounds are still far from the best known lower bound which is 176056 (see Table 2). To improve these bounds, we may decrease the number of partitions k. Unfortunately, this increases rapidly the running time of the approach. We observe also that the bounds don't improve regularly when the number of partitions decreases: the bound with 19 partitions is better than those with 18 and 17.

In fact, for a given instance, there exist some partitions better than others. In particular, the number of relaxed constraints should be used as a criterion for determining the partitions. This idea is studied in detail in the next section leading to a heuristic method for a more efficient partitioning.

6.3.2. A criterion for optimized partitioning. In this section, we introduce an optimization criterion for a better partitioning. The basic idea is the following. For a fixed number k of classes, a good partition should minimize the "number" of inter-class constraints. Such a partition will generate classes with a maximum of intra-class constraints, helpful for the exact algorithm.

Recall that there are different constraints in the DPSP01. The optimization criterion considers consistency, binary and ternary constraints and introduces a hierarchy for these constraints with a decreasing importance in that order.[5] Let $\{X_l\}_{1\leq l\leq k}$ be a partition of X. We represent this hierarchy by using the ω function:

$$\omega(x_i, x_j) = \begin{cases} 8 & \text{if } \exists k \in [1, m_1] \text{ such that} \quad A_{ki}.A_{kj} = 1 \\ 4 & \text{if } \exists k \in [m_1 + 1, m_1 + m_2] \text{ such that} \quad A_{ki}.A_{kj} = 1 \\ 1 & \text{si } \exists k \in [m_1 + m_2 + 1, m - 1] \text{ such that} \quad A_{ki}.A_{kj} = 1 \end{cases}$$

The values 8, 4 and 1 represent the relative importance that we give to each constraint and are empirically determined.

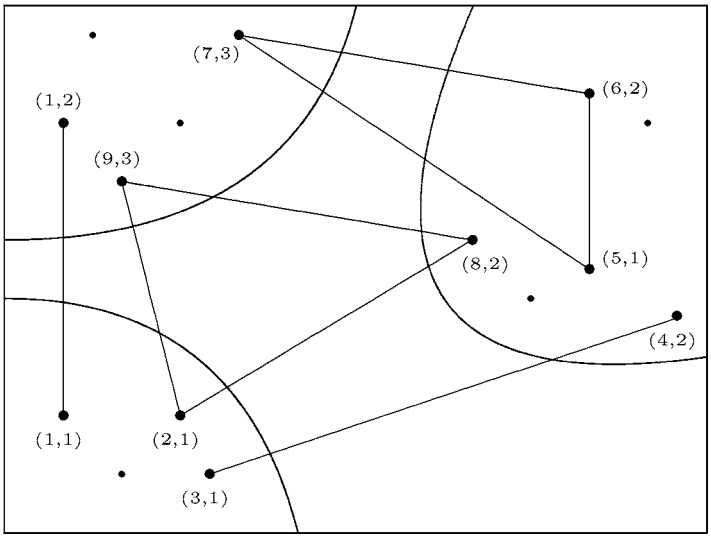

Figure 1. Example of a partition: $k = 3$, $\#RC = 4$ et $f = 17$.

We define also the following binary function κ:

$$\kappa(x_i, x_j) = \begin{cases} 0 & \text{if variables } x_i \text{ and } x_j \text{belong to a same class} \\ 1 & \text{otherwise} \end{cases}$$

Then we look for a partition $\{X_l\}^*$ of X which minimizes the function:

$$f(\{X_l\}) = \sum_{i<j} \kappa(i, j).\omega(i, j)$$

Figure 1 shows a small example with a partition of three classes. The points represent couples (photo, camera). The edges represent logical inter-class contraints. All the possible cases are given in this figure:

- one consistency constraint between couples (1,1) and (1,2);
- one inter-class binary constraint between couples (3,1) and (4,2);
- the two possible cases for inter-class ternary constraints: (2,1), (8,2) and (9,3) on the one hand, (5,1), (6,2) and (7,3) on the other hand.

Now, consider the graph whose nodes represent the classes of a partition and whose (weighted) edges represent constraints of the given DPSP01 instance. Then the task is to find a partition such that cut edges (those having their two end points in different classes) have a minimal total sum of their weights. This problem is thus a generalized version of the well-known NP-hard graph partitioning problem (Garey and Johnson, 1979, p. 209).

6.3.3. Partitioning with tabu search. Given the intrinsic difficulty of the above partitioning problem, we turn to a heuristic approach for generating good but not necessarily optimal partitions. For this purpose, we developed a tabu search algorithm, which improves iteratively an initial trivial partition. Notice that tabu search has been applied to the the conventional graph partitioning problem, see for instance (Friden et al., 1989; Rolland et al., 1996). Our tabu search algorithm shares some common features with these previous implementation. Some specific techniques are introduced to cope with the particular nature of our problem.

Given a DPSP01 instance and a fixed number k, the tabu search algorithm tries to find a partition $\{X_l\}_{1\leq l\leq k}$ of the initial variable set X while minimizing the previously defined f fonction (Section 6.3.2). In the tabu search algorithm, a configuration represents a partition composed of a fixed number k classes. We describe now the neighborhood relations and tabu tenure of this Tabu algorithm.

Two neighborhood relations are used by the algorithm in an alternative way. The first neighborhood is symmetric and the size of each class of a partition is kept constant.[6] A neighboring partition is obtained by simply exchanging two nodes between two classes. More precisely, at each iteration of the algorithm, we first select a node s such that it has the maximal number of edges going out from its class X_s. We then exchange s with a node t of another class X_t ($X_s \neq X_t$).

The second neighborhood is not symmetric. The size of each class of a partition may vary during the search. With this neighborhood, one chooses a variable s in a class X_s such that $|X_s| \geq s_{\min}$ and moves s into another class X_t such that $|X_t| < s_{\max}$. $s_{\min}$ and $s_{\max}$ are two symmetric values around n/k:

$$\left(s_{\max} - \frac{n}{k}\right) = \left(\frac{n}{k} - s_{\min}\right)$$

This principle, inspired of the work presented in Rolland et al. (1996) has, among others, the advantage of encouraging the grouping of the binary variables of a same photo in a same class.

The complexity of searching the first neighborhood is linear with respect to the instance size, because the first element of a move is chosen independently of the second element. For the second neighborhood, a trivial implementation will lead to a complexity of $O(n \times k)$. This complexity may be reduced using special data structures such as *buckets* (Battiti and Bertossi, 1999) when the number of the classes k is greater than two.

The tabu search algorithm uses mainly the first neighborhood during its search, i.e. during its search intensification phases and changes to the second neighborhood during its search diversification phases.

Each time a move is carried out, the reversing move is classified tabu for a number of iterations (tabu tenure). The tabu tenure of a move is determined dynamically using the following formula:

$$date(t, X_s) + freq(t) \geq iter$$

where *iter* is the current iteration number, $freq(t)$ the number of times node t has been exchanged with another node, $date(t, X_s)$ the time stamps (an iteration number) when t was moved into X_s.

Table 7. Advanced partitions with tabu search for the instance 1401.

k	$\#RC$	$\hat{z}$	k	$\#RC$	$\hat{z}$
20	3094	187062	15	2585	184062
19	3532	186064	14	3789	188062
18	3382	187063	13	3217	186062
17	3210	187062	12	2560	184062
16	3297	186063	11	1940	185062

In addition to the number of classes k for a partition, the tabu search algorithm requires some other parameters, in particular, $s_{\min}$ and $s_{\max}$.

In order to get a rough idea about the merit of this approach, we take again the instance 1401 used for the trivial partition (see Section 6.3.1). As before, we use 10 different partition sizes going from 20 classes to 11 classes. Using the approach described in this section, we obtain the results showed in Table 7.

We observe that these upper bounds $\hat{z}$ are much better than previous ones and the number of relaxed constraints $\#RC$ is much smaller. The running time to obtain these results is less than one minute on a PIII500 PC. Next section shows more detailed results for all the benchmark instances using this approach.

7. Upper bounds for the DPSP01 benchmark instances

7.1. New upper bounds for the single-orbit instances

For the single-orbit instances, we may compare directly our results with the optimal values z^* reported in Verfaillie et al. (1996). Comparisons are also given with previous known upper bounds obtained with column generation techniques (Gabrel, 1999). This allows one to further appreciate the advantage of the partition-based approach. Results are given in Table 8 ("-" means no result is available).

From the table, we see that the upper bound of 9 out of 13 instances coincides with the optimal value and for the 4 remaining instances, two bounds are very close to the optimal value. We remark that the best partition found has only two or three classes. Finally, we observe the partition-based bounds are much better than the previous known ones.

7.2. Tight upper bounds for the multi-orbit instances

We present now the most important results of this paper, i.e. the upper bounds for the multi-orbit instances, which remained unknown so far. Results are given in Table 9.

The first column z^* shows the *lower* bounds obtained with the tabu search algorithm described in Vasquez and Hao (2001). Columns $c_1, c_2, c_3, A_m.\hat{x}$ have the following meaning and together they give indications about the degree of unfeasibility of the solutions.

- c_1 = the number of relaxed *consistency* constraints;
- c_2 = the number of relaxed *binary* constraints;

Table 8. Comparison of upper bounds for the simple-orbit instances.

	Column-generation-based upper bounds		Partition-based upper bounds		
Pb.	$\hat{z}$	$(\hat{z} - z^*)/\hat{z}$ (%)	k	$\hat{z}$	$(\hat{z} - z^*)/\hat{z}$ (%)
54	71.44	2.02	2	70	0
29	–	–	3	12032	0
42	108067	0.00	3	108067	0
28	70053.00	19.98	2	58053	3.45
5	119.00	3.36	3	116	0.86
404	53.66	8.68	3	49	0
408	3091.75	0.32	3	3083	0.03
412	71.44	2.02	2	16102	0
11	–	–	2	22120	0
503	9595.00	5.20	2	9096	0
505	14107.75	7.14	3	13103	0.02
507	17639.22	14.19	2	15137	0
509	–	–	2	19125	0

Table 9. Tight upper bounds for the multi-orbit instances.

Pb.	z^*	k	$\hat{z}$	c_1	c_2	c_3	$A_m \cdot \hat{x}$	$(\hat{z} - z^*)/\hat{z}$ (%)
1401	176056	8	180062	13	25	1	221	2.22
1403	176140	7	180160	6	15	0	249	2.23
1405	176179	6	179226	2	11	3	283	1.70
1021	176246	10	177304	5	21	5	291	0.60
1502	*61158*	2	61158	0	0	0	148	0
1504	124243	3	124258	0	3	0	237	0.01
1506	168247	6	168294	2	13	0	283	0.03

- c_3 = the number of relaxed *ternary* constraints;
- $A_m \cdot \hat{x}$ = memory consummation which should be $\leq$200 according to the knapsack constraint.

The last column $(\hat{z} - z^*)/\hat{z}$ contains the most important information. We observe that the upper bounds $\hat{z}$ are close or very close to the lower bounds z^* for these instances. This observation has two immediate important implications. First, we may conclude that the heuristic solutions presented in Vasquez and Hao (2001) are of high quality. For three instances (1021, 1504, 1506), the heuristic solution is already very close to the optimal value, if it is not optimal. Second, we may conclude that these upper bounds are very tight, especially for the last four instances in the table. On the contrary, further improvements may be expected concerning the first three instances for both upper and lower bounds.

In general, few logical constraints are violated (relaxed). On the contrary, we observe an important memory excess with respect to the capacity constraint ($\leq$200). This is not surprising given that the knapsack constraint is not taken into account by the partition algorithm.

Finally, the total resolution time (partitioning with a tabu search plus exact resolution of each sub-problem) varies from several hours to several days on a PIII 500 MHZ.

8. Conclusion

The main result of this study is the achievement of a set of tight upper bounds for a set of benchmark instances of the daily photograph scheduling problem of earth observation satellites such as Spot 5. Contrasting these upper bounds with the lower bounds of Vasquez and Hao (2001) allows us not only to assess the quality of the previous heuristic solutions but also to assess the quality of the upper bounds on the other hand. These upper bounds may be also helpful for devising more efficient exact algorithms. Such an algorithm may hopefully solve optimally all the benchmark instances.

The upper bounds are obtained with an original partition-based approach. This approach is based on the well-known "divide and conquer" principle. The initial problem is divided into sub-problems which are solved exactly by an iterative enumeration algorithm. A tabu search algorithm is used to determine optimized partitions.

Two more conventional approaches were also experimented. The first one is based on LP relaxation with simplex. The second relaxes the logical constraints and solves a knapsack problem with dynamic programming. Both approaches led only to weak upper bounds.

Finally, this work shows the interest of combining a heuristic search and an exact search for large and difficult combinatorial optimization.

Acknowledgments

The authors highly appreciate the valuable and detailed comments of the anonymous reviewers, which helped to improve the presentation of the paper.

Notes

1. These benchmark instances are available from `ftp://ftp.cert.fr/pub/lemaitre/LVCSP/Pbs`
2. In Bensana et al. (1999), this instance is identified by 414.
3. In Bensana et al. (1999), this instance is identified by 1407.
4. If $k * \lfloor n/k \rfloor \neq n$, the last $n - k * \lfloor n/k \rfloor$ variables go to the last class.
5. Given the global nature of the knapsack constraint, this constraint is completely relaxed in this approach.
6. Two different classes may have different sizes however.

References

R. Battiti and A.A. Bertossi, "Greedy, prohibition, and reactive heuristics for graph partitioning," *IEEE Transactions on Computers*, vol. 48, no. 4, pp. 361–385, 1999.

R. Bellman and D. Stuart, *Applied Dynamic Programming*, Princeton University Press, 1962.

E. Bensana, M. Lemaître, and G. Verfaillie, "Earth observation satellite management," *Constraints*, vol. 4, no. 3, pp. 293–299, 1999.

E. Bensana, G. Verfaillie, J.C. Agnèse, N. Bataille, and D. Blumstein, "Exact and inexact methods for the daily management of an earth observation satellite," in *Proc. 4th Intl. Symposium on Space Mission Operations and Ground Data Systems*, Münich, Germany, 1996.

D. Fayard and G. Plateau, "An algorithm for the solution of the 0-1 knapsack problem," *Computing*, vol. 28, pp. 269–287, 1982.

C. Friden, A. Hertz, and D. de Werra, "Stabulus: A technique for finding stable sets in large graphs with tabu search," *Computing*, vol. 42, pp. 35–44, 1989.

V. Gabrel, "Improved linear programming bounds via column generation procedure for the daily scheduling of earth observation satellite," Research Report 99-01, LIPN, Paris XIII University, Jan. 1999.

M. Garey and D. Johnson, *Computers & Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979.

M. Gondran and M. Minoux, *Graphes & Algorithmes*, Eyrolles, 1985.

S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley: New York, 1990.

R.M. Nauss, "An efficient algorithm for 0-1 knapsack problem," *Management Science*, vol. 23, ppp. 27–31, 1976.

W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C*, Cambridge University Press: Cambridge, 1992.

E. Rolland, H.P. Pirkul, and F. Glover, "Tabu search for graph partitioning," *Annals of Operations Research*, vol. 63, pp. 209–232, 1996.

D.K. Smith, *Dynamic Programming: A Practical Introduction*, Ellis Horwood, 1990.

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," *Journal of Computational Optimization and Applications*, vol. 20, pp. 137–157, 2001.

G. Verfaillie, M. Lemaître, and T. Schiex, "Russian doll search for solving constraint optimization problems," in *Proc. 13th National Conference on Artificial Intelligence*, Portland, USA, 1996, pp. 182–187.