A "Logic-Constrained" Knapsack Formulation and a Tabu Algorithm for the Daily Photograph Scheduling of an Earth Observation Satellite

Michel Vasquez [1)] and Jin-Kao Hao [2)]
1) LGI2P, Ecole des Mines d'Alès, Parc Scientifique G. Besse, 30035 Nîmes Cedex 1, France
Michel.Vasquez@site-eerie.ema.fr
2) LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
Jin-Kao.Hao@univ-angers.fr

(*Journal of Computational Optimization and Applications, **accepted in February 2000***)

Abstract: The daily photograph scheduling problem of earth observation satellites such as Spot 5 consists of scheduling a subset of mono or stereo photographs from a given set of candidates to different cameras. The scheduling must maximize a profit function while satisfying a large number of constraints. In this paper, we first present a formulation of the problem as a generalized version of the well-known knapsack model, which includes large numbers of binary and ternary "logical" constraints. We then develop a tabu search algorithm which integrates some important features including an efficient neighborhood, a dynamic tabu tenure mechanism, techniques for constraint handling, intensification and diversification. Extensive experiments on a set of large and realistic benchmark instances show the effectiveness of this approach.

Key words: tabu search, heuristics, satellite photograph scheduling, multidimensional knapsack, constrained combinatorial optimization

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 problem is also important and interesting from a complexity point of view. Indeed, it can be modeled as a generalized Knapsack problem, which is known to be NP-hard. So far, several methods have been proposed to tackle this problem. These methods include *exact* algorithms based on Branch and Bound techniques [Verfaillie et al. 96], integer linear programming with CPLEX and constraint programming with the ILOG solver [Lemaître & Verfaillie 97] as well as *heuristics* based on greedy functions, simulated annealing and tabu search [Bensana et al. 96].

The goal of this paper is two fold. First, we introduce a new formulation of the DPSP using the well-known 0/1 knapsack model. Second, we develop an original and effective tabu search (TS) algorithm based on this formulation. This TS algorithm includes a set of important features including an efficient neighborhood, an incremental technique for move evaluation, a mechanism for dynamic tabu tenure, constraint handling techniques, and mechanisms for intensification and diversification.

This algorithm is extensively evaluated on a set of large and realistic instances provided by the French National Space Agency CNES (Centre National d'Etudes Spatiales). Experimental results show the effectiveness of this algorithm both in terms of solution quality and speed of execution. Indeed, the algorithm easily obtains the previously best known results for these instances. More importantly, it produces much better solutions for the most difficult instances with very reasonable computing times.

The paper is organized as follows, in the next section, the DPSP is described and modeled. Section 3 presents the details of the TS algorithm. Numerical results and comparisons are presented in Section 4. Implications are discussed in Section 5 and Section 6 concludes the paper.

2. Photograph daily scheduling problem (DPSP)

2.1. Problem definition

The photograph daily scheduling problem can be informally described as follows [Bensana et al. 98].

Problem Components

- A set $P = \{p_1, p_2, ..., p_n\}$ 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 hard constraints, which must be satisfied:

  1. any two trials must not overlap and the minimal transition time between two successive photographs on the same camera must be met;
  2. limitations on the instantaneous data flow through the satellite telemetry resulting from simultaneous photographs on different cameras;
  3. capacity constraint: the recording capacity on board must not be exceeded.

Problem Objective

The DPSP is to find a subset P' of P which satisfies all the imperative constraints and maximizes the sum of the profits of the photographs in P'. Thus, the goal is to maximize the total value of the items (photographs) packed in the "knapsack", subject to the constraint that the total size of all the packed items does not exceed the knapsack capacity (constraint 3) and subject to other "logical constraints" (constraints 1 and 2 above). The DPSP is therefore a constrained combinatorial optimization problem.

In practice, the number of photographs in P may be quite large (up to 1057 for the largest instance we tested), implying a huge search space. Moreover, the presence of a large number of hard constraints (up to tens of thousands) makes the problem difficult to solve.

2.2. Problem formulation

2.2.1 Representing a schedule

Let P be the set of candidate photographs and $n = |P|$. With each mono photograph p_i in P, we associate three pairs of elements (p_i, camera_1), (p_i, camera_2), (p_i, camera_3). Similarly, with each stereo photograph p_i in P, we associate one pair (p_i, camera_13). Letting n1 and n2 be respectively the number of mono and stereo photographs in P (n = n1 + n2), there are in total m = 3*n1 + n2 possible pairs of elements for the given set P of candidates. Now, associating a binary (decision) variable x_i with each such pair, a photograph schedule corresponds to a binary vector:

$$x = (x_1, x_2, ..., x_m)$$

where $x_i = 1$ if the corresponding pair (photo, camera) is present in the schedule, and $x_i = 0$ otherwise.

For example, if $P=\{p_1, p_2, p_3\}$ where p_1 and p_2 are mono photographs and p_3 is a stereo photograph, then x = (1, 0, 0, 0, 0, 0, 1) represents a schedule in which p_1 is taken by camera 1, p_2 is rejected and p_3 is taken by cameras 1 and 3. (Remember that a stereo photograph requires the front and the rear cameras simultaneously.)

2.2.2 Evaluation of a schedule

Define the profit of a pair (p, camera) (or its 0-1 variable) to be the profit of the photograph p. The total profit of all the pairs of the given set P is then represented by a vector:

$$g = (g_1, g_2,..., g_m)$$

where $g_i = g_j$ ($i \neq j$) if g_i and g_j correspond to two different pairs of elements involving the same photograph p, i.e. (p, camera_x) and (p, camera_y).

Then the total profit value of a schedule $x = (x_1, x_2,..., x_m)$ is the sum of the profits of the photographs in s, i.e.

$$f(x) = \Sigma_{1 \leq i \leq m}\, g_i \cdot x_i$$

2.2.3 Constraints

- **Capacity constraint**

Define the size of a pair (p, camera) (or its 0-1 variable) as the size of the photograph p. The total size of all the pairs of the given set P is then represented by a vector:

$$c = (c_1, c_2, ..., c_m)$$

where $c_i = c_j$ $(i \neq j)$ if c_i and c_j correspond to two different pairs of elements involving the same photograph p, i.e. (p, camera_x) and (p, camera_y).

The capacity constraint states that the sum of the sizes of the photographs in a schedule $x = (x_1, x_2, ..., x_m)$ cannot exceed the maximal recording capacity on board. This constraint is easily expressed as a knapsack constraint:

$$\Sigma_{1 \leq i \leq m} c_i \,.\, x_i \leq \mathit{Max\_capacity}$$

- **Binary constraints**

The constraints involving the non overlapping of two trials and the minimal transition time between two successive trials of a camera, and also some constraints involving limitations on instantaneous data flow are conveniently expressed by simple relations over two pairs (photo, camera). Such a binary constraint corresponds to forbidding the simultaneous presence of a pair (p_i, k_i) and another pair (p_j, k_j) in a schedule. If x_i and x_j are the corresponding decision variables of such two pairs, then a binary constraint is defined as follows:

$$x_i + x_j \leq 1$$

Let C2 denote the set of all such pairs (x_i, x_j) which should verify the above binary constraint.

- **Ternary constraints**

Some constraints involving limitations on instantaneous data flow cannot be expressed in the form of binary constraints as above. These remaining constraints may however be expressed by relations over three pairs (photo, camera). Such a ternary constraint corresponds to forbidding the simultaneous presence of three pairs (p_i, k_i), (p_j, k_j), and (p_l, k_l). Letting x_i, x_j and x_l be the decision variables corresponding to these pairs, then such a ternary constraint is written:

$$x_i + x_j + x_l \leq 2$$

Let C3_1 denote the set of all such triplets (x_i, x_j, x_l) which should verify this ternary constraint.

Finally, we need to be sure that a schedule contains no more than one pair from $\{(p, k_i), (p, k_j), (p, k_l)\}$ for any (mono) photograph p. Letting x_i, x_j and x_l be the decision variables corresponding to these pairs, then this (ternary) constraint is expressed as:

$$x_i + x_j + x_l \leq 1$$

Clearly there are exactly n1 ternary constraints of this type. Let C3_2 denote the set of all such triplets (x_i, x_j, x_l) which verify this second type of ternary constraints. Use C3 to denote the union of C3_1 and C3_2, i.e. C3 = C3_1 $\cup$ C3_2.

2.2.4 Final model

Now the DPSP can be formally stated as the following generalized 0/1 Knapsack problem:

Max $f(x) = \Sigma_{1\leq i\leq m}\, g_i \cdot x_i$

where $x = (x_1, x_2,..., x_m) \in \{0,1\}^m$ and $g = (g_1, g_2,..., g_m) \in Z^{+\,m}$

subject to

1) $\sum_{1\leq i\leq m} c_i \cdot x_i \leq$ *Max_capacity* with *Max_capacity* $\in Z^+$ and $c = (c_1, c_2,..., c_m) \in Z^{+\,m}$

2) $\forall\, (x_i, x_j) \in C2,\ x_i + x_j \leq 1$

3) $\forall (x_i, x_j, x_k) \in C3\_1,\ x_i + x_j + x_k \leq 2$

4) $\forall (x_i, x_j, x_k) \in C3\_2,\ x_i + x_j + x_k \leq 1$

This formulation is a special instance of the multidimensional knapsack problem (MKP) [Martello & Toth 90]. However, let us point out a notable difference. While the constraints in a MKP are all "knapsack constraints" like (Eq.1), the formulation above has a single "knapsack constraint" and three types of "logic constraints" (Eq. 2 to 4). Moreover, while the number of constraints in a MKP is rarely large (for instance, well-known benchmark problems have at most 30 constraints), the number of logic constraints may be very high (up to 36000 for some solved instances). Therefore, special techniques are needed for handling these constraints in an effective way.

3. A TS algorithm for DPSP

3.1. Review of TS

This section gives a brief review of Tabu Search, emphasizing the most important features which have been implemented in our TS algorithm. For a comprehensive presentation of TS, the reader is invited to consult the recent book by Glover and Laguna [Glover & Laguna 97].

Tabu Search is a meta-heuristic designed for tackling hard combinatorial optimization problems. Contrary to randomizing approaches such as SA where randomness is extensively used, TS is based on the belief that intelligent searching should embrace more systematic forms of guidance which are based on adaptive memory and learning.

TS can be described as a form of neighborhood search with a set of critical and complementary components. For a given instance of an optimization problem (S,f) characterized by a search space S and an objective function f, a neighborhood N is first introduced to associate, for each s in S, a non-empty subset N(s) of S. A typical TS algorithm begins then with an initial configuration s in S and then proceeds repeatedly to visit a series of locally best configurations following the neighborhood function. At each iteration, one of the *best* neighbors s' $\in$ N(s) is sought to replace the current configuration even if *s'* does not improve the current configuration in terms of the cost function. To avoid the problem of possible cycling and to allow the search to go beyond local optima, TS introduces the notion of *tabu list*, which is a foundation for the short term memory component of the method.

A tabu list maintains a selective history H, composed of previously encountered solutions or, more generally, pertinent attributes of such solutions. A simple TS strategy based on this short term memory H consists in preventing solutions of H from being reconsidered for the next k iterations, called the *tabu tenure*. The tabu tenure can vary for different attributes, and in general is problem dependent. At each iteration, TS searches for a best neighbor from this dynamically modified neighborhood N(H,s), instead of N(s) itself. Such a strategy prevents the search from being trapped in short term cycling and imparts rigor to the search.

By means of the tabu restriction mediated by this memory, some non-visited, yet interesting solutions may be prevented from being considered. Accordingly, *aspiration criteria* are introduced to overcome this problem. A simple and widely used aspiration criterion consists of removing a tabu classification from a move when the move leads to a solution better than the best obtained so far.

Two other important ingredients of TS are intensification and diversification [Glover & Laguna 97]. Intensification consists in focusing the search to exploit regions of the space, or characteristics of solutions, that the search history suggests are promising. For example, it may be applied to seek improved solutions by incorporating "good attributes" of previously encountered solutions. On the other hand, diversification undertakes to explore regions that differ in significant respects from regions previously visited.

3.2. Components of the TS algorithm

3.2.1. Unconstrained and constrained search space

- **Definition 1:** The *unconstrained search space* S is composed of all binary vectors of m elements:

$$S = \{(x_1, x_2,..., x_m) \in \{0, 1\}^m\}$$

The size of S may become huge for high values of m. For some instances we solved, the value of m can be as large as 2355, implying a search space of 2^{2355}. However, a solution must verify the constraints defined by Eq. (1)-(4) and thus belongs to a constrained space. This leads to the following definition.

- **Definition 2:** The *totally constrained search space* X is composed of all binary vectors of m elements, satisfying the "knapsack constraint" and the "logic constraints", i.e.

$$X = \{s \in S \mid s \text{ satisfies all the constraints defined by Eq. (1) - (4)}\}$$ [1]

In general, an algorithm may work with either S or X. It is equally possible for an algorithm to work with an intermediary space where some constraints are relaxed. It is this last approach which is adopted in this work. For this purpose, we define a *partially constrained search space* C where the knapsack constraint (Eq. (1)) is relaxed.

- **Definition 3:** The *partially constrained search space* C is composed of all binary vectors of m elements, satisfying the "logic constraints", i.e.

$$C = \{s \in S \mid s \text{ verifies the logic constraints defined by Eq. (2) - (4)}\}$$

[1] Hereafter, we will use the letter s instead of x to denote a configuration $(x_1, x_2,..., x_m)$.

Note that not all the configurations in C are equally interesting. For example, s = (0, 0, ..., 0) is trivially in C but far from any global optimum. The following definition identifies a subset of C which is particularly interesting from the point of view of the profit function.

- **Definition 4**: The *saturated partially constrained search space* M is a subset of C such that:

M={s∈C | ∀(x_i=0)∈s, setting x_i=1 violates some binary or ternary constraints defined by Eq. (2)-(3)}

Thus a configuration s∈C is saturated when no more pair (photo, camera) can be added without violating some logic constraints.

The saturated set M is similar to the notions of "critical events" [Glover & Kochenberger 96] and "promising zones" [Hanafi & Freville 97] developed for the MKP. The principle here is that the TS algorithm will search for its solutions in the partially constrained search space C and try to stay at the frontier of this saturated (promising) area.

3.3.2. Neighborhood and move

We introduce now the *neighborhood function* N over the partially constrained search space C. More precisely, this function N: $C \rightarrow (2^C - \emptyset)$is defined as follows.

Let s = $(x_1, x_2,...,x_m)$∈C and s′ = $(x'_1, x'_2,...,x'_m)$, then s′ is a neighbor of s, i.e. s′∈N(s), if and only if the following conditions are verified:

1) ∃ ! i such that $x_i = 0$ and $x'_i = 1$ (1≤i≤m)
2) for the above i, ∀ (x_i, x_j) ∈ C2, $x'_j = 0$ (1≤j≤m)
3) for the above i, ∀ (x_i, x_j, x_k) ∈ C3_1, $x'_j + x'_k \leq 1$ (1≤j, k≤m)

Thus, a neighbor of s can be obtained by *adding* a pair (photo, camera) (flipping a variable x_i from 0 to 1) in the current schedule and then *dropping* some pairs (photo, camera) (flipping some x_j from 1 to 0) to repair binary and ternary constraint violations. Consequently, a move *mv* to obtain a neighbor s′ from a configuration s = $(x_1, x_2, x_3, ..., x_m)$ is characterized by a series of flipping operations:

$$\begin{array}{c} x_i \text{ from 0 to 1} \\ x_j \text{ from 1 to 0} \\ \dots \dots \\ x_k \text{ from 1 to 0} \end{array}$$

where x_j ... x_k are variables linked to x_i by a binary or ternary constraint. We use mv(i) = (x_i: 0→1, x_j ... x_k: 1→0) to denote such a move.

The repair of a violation of a binary constraint (x_i, x_j) ∈ C2 is a simple operation: it suffices to set x_j’, to 0 in s’ (x_j = 1 in s). Repairing a ternary constraint violation (x_i, x_j, x_k) ∈ C3 is more complex since there are different ways to proceed. For example, one may set either x'_j or x'_k to 0 in s′ randomly or according to some criteria, one may set both x'_j and x'_k to 0 and so on. More details on this topic are discussed later (§5.3).

It should be clear that from a configuration $s = (x_1, x_2, ..., x_m)$, the number of possible moves equals the number of variables in s having value 0. Letting $Z = \{x_i \in s \mid s = (x_1, x_2, ..., x_m) \text{ and } x_i=0\}$, then N(s) has exactly |Z| neighboring configurations.

Note that similar neighborhoods based on adding-dropping have been used in many heuristic algorithms for MKP [Dammeyer & Voss 93, Fréville & Plateau 94, 97, Glover & Kochenberger 96, Lokketangen & Glover 98, Hanafi & Fréville 98, Chu & Beasley 98]. However, one difference remains that concerns the repair operation after adding an element: constraint repairing here is much simpler since it concerns only binary and ternary logic constraints.

3.3.3. Incremental evaluation of the neighborhood

TS uses an aggressive search strategy to exploit its neighborhood, i.e. at each iteration, the TS algorithm examines the value f(s') for each candidate neighbor $s' \in N(s)$ and chooses one that has the best value. In order to do this in an efficient way, we use an incremental evaluation technique. The main idea is to keep in a special data structure δ the *move value* for each possible move (neighbor) of the current configuration. Thus if s' = s + mv(i), then $\delta(i)$ is equal to the difference f(s') - f(s). Each time a move is carried out, the elements of this data structure affected by the move are updated accordingly.

Since the number of possible moves for each configuration is defined by |Z| ($|Z| \leq m$), the data structure δ can be implemented with a vector of m elements: $\delta(i)$ gives the move value f(s)-f(s') if the corresponding move mv(i) is carried out. The vector can be initialized at the beginning of the search and updated after a move in time *O*(|C2_i| + |C3_i|) where C2_i and C3_i are subsets of the sets of binary and ternary logic constraints (see §2.2) involving the element x_i. Searching for a best move within δ requires time *O*(m).

3.3.4. Tabu list management

The role of a tabu list is to prevent the search from short-term cycling (x_i: $1\rightarrow0\rightarrow1\rightarrow0$...). Remember that a move mv(i) = (x_i: $0\rightarrow1$, x_j ... x_q: $1\rightarrow0$) consists in flipping x_i from 0 to 1 and flipping then x_j, ..., x_q from 1 to 0 (to repair constraint violations). Each time such a move is carried out, the moves mv(j) = (x_j: $0\rightarrow1$, ...) ... mv(q) = (x_q: $0\rightarrow1$, ...) are classified tabu during some iterations (tabu tenure), forbidding to reset any of x_j, ..., x_q from 0 to 1. The number of iterations *k(t)* during which a move mv(t) (t = j...q) is classified tabu is dynamically defined as follows:

$$k(t) = C(t) + \alpha \times FREQ(t)$$

where *C(t)* is the number of binary and ternary constraints involving the element x_t, *FREQ(t)* the number of times x_t is flipped from 1 to 0 from the beginning of the search, and $\alpha = (\Sigma_{1\leq i\leq m} C(i))/m$ reflecting the hardness of an instance in terms of logic constraints. Therefore, the tabu tenure of a move depends on two factors: the number of constraints containing the variable of the move and the flipping frequency of the variable.

The idea for the first part of the function *C(t)* is the following. A variable involved in a large number of constraints has naturally more risk to be flipped during a move than a variable having few constraints on it. It is thus logic to give a longer tabu tenure for a move whose variable has many constraints on it. Therefore, each time x_t is flipped from 1 to 0, x_t cannot be reset to 1 until at least

C(t) iterations have been realized. Similarly, a move concerning a variable involved in a low number of constraints is given shorter tabu tenure.

The second part of the function is dynamic and proportional to the flipping frequency of the variable x_t (from 1 to 0). The basic idea is to penalize a move which repeats too often. The coefficient α is instance-dependent and defines a penalty factor for each move. For the instances we solved, the values of α vary from 10 to 30.

At this stage, let us notice that *FREQ(t)* is the essential part in the above tabu tenure function. Other frequency-based tabu tenures are also possible. For instance, the following ones have been experimented.

k2(t) = FREQ(t)
k3(t) = a×FREQ(t) (a is a constant)
k4(t) = C(t) + FREQ(t)

In practice, we observe that all these formulae lead to comparable results in term of quality of the solutions found. However, according to the formula used, the resolution time may be quite different. Indeed, for a same solution quality, the search using formula *k(t)* requires in general much shorter time.

It should now be clear that using the frequency of a move to tune the tabu tenure of the move may be considered as a general principle and thus applicable to other problems.

Finally, in order to implement the tabu list, a vector T of m elements is used. As suggested in [Glover & Laguna 97], each element T(i) ($1 \le i \le m$) records *k(i) + iter*, instead of *k(i)* where *iter* is the current number of iterations. In this way, it is very easy to know if a mv(i) is tabu or not at iteration j: if T(i) > j, mv(i) is a forbidden move; otherwise, mv(i) is a possible move.

3.3.5 Aspiration criteria

The tabu status of a move mv(i) is canceled if one of the two conditions (two-level aspiration criteria) below is verified.

1) if the move mv(i) leads to a configuration s strictly better than the best configuration s* found so far, i.e. f(s) > f(s*) even if s requires more memory,
2) if the move mv(i) leads to a configuration s of the same quality but s occupies less memory, i.e. f(s) = f(s*) and $\Sigma_{1\le i\le m}\ c_i \cdot x_i < \Sigma_{1\le i\le m}\ c^*_i \cdot x^*_i$.

3.3.6 Heuristics for carrying out a move

Let s be the current configuration at a given iteration. Let M be the set of candidate moves from s, i.e. the moves which are not forbidden by the tabu list and the moves which verifies the aspiration criteria. Now, all the best moves (i.e. those having the highest δ(i) values) are identified and then one of them is chosen at random. Once such a move is determined, a best neighbor s' of s is obtained (s' = s + mv(i)). Before moving to the next iteration, corresponding data structures such as T (tabu list) and δ (table of move values) are updated accordingly.

3.3.7 Intensification and diversification

The tabu mechanism may lead to a state where no move is admissible. This occurs when each possible move has been tried a large number of times without improving the best configuration found. When this happens, an intensification phase will be started.

The intensification used by the algorithm is based on a heuristic using long-term information: if an element is present in a large number of good configurations, then it is highly possible that this element is part of an optimal configuration. Heuristics of this kind are presented in [Glover & Laguna 97] and used by many TS algorithms.

To implement this heuristic, the algorithm uses a vector ker* of m elements ker* = $(k_1, k_2,...k_m)$ to collect high frequency elements. This vector is updated dynamically, following one of the two ways below.

- Each time the algorithm finds a configuration s = $(x_1, x_2,...,x_m)$ giving the same profit as the best one found, i.e. f(s) = f*, ker* is updated as follows: ker* = $(x_1.k_1, x_2.k_2, ...,x_m.k_m)$.
- Each time the algorithm finds a configuration s = $(x_1, x_2,...,x_m)$ having a better profit than the best one found, i.e. f(s)>f*, ker* is reset to s, i.e. ker* = s.

From the vector ker*, we construct the search space I (for intensification) of indices of ker* in the following way. Let To_1 be the set of indices of ker* that are equal to 1 and To_0 those equal to 0. Then the set I $\subseteq$ To_0 contains such indices of elements that the flipping of these elements does not affect the elements of To_1 even after repairing the logic constraints. The set I corresponds thus to regions of the search space where the elements of ker* are fixed. During an intensification phase, the algorithm runs on the set I and from the initial configuration s0 = ker*. In this way, the intensification forces the search to exploit exclusively the areas around the kernel ker*.

However, it is possible that ker* corresponds to a set of configurations trapped in a local optimum. It is for this reason that the algorithm builds dynamically another set D (for diversification) which collects the indices of elements having a flipping frequency lower than the average. Thus the set D corresponds to less visited regions of the search space. During a diversification phase, the algorithm runs exclusively on the set D and from the initial configuration s0 = 0. In this way, the diversification drives the search to explore areas that are either new or have not been visited very often.

3.3.8 Relaxation of the capacity constraint

During the search, the capacity (knapsack) constraint may be violated by the current configuration s = $(x_1, x_2,...,x_m)$, i.e., the total size of s may exceed the maximal allowed capacity ($\Sigma_{1\leq i\leq m}$ c_i . x_i>*Max_capacity*). To satisfy the capacity constraint, the following mechanism is devised. Each time the current configuration is improved, the capacity constraint is checked. If the constraint is violated, the configuration is immediately repaired by suppressing the elements x_i which have the worst ratio g_i/c_i until the capacity constraint is satisfied, i.e. $\Sigma_{1\leq i\leq m}$ c_i . x_i ≤ *Max_capacity*.

Note that since we have only one knapsack constraint, the relaxation handling is much simpler compared with the techniques used for the MKP [Pirkul 87, Glover & Kochenberger 96, Hanafi et al. 96, Hanafi & Fréville 98]. Indeed, for the MKP, one must decide how many and which knapsack constraints are to be relaxed. Non trivial special techniques are also needed to deal with the relaxed constraints.s

3.3.9 General algorithm

The TS algorithm follows a general schema composed of three iterative phases: exploration, intensification and diversification. The skeleton of the TS algorithm is given below.

```
                         General TS search algorithm

• s = (0...0)
• repeat the following steps

 1. Exploration:
    • search over all the elements of the partially constrained space C
      (§3.2) whenever a non tabu move exists
    • compute the set I and the set D (§3.3.7)
    • reset tabu list
    • set s = ker*  /* ker* is constructed during the search */
    • go to Intensification

 2. Intensification:
    • search over the elements of the restricted space I (§3.3.7) whenever
      a non tabu move exists
    • reset tabu list
    • set s = (0...0)
    • go to Diversification

 3. Diversification:
    • search over the elements of the restricted space D (§3.3.7) whenever
      a non tabu move exists
    • reset tabu list
    • set s = best solution found during this diversification step
    • go to Exploration
```

```
                              TS search engine

1. Let s and s* be respectively the current and the best configurations;
2. Let M(s) be the set of candidate moves from s (M(s) induces a particular
   search space C, I or D, see §3.3.6)
3. while ∃ possible moves in M(s) do
     choose a best move mv(i) (break ties randomly) (c.f. §3.3.3, §3.3.6)
     s' = s + mv(i);
     s = s';
     update tabu list T and table of move values δ; (c.f.§3.3.3, §3.3.4)
     update ker*;                                  (c.f. §3.3.6)
     handle (relaxed) capacity constraint          (c.f. §3.3.8)
     if s is better than s* then s * = s;
```

The general stop condition is defined by a maximum number of iterations allowed. The algorithm returns the best solution s* found during the search. An alternative condition can be used to stop the algorithm when a given value of the profit function is reached.

The different phases (exploration, intensification and diversification) use the same tabu search engine (with different search spaces C, I and D). Each phase is triggered and stopped automatically by the tabu list management (§ 3.3.4), i.e. whenever no more moves are admissible. This is particularly the case for intensification and diversification. This approach is quite different from previous work on MKP where the rhythm of intensification and diversification phases are controlled by some supplementary parameters [Glover & Kochenberger 96, Hanafi et al. 96, Hanafi & Fréville 98]. Finally, it should be noted that the tabu list management is claimed to be dynamic, however, in a different meaning than that described e.g. in [Dammeyer & Voss 93].

4. Experimentation and results

4.1. Test data

Experiments are carried out on a set of 20 realistic instances[2] provided by the French National Space Agency CNES and described in details in [Bensana et al. 98]. We sketch here only some main characteristics of these instances.

These instances belong to two different sets: without capacity constraint (13 instances) and with capacity constraint (7 instances).

1) **Set No.1 (without capacity constraint)**: The first set includes 13 instances having 67 to 364 candidate photographs, giving up to 809 binary variables and 14175 constraints.
2) **Set No.2 (with capacity constraint)**: The second set includes 7 instances having 209 to 1057 candidate photographs, giving up to 2355 binary variables and 35933 constraints.

For the instances of the first set, the optima are known thanks to two exact algorithms: a CPLEX commercial software using an ILP formulation and a non-standard Branch and Bound algorithm using a Valued Constraint Satisfaction Problem (VCSP) formulation [Verfaillie et al. 96]. For the instances of the second set, existing exact algorithms are unable to solve optimally these instances except the smallest one, due to the existence of the capacity constraint and the large size. For these instances, only sub-optimal solutions are known, which have been obtained by another tabu search algorithm [Bensana et al. 96].

4.2. Experimental settings

Our TS algorithm is programmed in C, and compiled using VisualC++ on a PC running Windows NT (32 MB RAM, 200 MHz). To obtain our computational results, the TS algorithm is run 100 times on each instance with different random seeds. Two stop conditions are used, when a given profit value is reached or when a given maximum number of iterations are realized. The tabu tenure is managed dynamically and automatically according to the formula given in §3.3.4.

[2] These instances are available via ftp from ftp.cert.fr/pub/lemaitre/LVCSP/Pbs/SPOT5.tgz

4.3. Results on instances without capacity constraint

Since the optima are known for this set of instances, the algorithm is stopped when a solution of an optimal profit value is found. We wish to know whether our algorithm is able to find the optimal solutions for the instances in this set. Table 1 shows the results of the TS algorithm on these instances.

For each instance, the following information is given. The first 4 columns give the name of the instance, the number of candidate photographs n in P, the number of 0/1 variables m and the known optimum. Columns 5-6 give the best (maximal) profit found by the TS algorithm, the minimal number of iterations needed to find a solution of such a profit. Columns 7-8 indicate the number of iterations and the running time needed to find a solution averaged over 100 runs.

				best		average perfor.		nb photo selected		similarity		distinct
Pb.	n	m	opt.	profit	iter	iter	time (s)	max.	min.	max.	min.	solution
54	67	125	70	70	71	280	1	45	45	45	29	63
29	82	120	12032	12032	35	81	1	34	34	34	18	97
42	190	304	108067	108067	110	291	1	80	80	76	38	100
28	230	346	56053	56053	88	1296	1	47	46	46	7	95
5	309	809	115	115	256	26007	7	96	93	71	15	100
404	100	158	49	49	36	596	1	33	31	31	15	100
408	200	328	3082	3082	684	8479	1	63	60	52	23	100
412	300	544	16102	16102	831	33278	5	79	77	67	23	100
11	364	692	22120	22120	1540	133889	29	98	95	84	33	100
503	143	259	9096	9096	84	907	1	70	69	55	25	100
505	240	448	13100	13100	910	25116	3	85	82	66	28	100
507	311	573	15137	15137	1885	97489	18	92	89	73	32	100
509	348	652	19125	19125	2094	104408	22	96	93	78	29	100

Table 1: Results on instances without capacity constraint

Columns 9-10 indicate the maximal and minimal numbers of selected photographs among 100 solutions. Columns 11-12 show how similar these solutions are when they are compared in pairs: the maximum and the minimum of the (100*99)/2 bit-wise comparisons are given. The last column indicates the number of distinct solutions among the hundred solutions found.

Thus the fifth line means that the instance called 28 has 230 mono and stereo candidates corresponding to 346 binary variables and an optimal schedule has a profit of 56053. The TS algorithm finds a solution of profit 56053 (optimal value) after 88 iterations (in average 1296 iterations and one second over 100 runs). Any schedule among 100 solutions found contains at least 46 and at most 47 photographs. When the 100 schedules are compared in pairs, at least 7 and at most 46 pairs (photo, camera) are shared by two schedules. Finally, among (100*99)/2 possible pairs of schedules, 95% of them are different.

Now let us make several comments about these results. First, the TS algorithm is very efficient and robust for these instances. Indeed, for each instance and for each of 100 runs, the algorithm is able to find an optimal solution. (It is for this reason that only the average iterations and computing times over 100 runs, but not the average profit are given in the table.)

Secondly, the TS algorithm is fast. Indeed, both the number of iterations and computing time needed to find an optimal solution are quite low (fewer than 140000 iterations for 29 seconds for the largest instance). On average, the algorithm performs 3715 to 8372 iterations per second.

Thirdly, these instances seem highly constrained since only a small number of candidate photographs are selected in an optimal solution (col. 9-10). We observe also that except for three instances, all the optimal solutions found (100 for each instance) are different, though they share always a subset of pairs (photo, camera). This implies that these instances probably have many optimal solutions, therefore are easy to solve.

4.4. Results for instances with capacity constraint

Contrary to the instances without capacity constraint, no optimal solutions are known for this set of instances except for the smallest one. Only sub-optimal ones are available, the best of them being produced by another TS algorithm developed by the CNES (we discuss this algorithm later in §5.1).

To solve an instance, our TS algorithm is allowed to run 9 million iterations. This corresponds to about one hour of computing time for the largest instance (delay considered reasonable by the CNES and used by the TS algorithm of the CNES). Once again, the TS algorithm is run 100 times on each instance with different random seeds. Table 2 gives the results of the TS algorithm together with the best known results (sub-optima) for these instances.

			best known		best		average			worst
Pb.	n	m	profit	time (s)	profit	iter*	profit	iter.	time (s)	profit
1401	488	914	174058	846	176056	4690	176055	547882	120	176053
1403	665	1317	174137	1324	176137	177354	176134	816099	332	176133
1405	855	1815	174174	1574	176179	519055	176175	1418907	1314	176171
1021	1057	2355	174238	2197	176246	8279411	176241	1707156	2422	176234
1502	209		61158*	13	61158	235	61158	1067	1	61158
1504	605	1253	124238	1011	124243	196064	124241	1092197	405	124239

Table 2: Results on instances with capacity constraint

For each instance, the following information is indicated. The first 3 columns have the same meaning as for Table 1. Columns 4-5 show the profit and computing time (in seconds)[3] of the best known solutions. Columns 6-7 give, for the best solution found by our TS algorithm after 100 runs, the profit value and the number of iterations needed to find such a solution. Columns 8-10 show the averaged profit, number of iterations and computing time over 100 runs. The last column gives the profit of the worst solution found by the TS algorithm over 100 runs.

From the data in Table 2, we may make the following remarks. First, these instances are much harder than those of the first set, essentially due to the large number of variables and "logic constraints" (up to 2355 0/1 variable or 1057 integer variables, and more than 30 000 constraints). Secondly, all the solutions (even the worst ones) produced by our TS algorithm improve on the best known results, both in terms of solution quality and speed of execution. The quality improvement is important, reaching a gain of more than 2000 units of the profit function in most cases. Moreover, the computing time needed to obtain such a solution satisfies by far the "one hour" constraint (40 minutes for the largest instance). Thirdly, the TS algorithm seems very robust because the quality variations of the 100 solutions for each instance remain small.

[3] The TS algorithm of the CNES is programmed in Fortran 77 and was run on a Sparc 20/50 workstation.

In order to see how difficult it is to reach a previously published best solution, another experiment is carried out. We re-ran 100 times the algorithm on each instance and stopped a run when the best known profit value is reached. Averaged results of this experiment is reported in Table 3.

Pb.	cost	time (s)	ratio best-known/TS
1401	174058	1	846
1403	174137	4	331
1405	174174	30	52
1021	174238	39	55
1502	61158	1	13
1504	124238	49	21
1506	165244	196	10

Table 3: Effort to reach a previously best result

From this table, we see that only one second upto a few minutes are needed to obtain the previously best solutions. Compared with the previous computing times (column 4), the running times of our TS algorithm represent only a very small fraction (last column).

Table 4 gives more information about the solutions found by indicating the maximal and minimal numbers of selected photographs among 100 solutions, and the similarity between these solutions.

			nb. photos		similarity		distinct
Pb.	n	m	max.	min.	max.	min.	solution
1401	488	914	148	144	127	40	100
1403	665	1317	213	208	170	65	100
1405	855	1815	253	242	189	71	100
1021	1057	2355	318	300	225	82	100
1502	209	413	166	166	114	69	100
1504	605	1253	280	274	188	102	100
1506	940	2060	314	299	221	95	100

Table 4: More information about solutions found

From Table 4, we see that as for the instances without capacity constraint, these instances are also highly constrained. Indeed, only about one third to one half of candidate photographs are selected in a solution. Once again, the solutions found for each instance always share some elements (photo, camera), but the solutions are all different. This implies that these instances have probably many (sub-optimal) solutions for a given profit value. This may be considered as an indicator that these results may be further improved.

5. Discussions

5.1. Related work

As mentioned before, both exact and non exact algorithms have been developed for the DPSP. We describe here briefly two such algorithms: an exact algorithm called Pseudo-Dynamic Search (PDS) [Verfaillie et al. 96] and another TS algorithm developed by the CNES [Bensana et al. 96]. Let us notice first both algorithms are based on an integer formulation of the problem: each variable represents a candidate photograph and is associated to a value domain of {0,1,2,3} or {0,13} according to whether it is a mono or stereo photograph. A configuration in the search space corresponds thus to a n-dimensional integer vector for a problem composed of n candidate photographs.

The PDS algorithm is an hybridization of dynamic programming and Branch & Bound (B&B) techniques. This algorithm consists in performing n searches (n being the number of variables of the problem), each solving, with a depth first B&B, a sub-problem limited to a subset of the n variables. Once a sub-problem is solved, its best profit value is recorded. This value is then used later as a lower bound for another unsolved sub-problem. This algorithm has proven the optimality for the 13 instances without capacity constraint (but failed for the instances with capacity constraint).

The TS algorithm of the CNES, denoted by TS-CNES hereafter, is quite different from our algorithm. First, TS-CNES uses a different (integer) *formulation* of the problem. Secondly, it manipulates only *feasible* configurations, i.e. those verifying all the constraints of the problem. Thirdly, it uses a different neighborhood. Fourthly, it considers only a *subset* of neighboring configurations to make a move. This is probably because no incremental evaluation technique is available. Fifthly, the tabu tenure for each move is randomly taken from predefined (very small) ranges: (3..5) for problems of size less than 500 candidate photographs and (7..10) for larger ones.

The TS-CNES algorithm has been compared with various forms of greedy and simulated annealing algorithms and proved to be the best non-exact algorithm. Compared with our TS algorithm, however, TS-CNES is less effective in terms of search power and speed of execution. Indeed, as shown in §4, our TS algorithm significantly improves upon the best results of the TS-CNES algorithm for both sets of instances (let us mention also that TS-CNES failed to find an optimal solution for 4 of the 13 instances without capacity constraint [Bensana et al. 96]).

5.2. Upper bounds

The computational results reported in this paper are much better than the previous best ones, in particular for the instances of the second set with capacity constraint. Now, we can ask how far these results are from optimal solutions. Currently, no definitive answer is known yet since all previously attempts for finding optimal solutions failed. Another possible answer to this question consists in seeking *strong* upper bounds. In this section, we review the known studies in this area.

One obvious solution concerns the continuous linear programming. However, it is well known that this approach could lead to a significant gap between the optimal discrete and continuous values. Eric Bensana et al. have applied this approach to the problem described in this paper [Bensana et al. 96, Verfaillie 99]. They have used CPLEX, an efficient implementation of the simplex algorithm, to solve the relaxed problem (integrity constraints relaxed). Eric Bensana et al. reported continuous optimal values (COV) which are largely above the discrete optimal values (DOV) for the instances of the first set (Recall that optimal values are known for these instances). In fact, the ratio (COV-DOV)/COV goes from 28% to 74% ! Even if they have not reported bounds on the instances of the second set, it is reasonable to believe that LP bounds on these instances should be much worse, given the fact that the instances of the second set are much larger and much more difficult.

Very recently, another attempt has been reported, which tries to tighten the bounds with a new ILP formulation of the problem and the column generation technique [Gabrel 99]. In this new formulation, the initial problem is decomposed into three independent sub-problems, one per camera. For the resolution of the problem, CPLEX 4.1 is employed, together with a column generation procedure. This approach improves the previous bounds for the instances of the first set, reducing the (COV-DOV)/COV ratio to the range of 0% to 20%. However, like for the initial ILP formulation, no bound was reported for the instances of the second set, due to the huge number of columns that have to be generated during the resolution.

Currently, studies continue on upper bounds for the instances of the second set. It should be clear that deriving good upper bounds for these instances is quite difficult and constitutes itself a challenging research topic.

5.3. Repairing logic constraints

As indicated in §3.3.2, binary or ternary constraint violations are repaired when a move is carried out. A binary constraint violation is easy to repair since there is a single way to do this. However, the case for a ternary constraint violation is more complicated since there are different ways to achieve such a repair.

Suppose that $x_j = x_k = 1$ and the move sets x_i to 1. The constraint $x_i + x_j + x_k \leq 2$ is thus violated and needs to be repaired. Four cases are possible:

1) set both x_j and x_k to 0,
2) randomly set either x_j or x_k to 0,
3) set to 0 the element which has a smaller profit,
4) look ahead in an exhaustive way to determine the best choice in terms of lost profit.

We have tested and compared these possibilities on all the instances used in this study. We observe that 3) and 4) give significantly better results than the other ones. In choosing between 3) and 4), the technique 4), which is more time consuming, gives slightly better results than 3) in a few cases. For the instances used in this study, experiments showed that no more than 5 elements need to be reset to repair a ternary constraint violation after a move, implying that an exhaustive look ahead is still possible. For the cases where more elements need to be forward-checked, the technique 3) should be the best choice.

5.4. Handling the capacity constraint

The algorithm presented so far works within the partially constrained search space C (§3.2.1 Definition 3). The capacity constraint is relaxed, i.e. during the search, the total capacity of the current configuration may exceed the maximal allowed capacity. An alternative is to impose the satisfaction of the capacity constraint during the search. In this case, the algorithm works within the totally constrained search space and manipulates only feasible configurations. However, the relaxation of the capacity constraint helps to obtain better results and to accelerate the search. Intuitively, optimal or high quality sub-optimal solutions are located at the frontier of feasibility. Often these solutions are difficult to reach uniquely from the feasible side. A more effective way is to allow the search to oscillate around the feasibility frontier, increasing the chance to reach good solutions. Strategies of this kind have been successfully applied to the MKP [Glover & Kochenberger 96, Hanafi & Fréville 98]. We show below that this is also the case for our application, even if the relaxation technique we used is different from those developed for the MKP.

To illustrate these points, experiments have been carried out to compare two versions of our algorithm with and without capacity constraint relaxation. We re-run these two versions for the same number of iterations and compare the quality of the solutions found. Fig. 1 and 2 show the comparative results on the instance 1504 (20 runs for each version of the algorithm, each run being given 300 000 iterations).

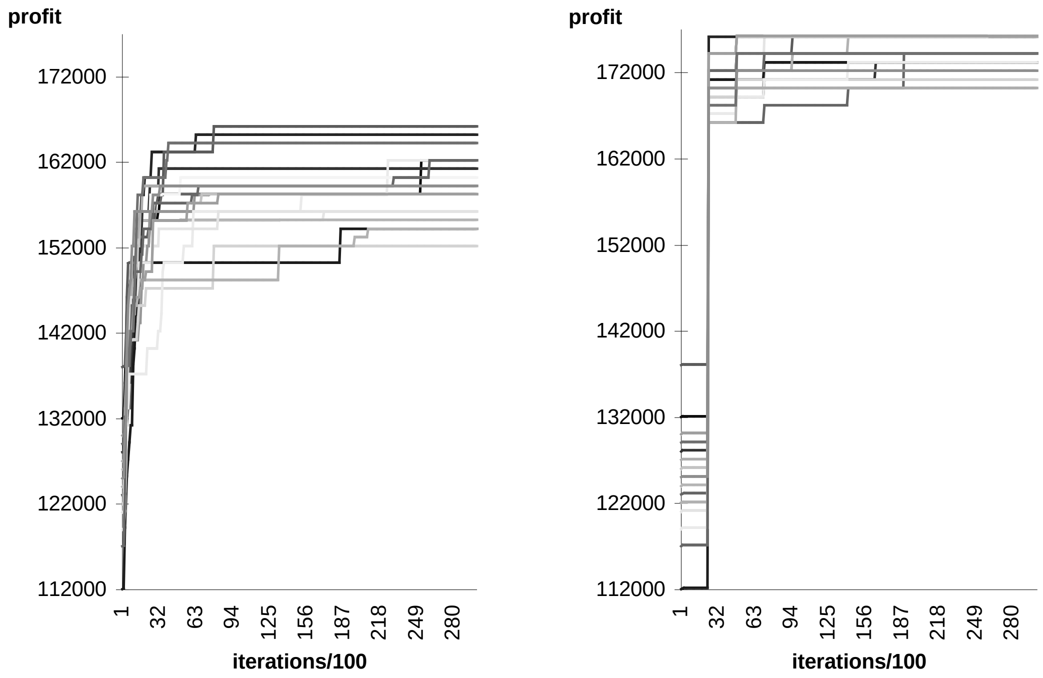

Fig. 1 & 2: Comparative results without (left) and with (right) capacity constraint relaxation

From these figures, we observe that the relaxation technique gives solutions of better quality for the same search effort. Moreover, the relaxation version of the algorithm needs fewer iterations to reach a solution of the same quality. Finally, the relaxation version is much more robust since it is less sensitive to the initial random seeds used.

6. Conclusions

In this paper, we have introduced a "logic constrained" knapsack formulation for a real world application, the photograph daily scheduling problem of the satellite Spot 5. Based on this formulation, we have developed a highly effective tabu search algorithm. This algorithm incorporates some important features including an efficient neighborhood, a fast and incremental technique for move evaluation, a method for automatic tabu tenure management, and intensification and diversification mechanisms.

Computational results on a set of realistic benchmark instances have showed the effectiveness of this algorithm. For the instances without capacity constraint, the algorithm finds all optimal solutions very easily. For the instances with capacity constraint, the algorithm finds with little computational effort previously best known results. More importantly, the algorithm significantly improves upon the best known results using less than one hour of computing time on a pentium PC.

An analysis of the solutions (100 for each instance) produced by the algorithm showed that only one third to half of the candidate photographs are retained in a schedule, indicating that these instances are highly constrained. We observed that the solutions found for a given instance are quite different, though they do share some elements in common. This implies that the density of solutions for a given profit is probably high. Therefore, we conjecture that if the obtained results (for the instances with capacity constraint) are not optimal, they may be further improved by more powerful search methods.

Unfortunately, for the moment nothing is known about the distance between the solutions found and optimal solutions. An exact algorithm would allow us to answer definitively this question. Given the hardness of these instances, a more realistic approach would be to develop more powerful techniques for deriving strong upper bounds.

Acknowledgment

We would like to thank gratefully the following people: Dr. Gérard Verfaillie from CERT Toulouse for many useful discussions on this application, Prof. Fred Glover and the referees of the paper for their very useful comments and careful corrections, Steve Hurley for his corrections on the final version of the paper. Thanks go also to the CNES for having defined and made available the tested instances.

References

[Bensana et al. 96] E. Bensana, G. Verfaillie, J.C. Agnèse, N. Bataille, and D. Blumstein. Exact and approximate methods for the daily management of an earth observation satellite. In *Proc. of the 4th Intl. Symposium on Space Mission Operations and Ground Data Systems (SpaceOps-96*), Munich, Germany. ftp://ftp.cert.fr/pub/verfaillie/spaceops96.ps.

[Bensana et al. 98] E. Bensana, M. Lemaître and G. Verfaillie, Earth observation satellite management. *Constraints: An International Journal*, 4(3): 293-299, 1999.

[Chu & Beasley 98] P.C. Chu and J.E. Beasley, A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics* 4: 63-86.

[Dammeyer & Voss 93] F. Dammeyer and S. Voss, Dynamic tabu list management using reverse elimination method. *Annals of Operations Research* 41: 31-46.

[Fréville & Plateau 94] A. Fréville and G. Plateau, An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. *Discrete Applied Mathematics* 49: 189-212.

[Fréville & Plateau 97] A. Fréville and G. Plateau, The 0-1 bidimensional knapsack problem: toward an efficient high-level primitive tool. *Journal of Heuristics* 2: 147-167.

[Gabrel 99] V. Gabrel, Improved linear programming bounds via column generation for daily scheduling of earth observation satellite. *Research Report 99-01*, LIPN, Université de Paris XIII, January 1999.

[Glover & Kochenberger 96] F. Glover and G.A Kochenberger, Critical event tabu search for multidimensional knapsack problems. In *Meta-heuristics: Theory and Applications*, I.H. Osman and J.P. Kelly (Eds), Kluwer Academic Publishers, pp407-428.

[Glover & Laguna 97] F. Glover and M. Laguna, *Tabu Search*. Kluwer Academic Publishers.

[Lokketangen & Glover 98] A. Lokketangen and F. Glover, Solving zero-one mixed integer programming problems using tabu search. *European Journal of Operational Research*, Special Tabu Search Issue, 106(2-3): 627-662.

[Hanafi et al. 96] S. Hanafi, A. Freville and A. EI. Abdellaoui, Comparison of heuristics for the 0-1 multidimensional knapsack problem. In *Meta-heuristics: Theory and Applications*, I.H. Osman and J.P. Kelly (Eds), Kluwer Academic Publishers, pp449-465.

[Hanafi & Freville 98] S. Hanafi and A. Freville, An efficient tabu search approach for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, Special Tabu Search Issue, 106(2-3): 663-697.

[Martello & Toth 90] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Wiley Chichester.

[Lemaître & Verfaillie 97] M. Lemaître and G. Verfaillie, Daily management of an earth observation satellite: comparison of Ilog solver with dedicated algorithms for valued constraint satisfaction problems. In *Proc. of the 3rd Ilog Intl. Users Meeting*, Paris, France. ftp://ftp.cert.fr/pub/verfaillie/ilog97.ps.

[Pirkul 87] H. Pirkul, A heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Naval Research Logistics* 34: 161-172.

[Verfaillie et al. 96] G. Verfaillie, M. Lemaître and T. Schiex, Russian doll search for solving constraint optimization problems. *In Proc. of the 13th National Conference on Artificial Intelligence (AAAI-96)*, Portland, USA, pp181-187.

[Verfaillie 99] G. Verfaillie, Personal communications. April 1999.