Università di Pisa

Dipartimento di Informatica

Technical Report: TR-01-10

A local search based heuristic for the demand constrained multidimensional knapsack problem

Paola Cappanera, *DI - Università di Pisa*
e-mail: `cappaner@di.unipi.it`

Marco Trubian, *DEI - Politecnico di Milano*
e-mail: `trubian@.elet.polimi.it`

August 1, 2001

ADDRESS: Corso Italia 40, 56125 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

A local search based heuristic for the demand constrained multidimensional knapsack problem

Paola Cappanera, *DI - Università di Pisa*
e-mail: `cappaner@di.unipi.it`

Marco Trubian, *DEI - Politecnico di Milano*
e-mail: `trubian@.elet.polimi.it`

August 1, 2001

Abstract

We consider an extension of the 0-1 multidimensional knapsack problem in which there are greater-than-equal inequalities, called *demand constraints*, besides the standard less-than-equal constraints. Moreover the objective function coefficients are not constrained in sign. This problem is worth considering because it is embedded in the models of practical applications, it has an intriguing combinatorial structure and it appears to be a challenging problem for commercial ILP solvers. Our approach is based on a nested tabu search algorithm in which neighborhoods of different structure are exploited. A first tabu search procedure is carried on in which mainly the infeasible region is explored. Once feasibility has been gained, a second tabu search procedure, which analyses only feasible solutions, is applied. The algorithm has been tested on a wide set of instances. Computational results are discussed.

Keywords: integer programming, heuristic algorithms, multidimensional knapsack.

1 Introduction

We consider an extension of the multidimensional knapsack problem in which there are greater-than-equal inequalities, besides the standard less-than-equal constraints. Moreover the objective function coefficients are not constrained in sign. We call it *Multi Demand Multidimensional Knapsack* problem, *MDMKP* for short. It can be formulated as

$$\text{MDMKP} \qquad \max \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad \forall i \in \{1, \ldots, m\} \tag{1}$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i \qquad \forall i \in \{m+1, \ldots, m+q\} \tag{2}$$

$$x_j \in \{0,1\} \qquad \forall j \in \{1, \ldots, n\} \tag{3}$$

where $b_i > 0 \ \forall i \in \{1, \ldots, m+q\}$, $a_{ij} \geq 0 \ \forall i \in \{1, \ldots, m+q\}$, $\forall j \in \{1, \ldots, n\}$.
A *well-stated MDMKP* assumes that $\sum_{j=1}^{n} a_{ij} > b_i \ \forall i \in \{1, \ldots, m+q\}$, $\max_j\{a_{ij}\} \leq b_i \ \forall i \in \{1, \ldots, m\}$, $\min_j\{a_{ij}\} < b_i \ \forall i \in \{m+1, \ldots, m+q\}$, since any violation of these conditions will result in some x_j's being fixed to zero or some constraints being never satisfied or redundant but for trivial solutions. Each one of the m constraints of family (1) is called *knapsack constraint*, while we refer to each one of the q constraints of family (2) as *demand constraint*.
While the multidimensional knapsack problem (*MKP*), in which only the knapsack constraints are present, has received wide attention in the literature, to our knowledge, no literature exists specifically devoted to *MDMKP*. A rich survey on the *MKP* can be found, e.g., in the paper of Chu and Beasley (1998) [7]. Hereafter we briefly recall some of the main contributions to the solution of *MKP*, being its structure embedded in that of the proposed problem.
The decision version of the well known knapsack problem ($m = 1$) is weakly NP–complete and effective approximation algorithms for the optimization version have been proposed, see e.g. Martello and Toth (1990) [21] for an extended review on the subject and more recently the works of Pisinger (1999) [24] and Martello, Pisinger and Toth (2000) [20]. Also the decision version of the *MKP* is weakly NP–complete, but indeed Korte and Schrader (1981) [16] have shown that no fully polynomial approximation

scheme exists for *MKP* unless P=NP, whereas polynomial approximation schemes are known for m fixed (see e.g. Magazine and Chern (1984) [18]).
Exact algorithms for the *MKP* have been introduced since long ago, being, as far as we are aware, the papers of Dantzig (1957) [9] and Balas (1965) [1] the earliest references. Almost every exact technique has been applied to the *MKP* including a dynamic programming based method (Gilmore and Gomory (1966) [12]), an enumerative algorithm based on the Fourier-Motzkin elimination (Cabot (1970) [5]) and branch and bound frameworks, see e.g. Shih (1979) [27] and Gavish and Pirkul (1985) [11].
Works which study the theoretical relationships among the bounds generated by different relaxations (Lagrangean, surrogate and their composition) have also been proposed, see Gavish and Pirkul (1985) [11] and Crama and Mazzola (1994) [8].
Recently, Osorio et al. (2000) [22] generated cuts from surrogate constraint analysis and constraint pairing ideas. They used the cut generator in combination with a commercial ILP solver and analyzed the quality of the improvement in the performances obtained.
A number of heuristic algorithms has been developed for the *MKP*. The bound based heuristics make use of the information given by linear, Lagrangean or surrogate relaxations (see e.g. Balas and Martin (1980) [2], Magazine and Oguz (1984) [19] and Pirkul (1987) [23]). In this context Freville and Plateau (1994) [10] introduced efficient preprocessing procedures for fixing variables at their optimal values in large size instances. Also tabu search heuristics (see e.g. Voß(1993) [28], Glover and Kochenberger (1996) [13], Løkketangen and Glover (1998) [17], Hanafi and Freville (1998) [14]) and genetic algorithms (see e.g Chu and Beasley (1998) [7] and Haul and Voß(1998) [15]) have been applied to the *MKP*. In particular, in order to test their genetic algorithm, Chu and Beasley (1998) [7] generated 270 test problems of larger sizes and of more difficult types than the standard problems used by many authors in the past to validate their approaches. Such a set of large *MKP* problems, which has been generated using the procedure suggested by Freville and Plateau (1994) [10], is publically available in the ORlibrary (Beasley (1995) [3]).
As it is the case for the *MKP*, the structure of the *MDMKP* is embedded in the models of many practical applications, e.g. obnoxious and semi obnoxious facility location problems (see Romero-Morales et al. (1997) [26], Cappanera (1999) [6] and Plastria (2001) [25]), capital budgeting and portfolio selection problems (see Beaujon et al. (2001) [4]), thus suggesting that the *MDMKP* is worth considering at least as well as *MKP*. Moreover the *MDMKP* can be seen as a general model for any kind of binary problems with positive coefficients on the constraint matrix and on the right hand side vector.

Since in the *MDMKP* there are knapsack constraints which force us to keep low the number of variables set to one and demand constraints which drive in the opposite direction even identifying feasible solutions can become a hard task. Not only to identify feasible solutions of the *MDMKP* is an NP-complete problem, since *Subsetsum* problem immediately reduces to the satisfiability of two opposite binary constraints, but it can be, in our experience, a really nasty problem. As we will show analyzing the computational results in the final section, state of the art commercial integer linear programming solvers can be unable to find a feasible solution (within reasonable computational time) for instances for which a feasible solution exists. And this can happen with a few hundreds of variables and a few tens of constraints. As a consequence the search for feasibility becomes a main concern when trying to solve the *MDMKP*. On this basis we designed a procedure in which the search for feasibility and the optimization of feasible solutions are separated phases. This is the reason why the approach we propose for solving the *MDMKP* is a nested tabu search algorithm in which neighborhoods of different structure are exploited. A first tabu search procedure is carried on in which mainly the infeasible region is explored. In this phase an oscillation method proceeds alternating between constructive and destructive steps. The main objective of this local search is to gain the feasibility as soon as possible and the criterion used to choose variables to add in the constructive phase and variables to drop in the destructive one is only based on the coefficients a_{ij} and b_i disregarding the cost coefficients c_j: only information concerning the feasible region of *MDMKP* are thus taken into account to guide the search. On the other side, once feasibility has been gained, a second tabu search procedure, which implements an intensification strategy, is applied. This second phase attempts at improving the current solution: here moves are performed on a cost basis exclusively and the feasibility is always assured.

We make our computational experiments on general benchmarks derived from unstructured *MKP* instances since our approach has been developed without exploiting the particular structure that instances derived from particular applications may exhibit. To generate the 810 test instances which we used we modified the Chu and Beasley (1998) [7] data set, by adding the demand constraints.

In the next section we describe the proposed local search based algorithm. In the third section we describe some dominance criteria used to fix variables which are derived from a surrogate analysis of conflicting constraints. In the fourth section we give an account of the way test instances have been generated, we present the computational experiments and analyze the results. Finally some conclusions are drawn.

2 A Tabu Search heuristic for MDMKP

The Nested Tabu Search heuristic provided is composed of two parts, which we call *inner* and *outer* part, respectively. In the outer part an oscillation method proceeds alternating between *constructive* and *destructive* phases, and it aims at generating feasible solutions. The proposed approach is similar to that used by Glover and Kochenberger (1996) [13]. A constructive phase consists of progressively setting variables to one, while a destructive phase progressively sets variables to zero. For this purpose variables are ordered according to pseudoutility values which depend only on the coefficients a_{ij}'s and on the right hand sides b_i's. In the inner part a standard tabu search approach attempts to iteratively improve a starting feasible solution supplied by the outer part: here only feasible solutions are analyzed and the search is driven on the base of cost information.

2.1 The outer local search

The outer part proceeds alternating between constructive and destructive phases, i.e. progressively setting variables to one and to zero, respectively. The goal of this part is to gain feasibility.

Since a feasible solution satisfies both knapsack and demand families of constraints, each phase basically tries to gain the feasibility of a not fully satisfied family of constraints while trying to preserve the possible full satisfaction of the other family. We base our approach on splitting the feasibility test into two separate conditions which we denote as $\leq$-*feasibility* and $\geq$-*feasibility*: a solution is $\leq$-feasible when all the knapsack constraints are satisfied, and it is $\geq$-feasible when all the demand constraints are fulfilled. Clearly a feasible solution is both $\leq$-feasible and $\geq$-feasible.

Figure 1 depicts the framework of the outer local search. With *LScnt* we denote the feasible solutions counter and with `MaxNumLS` the maximum number of feasible solutions which we want to generate. With *TPcnt* we denote the iteration counter of outer local search, and with `MaxNumTP` the maximum number of iterations allowed. With *ILScnt* we denote the counter of the performed inner local searches and with `MaxNumILS` the maximum number of inner local searches which we want to perform. With x^* we denote the best solution found so far and with z^* its objective value, i.e. x^* is the feasible solution that gives a maximum value for cx. Finally, x denotes the incumbent solution and z is its value.

The main framework. In the first three steps the tabu memory and counters are initialized and the all-zero solution is chosen as starting solution (but any other solution can be used as well). Then, in step 4, a loop begins in which five procedures are iteratively called until a stopping criterion is met. Within this loop, which implements the outer local search, all the feasible solutions generated are stored. In the first procedure, *Constructive_Phase1()*, variables (which are not tabu) are iteratively set to one until a solution is found which is not $\leq$-feasible but $\geq$-feasible (critical event). In the second procedure, *Constructive_Phase2()*, `span` variables (which are not tabu) are set to one. In both procedures, if all candidate variables are tabu, those which have been tabu for the longest time are selected. These procedures implement the constructive phase. The third procedure, *Update_span()*, manages the critical parameter `span`. Such a parameter determines the depth of the search within the infeasible region (see below). Also the destructive phase is, at its turn, organized into two parts. In the fourth procedure, *Destructive_Phase1()*, variables (which are not tabu) are iteratively set to zero until a solution is found which is not $\geq$-feasible but $\leq$-feasible (critical event). In the fifth procedure, *Destructive_Phase2()*, `span` variables (which are not tabu) are set to zero. As above, in both procedures, if all candidate variables are tabu those which have been tabu for the longest time are selected.

In both phases a feasible solution can be generated only during the first part, i.e. within procedure *Constructive_Phase1()* or procedure *Destructive_Phase1()*. For each call of these procedures, the first time a feasible solution, namely x, is obtained it is stored.

Step 4 terminates when a total iteration limit has been reached, i.e. when a maximum number of outer phases, `MaxNumTP`, or a maximum number of feasible solutions, `MaxNumLS`, has been generated. Indeed the counter *TPcnt*, is reset to zero after that the first feasible solution has been found.

When the outer local search stops, all the stored feasible solutions are analyzed and on the more promising ones an inner local search is performed. In step 5 all the feasible solutions collected are replaced by their local minima which are then sorted in non increasing order of the objective function value. For each feasible solution its local minimum is computed using the neighborhood $N1$ (to be described in Section 2.2).

Then, in step 6, a loop begins in which the inner local search procedure is iteratively applied to the best (according to the value of the corresponding local minimum) `MaxNumILS` feasible solutions. In step 6.1 the next solution, in the order given by step 5, is compared with the previous ones and, if it is different from all of them, it is given in input to the inner local search procedure. The inner local search is applied to all the (different) feasible solutions generated by the outer local search, if they are less than `MaxNumILS`.

```
procedure Nested_Tabu_Search()
begin
1) x := 0; z := 0; z* := −∞;
2) Initialize_Tabu_Memory();
3) LScnt := TPcnt := ILScnt := 0;
4) repeat
4.1)    Constructive_Phase1(LScnt,x, z, x*, z*);
4.2)    Constructive_Phase2(span,x, z);
4.3)    Update_span(span);
4.4)    Destructive_Phase1(LScnt,x, z, x*, z*);
4.5)    Destructive_Phase2(span,x, z);
4.6)    TPcnt := TPcnt + 1;
   until( ( TPcnt ≥ MaxNumTP ) or ( LScnt ≥ MaxNumLS ) )
5) sort_feasible_solutions();
6) repeat
6.1)    select_next_stored_feasible_solution(x);
6.2)    apply_the_inner_local_search(x, z, x*, z*);
6.3)    ILScnt := ILScnt + 1;
   until( ILScnt ≥ MaxNumILS )
7) return(z*, x*);
end
```

Figure 1: The outer local search

The inner local search explores only the feasible region and eventually updates x^*. Finally the best feasible solution, x^*, generated either within the first or (more likely) within the second loop, is the final output of the procedure, step 7.

Parameter span. Once a critical event has been identified, variables are still added in the constructive phase (loosing more and more the $\leq$-feasibility) and dropped in the destructive one (loosing more and more the $\geq$-feasibility). The number of variables added and dropped respectively (i.e. how far to go within the infeasible solution space) is controlled by the parameter span. This parameter is the most critical one in the outer local search and we manage it as in the oscillation method proposed by Glover and Kochenberger (1996) [13]. At the beginning span is equal to 1 and gradually increases to a limiting value. Once the limiting value has been reached, span is gradually decreased. For each value of span (a series of) alternating constructive and destructive phases are executed.

Intensification vs. diversification. We interpret the outer local search as a diversification phase. Higher is the value of the parameter span and wider is the set of feasible solutions which can be eventually generated and then used as starting points for the inner local search. This approach is not only a strategic choice, but it is imposed by the fact that the solution space of feasible solutions is not connected with respect to the neighborhoods used within the inner local search (as described in Section 2.2). In other words, there is not any guarantee that starting from a feasible solution we can reach an optimal one adopting only the moves used to define the neighborhoods. As a consequence, for a suitable choice of the cost coefficients, the set of all the feasible solutions which can be reached from a given starting solution, by means of the moves used to define the neighborhoods, could contain solutions of only arbitrarily bad quality. For this reason a multistart approach should perform better on the average than any single run approach, when the same neighborhood is used and the same number of iterations is allowed. The oscillation method which we adopt tries to generate as many and different starting solutions as possible with the aim of counterbalance the non connectivity property of the proposed neighborhoods.

Constructive phase. During a constructive phase variables which minimize the loss of the $\leq$-feasibility and maximize the gain of the $\geq$-feasibility are set to one. Figures 2 and 3 contain the pseudocode of procedures *Constructive_Phase1()* and *Constructive_Phase2()*, respectively. We denote with N the overall set of variables, i.e. $N = \{1, \ldots, n\}$, with S_1 the set of variables set to one, and with F_0 the set of variables fixed to zero. Finally, with *span_cnt* we denote the counter of parameter `span`.

```
procedure Constructive_Phase1(LScnt, x, z, x*, z*)
begin
1)      Enable_Dominance_Criterion();
2)      while( not ≥-feasible(x) or ≤-feasible(x)) do
2.1)        Dominance_Test(F_0, S_1);
2.2)        Compute_s_Constructive_Phase(s);
2.3)        Sort variables in not decreasing s order;
2.4)        repeat
2.4.1)          Choose the next j ∉ F_0 in the given order with x_j = 0;
2.4.2)          if( x or (x|x_j = 1) is the first feasible solution in this phase ) then
2.4.3)              Store_feasible_solution(x);
2.4.4)              Disable_Dominance_Criterion(F_0);
2.4.5)              LScnt := LScnt + 1;
                endif
            until x_j is not tabu;
2.5)        if( all variables are tabu ) then Look_for_Oldest_Tabu_Var(); endif
2.6)        x_j := 1;   z := z + c_j;   S_1 := S_1 ∪ {j}
2.7)        Update_Tabu_Memory();
        endwhile
end
```

Figure 2: The first part of the constructive phase

In step 1 of procedure *Constructive_Phase1()* a dominance criterion, to be described in Section 3, is eventually enabled. As a consequence in step 2.1 some variables are set to the value one, some others are forced to the value zero, and the sets S_1 and F_0 are updated consequently. Since dominance criteria have been introduced only to gain feasibility as soon as possible, every time a feasible solution is generated the dominance criterion is disabled (step

```
procedure Constructive_Phase2(span,x, z)
begin
1) span_cnt := 0;
2) while( ( span_cnt < span ) and (S_1 ≠ N ) ) do
2.1)   Compute_s_Constructive_Phase(s);
2.2)   Sort variables in not decreasing s order;
2.3)   Choose the first not tabu j in the given order with x_j = 0;
2.4)   if( all variables are tabu ) then Look_for_Oldest_Tabu_Var(); endif
2.5)   x_j := 1;   z := z + c_j;   S_1 := S_1 ∪ {j};
2.6)   Update_Tabu_Memory();
2.7)   span_cnt := span_cnt + 1;
   endwhile
end
```

Figure 3: The second part of the constructive phase

2.4.4) and the set F_0 is emptied out. During the execution of this phase the $\geq$-feasibility is progressively gained by setting variables to one in step 2.6, while the $\leq$-feasibility is progressively lost. When the $\geq$-feasibility is assured and the $\leq$-feasibility is lost a *critical event* is identified and the procedure terminates. In this situation going on adding variables will only imply a deterioration of the $\leq$-feasibility. Steps 2.2 and 2.3 determine the order in which the candidate variables are examined within the loop of step 2.4. A candidate variable j is selected in step 2.4.1. If no feasible solution has been generated in this phase the procedure tests in step 2.4.2 if the current solution x or the solution which can be obtained setting j to one is feasible or not. If this is the case the feasible solution is stored and the corresponding counter is updated (steps 2.4.3 and 2.4.5). The loop stops only when a not tabu variable is examined (or all candidate variables have been examined). After that, a non tabu variable (or the oldest tabu variable) has been chosen; it is set to one and the tabu memory is updated.

Procedure *Constructive_Phase2()* begins after that a critical event has been generated. After initializing, in step 1, the counter `span_cnt`, in step 2 a loop begins in which at most `span` variables are set to one. This behavior implies a deterioration of the $\leq$-feasibility. Steps 2.1 and 2.2 determine the order in which the candidate variables are selected in step 2.3. If all candidate variables are tabu those which have been tabu for the longest time are selected in step 2.4. Then the chosen variable is set to one, step 2.5, the tabu memory and the counter `span_cnt` are updated, step 2.6 and 2.7.

Destructive phase. Destructive phase is perfectly symmetric to constructive one. As a consequence we do not report the details concerning procedures *Destructive_Phase1()* and *Destructive_Phase2()*. We simply recall that here variables are chosen which minimize the loss of the $\geq$-feasibility and maximize the gain of the $\leq$-feasibility when set to zero. In this phase, when proceeding toward the $\leq$-feasibility by dropping variables, a variable is finally identified such that if dropped the resulting solution would be $\geq$-infeasible. Going on, when the $\leq$-feasibility is finally assured and the $\geq$-feasibility is lost a *critical event* is identified. At this point the second part begins and a loop in which at most `span` variables are set to zero is executed. This behavior results in a further deterioration of the $\geq$-feasibility.

Ranking the variables. When starting from a $\geq$-infeasible solution, either $\leq$-feasible or not, adding variables allows to gain the $\geq$-feasibility eventually to the detriment of $\leq$-feasibility. So in a constructive phase, in all the cases but the one in which a feasible solution has been eventually identified, there is a tradeoff between gaining the $\geq$-feasibility and destroying the $\leq$-feasibility. On such tradeoff is based the procedure *Compute_s_Constructive_Phase()*. As we announced at the beginning of this section only the coefficients a_{ij}'s and the right hand sides b_i's are used to guide the decisions of the method in the outer phase. In particular, given the current solution x and a constraint i we make use of the following information which we refer to as the *residual* of i, i.e.

$$b'_i = b_i - \sum_{j \in N} a_{ij} x_j.$$

Without loss of generality, we assume in this section that $\geq$-constraints have been turned into $\leq$-constraints, so that $b'_i < 0$ always identifies a violated constraint i, whether it is a knapsack or a demand constraint.

In a constructive phase variables are ordered according to non-decreasing pseudoutility values s_j's with $s_j = \alpha_j / \beta_j$ where α_j measures how far we go from the $\leq$-feasibility if x_j was set to one, while β_j measures how much we get closer the $\geq$-feasibility: variable j is chosen for which α_j is minimum and β_j is maximum. Two ranking criteria have been proposed: while in the former one, referred to as *bottleneck criterion*, the most critical constraints are identified in order to select variables, in the latter one, referred to as *surrogate criterion*, surrogate information are used.

Let now go into details concerning the bottleneck criterion. For a given variable j, the $\leq$-constraint which gives the greatest loss of the $\leq$-feasibility, namely i, is the one for which the ratio of a_{ij} to the absolute value of the residual b'_i is maximum (step 1.1). On the other hand, as far as β_j is concerned, if the $\geq$-feasibility is assured, setting a variable to one would only improve the fulfillment of the $\geq$-constraints (step 1.3); otherwise, for a given j, the most critical $\geq$-constraint i is the one for which the ratio of a_{ij} over the residual b'_i is minimum (step 1.5). More formally stated the procedure that computes the values s_j's in the constructive phase is reported in Figure 4. Since the destructive phase is perfectly symmetric to the constructive one, during a destructive phase, variables are again ordered according to non-decreasing pseudoutility values, but this time α_j measures the detriment of $\geq$-feasibility and β_j the improvement of the $\leq$-feasibility when variable j is set to zero; variables which minimize the loss of the $\geq$-feasibility and maximize the gain of the $\leq$-feasibility are chosen

procedure Compute_s_Constructive_Phase(s)
begin
1) **for each** $j : 1 \leq j \leq n$ **do**
1.1) $\alpha_j := \max\left\{\frac{a_{ij}}{|b_i^{'}|};\quad 1 \leq i \leq m\right\}$; /* constraint i can be violated or not */
1.2) **if** ($b_i^{'} \geq 0,\ m+1 \leq i \leq m+q$) **then**
1.3) $\beta_j := 1$; /* there are not $\geq$ violated constraints */
1.4) **else**
1.5) $\beta_j := \min\left\{\frac{a_{ij}}{|b_i^{'}|};\quad m+1 \leq i \leq m+q;\quad b_i^{'} < 0\right\}$;
endif;
1.6 $s_j := \frac{\alpha_j}{\beta_j}$;
endfor;
end;

Figure 4: Ranking the candidate variables: bottleneck criterion

first. To this end, for a given variable j, the most critical $\geq$-constraint, namely i, is the one for which the ratio $a_{ij}/|b_i'|$ is maximum. Again, as far as β_j is concerned, if the $\leq$-feasibility is guaranteed, setting a variable to zero would only improve the satisfaction of the $\leq$-constraints; otherwise, for a given j the most critical $\leq$-constraint i is the one for which the ratio of a_{ij} over the residual b_i' is minimum.
Because of symmetry we do not report the details of this procedure.

As announced before, another choice for the ranking criterion is based on surrogate information and consists in weighting opportunely the constraints, by taking into account whether they are violated or not. The procedure used to compute the values s_j's during a constructive phase is sketched in Figure 5: here α_j is the j-th coefficient of the surrogate constraint computed as linear combination of the $\leq$-constraints by giving a penalty equal to a_{ij} to violated constraints i's and a relative weight a_{ij}/b_i' to all non violated constraints i's. The rationale is that if the $\leq$-constraint i is satisfied, α_j takes into account how far we go from the $\leq$-feasibility when x_j is set to one; on the other side, the aim of such a procedure is to inflate the influence of violated constraints. Analogously β_j is the j-th coefficient of the surrogate constraint computed as linear combination of violated $\geq$-constraints only, by using the relative weights a_{ij}/b_i'.

The tabu memory. The attributes associated to the moves are *both the names and the values* of the variables which are modified. Setting a variable to one (zero) is considered tabu if the variable has been set to one (zero) less than `tabu_tenure` outer iterations before the current one. The tabu status of a move is not always considered: it is evaluated only for the first `k` moves of each phase.

```
procedure Compute_s_Constructive_Phase(s)
begin
    for each j : 1 ≤ j ≤ n do
        α_j := β_j := 0;
        for each i : 1 ≤ i ≤ m do
            if (b'_i ≤ 0) then
                α_j := α_j + a_ij
            else
                α_j := α_j + a_ij / b'_i
        endfor;
        for each i : m + 1 ≤ i ≤ m + q do
            if (b'_i < 0) then
                β_j := β_j + a_ij / |b'_i|;
        endfor;
        s_j := α_j / β_j;
    endfor;
end;
```

Figure 5: Ranking the candidate variables: surrogate criterion

2.2 The inner local search

The inner local search is a tabu search based procedure in which only feasible solutions are examined and which terminates after that a maximum number of iterations has been performed or no more moves exist.

The neighborhood structure. We associate each solution x with two neighborhoods, $N1$ and $N2$. The neighborhood $N1$ is made by all the feasible solutions which can be generated by applying the following three moves:

- $0 \rightarrow 1$ MOVE: a zero variable is set to one.
- $1 \rightarrow 0$ MOVE: a one variable is set to zero.
- SWAP MOVE: a zero variable and a one variable interchange their values.

This neighborhood has cardinality of order $O(n^2)$.
The neighborhood $N2$ is made by all the feasible solutions which can be generated by applying the following move:

- DOUBLE SWAP MOVE: two zero variables and two one variables interchange their values.

This neighborhood has cardinality of order $O(n^4)$.
At each iteration we first evaluate the neighborhood $N1$. In the case in which either the neighborhood $N1$ is empty or we detect a cycling behavior of the search we evaluate the

neighborhood $N2$. Each solution in both neighborhoods is evaluated only with respect to the objective function value. When the neighborhood $N1$ is evaluated the algorithm looks for the best non tabu solution, and if such solution does not exist it looks for the oldest tabu solution, i.e. the solution which has been tabu for the longest time. When the neighborhood $N2$ is evaluated the algorithm does not look for the best non tabu solution, since it would be too computationally expensive. Instead we explore $N2$ with the aim of obtaining both diversification of the search and quality of solutions. First we sort the variables in non decreasing c_j order and we initialize two indices, l and r, to the values 0 and $n+1$, respectively: they identify the range into which the search is done. Then we search for the two variables which give the maximum increase to the objective function value when interchanged, i.e. starting from the left we look for the first variable in the given order set to one: let $j_1 > l$ be its position. In a symmetric way variables are screened from the right to the left seeking for the first variable set to 0; let $j_2 < r$ be its position. Then, adopting the same criterion, we look for the next two variables in the range comprised between positions j_1 and j_2, set to one and zero respectively, such that if we interchange the values of the four variables the resulting solution is feasible. If a feasible solution is generated we set $l = j_1 + 1$ and $r = j_2 - 1$. In this way, the next time the neighborhood $N2$ is evaluated, the first feasible move found by the search procedure, will interchange the values of a different set of variables. In this way we impose a diversification of the search. Moreover, all the indices are managed in a circular way, i.e. index l which is running from 0 to $n + 1$, takes again the value 0 after the value $n + 1$, and index r, which is running from $n + 1$ down to 0, takes the value $n + 1$ after the value 0. No matter which neighborhood has been adopted, the selected solution becomes the new current inner solution x. Every time the search generates a solution x whose value z is greater than z^*, then the best solution x^* is updated.

To see that the neighborhoods $N2$ and $N1$ are not connected we propose the following example. Let us consider an equality constraint, since it can be modeled as two identical, but opposite, inequality constraints. Let be $n = 2\Delta + 2$ and $b = 4M + \Delta$, where $M >> \Delta \geq 4$ (e.g. $M = \Delta^2$) and Δ is an even integer value. Let be $a_i = i$ for $i = 1, \ldots, \Delta$, $a_{\Delta+i} = M + i$, for $i = 1, \ldots, \Delta$, $a_{n-1} = 4M$ and $a_n = 2M$. Let us consider the set $S^1 = \{x^1(1), \ldots, x^1(\Delta/2)\}$ of all the feasible solution vectors $x^1(j)$, for $j = 1, \ldots, \Delta/2$, whose components j, $\Delta - j$ and $n - 1$ assume the value one and all the other components are set to zero. From any solution in S^1 the only possible moves are those which define $N2$ and such moves can only lead to solutions in S^1. On the other hand there exist another set of feasible solutions, $S^2 = \{x^2(1), \ldots, x^2(\Delta/2)\}$ which is disjoint from S^1. It is given by all the feasible solution vectors $x^2(j)$, for $j = 1, \ldots, \Delta/2$, whose components $\Delta + j$, $2\Delta - j$ and n assume the value one and all the other components are set to zero. Even here from any solution in S^2 the only possible moves are those which define $N2$ and such moves can only lead to solutions in S^2.

The stopping criteria. The inner search stops whenever (a) a maximum number of iterations, `MaxNumLSSteps`, has been performed or (b) no feasible solution in both the neighborhoods of x exists (besides the last visited one). The value of `MaxNumLSSteps` is increased of the value `MinNumLSSteps`, if a solution which improves over the current best solution is generated and less than `MinNumLSSteps` iterations are still available for the current inner local search.

The tabu memory. Differently from the outer local search the attributes associated with the moves are only *the names* of the variables whose values are modified. The tabu tenure is the same for all kinds of move and its value is stored into the parameter `tabu_tenure`. A $0 \to 1$ MOVE ($1 \to 0$ MOVE) is considered tabu if the variable which should be set to one (zero) has been modified less than `tabu_tenure` iterations before the current one. A SWAP MOVE is considered tabu if the variables whose values should be interchanged have been *both* modified less than `tabu_tenure` iterations before the current one. A DOUBLE SWAP MOVE is considered tabu if *all* the four variables whose values should be interchanged are the same which have been modified last time the neighborhood $N2$ has been evaluated. A dynamic handling of parameter regarding tabu tenure is done. If the performed move leads to a solution whose value is better (worse) than that of the current one, the `tabu_tenure` parameter is decreased (increased) by one unit. Anyway parameter `tabu_tenure` is always comprised between two limiting values.

The aspiration criteria. We adopt one simple aspiration criterion. A tabu move is performed if the generated solution is better than the best current solution x^*.

3 Fixing rules

As announced in previous sections, a dominance test is invoked which aims at identifying variables which should be fixed to 0 or to 1 in a feasible solution. Such a procedure exploits a set of information coming from surrogate constraints.
We recall that every time a variable is set to 0 or to 1 the dominance test is performed if enabled. The dominance test is disabled whenever the inner local search is invoked.

Let i and h be indices of a $\leq$ and a $\geq$-constraint respectively, x be the current solution and let b'_i and b'_h be the corresponding residuals as defined above, i.e.

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \tag{4}$$

$$\sum_{j=1}^{n} a_{hj}x_j \geq b_h \tag{5}$$

where $a_{ij}, a_{hj} \geq 0\ \forall j$ and $a_{ij} \leq b'_i\ \forall j$ by hypothesis.
Let us state two properties which allow to fix a variable to 0 or to 1 respectively.

Property 3.1. *Let i and h be defined as above; then $x_k = 0$ for all indices k such that the following relation holds:*

$$\max_{j|a_{ij}\neq 0} \left\{ \frac{a_{hj}}{a_{ij}} \right\} < \frac{b'_h - a_{hk}}{b'_i - a_{ik}}.$$

Property 3.2. *Let i and h be defined as above and let $\sum_j \alpha_j x_j \geq \beta$ be the surrogate constraint obtained as linear combination of constraints (4) and (5) by using -1 and μ respectively as multipliers, where $\mu = \max_{j|a_{hj}\neq 0} \left\{ \frac{a_{ij}}{a_{hj}} \right\}$; then $x_k = 1$ for all indices k such that $\sum_{j\neq k} \alpha_j < \beta$.*

The base idea underlying the fixing rules consists of defining opportunely two non-negative multipliers in such a way the surrogate constraint obtained as linear combination of constraints (4) and (5) satisfies some properties we are going to state. Note that such a problem really results in computing only a multiplier since two given non-negative multipliers can be always normalized simply dividing them by the maximum of the two.
As far as the fixing to zero rule is concerned, constraint (5) is turned into a $\leq$-constraint by multiplying it by -1 and a multiplier μ is looked for such that the coefficients α_j's of the surrogate constraint are non-negative. More formally α_j is defined as

$$\alpha_j = \mu a_{ij} - a_{hj} \quad \forall j \in \{1. \dots, n\}$$

and we are asked for computing a multiplier μ such that $\mu \geq a_{hj}/a_{ij} \forall j | a_{ij} \neq 0$. So, let us define μ as follows

$$\mu = \max_{j|a_{ij}\neq 0} \frac{a_{hj}}{a_{ij}}.$$

The right hand side β of the surrogate constraint is defined as $\beta = \mu b'_i - b'_h$ and trivially if an index k exists such that $\alpha_k > \beta$, then variable x_k is to be fixed to 0. It is straightforward to see that $\alpha_k > \beta$ if and only if $\mu < (b'_h - a_{hk})/(b'_i - a_{ik})$.

Example 3.1. *Consider the following constraints*

$$2x_1 + 4x_2 + 10x_3 \leq 12 \tag{6}$$

$$6x_1 + 4x_2 + 3x_3 \geq 10 \tag{7}$$

and define the multiplier μ according to the rule above stated, i.e.

$$\mu = \max\{6/2, 4/4, 3/10\} = 3.$$

The surrogate constraint obtained from the linear combination of constraints (6) and (7) by using respectively μ and -1 as multipliers is the following

$$8x_2 + 27x_3 \leq 26$$

and trivially we get that x_3 must be equal to 0.

On the other side in order to state the fixing to one rule, constraint (4) is first turned into a $\geq$-constraint and then a multiplier μ is searched for such that the surrogate constraint coefficients α_j's are non-negative. The coefficient α_j is defined as

$$\alpha_j = \mu a_{hj} - a_{ij} \quad \forall j \in \{1. \dots, n\}$$

and the non-negativeness of the coefficients is assured if and only if $\mu \geq a_{ij}/a_{hj} \forall j$. So, let us define μ as follows

$$\mu = \max_{j | a_{hj} \neq 0} \frac{a_{ij}}{a_{hj}}.$$

The right hand side β of the surrogate constraint is defined as $\beta = \mu b'_h - b'_i$ and trivially if an index k exists such that $\sum_{j \neq k} \alpha_j < \beta$, then variable x_k is to be fixed to 1. Such a relation states that even if all the variables but variable x_k were set to one, the surrogate $\geq$-constraint would not be satisfied.

Example 3.2. *Consider the following constraints*

$$x_1 + 2x_2 + x_3 \leq 2 \tag{8}$$
$$3x_1 + 3x_2 + 2x_3 \geq 5 \tag{9}$$

and define the multiplier μ according to the rule above stated, i.e.

$$\mu = \max\{1/3, 2/3, 1/2\} = 2/3.$$

The surrogate constraint obtained as linear combination of constraints (8) and (9) by using respectively -1 and μ as multipliers is the following

$$x_1 + 1/3x_3 \geq 4/3$$

and trivially we get that x_1 and x_3 must be equal to 1.

```
procedure Dominance_Test(F_0,S_1)
begin
1)      for each i,h :  1 ≤ i ≤ m,  m+1 ≤ h ≤ m+q do
1.1)        μ := max_{1≤j≤n} { a_hj / a_ij | a_ij ≠ 0 };
1.2)        β := μ * b'_i − b'_h;
1.3)        for each j : 1 ≤ j ≤ n do
1.3.1)          α_j := μ * a_ij − a_hj;
1.3.2)          if  α_j > β then  F_0 := F_0 ∪ {j};
            end for;
        end for;
2)      for each i,h :  1 ≤ i ≤ m,  m+1 ≤ h ≤ m+q do
2.1)        μ := max_{1≤j≤n} { a_ij / a_hj | a_hj ≠ 0 };
2.2)        β := μ * b'_h − b'_i;
2.3)        T := Σ_{j=1}^{n} (μ * a_hj − a_ij);
2.4)        for each j : 1 ≤ j ≤ n do
2.4.1)          α_j := μ * a_hj − a_ij;
2.4.2)          if  T − α_j < β then  S_1 := S_1 ∪ {j};
            end for;
        end for;
end;
```

Figure 6: The Dominance_Test procedure

The Dominance_Test procedure is sketched in Figure 6.

4 Computational results

The Nested Tabu Search has been tested on a wide set of instances generated modifying the *MKP* instances by Chu and Beasley [7] (which are publically available from [3]).
Let us briefly describe how these instances have been generated. The number of $\leq$-constraints m is set to 5, 10 and 30 while the number q of $\geq$-constraints depends on m: for a fixed value of m, possible values for q are 1, $m/2$ and m respectively. The number of variables n is set to 100, 250 and 500.
The coefficients a_{ij}'s are integer numbers uniformly drawn in the range $[0, 1000]$. The objective function coefficients c_j's are correlated to a_{ij}; two kinds of instances have been generated according to the sign of the cost coefficients: in the former ones cost coefficients are all assumed positive while in the latter ones cost coefficients are not constrained in sign.
Let us go into details concerning the way the cost coefficients are computed:

- **Positive cost coefficients.** The cost c_j of variable j is defined as

$$c_j = \sum_{i=1}^{m} \frac{a_{ij}}{m} - \sum_{i=m+1}^{m+q} \frac{a_{ij}}{q} \quad \forall j \in \{1, \ldots, n\}$$

which is the difference between the average coefficient of the j-th column of A computed over the first m $\leq$-constraints and the average value over the last q $\geq$-constraints. Cost coefficients are then opportunely scaled of quantity Δ so as to become all positive and then perturbed as follows:

$$c_j = c_j + \Delta + 500r_j$$

where r_j is a real number uniformly drawn in the range $[0,1]$ and the quantity Δ is defined as

$$\Delta = \begin{cases} 1 + \min_j\{c_j\} & \text{if } \min_j\{c_j\} < 0 \\ \\ 0 & \text{otherwise.} \end{cases}$$

- **Mixed cost coefficients.** The cost c_j of variable j is defined as

$$c_j = \sum_{i=1}^{m} \frac{a_{ij}}{m} - \sum_{i=m+1}^{m+q} \frac{a_{ij}}{q} + 500r_j \quad \forall j \in \{1, \dots, n\}$$

  where r_j is a real number uniformly drawn in the range $[0,1]$. A set of coefficients is then properly scaled so as to have exactly $n \cdot q/(m+q)$ negative coefficients, i.e. the number of negative coefficients is proportional to the number q of $\geq$-constraints.

The right hand side coefficients b_i's are computed as $b_i = \alpha \sum_{j=1}^{n} a_{ij}$ where α is a tightness ratio and $\alpha \in \{0.25, 0.5, 0.75\}$.
Five problems were generated for each n–m–q–α combination and for each type of cost coefficients, either positive or mixed, summing up to 810 instances.

Three versions of the Nested Tabu Search (NT) have been tested: in the first one the dominance rules which allow to setting variables are disabled. In the other two cases the fixing rules are enabled but different criteria are used; on one hand surrogate information is used, on the other hand a complete screening of the constraints is done. Recall the reader that in order to apply the fixing rules, two candidate conflicting constraints have to be identified: when surrogate information is used, the $\leq$-constraint is obtained as linear combination of the m $\leq$-constraints, while the $\geq$-constraint is the surrogate of the q $\geq$-constraints. On the other hand, in the third version, each couple of $\leq$ and $\geq$-constraints is in turn chosen as candidate for the fixing rules shown in Section 3. We refer to these versions as *surrogate setting* and *complete setting* respectively.
In all the versions of NT, variables to add and drop are chosen via the surrogate criterion since computational results shown later on, have disclosed that on average such a criterion

outperforms the bottleneck one, both in terms of computational time and robustness, being however less accurate in general.

The parameters involved in both the outer and the inner phases of NT have been set as shown in Tables 1 and 2, where Δ_{span} is the number of values parameter `span` assumes (Δ_{span} = Max `span` - Min `span` + 1) and similarly Δ_{tt} is the number of values parameter `tabu_tenure` takes (Δ_{tt} = Max `tt` - Min `tt` + 1). So, for each phase, either constructive or destructive, each combination of `span` and `tabu_tenure` values in the ranges considered, is taken into account resulting in a maximum number `MaxNumTP` of iterations which is equal to $2 * \Delta_{span} * \Delta_{tt}$. The value of parameter `k`, dealing with the tabu memory management, is a percentage of the number of variables n ($\mathtt{k} = 0.08 * n$).

	Oscillation		Tabu Mem			Stopping Criteria	
n	Min Span	Max Span	Min tt	Max tt	k	MaxNumTP	MaxNumLS
100	1	40	2	15	8	$2 * \Delta_{span} * \Delta_{tt}$	2* MaxNumTP
250	1	40	2	15	20	$2 * \Delta_{span} * \Delta_{tt}$	2* MaxNumTP
500	1	40	2	15	40	$2 * \Delta_{span} * \Delta_{tt}$	2* MaxNumTP

Table 1: Outer Phase parameters

Tabu Mem		Stopping Criteria		
Min tt	Max tt	MaxNumILS	MinNumLSSteps	MaxNumLSSteps
3	6	50	150	2000

Table 2: Inner Phase parameters

All the versions of NT have been compared with CPLEX 7.0 on a Pentium III at 600Mhz with 384 MB of main memory. All times are expressed in CPU seconds.

The problems generated according to the criteria above described seem to be very difficult to solve for CPLEX which very often runs out of memory without yielding a feasible solution: we have thus fixed an upper bound of 250 MB on the size of the Branch and Bound tree and a time limit of 3600 CPU seconds. The best solution obtained is then reported.

In Table 3 a comparison between CPLEX and NT without fixing rules is reported in terms of percentage gap with respect to the upper bound given by CPLEX (column Gap), time required to compute the best solution (column Best Time), overall CPU time (column Time) and number of failures, i.e. number of trials for which the solver does not provide a feasible solution.

We stress here that hereafter the average values are computed over all of the successful trials thus meaning that it makes little sense to directly compare the results obtained with CPLEX and NT when the number of unsuccessful trials is different.

Table 3 reports a row for each combination of n-m and for each different choice of cost coefficients (either positive, case 0 or mixed, case 1). Each row thus shows the average values computed over 45 instances. The table clearly shows that the NT is well able to provide feasible solutions of good quality within reasonable computational time: first of all observe that NT fails 28 times over 810 trials against the 173 failures of CPLEX. We suspect that some of the 100-30 instances are infeasible since neither CPLEX nor NT were able to provide a feasible solution but we were not able to prove such a fact within reasonable computational time. Observe that the higher gaps are obtained by NT on instances with 30 $\leq$-constraints, i.e. when the number of failures reported by CPLEX is the maximum. This is probably due to the fact that such instances seem to be really difficult to solve and on such problems the gap between the optimal solution and the upper bound given by the linear relaxation could be quite high. On the other hand, NT is much more effective than CPLEX both in terms of best time and total time: NT is on average 4 times faster than CPLEX and the time required by NT to give the best solution is smaller than the one reported by CPLEX by a factor approximately equal to 2.6.

	Cplex				NT			
Instances	Gap	BestTime	Time	Failed	Gap	BestTime	Time	Failed
100-5-0	0.00	44.59	189.02	0	0.07	18.70	38.93	0
100-5-1	0.00	64.81	377.02	0	0.26	20.09	36.69	0
100-10-0	0.37	601.09	2393.63	0	0.69	25.03	49.43	0
100-10-1	0.40	410.77	1596.03	0	0.88	27.26	47.01	0
100-30-0	4.52	1738.11	3600.00	26	5.92	94.47	207.12	14
100-30-1	3.10	1165.92	2768.95	29	10.76	84.03	266.65	14
250-5-0	0.10	254.87	867.05	0	0.27	140.30	217.34	0
250-5-1	0.36	252.25	844.51	0	1.15	151.90	198.25	0
250-10-0	0.62	472.19	1085.55	0	0.96	145.73	230.28	0
250-10-1	1.26	498.15	1076.05	0	1.80	147.44	212.06	0
250-30-0	0.99	1373.32	3600.00	30	2.75	385.03	627.39	0
250-30-1	2.86	1876.21	3600.00	29	5.63	371.64	588.13	0
500-5-0	0.07	583.03	1221.57	0	0.25	605.86	875.29	0
500-5-1	0.24	748.77	1283.72	0	0.99	614.55	790.92	0
500-10-0	0.29	1582.05	3138.26	0	0.64	627.13	867.44	0
500-10-1	0.72	1390.98	2942.64	0	1.50	616.74	814.24	0
500-30-0	0.60	1745.85	3600.00	30	1.73	1175.85	1836.11	0
500-30-1	0.51	1378.70	3600.00	29	2.94	945.85	1360.80	0

Table 3: Cplex vs NT: average results

We now go into details by analyzing disaggregated results relative to the 100-30 problems with positive costs; in particular we outline the performance of the three versions of NT. We choose these instances for our analysis since they are the smallest problems among the ones which are difficult to solve for CPLEX and the computational burden required by the fixing rules is still acceptable. In Table 4 computational results given by the outer phase of the 3 versions of NT above described, are reported. Each row of Table 4 reports average values obtained over the 5 instances with the same value of α ($\alpha \in \{0.25, 0.5, 0.75\}$) and the same number of $\geq$-constraints ($q \in \{1, 15, 30\}$). Observe that the number of unsuccessful runs is different according to the dominance criterion selected. For the cases when $q = 30$ and $\alpha = 0.25$ or $\alpha = 0.75$ all of the three versions of NT give 5 failures thus strengthening our opinion that such instances are perhaps infeasible. On the other side we observe that the complete setting version attains the minimum number of failures for the case $\alpha = 0.5$ and $q = 30$: on such problems the quality of the solution obtained by the complete setting is meaningfully better (10%) than the one obtained with the other versions with, unfortunately, a deterioration of the computational time. As a matter of fact there is a trade-off between the quality of the solution obtained and the computational burden required by this version of NT: on bigger instances the cost of such dominance rules becomes soon prohibitive and much clever dominance criteria should be used. On the other side the surrogate setting is much less time consuming than the complete setting but it exploits weaker relationships resulting in quite poor performance. We further stress here that CPLEX fails to provide a feasible solution 11 times over 15 trials when $q = 15$ while an optimal solution exists as NT versions one and three highlights reporting 0 failures. And it is important to observe that on some instances (case $\alpha = 0.5$ and $q = 15$) the outer phase of NT only, outperforms CPLEX both in terms of total time and gap, providing very soon solution of good quality.

Instances		Cplex			NT			NT + Surr. Setting			NT + Compl. Setting		
α	q	Gap	Time	Fail	Gap	Time	Fail	Gap	Time	Fail	Gap	Time	Fail
0.25	1	2.06	3600.00	0	6.76	22.95	0	6.76	26.81	0	6.76	31.38	0
0.50	1	1.41	3600.00	0	3.76	31.29	0	3.76	38.12	0	3.70	54.05	0
0.75	1	0.57	3600.00	0	2.17	27.98	0	2.10	37.41	0	2.10	57.82	0
0.25	15	10.31	3600.00	3	18.45	53.45	0	19.55	68.12	0	18.24	268.90	0
0.50	15	8.23	3600.00	3	7.98	24.33	0	7.85	35.09	0	8.04	402.50	0
0.75	15	-	3600.00	5	6.92	62.00	0	6.59	84.83	1	6.85	669.79	0
0.25	30	-	3600.00	5	-	82.18	5	-	99.76	5	-	647.58	5
0.50	30	-	3600.00	5	25.76	88.62	4	-	119.75	5	15.26	1587.58	3
0.75	30	-	3600.00	5	-	82.01	5	-	113.78	5	-	1440.91	5

Table 4: About the fixing rules on 100-30 instances

Table 5 reports again on aggregated results: in such a table we point out the behavior of both the outer and the inner phases of NT in terms of overall time and gaps. In all cases remarkable reductions of the gap are obtained with the inner local search and such gaps are quite low but for the cases when $m = 30$ which corresponds to the most critical situations: as we already observed the gap between integer and linear solution could be quite high on such problems.

	Cplex		NT: Outer Phase		NT: Inner Phase	
Instances	Gap	Time	Gap	Time	Gap	Time
100-5-0	0.00	189.02	0.84	12.07	0.07	26.86
100-5-1	0.00	377.02	3.97	11.97	0.26	24.72
100-10-0	0.37	2393.63	2.10	14.90	0.69	34.53
100-10-1	0.40	1596.03	4.39	15.00	0.88	32.01
100-30-0	4.52	3600.00	10.26	52.76	5.92	154.37
100-30-1	3.10	2768.95	18.41	52.60	10.76	214.05
250-5-0	0.10	867.05	1.23	65.99	0.27	151.35
250-5-1	0.36	844.51	6.19	65.99	1.15	132.26
250-10-0	0.62	1085.55	2.41	73.50	0.96	156.78
250-10-1	1.26	1076.05	5.32	74.83	1.80	137.23
250-30-0	0.99	3600.00	4.79	106.01	2.75	521.38
250-30-1	2.86	3600.00	11.41	106.14	5.63	482.00
500-5-0	0.07	1221.57	1.15	310.81	0.25	564.47
500-5-1	0.24	1283.72	6.31	302.95	0.99	487.98
500-10-0	0.29	3138.26	2.04	309.36	0.64	558.08
500-10-1	0.72	2942.64	5.91	318.60	1.50	495.64
500-30-0	0.60	3600.00	3.73	422.45	1.73	1413.66
500-30-1	0.51	3600.00	7.19	408.54	2.94	952.25

Table 5: Outer and Inner Phases vs Cplex

In Table 6 a comparison between CPLEX and the outer phase of NT (without fixing rules) is given for problems with positive costs. Here for a given combination of n-m the results relative to problems with the same values of α are shown. It is straightforward to see that the problems in the test bed becomes much more hard to solve as the number of $\geq$-constraints grows: for example, as far as the problems with $n = 100$ are concerned, the maximum number of failures is reported by CPLEX for the case when $m = 30$. We observe that CPLEX requires just an hundred of seconds to solve problems with 5 $\leq$-constraints but the computational time exponentially increases when only few further $\leq$-constraints are added, up to an hour and even more; and the number of problems solved to optimality soon degrades. On the other side the increase of the computational time required by NT when the dimension of the problems grows is much more limited: the outer phase of NT is able to provide good solutions (gap not over 6%) for the bigger problems in the test bed (500-30) within few minutes. It is quite nice to observe that the behavior of NT in terms of gap is almost identical for problems with the same number of constraints: see for example the three rows corresponding to the 250-5 and the other 3 ones corresponding to 500-5. Also the 250-10 and the 500-10 show a similar behavior in terms of gap: such a fact allow us to believe that NT is quite robust and effective. Moreover it is worth noting that NT does not collect failures for problems belonging to 250 and 500 classes with 30 $\leq$-constraints, while, on the contrary, on the same instances the number of unsuccessful trails for CPLEX is equal to 2/3 of all the tests.

	Cplex				NT		
Instances	Gap	Time	Fail	Opt	Gap	Time	Fail
100-5-0.25	0.00	108.14	0	15	1.20	11.28	0
100-5-0.5	0.00	367.64	0	15	0.98	13.95	0
100-5-0.75	0.00	91.28	0	15	0.34	10.98	0
100-10-0.25	0.69	2538.60	0	6	3.40	13.74	0
100-10-0.5	0.27	2545.24	0	6	1.76	16.96	0
100-10-0.75	0.14	2097.04	0	7	1.15	14.00	0
100-30-0.25	6.18	3600.00	8	0	12.60	52.86	5
100-30-0.5	4.82	3600.00	8	0	12.50	48.08	4
100-30-0.75	0.57	3600.00	10	0	4.54	57.33	5
250-5-0.25	0.12	922.36	0	2	1.94	64.54	0
250-5-0.5	0.10	625.16	0	0	1.18	75.69	0
250-5-0.75	0.07	1053.65	0	2	0.57	57.74	0
250-10-0.25	0.83	1066.91	0	0	3.74	71.95	0
250-10-0.5	0.52	1086.32	0	0	2.19	83.93	0
250-10-0.75	0.51	1103.41	0	0	1.30	64.62	0
250-30-0.25	1.61	3600.00	10	0	7.58	98.44	0
250-30-0.5	0.84	3600.00	10	0	3.98	118.69	0
250-30-0.75	0.52	3600.00	10	0	2.80	100.89	0
500-5-0.25	0.09	1068.26	0	0	1.78	306.94	0
500-5-0.5	0.07	1322.13	0	0	1.10	381.93	0
500-5-0.75	0.05	1274.32	0	0	0.56	243.57	0
500-10-0.25	0.41	3054.40	0	0	3.30	309.81	0
500-10-0.5	0.25	3287.74	0	0	1.79	369.10	0
500-10-0.75	0.22	3072.65	0	0	1.03	249.18	0
500-30-0.25	0.98	3600.00	10	0	5.90	406.96	0
500-30-0.5	0.49	3600.00	10	0	3.29	488.48	0
500-30-0.75	0.33	3600.00	10	0	2.00	371.90	0

Table 6: Cplex vs NT: disaggregated results

Hereafter, as announced at the beginning of the section, a comparison between the two criteria used to rank the variables to add and drop is done: Table 7 reports on aggregated results obtained for mixed cost coefficients.

Observe (column Gap) that generally a greater accuracy is yielded with the bottleneck ranking criterion which however requires a greater computational burden (column Time) as the dimension of the problem increases. Furthermore the number of unsuccessful trials reported by the bottleneck criterion is equal to 18, against the 14 failures given by the surrogate criterion. The surrogate criterion seems thus to be more robust: this is the reason why we preferred it to the bottleneck one. Anyway the bottleneck criterion was not fully discarded and we adopted the following strategy in our computational experiments: bottleneck ranking is used should surrogate one fail to provide a meaningful number of feasible solutions.

Finally, we observe that mixed cost instances are consistently more difficult to solve than positive cost ones, both for Cplex and for our approach. However this behavior is more evident

	Surrogate Setting			Bottleneck Setting		
Instances	Gap	Time	Fail	Gap	Time	Fail
100-5-0.25	5.71	11.18	0	4.42	15.21	0
100-5-0.5	4.48	13.74	0	3.68	18.77	0
100-5-0.75	1.72	10.99	0	1.39	15.24	0
100-10-0.25	6.53	13.82	0	5.94	19.53	0
100-10-0.5	4.32	17.03	0	3.83	23.02	0
100-10-0.75	2.30	14.16	0	2.47	19.31	0
100-30-0.25	24.04	37.90	5	24.65	56.43	5
100-30-0.5	25.47	47.72	4	14.25	64.45	5
100-30-0.75	10.41	45.13	5	11.18	59.20	8
250-5-0.25	9.09	66.57	0	7.53	76.35	0
250-5-0.5	6.44	75.03	0	4.79	87.42	0
250-5-0.75	3.05	56.36	0	2.76	67.75	0
250-10-0.25	8.07	75.29	0	7.23	90.85	0
250-10-0.5	4.96	84.46	0	4.56	103.21	0
250-10-0.75	2.94	64.73	0	2.89	82.26	0
250-30-0.25	15.83	98.15	0	18.47	148.43	0
250-30-0.5	11.14	118.64	0	12.94	162.44	0
250-30-0.75	7.25	101.62	0	8.27	150.06	0
500-5-0.25	8.73	314.23	0	7.65	343.06	0
500-5-0.5	6.34	357.93	0	5.28	393.07	0
500-5-0.75	3.87	236.68	0	3.22	270.57	0
500-10-0.25	8.56	332.99	0	7.99	368.92	0
500-10-0.5	5.74	375.97	0	4.69	415.36	0
500-10-0.75	3.41	246.84	0	3.12	288.57	0
500-30-0.25	10.71	398.80	0	11.13	496.92	0
500-30-0.5	6.52	463.51	0	7.01	572.24	0
500-30-0.75	4.35	363.32	0	4.80	497.12	0

Table 7: Surrogate vs bottleneck ranking for mixed costs problems

for Cplex when 250-30 instances are considered whilst for the heuristic approach the mixed cost instances lead to a gap value which is almost always twice that of positive cost ones. These computational results seem to be particularly encouraging since they show that NT attains its best performance just when the number of $\geq$-constraints increases which are exactly the most difficult problems in the test bed: in fact, they correspond to crucial situations with many conflicting constraints and solving such problems was one of the main objective the Nested Tabu search has been designed for. When there are $\leq$-constraints which force to keep low the number of variables set to one and $\geq$-constraints that drive in the opposite direction, gaining the feasibility is a very critical task, and NT which operates at two different levels, and takes into account the feasibility target and optimality target separately, seems to be a good choice.

References

[1] E. Balas. An Additive Algorithm for Solving Linear Programs with 0-1 Variables. *Operations Research*, 13:517–546, 1965.

[2] E. Balas and C.H. Martin. Pivot and Complement - a Heuristic for 0-1 Programming. *Management Science*, 26:86–96, 1980.

[3] J. Beasley. OR Library, 1995. Available by anonimous ftp to `msemga.ms.ic.ac.uk`.

[4] G.J. Beaujon, S.P. Marin, and G.C. McDonald. Balancing and Optimizing a Portfolio of R&D Projects. *Naval Research Logistics*, 48:18–40, 2001.

[5] A.V. Cabot. An Enumeration Algorithm for Knapsack Problems. *Operations Research*, 18:306–311, 1970.

[6] P. Cappanera. *Discrete Facility Location and Routing of Obnoxious Activities.* PhD thesis, University of Milano, 1999. Available at `http://www.di.unipi.it/~cappaner`.

[7] P.C. Chu and J.E. Beasley. A Genetic Algorithm for the Multidimensional Knapsack Problem. *Journal of Heuristics*, 4:63–86, 1998.

[8] Y. Crama and J.B. Mazzola. On the Strenght of Relaxations of Multidimensional Knapsack Problems. *INFOR*, 32:219–225, 1994.

[9] G.B. Dantzig. Discrete Variable Extremum Problems. *Operations Research*, 5:266–277, 1957.

[10] A. Freville and G. Plateau. An Efficient Preprocessing Procedure for the Multidimensional 0-1 Knapsack Problem. *Discrete Applied Mathematics*, 49:189–212, 1994.

[11] B. Gavish and H. Pirkul. Efficient Algorithms for Solving Multiconstraint Zero-One Knapsack Problems to Optimality. *Mathematical Programming*, 31:78–105, 1985.

[12] P.C. Gilmore and R.E. Gomory. The Theory and Computation of Knapsack Functions. *Operations Research*, 14:1045–1074, 1966.

[13] F. Glover and G.A. Kochenberger. Critical Event Tabu Search for Multidimensional Knapsack Problems. In I.H. Osman and J.P. Kelly, editors, *Meta Heuristics: Theory and Applications*, pages 407–427. Kluwer Academic Publishers, 1996.

[14] S. Hanafi and A. Freville. An Efficient Tabu Search Approach for the 0-1 Multidimensional Knapsack Problem. *European Journal of Operational Research*, 106:659–675, 1998.

[15] C. Haul and S. Voß. Using Surrogate Constraints in Genetic Algorithms for Solving Multidimensional Knapsack Problems. In D.L. Woodruff, editor, *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search*, pages 235–251. Kluwer Academic Publishers, 1998.

[16] B. Korte and R. Schnader. On the Existence of Fast Approximation Schemes. In O.L. Mangasarian, R.R. Meyer, and S.M. Robinson, editors, *Nonlinear Programming 4*, pages 415–437. Academic Press, New York, 1981.

[17] A. Løkketangen and F. Glover. Solving Zero-One Mixed Integer Programming Problems using Tabu Search. *European Journal of Operational Research*, 106:624–658, 1998.

[18] M.J. Magazine and M.S. Chern. A Note on Approximation Schemes for Multidimensional Knapsack Problem. *Mathematics of Operations Research*, 9:244–247, 1984.

[19] M.J. Magazine and O. Oguz. A Heuristic Algorithm for the Multidimensional Zero-One knapsack problem. *European Journal of Operational Research*, 16:319–326, 1984.

[20] S. Martello, D. Pisinger, and P. Toth. New Trends in Exact Algorithms for the 0-1 Knapsack Problem. *European Journal of Operational Research*, 123:325–332, 2000.

[21] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[22] M.A. Osorio, F. Glover, and P. Hammer. Cutting and Surrogate Constraint Analysis for Improved Multidimensional Knapsack Solutions. X Congreso Latino-Iberoamericano de Investigación de Operaciones y Sistemas, México, D.F., MEXICO, 4–8 September 2000. Available at `http://optnet.itwm.uni-kl.de/opt-net/documents/v00w39n1`.

[23] H. Pirkul. A Heuristic Solution Procedure for the Multiconstraint Zero-One Knapsack Problem. *Naval Research Logistics*, 34:161–172, 1987.

[24] D. Pisinger. Core Problems in Knapsack Algorithms. *Operations Research*, 47:570–575, 1999.

[25] F. Plastria. Static Competitive Facility Location: an Overview of Optimisation Approaches. *European Journal of Operational Research*, 129:461–470, 2001.

[26] D. Romero-Morales, E. Carrizosa, and E. Conde. Semi-Obnoxious Location Models: a Global Optimization Approach. *European Journal of Operational Research*, 102:295–301, 1997.

[27] W. Shih. A Branch and Bound Method for the Multiconstraint 0-1 Knapsack Problem. *Journal of the Operational Research Society*, 30:369–378, 1979.

[28] S. Voß. Tabu Search: Applications and Prospects. In D.Z. Du and P.M. Pardalos, editors, *Network Optimization Problems*, pages 333–353. World Scientific Publishing Co., 1993.