ELSEVIER

European Journal of Operational Research 92 (1996) 310–325

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

Linear programming for the 0–1 quadratic knapsack problem

Alain Billionnet *, Frédéric Calmels

CEDRIC, Institut d'Informatique d'Enterprise, 18 allée Jean Rostand, 91025 Evry cedex, France

Received April 1993; revised August 1993

Abstract

In this paper we consider the quadratic knapsack problem which consists in maximizing a positive quadratic pseudo-Boolean function subject to a linear capacity constraint. We propose a new method for computing an upper bound. This method is based on the solution of a continuous linear program constructed by adding to a classical linearization of the problem some constraints redundant in 0–1 variables but nonredundant in continuous variables. The obtained upper bound is better than the bounds given by other known methods. We also propose an algorithm for computing a good feasible solution. This algorithm is an elaboration of the heuristic methods proposed by Chaillou, Hansen and Mahieu and by Gallo, Hammer and Simeone. The relative error between this feasible solution and the optimum solution is generally less than 1%. We show how these upper and lower bounds can be efficiently used to determine the values of some variables at the optimum. Finally we propose a branch-and-bound algorithm for solving the quadratic knapsack problem and report extensive computational tests.

Keywords: Zero-one quadratic programming; Knapsack problem; Linearization; Branch-and-bound; Computational analysis

1. Introduction

The 0–1 quadratic knapsack problem consists in maximizing a pseudo-Boolean quadratic function subject to a linear capacity constraint. This problem was introduced by Gallo, Hammer and Simeone (1980) and can be expressed as follows:

$$\text{Max } f(x) = \sum_{i=1}^{n} q_{ii}x_i + \sum_{1 \leqslant i < j \leqslant n} q_{ij}x_i x_j$$

$$\text{subject to } \sum_{i=1}^{n} a_i x_i \leqslant b, \tag{1}$$

$$x_i \in \{0, 1\}, \quad i = 1, \ldots, n, \tag{2}$$

* Corresponding author.

SSDI 0377-2217(94)00229-0

where the coefficients a_i $(i = 1, \ldots, n)$ and q_{ij} $(1 \leqslant i \leqslant j \leqslant n)$ are non-negative integers and b is an integer such that $0 < b < \sum_{i=1}^{n} a_i$. Note that problem (P) is NP-hard since the case $q_{ij} = 0$ $(1 \leqslant i < j \leqslant n)$ gives the usual (linear) knapsack problem.

The problem of maximizing an unconstrained quadratic pseudo-Boolean function has been studied by many authors (see, for example Carter, 1984, Hammer, Hansen and Simeone, 1984, Williams, 1985, Barahona, Jünger and Reinelt, 1989, Pardalos and Rodgers, 1990, Billionnet and Sutter, 1994, and Chardaire and Sutter, 1995). It is difficult to find the maximum of a function of some 50 variables. The problem of maximizing a quadratic pseudo-Boolean function subject to one or several linear constraints has been much less investigated although it is a very general formulation of a set of combinatorial optimization problems and it seems at least as difficult as the unconstrained problem. The quadratic knapsack problem is therefore interesting from a theoretical point of view since it is a particular case of quadratic 0–1 optimization with linear constraints. It is also interesting from a practical point of view since, as it is mentioned by Gallo, Hammer and Simeone (1980), this problem has several applications. For example, the determination of the optimal, sites for communication satellite earth stations with a budget constraint (Witzgall, 1975) can be formulated as (P). Similar problems concern the location of railway stations, freight handling terminals and airports. The determination of the optimal sites for pluviometers in hydrological studies or of best investment in portfolio selections (Laughhunn, 1970) are other instances of the quadratic knapsack problem quoted by Gallo, Hammer and Simeone (1980). It is also clear that some graph theory problems can be easily formulated in this way.

Consider the example of a classical task assignment problem in a host-satellite system with limited memory (see, for example, Billionnet, Costa and Sutter, 1989). Assuming a program has been partitioned into a set of tasks $T = \{1, \ldots, n\}$, how should we optimally allocate these tasks to the satellite (processor 1) and to the host (processor 2)? Program partitioning and task allocation are two important steps for processing a program on a distributed system and it is known that if these steps are not done properly, an increase in the number of processors in a system may actually result in a decrease of the total throughput. The two processors of the system are heterogeneous. A single task if executed on different processors will therefore require different amounts of running time. The cost of executing task i $(i = 1, \ldots, n)$ on processor j $(j = 1, 2)$ is assumed to be known and is denoted by e_{ij}. Each task i requires a memory capacity equal to m_i. The satellite processor has limited memory M and we assume that the host processor has unlimited memory. As mentioned by Rao, Stone and Hu (1979), this assumption is justified in many host-satellite systems. The capacities of the link between the two processors are assumed to be unlimited and if tasks i and j communicate, then c_{ij} denotes the communication cost between tasks i and j when they are not assigned to the same processor; c_{ij} also is assumed to be known ($c_{ij} = 0$ if tasks i and j do not have to communicate). The cost of transmitting data between coresident tasks is assumed to be zero. The problem is to find an assignment of the tasks to the two processors which minimizes a program's total cost defined as the sum over the two processors of both processing costs and communication costs. It is easy to formulate this problem as follows with $x_i = 1$ if task i is assigned to the satellite processor and $x_i = 0$ if task i is assigned to the host processor.

$$\text{Min} \sum_{i=1}^{n} e_{i1} x_i + \sum_{i=1}^{n} e_{i2}(1 - x_i) + \sum_{1 \leqslant i < j \leqslant n} c_{ij}\left[(1 - x_i)x_j + x_i(1 - x_j)\right]$$

$$\text{subject to} \sum_{i=1}^{n} m_i x_i \leqslant M,$$

$$x_i \in \{0,1\}, \quad i = 1, \ldots, n.$$

This problem, can be written as

$$\text{Max} - \sum_{i=1}^{n} e_{i2} - \sum_{i=1}^{n} x_i \left(e_{i1} - e_{i2} + \sum_{j=i+1}^{n} c_{ij} + \sum_{j=1}^{i-1} c_{ji} \right) + 2 \sum_{1 \leqslant i < j \leqslant n} c_{ij} x_i x_j$$

$$\text{subject to } \sum_{i=1}^{n} m_i x_i \leqslant M,$$

$$x_i \in \{0, 1\}, \quad i = 1, \ldots, n,$$

and we obtain a 0–1 quadratic knapsack problem where the coefficients of the linear terms in the objective function can be negative. It is not difficult by using the constraint $\sum_{i=1}^{n} m_i x_i \leqslant M$ to transform this problem into problem (P).

A few authors have proposed algorithms for the quadratic knapsack problem. Briefly consider that of Gallo, Hammer and Simeone (1980). They express the problem as

$$\max\{f(x) = x^t Q x : a^t x \leqslant b, x \in \{0,1\}^n\}, \tag{3}$$

where Q is a non-negative symmetric square matrix of order n, a is a positive n-vector and b is a positive scalar. The algorithm proposed by these authors is of branch-and-bound type. The upper bound is obtained by replacing the objective function $f(x)$ by a linear function $g(x)$ which dominates $f(x)$ in all feasible points (upper plane) and by solving the linear knapsack problem

$$\max\{g(x) : a^t x \leqslant b, x \in \{0,1\}^n\}. \tag{4}$$

This approach is motivated by the ease with which even large linear knapsack problems can be solved. The function $f(x)$ can be written $\sum_{j=1}^{n} (q_j^t x) x_j$ where q_j is the j-th column of Q. Obviously, $g(x) = \sum_{j=1}^{n} v_j x_j$, where v_j is an upper bound for $\max\{q_j^t x : x \in \{0,1\}^n, ax \leqslant b\}$, is an upper plane for $f(x)$. For each considered subproblem in the branch-and-bound algorithm, they construct the upper plane $v_0 + \sum_{j \in F} v_j x_j$ where F is the index set of the free variables. Obviously, the efficiency of this method depends on the choice of the bound v_j. A compromise must be found between the tightness of this bound and the time required for computing it. Note that solving (4) gives immediately a feasible solution of (3) and therefore a lower bound. Some theoretical results concerning the class of all upper planes, as well as extensive computational experience are reported in Gallo, Hammer and Simeone (1980).

Chaillou, Hansen and Mahieu (1983) have also proposed a branch-and-bound algorithm for the quadratic knapsack problem. The computation of an upper bound is based on a Lagrangean relaxation.

The Lagrangean function associated with problem (P) is

$$L(x_1, x_2, \ldots, x_n, \lambda) = \sum_{1 \leqslant i \leqslant j \leqslant n} q_{ij} x_i x_j + \lambda \left(b - \sum_{i=1}^{n} a_i x_i \right),$$

the dual function is

$$w(\lambda) = \max\{L(x_1, x_2, \ldots, x_n, \lambda) : x \in \{0,1\}^n\}$$

and the dual problem can be written

$$\text{Min}\{w(\lambda) : \lambda \geqslant 0\}.$$

Classically, the solution of this dual problem gives an upper bound of the optimal value of (P). The authors show that the dual problem can be solved by applying $n - 1$ times a maximum flow algorithm to a graph composed of $n + 2$ vertices and $n + m$ arcs where m is the number of quadratic terms of the function f. They report computational experience concerning 400 problems with 10 to 50 variables.

Michelon and Veuilleux (1991) use a Lagrangean decomposition method to compute an upper bound of the optimal value of (P). The bound obtained by this method is better than the bounds obtained by both methods presented above. The problem (P) is replaced by the following:

$$\text{Max} \sum_{i=1}^{n} q_{ii} x_i + \sum_{1 \leqslant i < j \leqslant n} q_{ij} x_i x_j$$

$$\text{subject to } \sum_{i=1}^{n} a_i y_i \leqslant b,$$
$$x_i = y_i, \quad i = 1,\ldots,n,$$
$$x_i \in \{0,1\}, \quad i = 1,\ldots,n,$$
$$y_i \in \{0,1\}, \quad i = 1,\ldots,n.$$

The multiplier u_i is associated with the constraints $x_i = y_i$ $(i = 1,\ldots,n)$. The dual function obtained by relaxation of these constraints is

$$L(u) = \text{Max}\left\{ \sum_{i=1}^{n} (q_{ii} - u_i)x_i + \sum_{1 \leqslant i < j \leqslant n} q_{ij}x_i x_j : x \in \{0,1\}^n \right\}$$
$$+ \text{Max}\left\{ \sum_{i=1}^{n} u_i y_i : y \in \{0,1\}^n, \sum_{i=1}^{n} a_i y_i \leqslant b \right\}$$

and an upper bound of the optimal value of (P) is given by the solution of the dual problem $\text{Min}\{L(u): u \in \mathbb{R}^n\}$. This bound is better than that obtained by Lagrangean relaxation but to compute $L(u)$ is an NP-hard problem since a linear knapsack problem has to be solved. Michelon and Veuilleux propose an efficient heuristic method for solving the dual problem and construct a branch-and-bound algorithm based on this bound. Extensive computational results on the same class of randomly generated test problems which are used in Gallo, Hammer and Simeone (1980) and Chaillou, Hansen and Mahieu (1983) are reported.

Section 2 presents heuristic algorithms to compute a good feasible solution of (P), i.e. a lower bound to the optimal value. In Section 3 we consider a classical linear programming formulation of the problem. The continuous relaxation of this reformulation gives a poor upper bound. We propose to dynamically add additional linear constraints (cuts) to improve this bound. Section 4 presents a method for fixing certain variables at the optimum of (P) once we know an upper bound. These fixations allow a reduction of the size of the problem. Section 5 shows how to incorporate the previous algorithms into a branch-and-bound scheme. Computational experiments are reported in Section 6 and the conclusion is presented in Section 7.

2. Computation of a lower bound

In this section we propose a new method for computing a good feasible solution of (P), i.e. a lower bound of the optimal value. This computation is composed of two stages. The first stage uses the heuristic method presented in Chaillou, Hansen and Mahieu (1983); the second stage consists of trying to improve the obtained feasible solution by a fill-up and exchange procedure as proposed in Gallo, Hammer and Simeone (1980).

Algorithm 1. Finding a first feasible solution x' $(x'_1, x'_2, \ldots, x'_n)$ of (P) (Chaillou, Hansen and Mahieu, 1983).

begin
$K_1 \leftarrow \{1, 2, \ldots, n\}$; $K_0 \leftarrow \emptyset$
for $j = 1$ to n **do** $c_j \leftarrow (\sum_{i=1}^{n} q_{ij})/a_j$ **endfor**
while $\sum_{j \in K_1} a_j > b$ **do**
  let $k \in K_1$ such that $c_k = \min\{c_j \mid j \in K_1\}$

$K_1 \leftarrow K_1 \setminus \{k\}$, $K_0 \leftarrow K_0 \cup \{k\}$
"Comment: update c_j for $j \in K_1$"
for all $j \in K_1$ **do**
if $k < j$ **then** $c_j \leftarrow c_j - q_{kj}/a_j$ **else** $c_j \leftarrow c_j - q_{jk}/a_j$ **endif**
endfor
endwhile
"Comment: the vector x' such that $x'_j = 1$ for all $j \in K_1$ and $x'_j = 0$ for all $j \in K_0$ is a feasible solution of (P)"
end

We now try to improve the obtained bound by using the fill-up and exchange procedure presented in Gallo, Hammer and Simeone (1980). Let $x' = (x'_1, x'_2, \ldots, x'_n)$ be the feasible solution obtained at the end of Algorithm 1. Let e_i be the i-th unit vector. Denote

$$\Delta_k(x') = f(x'_1, \ldots, 1_{(k)}, \ldots, x'_n) - f(x'_1, \ldots, 0_{(k)}, \ldots, x'_n)$$

and

$$\Delta_{ij}(x') = f(x'_1, \ldots, 1_{(i)}, \ldots, 0_{(j)}, \ldots, x'_n) - f(x'_1, \ldots, 0_{(i)}, \ldots, 1_{(j)}, \ldots, x'_n), \quad i \neq j,$$

and remark that $\Delta_{ij}(x') = \Delta_i(x') - \Delta_j(x') + q_{ij}(x_i - x_j)$.

The following algorithm tries to improve the feasible solution x' given by Algorithm 1.

Algorithm 2. Improvement of the feasible solution given by Algorithm 1.
begin
"Comment: fill up"
for $i = 1$ to n **do**
if $x'_i = 0$ and $x' + e_i$ is a feasible solution **then** $x' \leftarrow x' + e_i$ **endif**
endfor
"Comment: exchange"
for all $(i,j) \in \{1, \ldots, n\}^2$ **do**
if $x' - e_i + e_j$ is a feasible solution and $\Delta_{ij}(x') < 0$ **then** $x' \leftarrow x' - e_i + e_j$ **endif**
endfor
end

The 0–1 vector obtained at the end of Algorithm 2 is a feasible solution of (P). Of course the value of this solution is a lower bound of the optimum of (P).

3. Computation of an upper bound

Consider the classical linearization of problem (P) obtained by replacing the quadratic terms $x_i x_j$ of the objective function by the 0–1 variables x_{ij}:

(P_L)

$$\text{Max } z(x) = \sum_{i=1}^{n} q_{ii} x_i + \sum_{1 \leqslant i < j \leqslant n} q_{ij} x_{ij}$$

$$\text{subject to } \sum_{i=1}^{n} a_i x_i \leqslant b, \tag{1}$$

$$x_{ij} \leqslant x_i, \quad 1 \leqslant i < j \leqslant n, \tag{5}$$

$$x_{ij} \leqslant x_j, \quad 1 \leqslant i < j \leqslant n, \tag{6}$$

$$x_i + x_j - 1 \leqslant x_{ij}, \quad 1 \leqslant i < j \leqslant n, \tag{7}$$

$$x_i \in \{0, 1\}, \quad i = 1, \ldots, n, \tag{2}$$

$$x_{ij} \in \{0, 1\}, \quad 1 \leqslant i < j \leqslant n. \tag{8}$$

Note that since all the coefficients q_{ij} in the quadratic knapsack problem are non-negative, the constraints (7) are not necessary.

An upper bound of the optimal value of (P) is immediately obtained by solving the continuous relaxation of (P_L) obtained by replacing constraints (2) and (8) by $0 \leqslant x_i \leqslant 1$ ($i = 1, \ldots, n$) and $x_{ij} \geqslant 0$ ($1 \leqslant i < j \leqslant n$). We refer to this problem as (P_{RL}). Denote by y_i and y_{ij} the variables of this programming problem.

(P_{RL})

$$\text{Max } z(y) = \sum_{i=1}^{n} q_{ii} y_i + \sum_{1 \leqslant i < j \leqslant n} q_{ij} y_{ij}$$

$$\text{subject to } \sum_{i=1}^{n} a_i y_i \leqslant b, \tag{9}$$

$$y_{ij} \leqslant y_i, \quad 1 \leqslant i < j \leqslant n, \tag{10}$$

$$y_{ij} \leqslant y_j, \quad 1 \leqslant i < j \leqslant n, \tag{11}$$

$$y_i + y_j - 1 \leqslant y_{ij}, \quad 1 \leqslant i < j \leqslant n, \tag{12}$$

$$0 \leqslant y_i \leqslant 1, \quad i = 1, \ldots, n, \tag{13}$$

$$y_{ij} \geqslant 0, \quad 1 \leqslant i < j \leqslant n. \tag{14}$$

A few tests have shown that the solution of (P_{RL}) is often very far from the optimum of (P_L) and from the optimum of (P). Therefore we add to (P_{RL}) some new constraints which are redundant for (P_L) but non-redundant for (P_{RL}).

The first set of constraints that we add to the problem (P_{RL}) is deduced from the capacity constraint (1) by multiplying both sides of this inequality by x_j ($j = 1, \ldots, n$). Such constraints are used by Adams and Sherali (1986) for solving linearly constrained 0–1 quadratic programing problems. Obviously, the n constraints obtained in this way lead to redundant constraints for the problem (P_L) but computational experiments have shown that they efficiently reduce the set of feasible solutions of (P_{RL}). These constraints are

$$\sum_{i<j} a_i y_{ij} + \sum_{i>j} a_i y_{ji} \leqslant (b - a_j) y_j, \quad j = 1, \ldots, n. \tag{15}$$

The second set of constraints that we consider are similar to constraints (12) but involve the six variables y_i, y_j, y_k, y_{ij}, y_{ik}, y_{jk}:

$$y_i + y_j + y_k - y_{ij} - y_{ik} - y_{jk} \leqslant 1, \quad 1 \leqslant i < j < k \leqslant n. \tag{16}$$

Finally, we consider the following linear programming problem:

(P'_{RL})

$$\text{Max} \quad z(y) = \sum_{i=1}^{n} q_{ii} y_i + \sum_{1 \leqslant i < j \leqslant n} q_{ij} y_{ij}$$

$$\text{subject to} \quad \sum_{i=1}^{n} a_i y_i \leqslant b, \tag{9}$$

$$y_{ij} \leqslant y_i, \quad 1 \leqslant i < j \leqslant n, \tag{10}$$

$$y_{ij} \leqslant y_j, \quad 1 \leqslant i < j \leqslant n, \tag{11}$$

$$y_i + y_j - 1 \leqslant y_{ij}, \quad 1 \leqslant i < j \leqslant n, \tag{12}$$

$$0 \leqslant y_i \leqslant 1, \quad i = 1, \ldots, n, \tag{13}$$

$$y_{ij} \geqslant 0, \quad 1 \leqslant i < j \leqslant n, \tag{14}$$

$$\sum_{i<j} a_i y_{ij} + \sum_{i>j} a_i y_{ji} \leqslant (b - a_j) y_j, \quad j = 1, \ldots, n, \tag{15}$$

$$y_i + y_j + y_k - y_{ij} - y_{ik} - y_{jk} \leqslant 1, \quad 1 \leqslant i < j < k \leqslant n. \tag{16}$$

Computational tests show that the optimum value of (P'_{RL}) gives a good upper bound for the optimum value of (P). However the computational time is too long because of the large number of constraints. For example, a problem (P'_{RL}) with $n = 20$ includes more than 1500 constraints. On the other hand, we have noted that the solution of (P'_{RL}) without constraints (10), (11), (12) and (16) satisfied many of these constraints. We have therefore chosen to solve problem (P'_{RL}) by progressively considering the constraints. To this end, we minimize $z(y)$ subject to constraints (9), (13), (14) and (15). Then we search for the constraints which are not satisfied by the obtained solution, among (10), (11), (12) and (16) and we dynamically generate these constraints, one by one, from the last obtained optimum. Each new constraint is expressed as a function of the non-basic variables at the current optimal basic solution and the slack variable is chosen as basic variable. We thus obtained a non-feasible basic solution and we apply the dual simplex algorithm to find the new optimum. For example in a problem with $n = 20$, the total number of constraints will typically be about 300 instead of 1500.

Consider now how the upper bound given by an optimal solution, y^*, of (P'_{RL}) can be improved in some cases.

Let $U = \{i \mid y_i^* = 1\}$ and $F = \{i \mid y_i^* \neq 0 \text{ and } y_i^* \neq 1\}$. If F is empty, the optimum y^* is integer and it is a solution of problem (P). Otherwise by noting that at the optimum of (P) all the variables whose index belongs to $U \cup F$ cannot simultaneously be equal to 1 (otherwise we would have $f(x^*) \geqslant z(y^*)$ and that would contradict the fact that $z(y^*)$ is an upper bound), we can add to (P'_{RL}) two new constraints. The first one states that the sum of the variables of index belonging to $U \cup F$ is strictly less than the number of these variables:

$$\sum_{i \in U \cup F} y_i \leqslant |U \cup F| - 1. \tag{17}$$

The second constraint concerns the variables y_{ij} associated with the variables y_i $(i \in U \cup F)$ and y_j $(j \in U \cup F)$. Since at least one of the variables x_i $(i \in U \cup F)$ is equal to zero at the optimum of (P), the sum of the variables y_{ij} must be less than or equal to the number of possible quadratic terms involving $|U \cup F| - 1$ variables:

$$\sum_{\substack{(i,j) \in U \times F \\ i<j}} y_{ij} \leqslant \tfrac{1}{2}(|U \cup F| - 1)(|U \cup F| - 2). \tag{18}$$

Of course these constraints will be added to (P'_{RL}) only if the optimum y^* does not already satisfy them. They will be effective if the values of the variables of index belonging to F are close to 1 and, experimentally, that is often the case. The following algorithm computes an upper bound of the optimum value of (P) based on linear programming problem (P'_{RL}) and on constraints (17) and (18).

Algorithm 3. Computation of an upper bound of the optimum value of (P).
begin
Let y^* be an optimal solution of the current linear program Max $\sum_{i=1}^{n} q_{ii} y_i + \sum_{i<j} q_{ij} y_{ij}$ subject to the constraints (9), (13), (14) and (15)
"Comment: dynamic generation of new constraints"
while y^* does not satisfy all the constraints (9) to (16), (17) and (18) **do**
  while y^* does not satisfy all the constraints (10) and (11) **do**
    let (i', j') be such that $y_{i'j'} - y_{i'} = \max_{i<j}(y_{ij} - y_i)$
    let (i'', j'') be such that $y_{i''j''} - y_{j''} = \max_{i<j}(y_{ij} - y_j)$
    if $y_{i'j'} - y_{i'} \geqslant y_{i''j''} - y_{j''}$
      then add to the current linear program the constraint $y_{i'j'} \leqslant y_{i'}$
      else add to the current linear program the constraint $y_{i''j''} \leqslant y_{j''}$
    endif
    compute by the dual simplex algorithm the solution y^* of the new current linear program
  endwhile
  if $\sum_{i \in U \cup F} y_i^* > |U \cup F| - 1$
    then add to the current linear program the constraint $\sum_{i \in U \cup F} y_i \leqslant |U \cup F| - 1$
    compute by the dual simplex algorithm the solution y^* of the new current linear program
  endif
  if $\sum_{\substack{(i,j) \in U \times F \\ i<j}} y_{ij}^* > \frac{1}{2}(|U \cup F| - 1)(|U \cup F| - 2)$
    then add to the current linear program the constraint
    $\sum_{\substack{(i,j) \in U \times F \\ i<j}} y_{ij} \leqslant \frac{1}{2}(|U \cup F| - 1)(|U \cup F| - 2)$ and compute by the dual simplex algorithm the solution y^* of the new current linear program
  endif
  while there exists a constraint (12) which is not satisfied **do**
    let (i', j') be such that $y_{i'} + y_{j'} - y_{i'j'} = \max_{i<j}(y_i + y_j - y_{ij})$
    add to the current linear program the constraint $y_{i'} + y_{j'} - 1 \leqslant y_{i'j'}$
    compute by the dual simplex algorithm the solution y^* of the new current linear program
  endwhile
  add to the current linear program the constraints (16) which are not satisfied by y^*
  compute by the dual simplex algorithm the solution y^* of the new current linear program
endwhile
end

4. Fixing the variables

When an upper bound has been computed by Algorithm 3, we try to determine the values of certain variables at the optimum of (P). These fixations will allow a reduction of the size of the problem (sometimes all the variables will be fixed) before solving it by a branch and bound procedure. Let

$y^* = (y_1^*, y_2^*, \ldots, y_n^*)$ be the solution obtained by Algorithm 3, LB the lower bound computed by Algorithms 1 and 2 and x^*, the optimum of the quadratic knapsack problem (P).

Property 1. *For all* $i \in \{1, \ldots, n\}$ *let* $\mathrm{UB}_j(0)$ [*respectively* $\mathrm{UB}_i(1)$] *be the upper bound obtained by solving the linear programming problem obtained at the end of Algorithm* 3, *to which is added the constraint* $y_i = 0$ [*respectively* $y_i = 1$]. *If* $\mathrm{UB}_i(0) < \mathrm{LB}$ [*respectively* $\mathrm{UB}_i(1) < \mathrm{LB}$], *then we have* $x_i^* = 1$ [*respectively* $x_i^* = 0$].

Proof. The proof is obvious.

Note that adding the constraint $y_i \geqslant 1$ is equivalent to fixing y_i to 1 and adding the constraint $y_i \leqslant 0$ is equivalent to fixing y_i to 0. This new constraint is expressed as a function of the non-basic variables and the new linear programming problem is solved by the dual simplex algorithm. Practically. we will add the constraint $y_i \geqslant 1$ if $y_i^* < 0.5$ and the constraint $y_i \leqslant 0$ if $y_i^* \geqslant 0.5$ (0.5 is an arbitrary chosen cut-off point).

In some cases, the following properties allow an easy proof that certain variables must take the value 0 at the optimum of (P). The proofs of these properties are obvious.

Property 2. *Let* y^* *be the optimum of value* $z(y^*)$ *obtained at the end of Algorithm* 3 *and, for all* $i \in \{1, \ldots, n\}$, *let* δ_i *be the marginal cost of the variable* y_i. *Then for all* $i \in \{1, \ldots, n\}$ *and such that* $\delta_i < \mathrm{LB} - z(y^*)$, $x_i^* = 0$.

Property 3. *Let* F_1 *be the set of the indices of the variables equal to* 1 *at the optimum of* (P). *For all* $j \in \{1, \ldots, n\}$, *if* $a_j > b - \sum_{i \in F_1} a_i$, *then* $x_j^* = 0$.

Note that, in some cases, the procedure for fixing the variables allows the optimum of (P) to be found or the lower bound to be improved. When computing $\mathrm{UB}_i(\gamma)$, $\gamma \in \{0,1\}$, if we obtain a 0–1 integer solution for y^* with $z(y^*) \geqslant \mathrm{LB}$, we would then compute a new upper bound $\mathrm{UB}_i(1-\gamma)$. If $\mathrm{UB}_i(1-\gamma) \leqslant \mathrm{LB}$, then y^* is optimal to (P), otherwise, $\mathrm{UB}_i(\gamma)$ can serve as a better LB.

Let y^* be the optimal solution obtained at the end of Algorithm 3 with its optimal value $z(y^*)$ serving as an UB of (P). Denote C as the corresponding subset of the constraints. The following algorithm allows us to fix some variables in (P).

Algorithm 4. Fixation of some variables.

begin
for $i = 1$ to n and y_i not yet fixed **do**
  compute $\mathrm{UB}_i(\gamma)$ with $\gamma = 1$ if $y_i^* < 0.5$ and $\gamma = 0$ if $\gamma_i^* \geqslant 0.5$, i.e. solve the linear program
  {max $z(y)$ s.t. $C \cup \{y_i = 1 - \gamma\}$}; let y^* be the obtained optimum
  if $z(y^*) \geqslant \mathrm{LB}$ **then**
    if y^* is a 0–1 vector **then**
      compute the new upper bound $\mathrm{UB}_i(1-\gamma)$
      if $\mathrm{UB}_i(1-\gamma) \leqslant \mathrm{LB}$ **then**
        y^* is an optimum of problem (P)
      else
        $\mathrm{LB} \leftarrow \mathrm{UB}_i(\gamma)$
      endif
  else
    $C \leftarrow C \cup \{y_i = 1 - \gamma\}$

if $\gamma = 0$ **then**
fix the variables which must be equal 0 by using Properties 2 and 3;
let N be the set of the indices of these variables
$C \leftarrow C \cup_{i \in N} \{y_i = 0\}$
endif
endif
endfor
end

Remark. Many computational tests have shown that the computation time is strongly dependent on the order of the fixation of the variables. Indeed, in order to decrease the size of the problem, it is regularly updated as soon as a certain number of variables are fixed. It is thus important to try to quickly fix the maximum number of variables. To this end we have seen that it was interesting to first examine the variables whose coefficients in the capacity constraint were small or large.

5. The branch-and-bound scheme

By iteratively applying Algorithm 3 and Algorithm 4 we finally obtain a reduced quadratic knapsack problem for which no more variables can be fixed. Let y^* be the optimum of the last considered linear program. We then terminate the resolution by a branch-and-bound algorithm. At each node of the search tree we compute an upper bound by adding to the current linear program a new constraint concerning the branching variable and by using Algorithm 3. Let UB be this bound: If UB < LB, then we backtrack; otherwise we branch on the free variable which maximizes the quantity $|y_i^* - \frac{1}{2}|$. Note that this choice favours the observation of the variables which are equal to 0 or 1 but which are not yet fixed.

Algorithm 5. The branch and bound algorithm.
begin
choose the variable x_i for the first branching
if $y_i^* < 0.5$ **then** $\gamma_i \leftarrow 1$ **else** $\gamma_i \leftarrow 0$ **endif**
branch (i, γ_i)
branch $(i, 1 - \gamma_i)$
end

Description of the function 'branch(i, γ_i)':
begin
$x_i \leftarrow \gamma_i$
if $\gamma_i = 1$ **then** fix to 0 the variables which are necessarily equal to 0 (see Property 3) **endif**
compute the upper bound UB by Algorithm 3 and let y^* be the corresponding optimum
if UB $\geqslant$ LB **then**
if y^* is a 0–1 vector **then**
LB $\leftarrow z(y^*)$
else

choose the new branching variable x_i

```
        compute γ_i
        branch (i, γ_i)
        branch (i, 1 − γ_i)
    endif
else
    backtrack
endif
end
```

6. Computational results

We have implemented the branch and bound scheme of the previous section in C language on a HP 9000 and computational experiments have been performed on randomly generated test problems. The solution of the linear programming problems by the dual simplex algorithm are determined by software developed at CEDRIC. As in Gallo, Hammer and Simeone (1980), Chaillou, Hansen and Mahieu (1983) and Michelon and Veuilleux (1991), the coefficients q_{ij} of the objective function are integers uniformly

Table 1
Average computational results for 20 problems with 10 variables and density ranging from 25% to 100%

	1	2	3	4	5	6	7
	$b/\Sigma a_i$	$\%\frac{UB-LB}{LB}$	t_1	$\%\frac{opt-LB}{opt}$	% Fix. var.	Nodes	t_2
25%							
Av.	0.58	8.82	0.21	2.21	70(64)	14.1(7.7)	0.23(0.18)
St. dev.	0.23	9.38	0.12	5.22	34	25.2	0.16
50%							
Av.	0.58	5.01	0.30	0.34	86(79)	6.9(0.1)	0.29(0.11)
St. dev.	0.23	7.03	0.20	1.48	27	17.9	0.20
75%							
Av.	0.58	6.31	0.40	0.18	79(69)	8.1(7.4)	0.41(0.26)
St. dev.	0.23	7.21	0.27	0.79	28	13.7	0.36
100%							
Av.	0.58	5.40	0.43	0.20	82(68)	4.0(6.5)	0.44(0.32)
St. dev.	0.23	6.62	0.24	0.89	23	6.1	0.28

Column 1: $b/\Sigma a_i$: The closer this coefficient is to 1, the less the problem is constrained and the number of generated constraints will in general be large; *column 2:* (UB − LB)/LB = Relative gap between the value of the feasible solution given by Algorithms 1 and 2 and the value of the upper bound obtained by our method (before fixation of variables). i.e. UB is exactly the value obtained by Algorithm 3; *column 3:* t_1 = CPU time (in seconds) for computing UB and LB (the CPU time required for computing LB is negligible); *column 4:* (opt − LB)/opt = relative gap between the optimum value of the quadratic knapsack problem and the value of the feasible solution given by Algorithms 1 and 2. Columns 5, 6 and 7 concern the solution of the whole problem. In this case, in order to not lose time in computing too precise a bound, we stop the computation of UB as soon as (UB − LB)/LB is less than or equal to 5% or 7% (depending on the problem). Moreover we do not consider constraints (12) and (16). The values between parentheses are the results corresponding to the fastest method (among four) proposed in Michelon and Veuilleux (1991) and run on a Sun 4.0 workstation. *Column 5:* % fix. var. = Percentage of fixed variables, by Algorithm 4 before applying the branch-and-bound procedure, out of total variables; *column 6:* nodes = Number of nodes visited in the branch and bound tree; *column 7:* t_2 = CPU time (in seconds) for solving the whole problem (total CPU time required by the execution of Algorithm 1 + Algorithm 2 + Algorithm 3 stopped as soon as (UB − LB)/LB $\leq$ 5% or 7% + Algorithm 4 + Algorithm 5). Note that t_2 is often less than t_1.

Table 2
Average computational results for 20 problems with 20 variables and density ranging from 25% to 100%

	1	2	3	4	5	6	7
	$b/\Sigma a_i$	$\%\frac{UB-LB}{LB}$	t_1	$\%\frac{opt-LB}{opt}$	% Fix. var.	Nodes	t_2
25%							
Av.	0.55	3.23	8.4	0.39	76(61.5)	58.7(34.5)	5.8(2.90)
St. dev.	0.29	3.05	9.8	1.11	27.5	164.1	15.9
50%							
Av.	0.55	3.24	15.9	0.75	69(78.5)	74.6(8.8)	8.7(3.05)
St. dev.	0.29	2.74	13.5	1.29	26.5	130.2	12.6
75%							
Av.	0.55	3.07	26.6	0.46	75(54.5)	57.1(103.2)	17.2(9.09)
St. dev.	0.29	2.60	25.4	1.38	27	137.0	46.3
100%							
Av.	0.55	3.22	26.0	0.22	77.5(53.5)	21.0(37.5)	4.9(8.16)
St. dev.	0.29	3.30	29.2	0.69	17.5	29.4	4.4

Legend as Table 1.

distributed between 0 and 100, the coefficients a_i of the constraint are integers uniformly distributed between 1 and 50, while b is an integer randomly chosen between 50 and $\Sigma_{i=1}^{n} a_i$. Computational results concerning problems involving 10, 20, 30 and 40 variables with a density of the objective function (number of non-zero coefficients divided by $\frac{1}{2}n(n+1)$) equal to 25%, 50%, 75% and 100% are reported in Tables 1 to 4. For 10, 20 and 30 variables, the tests have been performed on 20 instances for each density; for 40 variables, 10 test instances have been run for a density equal to 100%. The average results and the corresponding standard deviations are reported with a comparison to the results obtained by Michelon and Veuilleux (1991) for problems randomly generated in the same way. Tables 5 to 8 present the detailed results corresponding to the 70 test instances of 100% density. These tests have been

Table 3
Average computational results for 20 problems with 30 variables and density ranging from 25% to 100%

	1	2	3	4	5	6	7
	$b/\Sigma a_i$	$\%\frac{UB-LB}{LB}$	t_1	$\%\frac{opt-LB}{opt}$	% Fix. var.	Nodes	t_2
25%							
Av.	0.46	2.73	234.2	0.64	75.3(67.3)	71.2(44.2)	109.0(22.12)
St. dev.	0.29	2.29	292.1	0.96	21.7	25.2	241.7
50%							
Av.	0.46	2.54	138.9	0.32	73(66.3)	59.6(201.3)	58.8(52.97)
St. dev.	0.29	2.08	136.4	0.71	18	78.9	56.0
75%							
Av.	0.46	2.93	246.3	0.45	68(47.3)	83.0(258.1)	101.8(104.09)
St. dev.	0.29	2.81	313.0	1.14	16.74s100.3	115.9	
100%							
Av.	0.46	2.50	227.6	0.17	78.7(59)	102.2(327.1)	65.0(111.89)
St. dev.	0.29	2.71	250.4	0.48	19.3	150.9	65.1

Legend as Table 1.

Table 4
Average computational results for 10 problems with 40 variables and density 100%

	1	2	3	4	5	6	7
	$b/\Sigma a_i$	$\%\frac{UB-LB}{LB}$	t_1	$\%\frac{opt-LB}{opt}$	% Fix. var.	Nodes	t_2
100%							
Av.	0.51	1.33	1082	0.16	72(39.5)	346.1(476.6)	517.3(523.7)
St. dev.	0.23	1.14	944	0.33	1.8	399	623

Legend as Table 1.

performed, on the one hand, to determine the sharpness of the upper bound obtained by Algorithm 3 and, on the other hand, to try to minimize the processing time required for solving the whole problem.

7. Comments and conclusion

The computational experiments reported in Tables 1 to 8 allow us to note several results. First, note the tightness of the lower bound. Tables 5 to 8 show that the joint use of two known methods allows us to obtain the optimum in 62 cases out of 70 and Tables 1 to 4 show that the average value of (opt–LB)/opt is almost always less than 1%. We have noted that Algorithm 2 (exchange and fill-up procedure) improved the solution given by Algorithm 1 in 60% of cases and that this improvement was about 2%.

Table 5
Detailed computational results for the 20 problems of Table 1 with 10 variables and 100% density

1	2	3	4	5	6	7
$b/\Sigma a_i$	$\%\frac{UB-LB}{LB}$	t_1	$\%\frac{opt-LB}{opt}$	% Fix. var.	Nodes	t_2
0.80	0	0.6	0	100	0	0.2
0.85	0.22	0.5	0	100	0	0.2
0.54	7.51	0.8	0	40	18	1.0
0.59	1.98	0.4	0	60	8	0.2
0.49	5.18	0.4	0	60	4	0.6
0.82	11.07	0.3	0	50	16	1.0
0.61	11.63	0.1	0	50	12	0.3
0.96	0	0.8	0	100	0	0.6
0.35	3.25	0.3	0	100	0	0.2
0.63	1.32	0.8	0	100	0	0.6
0.20	28.08	0.1	0	100	0	0.1
0.60	2.80	0.6	0	100	0	0.8
0.28	1.13	0.1	0	100	0	0.4
0.47	8.33	0.3	3.97	40	14	0.6
0.91	0	0.3	0	100	0	0.3
0.20	10.93	0.1	0	90	2	0.2
0.71	2.59	0.7	0	100	0	0.4
0.28	1.18	0.5	0	100	0	0.2
0.56	6.97	0.5	0	70	4	0.7

Legend as Table 1.

Table 6
Detailed computational results for the 20 problems of Table 2 with 20 variables and 100% density

1	2	3	4	5	6	7
$b/\Sigma a_i$	$\%\frac{UB-LB}{LB}$	t_1	$\%\frac{opt-LB}{opt}$	% Fix. var.	Nodes	t_2
0.92	0.08	60.6	0	100	0	3.8
0.64	0.52	27.3	0	100	0	2.6
0.52	3.03	11.7	0	55	76	9.0
0.29	2.47	4.4	0	80	10	2.0
0.89	3.31	104.6	0	45	102	21.2
0.98	0	70.1	0	100	0	5.0
0.82	3.68	75.7	0	65	36	4.9
0.53	3.25	4.8	1.92	75	10	3.0
0.98	0	33.4	0	100	0	3.2
0.34	6.41	6.8	0	50	63	8.8
0.49	2.44	21.4	0	60	29	5.1
0.62	0.81	22.6	0	85	4	6.6
0.16	8.77	4.0	0	80	2	3.2
0.13	11.85	1.0	0	60	47	2.7
0.56	0.53	15.5	0	85	2	5.2
0.85	1.53	23.1	0	75	8	3.7
0.59	0.93	25.7	0	100	0	2.2
0.46	2.24	5.1	0	75	10	3.0
0.10	8.72	0.6	0	75	18	0.9
0.18	3.73	2.4	2.54	85	2	1.7

Legend as Table 1.

The results presented in the column 2 of Tables 1 to 4 ((UB–LB)/LB) show that our method gives a good upper bound especially for large problems with 100% density. Indeed, the relative gap between this bound and the value of the feasible solution computed by Algorithms 1 and 2 is almost always less than 3% for problems with 40 variables (see Table 8) and, to the best of our knowledge, it is one of the best methods regarding the tightness of the upper bound. However, the computation time (t_1) required to obtain this bound is sometimes significant, particularly when the relative gap is very small (less than 1%) or when the problem is only slightly constrained. That is due to the fact that in these cases the number of generated constraints is large (for example, a 20 variables problem, for which the upper bound is equal to the optimum value, requires the solution of a linear programming problem with 422 constraints). Many tests have shown that it was not interesting, from a computation time point of view, to compute the best possible upper bound at the root of the search tree when one wants to solve the whole problem. We have thus limited the relative gap, (UB–LB)/LB, to 5% for most of the problems and to 7% for the slightly constrained problems in order to have a number of generated constraints which is not too large. Moreover we no longer consider constraints (12) and (16). Of course after the computation of this upper bound (which is not as tight as possible) we use Algorithm 4 for fixing some variables and then apply the branch-and-bound algorithm. Columns 5, 6 and 7 of Tables 1 to 4 report the corresponding results.

The tightness of the lower bound associated with the tightness of the upper bound (limited to 5% or 7%) allows us to fix many variables at the root of the branch-and-bound tree. Tables 1 to 4 show that our method always allows us to fix more variables than the best of the four methods presented in Michelon and Veuilleux (1991) except in one case. For example in the 40 variables problems with 100% density an average of 28.8 variables are fixed by our method while 15.8 are fixed by the other method. This result

Table 7
Detailed computational results for the 20 problems of Table 3 with 30 variables and 100% density

1	2	3	4	5	6	7
$b/\Sigma a_i$	$\%\frac{UB-LB}{LB}$	t_1	$\%\frac{opt-LB}{opt}$	% Fix. var.	Nodes	t_2
0.14	8.16	18.1	0	60	103	37.6
0.65	0.20	230.9	0	100	0	63.5
0.27	3.27	47.5	0	63.3	142	24.0
0.40	0.79	86.6	0	63.3	98	42.1
0.52	2.79	230.8	0	50	254	67.8
0.91	1.96	940.0	0	100	0	81.3
0.52	0.86	223.3	0	76.7	24	64.2
0.51	0	313.1	0	100	0	125.8
0.18	4.95	26.6	1.08	70	68	22.0
0.86	0.43	469.8	0	100	0	39.8
0.37	0.57	97.7	0	100	0	10.3
0.19	2.86	21.2	0.47	66.7	77	24.4
0.86	1.90	728.7	0	53.3	389	277.5
0.08	1.76	15.2	0	100	0	4.8
0.65	4.48	148.6	1.89	53.3	569	155.2
0.10	10.23	7.4	0	70	86	10.3
0.17	2.62	15.1	0	80	28	9.8
0.68	1.57	282.8	0	63.3	206	115.4
0.97	0	284.2	0	100	0	40.5
0.64	0.51	364.3	0	100	0	83.4

Legend as Table 1.

has an immediate consequence on the number of nodes generated in the branch-and-bound tree: When the number of variables fixed by our method is significantly greater than the number of variables fixed by the other method there is a significant decrease of the number of nodes of the search tree. Note how

Table 8
Detailed computational results for the 10 problems of Table 4 with 40 variables and 100% density

1	2	3	4	5	6	7
$b/\Sigma a_i$	$\%\frac{UB-LB}{LB}$	t_1	$\%\frac{opt-LB}{opt}$	% Fix. var.	Nodes	t_2
0.48	2.10	1865	0.87	50	817	886
0.22	0	91	0	100	0	23
0.85	* [a]	*	0	57.5	980	2220
0.59	0.15	1120	0	80	26	285
0.43	0.59	444	0	87.5	8	158
0.22	0	137	0	100	0	58
0.58	1.20	1035	0	77.5	76	270
0.94	2.62	3300	0	55	293	665
0.39	3.08	635	0.78	55	1003	433
0.37	2.24	1110	0	57.5	258	175

Legend as Table 1.
[a] *: The computation of UB requires too many constraints for our linear programming software.

little influence the density has on the number of fixed variables and thus on the number of nodes. Unfortunately the quality of the two bounds and the significant improvement in the fixation of variables do not actually lead to a substantial decrease in the computation time. However several directions can be investigated to try to improve this computation time: Use a more efficient dual simplex algorithm, determine the best frequency to simplify the problem when some variables are fixed and find a method which avoids the consideration of redundant constraints in the linear programming problem.

References

Adams, W.P., and Sherali, H.D. (1986), "A tight linearization and an algorithm for zero-one quadratic programming problems", *Management Science* 32/10, 1274–1290.

Barahona, F., Jünger, M., and Reinelt, G. (1989), "Experiments in quadratic 0–1 programming", *Mathematical Programming* 44, 127–137.

Billionnet, A., and Sutter, A. (1994), "Minimization of a quadratic pseudo-Boolean function", *European Journal of Operational Research* 78, 106–115.

Billionnet, A., and Costa, M.-C., and Sutter, A. (1989), "Les problèmes de placement dans les systèmes distribués", *Technique et Sciences Informatiques* 8/4, 307–337.

Carter, M.W. (1984), "The indefinite zero-one quadratic problem", *Discrete Applied Mathematics* 7, 23–44.

Chaillou, P., Hansen, P., and Mahieu, Y. (1983), "Best network flow bound for the Quadratic Knapsack Problem", presented at NETFLOW 83 International Workshop, Pisa, Italy.

Chardaire, P., and Sutter, A. (1995), "A decomposition method for quadratic 0–1 programming", *Management Science* 41/4, 704–712.

Gallo, G., and Simeone, B. (1988), "On the Supermodular Knapsack Problem", *Mathematical Programming* 45, 295–309.

Gallo, G., Grigoriadis, M.D., and Tarjan, R.E., (1989), "A fast parametric maximum flow algorithm and application", *SIAM Journal on Computing* 18/1, 30–35.

Gallo, G., Hammer, P.L., Simeone, B. (1980), "Quadratic Knapsack Problems", *Mathematical Programming* 12, 132–149.

Hammer, P.L., Hansen, P., and Simeone, B. (1984), "Roof duality, complementation and persistency in quadratic 0–1 optimization", *Mathematical Programming* 28, 121–155.

Hansen, P. (1979), "Methods of non-linear 0–1 programming", *Annals of Discrete Mathematics* 5, 53–70.

Körner, F. (1986), "A new bound for the quadratic knapsack problem and its use in a branch and bound algorithm", *Optimization* 17, 643–648.

Laughhunn, D.J. (1970), "Quadratic binary programming with applications to capital budgeting problems", *Operations Research* 18, 454–461.

Michelon, P., and Maculan, N. (1993), "Lagrangean methods for 0–1 quadratic programming", *Discrete Applied Mathematics* 42/2–3, 257–269.

Michelon, P., and Veuilleux, L. (1991), Lagrangean methods for the 0–1 Quadratic Knapsack Problem", Publication #779, D.I.R.O., Université de Montréal.

Pardalos, P.M., and Rodgers, G. (1990), "Computational aspects of a branch and bound algorithm for quadratic zero-one programming", *Computing* 45, 131–144.

Rao, G.S., Stone, H.S., and Hu, T.C. (1979), "Assignment of tasks in a distributed processor system with limited memory", *IEEE Transactions on Computers* 28/4, 291–299.

Williams, A.C. (1985), "Quadratic zero-one programming using the roof dual with computational results", Rutcor Research Report, 8–85, Rutgers University.

Witzgall, C. (1975), "Mathematical methods of site selection for Electronic Message System (EMS)", NBS Internal Report.