Discrete Applied Mathematics 3 (1981) 275–288
North-Holland Publishing Company

A BOUND AND BOUND ALGORITHM FOR THE ZERO-ONE MULTIPLE KNAPSACK PROBLEM

Silvano MARTELLO and Paolo TOTH

Istituto di Automatica, University of Bologna, Italy

Received 28 February 1980
Revised 18 February 1981

By the term "Bound and Bound" we define a particular tree-search technique for the ILP, which, for a maximization problem, makes use of a lower bound to determine the branches to follow in the decision tree. This technique is applied to the solution of the Zero-One Multiple Knapsack Problem and an algorithm is derived; an illustrative example of the procedure is provided. We present extensive computational results showing that the method is capable of solving problems up to 4 knapsacks and 200 variables with running times considerably smaller than those of the most commonly utilized algorithms.

Introduction

Let $N=\{1,2,\ldots,n\}$ be a set of integers labelling n items, each having a profit p_j and a weight w_j; let $M=\{1,2,\ldots,m\}$ be a set of integers labelling m knapsacks, each having a capacity c_i; define the elements of an $(m\times n)$ matrix $(x_{i,j})$ as

$$x_{i,j}\begin{cases}=1 & \text{if item } j \text{ is assigned to knapsack } i;\\ =0 & \text{otherwise.}\end{cases}$$

Formally, the *Zero-One Multiple Knapsack Problem* consists in determining $(x_{i,j})$ so as to

(P)

$$\text{maximize} \quad \sum_{i\in M}\sum_{j\in N} p_j x_{i,j}, \tag{1}$$

$$\text{subject to} \quad \sum_{j\in N} w_j x_{i,j}\le c_i \quad \text{for all } i\in M; \tag{2}$$

$$\sum_{i\in M} x_{i,j}\le 1 \quad \text{for all } j\in N; \tag{3}$$

$$x_{i,j}\in\{0,1\} \quad \text{for all } i\in M, j\in N. \tag{4}$$

Informally, the problem consists in assigning items to the knapsacks such that the total profit of the assigned items is maximum (1), the total weight assigned to each knapsack does not exceed the corresponding capacity (2), each item is either

0166-218X/81/0000–0000/$02.75

assigned to one of the knapsacks or rejected (3), (4).

Without loss of generality, the following assumptions can be made:

(a) $p_j,\ w_j > 0$ and integers for all $j \in N$;

(b) $c_i > 0$ and integer for all $i \in M$;

(c) $\min_j \{w_j\} \leq \min_i \{c_i\}$;

(d) $\max_j \{w_j\} \leq \max_i \{c_i\}$;

(e) $\sum_{j \in N} w_j > \max_i \{c_i\}$.

In addition, we will assume that the items are ordered according to decreasing values of the profit per unit weight:

(f) $p_1/w_1 \geq p_2/w_2 \cdots \geq p_n/w_n$.

The well-known *Zero-One Single Knapsack Problem*, which generally arises as a subproblem when solving (P), is a particular case easily obtainable from eqns. (1), (2), (3) and (4) by setting $m = 1$:

$$\text{(P')} \qquad \begin{aligned} &\text{maximize} && \sum_{j \in N} p_j x_j, \\ &\text{subject to} && \sum_{j \in N} w_j x_j \leq c; \\ & && x_j \in \{0, 1\} \quad \text{for all } j \in N. \end{aligned}$$

It is known that (P') belongs to the set of NP-complete problems; it follows that (P) too is NP-complete, so enumerative algorithms are generally used for its solution.

This paper firstly describes the relaxation techniques commonly utilized for the computation of upper bounds to (P), and reviews the previous works on the subject.

The concept of "bound and bound" tree-search technique is then introduced and a "bound and bound" algorithm for (P) is derived.

Extensive computational results are employed to compare the proposed algorithm with that of Hung and Fisk [2] and with a previous algorithm of the authors [6].

1. Relaxations

Two relaxation methods are generally employed to determine good upper bounds for (P): the Lagrangean Relaxation and the Surrogate Relaxation.

The *Lagrangean Relaxation* of (P), relative to a nonnegative vector (λ_j), can be defined as

$$(\mathrm{PL}_\lambda)\qquad \begin{aligned} &\text{maximize} && \sum_{i\in M}\sum_{j\in N} p_j x_{i,j} - \sum_{j\in N}\lambda_j\Big(\sum_{i\in M} x_{i,j} - 1\Big),\\ &\text{subject to} && \sum_{j\in N} w_j x_{i,j} \le c_i \quad \text{for all } i\in M;\\ & && x_{i,j}\in\{0,1\} \quad \text{for all } i\in M,\ j\in N. \end{aligned}$$

Since the objective function of (PL_λ) can be written as

$$\sum_{j\in N}\lambda_j + \max\left\{\sum_{i\in M}\sum_{j\in N}(p_j-\lambda_j)x_{i,j}\right\},$$

it is easy to see that (PL_λ) decomposes into a series of single knapsack problems.

The *Surrogate Relaxation* of (P), relative to a nonnegative vector (σ_i), can be defined as

$$(\mathrm{PS}_\sigma)\qquad \begin{aligned} &\text{maximize} && \sum_{i\in M}\sum_{j\in N} p_j x_{i,j},\\ &\text{subject to} && \sum_{i\in M}\sigma_i\sum_{j\in N} w_j x_{i,j} \le \sum_{i\in M}\sigma_i c_i;\\ & && \sum_{i\in M} x_{i,j}\le 1 \quad \text{for all } j\in N;\\ & && x_{i,j}\in\{0,1\} \quad \text{for all } i\in M,\ j\in N. \end{aligned}$$

Let z be an index such that $\sigma_z = \min_{i\in M}\{\sigma_i\}$ and suppose an optimal solution to (PS_σ) is given by $(x^*_{i,j})$; an equivalent solution can be obtained by setting $x^*_{i,j}=0$ and $x^*_{z,j}=1$ for each j such that $x^*_{i,j}=1$ with $i\neq z$. Thus (PS_σ) is equivalent to the single knapsack problem:

$$\begin{aligned} &\text{maximize} && \sum_{j\in N} p_j x_{z,j},\\ &\text{subject to} && \sum_{j\in N} w_j x_{z,j} \le \left[\sum_{i\in M}\sigma_i c_i/\sigma_z\right];^{1}\\ & && x_{z,j}\in\{0,1\} \quad \text{for all } j\in N. \end{aligned}$$

Since $[\sum_{i\in M}\sigma_i c_i/\sigma_z] \ge \sum_{i\in M} c_i$, the choice $\sigma_i = k$ (k any positive constant) for all $i\in M$ leads to the minimum value of the optimal solution to (PS_σ), that is to the tightest upper bound the Surrogate Relaxation (PS_σ) can give for (P).

2. Previous works

Algorithms for the Zero-One Multiple Knapsack Problem can be subdivided in

[1] [a] = largest integer not greater than a.

two classes, depending on the relative values of n and m. The problem, in fact, is hard, in the sense that all the currently known algorithms involve running times that grow steeply when n and/or m grow; so, the algorithms are especially oriented either to the case of low values of the ratio n/m (as, for example, when m liquids, that cannot be mixed, have to be loaded into n tanks) or to the case of high values of this ratio (as, for example, when m ships have to be loaded with n containers).

Algorithms for the first class have been given by Christofides, Carpaneto, Mingozzi and Toth [1] and by Neebe and Dannenbring [7].

This paper is devoted to the case of high values of the ratio n/m, so only algorithms for this class of problem will here be summarized.

Hung and Fisk [2] have proposed a depth-first branch and bound algorithm, where successively higher levels of the decision-tree are constructed by selecting an item and assigning it to knapsacks in decreasing order of their capacities; when all the knapsacks have been considered, the item is assigned to a dummy knapsack, implying its exclusion from the solution: so, each node of the decision-tree generates $m+1$ descendent nodes. The upper bound associated with each node can be computed either as the solution of the Lagrangean Relaxation, or the Surrogate Relaxation, of the current problem, or as the smaller of the two; the (λ_j) and the (σ_i) are set respectively equal to the optimal dual multipliers of constraints (3) and (2) in the continuous problem given by the current problem with constraint $0 \leq x_{i,j} \leq 1$ instead of $x_{i,j} \in \{0,1\}$.

Martello and Toth [6] have obtained better results, as regards running times, through a different branching scheme where, at each node, the Lagrangean Relaxation of the current problem is computed (it is assumed that $\lambda_j = 0$ for all j, so each knapsack of the relaxed problem can be solved independently of the others). If no item appears in two or more knapsacks, a feasible solution to the current problem has been found and a backtracking can be performed. Otherwise, an item which appears in m' $(2 \leq m' \leq m)$ knapsacks is selected and m' nodes are generated ($m'-1$ by assigning the item to the first $m'-1$ knapsacks where it appears, the m'-th one by excluding it from them); the m' upper bounds are computed by solving only m' single knapsacks and utilizing part of the solutions previously found for the ascendent nodes; each bound can be improved by assuming the smaller between it on the one hand and the solution of the corresponding Surrogate Relaxation on the other.

Ingargiola and Korsh [3] have proposed a reduction procedure which utilizes dominance relations among the items in order to determine which items must be included and which must be excluded for an optimal solution to (P). This method, starting from a feasible solution to (P), allows one to define two disjoint sets of integers:

$$J1 = \left\{ j \in N \mid \sum_{i \in M} x_{i,j} = 1 \text{ in an optimal solution to (P)} \right\};$$

$$J0 = \left\{ j \in N \mid \sum_{i \in M} x_{i,j} = 0 \text{ in an optimal solution to (P)} \right\}.$$

The original problem can then be reduced by eliminating the items whose label is in $J0$; in addition, the difficulty of the reduced problem is further lessened if one reflects that the items whose labels are in $J1$ must be included in the optimal solution to (P).

3. The bound and bound technique

By the term "Bound and Bound" we define a particular depth-first tree-search technique that will here be described for the 0-1 linear programming problem:

$$\text{(IP)} \qquad \begin{aligned} &\text{maximize} && \sum_{j \in R} c_j x_j, \\ &\text{subject to} && \sum_{j \in R} a_{i,j} x_j \leq b_i \quad \text{for all } i \in T; \\ & && x_j \in \{0,1\} \quad \text{for all } j \in R, \end{aligned}$$

with $R = \{1, 2, \ldots, r\}$, $T = \{1, 2, \ldots, t\}$.

Let *partial solution* S be a stack containing the labels of those variables that are fixed: a label in S is *marked* (*unmarked*) if the corresponding variable is *fixed to* 0 (*fixed to* 1).

We define the *current problem* corresponding to S as (IP) with the additional constraints given by fixing the variables whose labels are in S; let *upper*(S) be an upper bound to this problem.

Let Π be a heuristic procedure which, when applied to the current problem corresponding to S, has the following properties:

(a) a feasible solution $(\bar{x}_j)$ is always found, if one exists;

(b) no nul element in $(\bar{x}_j)$ can be set to 1 without violating the constraints.

Obviously, a lower bound to the current problem corresponding to S is $lower(S) = \sum_{j \in R} c_j \bar{x}_j$.

A bound and bound algorithm, producing an optimal solution (x_j^*) of value V^* to (IP), consists of the following steps:

Step 1. [*Initialization*]
Set $S = \emptyset$, $V^* = -\infty$.

Step 2. [*Heuristic*]
Apply Π to the current problem corresponding to S.
If no feasible solution exists, go to Step 4.
If $lower(S) \leq V^*$, go to Step 3.
Set $V^* = lower(S)$, $(x_j^*) = (\bar{x}_j)$.
If $lower(S) = upper(S)$, go to Step 4.

Step 3. [*Updating*]
Let j be the first label $\in (R-S)$ such that $\bar{x}_j = 1$.
If no such j exists, go to Step 4.
Set $S = S \cup \{j\}$.
If $upper(S) > V^*$, repeat Step 3.

Step 4. [*Backtracking*]
Let h be the last unmarked label in S (if no such h exists, stop): mark h and set $S = S - \{j \in S \mid j \text{ follows } h \text{ in } S\}$.
If $upper(S) \leq V^*$, repeat Step 4.
Go to Step 2.

The main difference between the above approach and a depth-first branch and bound technique is that the branching phase is here performed by updating the partial solution through the solution obtained from the computation of a lower bound; this gives two important advantages:

(a) For all S for which $lower(S) = upper(S)$, $(\bar{x}_j)$ is obviously an optimal solution to the corresponding current problem, so that it is possible to avoid the updating of all the variables whose labels are in $\{j \in (R-S) \mid \bar{x}_j = 1\}$ and the corresponding useless upper bound computations and backtrackings.

(b) For all S for which $lower(S) < upper(S)$, iterated executions of step 3 update S through the solution previously found by Π: the resulting partial solution is generally better than that which would be obtained by a series of forward steps, each fixing a variable independently of the following ones.

On the other hand, in case (b), it is possible that the computational effort spent to obtain $lower(S)$ through Π may be partially useless: this happens when, after a few iterations of Step 3, condition $upper(S) \leq V^*$ holds.

In general, we feel that the bound and bound technique can be successfully applied to problems satisfying the following conditions:

(I) There can be found a "fast" heuristic procedure producing "good" lower bounds;

(II) The relaxation technique utilized to obtain the upper bounds leads to solutions whose feasibility for the current problem is difficult to check or is seldom verified.

If condition (II) is not satisfied, it is generally better to employ a different tree-search technique, making use of the property that solutions to the relaxed problem are often feasible; this is the case, for example, with Martello-Toth's algorithm [6] outlined in Section 2 and with the most efficient algorithms for the Travelling Salesman Problem.

4. A bound and bound algorithm for the zero-one multiple knapsack problem

We will solve the zero-one multiple knapsack problem (P) through a bound and

bound scheme, where each node of the decision-tree generates two branches either by assigning an item j to a knapsack i or by excluding j from i; this is done by fixing the elements of a matrix $(\hat{x}_{i,j})$ respectively to 1 or to 0. In this case a partial solution S contains those pairs (i,j) of labels such that $\hat{x}_{i,j}$ is fixed.

Because of the structure of (P), a feasible solution to a current problem always exists. The following heuristic can be employed to obtain a "good" feasible solution to the current problem corresponding to S: find an optimal solution for the first knapsack, then exclude the items inserted in it and find an optimal solution for the second knapsack, and so on. In detail:

Procedure Π

Step 1. Let $\bar{V}$ be the sum of the profits of the items fixed to 1 in S and let N' be the set of the remaining items. Set $u=1$.

Step 2. Let k_u be the "unfilled" capacity of knapsack u and $\bar{N}_u$ be the subset of N' containing the items not fixed to 0 for knapsack u. Set $(\bar{x}_{u,v}=\hat{x}_{u,v}$ for all (u,v) in $S)$.
If $\bar{N}_u=\emptyset$, go to Step 3.
Exactly solve the single knapsack problem given by (P′) with N and c replaced, respectively, by $\bar{N}_u$ and k_u: store the solution vector in the uth row of matrix $(\bar{x}_{q,j})$.
Set $\bar{V}=\bar{V}+$ value of the solution to (P′).

Step 3. If $u=m$, set $lower(S)=\bar{V}$ and return.
Remove from N' those items j for which $\bar{x}_{u,j}=1$, set $u=u+1$ and go to Step 2.

An upper bound for the current problem can be computed through the surrogate relaxation (PS_σ) with $\sigma_i=1$ for all $i\in M$:

Procedure Σ

Let V be the sum of the profits of the items fixed to 1 in S, let N' be the set of the remaining items and let c' be the sum of the "unfilled" capacities of the knapsacks.
Exactly solve the single knapsack problem given by (P′) with N and c replaced, respectively, by N' and c'.
Set $upper(S)=V+$ value of the solution to (P′).
Return.

In order to outline a bound and bound scheme for (P) it only remains for us to establish the order in which labels are inserted in stack S at Step 3 of the algorithm of the previous section: since the heuristic finds a solution for each single knapsack, it is worthwhile to insert in S couples of labels (i,j) for which $\bar{x}_{i,j}=1$ in increasing order of i and then of j.

This implies that, when considering the node of the decision-tree associated to (i,j) – that is the node which generates the branches corresponding to $\hat{x}_{i,j}=1$ and $\hat{x}_{i,j}=0$ –, all the knapsacks whose labels are less than i are "completely loaded" (in

the sense that no further item can be inserted in them) and all the knapsacks whose labels are greater than i are "empty", while only knapsack i is "partially loaded". The following considerations can then be derived in order to improve on the algorithm's efficiency (when considering the node associated to (i, j)):

(a) In procedure Π, at Step 1, index u can be initialized to i instead of to 1 (it is then necessary to set $(\bar{x}_{u,v} = \hat{x}_{u,v}$ for all $(u, v) \in S)$ and $(\bar{x}_{u,v} = 0$ for all $v \in N')$ for all $u < i)$; in procedure Σ we can compute c' by considering only the knapsacks from i to m.

(b) If $i = m$, it emerges that $upper(S) = lower(S)$ for any S; it follows that no couple (m, j) will be inserted in S, so that no updating nor backtracking steps will be executed on items inserted in the m-th knapsack.

Because of consideration (b), it is worthwhile to arrange the knapsacks so that

$$c_1 \leq c_2 \leq \cdots \leq c_m.$$

The detailed statement of the algorithm we propose for the Zero-One Multiple Knapsack Problem follows:
Meaning of the variables utilized:

$$x^*_{q,j} = \begin{cases} 1 & \text{if item } j \text{ is assigned to knapsack } q \text{ in the best solution so far;} \\ 0 & \text{otherwise;} \end{cases}$$

$$V^* = \sum_{q \in M} \sum_{j \in N} p_j x^*_{q,j};$$

$i =$ knapsack currently considered $(i \leq m - 1)$;

$$\hat{x}_{q,j} = \begin{cases} 1 & \text{if item } j \text{ is assigned to knapsack } q \text{ in the current solution;} \\ 0 & \text{otherwise;} \end{cases}$$

$$V = \sum_{\substack{q \in M \\ q \leq i}} \sum_{j \in N} p_j \hat{x}_{q,j};$$

$S_{q,0} =$ pointer to the last item inserted in knapsack q ($S_{q,0} = -1$ if knapsack q is empty);

$S_{q,j} =$ pointer to the item inserted in knapsack q just before item j ($S_{q,j} = -1$ if j is the first item inserted in knapsack q); $S_{q,j} \neq 0$ iff $\hat{x}_{q,j}$ is fixed;

$\bar{N}_q =$ set of the items which can be inserted in knapsack q;

$$b_j = 1 - \sum_{\substack{q \in M \\ q \leq i}} \hat{x}_{q,j};$$

$$k_q = \begin{cases} c_q - \sum_{j \in N} w_j \hat{x}_{q,j} & \text{for } q \leq i; \\ c_q & \text{for } q > i; \end{cases}$$

$(\bar{x}_{q,j}) =$ matrix whose rows give the solutions found in the last application of procedure Π.

$(x^*_{q,j})$, (b_j) and (k_q) are defined for all $q \in M$ and $j \in N$; $(\hat{x}_{q,j})$ and $(S_{q,j})$ for all $q \in \{u \in M \mid u \leq i\}$ and $j \in N$; $(\bar{N}_q)$ and $(\bar{x}_{q,j})$ for all $q \in \{u \in M \mid u \geq i\}$ and $j \in \bar{N}_q$.

Algorithm

Step 1. [*Initialization*]
Arrange the knapsacks in increasing order of their capacities c_i.
Set $(b_v = 1$ for all $v \in N)$, $(k_q = c_q$ for all $q \in M)$, $(\hat{x}_{q,v} = 0$ for all $q \in M - \{m\}$, $v \in N)$.
Set $(S_{q,v} = 0$ for all $q \in M - \{m\}$, $v \in N)$, $(S_{q,0} = -1$ for all $q \in M - \{m\})$, $V^* = V = 0$, $i = 1$.
Apply Σ to the current problem corresponding to S and set $U = upper(S)$.

Step 2. [*Heuristic*]
Apply Π to the current problem corresponding to S and set $L = lower(S)$.
If $L \leq V^*$, go to Step 3.
Set $V^* = L$; for all $v \in N$, set $(x^*_{q,v} = \hat{x}_{q,v}$ for all $q \in \{u \in M \mid u < i\})$, $(x^*_{i,v} = \hat{x}_{i,v} + \bar{x}_{i,v})$, $(x^*_{q,v} = \bar{x}_{q,v}$ for all $q \in \{u \in M \mid u > i\})$.
If $L = U$, go to Step 4.

Step 3. [*Updating*]
Set $i = \min\{u \in M \mid i \leq u < m, \bar{x}_{u,j} = 1$ for some $j \in \bar{N}_u\}$ (if this set is empty, set $i = m - 1$ and go to Step 4).
3.1. Let j be the next label in $\bar{N}_i$ such that $\bar{x}_{i,j} = 1$.
Set $\hat{x}_{i,j} = 1$, $V = V + p_j$, $k_i = k_i - w_j$, $b_j = 0$, $S_{i,j} = S_{i,0}$, $S_{i,0} = j$.
If j is the last label in $\bar{N}_i$ such that $\bar{x}_{i,j} = 1$, set $i = \min\{u \in M \mid i < u < m, \bar{x}_{u,j} = 1$ for some $j \in \bar{N}_u\}$ (if this set is empty, set $i = m - 1$ and go to Step 4).
Apply Σ to the current problem corresponding to S and set $U = upper(S)$.
If $U > V^*$, repeat Step 3.1.

Step 4. [*Backtracking*]
If $S_{i,0} = -1$, set $(S_{i,v} = 0$ for all $v \in N)$, $i = i - 1$. If $i = 0$, stop; otherwise, repeat Step 4.
Set $j = S_{i,0}$, $\hat{x}_{i,j} = 0$, $V = V - p_j$, $k_i = k_i + w_j$, $b_j = 1$, $(S_{i,v} = 0$ for each $v \mid S_{i,v} = j)$, $S_{i,0} = S_{i,j}$.
Apply Σ to the current problem corresponding to S and set $U = upper(S)$.
If $U \leq V^*$, repeat Step 4.
Go to Step 2.

Procedure Π

Step 1. Set $N' = N - \{z \in N \mid b_z = 0\}$, $\bar{N}_i = N' - \{z \in N' \mid S_{i,z} \neq 0\}$.
Set $\bar{V} = V$, $u = i$.

Step 2. Set $(\bar{x}_{u,z} = 0$ for all $z \in N - \bar{N}_u)$.
If $\bar{N}_u = \emptyset$, go to Step 3.

Exactly solve the single knapsack problem given by (P') with N and c replaced, respectively, by $\bar{N}_u$ and k_u: store the solution vector in the uth row of matrix $(\bar{x}_{q,j})$. Set $\bar{V} = \bar{V}$ + value of the solution to (P').

Step 3. If $u = m$, set *lower*$(S) = \bar{V}$ and return.
Set $N' = N' - \{z \in \bar{N}_u \mid \bar{x}_{u,z} = 1\}$, $u = u + 1$, $\bar{N}_u = N'$ and go to Step 2.

Procedure Σ

Set $N' = N - \{z \in N \mid b_z = 0\}$, $c' = k_i + \sum_{q \in M, q > i} c_q$.
Exactly solve the single knapsack problem given by (P') with N and c replaced, respectively, by N' and c'.
Set *upper*$(S) = V$ + value of the solution to (P').
Return.

5. Parametric computation of the upper bounds

The following property of the algorithm of the previous section can be utilized to reduce the computational effort spent in determining the upper bounds.

Consider a node a_1 associated to (i_1, j_1): procedure Σ defines a solution vector (y_z^1) to (P') and an upper bound UB^1; let $r^1 = c' - \sum_{z \in N'} w_z y_z^1$ be the residue of the surrogate knapsack c'. Now consider a node a_2 associated to (i_2, j_2), descending from a_1; it can be easily verified that, if the following conditions hold:

(1) $y_z^1 = 1$ for all items z for which $\hat{x}_{u,z}$ has been set to 1 for some u on the branches leading from a_1 to a_2,

(2) $r^1 \geq \sum_{i_1 \leq u < i_2} k_u$,

then the upper bound UB^2computed from procedure Σ is equal to UB^1.

From this property we can derive two conclusions of use in parametrically computing the upper bounds in the algorithm of the previous section:

(a) At Step 4, *upper*(S) is equal to the upper bound previously defined for the node associated to (i, j). So, at Step 3.1, after the first line we can insert the statement "Set $UB_j = U$" and at Step 4 we can replace the application of Σ by the statement "Set $U = UB_j$".

(b) In order to reduce to a minimum the number of applications of Σ at Step 3.1, it would be necessary to store, at each node in which Σ must be applied, not only *upper*(S), but also the corresponding solution vector and residue of the surrogate knapsack; then, at Step 3.1, before applying Σ, we could check conditions (1) and (2). The large storage requirement generally makes this approach impractical, but an alternative (reduced) improvement can be obtained by merely storing the above information at the root of the decision tree and at the node $\tilde{a}$ corresponding to the last application of procedure Σ; then, at Step 3.1, we will check conditions (1) and (2) for node $\tilde{a}$ if the current node descends from $\tilde{a}$, and for the root otherwise.

A further improvement can be obtained by considering that the solution to sub-

problem (P'), in procedures Π (if applied to the mth knapsack) and Σ, must be obtained only if it is greater than $(V^* - \bar{V})$ in Π, or greater than $(V^* - V)$ in Σ. If (P') is solved through a branch and bound algorithm, a valid initial optimal solution value can then be given by $(V^* - \bar{V})$ or $(V^* - V)$; so, it is sure that all the nodes of the decision-tree for which the local upper bound is not greater than that quantity will be fathomed.

Let us consider an example in order to illustrate the algorithm presented. Let $n = 10$, $m = 2$ and

$$(p_j) = (78, 35, 89, 36, 94, 75, 74, 79, 80, 16);$$
$$(w_j) = (18, 9, 23, 20, 59, 61, 70, 75, 76, 30);$$
$$(c_i) = (103, 156).$$

Fig. 1 gives the branch-decision tree determined by the algorithm: $\mathrm{UB}^{\mathrm{root}}$, (y_j^{root}) and r^{root} are the information stored at the root of the decision tree; $\mathrm{UB}^{\mathrm{last}}$, (y_j^{last}) and r^{last} the information stored at the last node of the decision tree for which procedure Σ was applied.

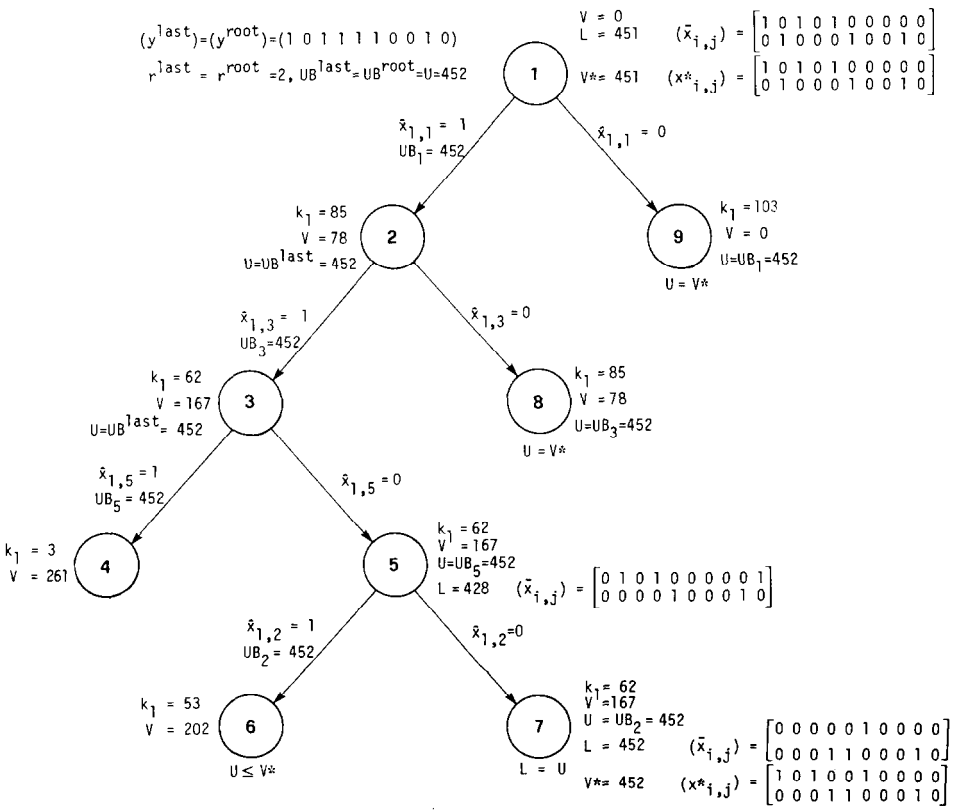

Fig. 1.

6. Computational results

In this section we give a computational comparison between the algorithm of Sections 4, 5 (here referred to as BB) and the methods outlined in Section 2.

Hung and Fisk [2] proposed two specializations of their algorithm: the former (HFS) based on the surrogate relaxation and the latter (HFL) based on the lagrangean relaxation of (P). Martello and Toth [6] presented a method from which two algorithms (MTL and MTLS) can be derived by utilizing either the lagrangean relaxation alone or both the lagrangean and the surrogate relaxation; in [6] it was also shown that a combined application of the two relaxation techniques in the Hung-Fisk method gives an algorithm (HFLS) generally faster than HFL. All the above algorithms can be applied either directly or after the reduction procedure (IKR) of Ingargiola and Korsh [3].

The authors have coded the algorithms in FORTRAN IV and run them on a CDC-6600 at times when the demand for computer use was comparable.

The single knapsack problems (P'), generated by all the methods, have been solved through the Martello-Toth branch and bound algorithm [4] as coded by the authors in [5].

Test problems have been obtained by independently generating the values p_j and w_j from a uniform distribution in the interval (10, 100). Two classes of data sets have subsequently been obtained by independently generating the values c_i from a uniform distribution according to the conditions:

$$\text{Class 1: } \left\lceil 0.4 \sum_{j \in N} w_j/m \right\rceil \leq c_i \leq \left\lceil 0.6 \sum_{j \in N} w_j/m \right\rceil \quad \text{for } i = 1, \ldots, m-1;$$

$$\text{Class 2: } 0 \leq c_1 \leq \left\lceil 0.5 \sum_{j \in N} w_j \right\rceil;$$

$$0 \leq c_i \leq \left\lceil 0.5 \sum_{j \in N} w_j \right\rceil - \sum_{\substack{u \in M \\ u < i}} c_u \qquad \text{for } i = 2, \ldots, m-1.$$

For both classes the last knapsack capacity c_m has been chosen such that $\sum_{i \in M} c_i = 0.5 \sum_{j \in N} w_j$; if $c_i < \min_{j \in N} \{w_j\}$ for some i or $\max_{i \in M} \{c_i\} < \max_{j \in N} \{w_j\}$, a new set of knapsack capacities has been generated.

For each class and for each value of $m \in \{2, 3, 4\}$ a data set has been obtained by generating 120 problems (30 for each value of $n \in \{25, 50, 100, 200\}$). All the algorithms had a time limit of 250 seconds assigned to solve each data set: the entries of the tables give the average running times and, in brackets, the maximum times obtained by the algorithms; when the time limit was not enough to solve the 120 problems of the data set, only the number of solved problems is indicated.

Only two columns (HF and MT) are given, respectively, for the six algorithms of Hung and Fisk [2] and for the four algorithms of Martello and Toth [6]: the entries give the lowest average and maximum running times obtained (an asterisk indicates

that the corresponding time was obtained by previous application of the reduction procedure; the reduction time is included in such entries). Columns BB* and BB give the times obtained by the algorithm of Sections 4 and 5, respectively with and without previous application of the reduction procedure (in the first case the time needed by IKR is included in the entries). Column IKR gives the average times needed by procedure IKR. The times in the tables are not comprehensive of the time needed to sort out the items and the knapsacks.

Tables 1 and 2 give the results obtained for data sets of Classes 1 and 2, respectively.

Table 1
Data sets from Class 1; 30 trials for each entry; average (maximum) times in CDC-6600 seconds

m	*n*	HF	MT	BB	BB*	IKR
2	25	0.136 (1.590)	0.067 (0.249)	0.092 (1.389)	0.106 (1.324)	0.030
	50	0.327 (2.034)	0.231 (1.035)	0.160 (2.424)	0.237 (1.940)	0.122
	100	0.820 (2.634)	0.390 (1.688)	0.067 (0.454)	0.636 (0.971)	0.600
	200	3.035 (7.995)	2.466 (6.044)	0.141 (2.090)	3.344 (4.888)	3.208
3	25	1.644 (14.682)*	0.589 (5.062)*	0.608 (6.218)	0.543 (5.074)	0.032
	50	9 problems*	1.171 (9.440)*	0.276 (1.376)	0.377 (1.252)	0.127
	100	–	2.986 (11.951)*	0.200 (0.632)	0.802 (1.251)	0.614
	200	–	6 problems*	0.273 (1.292)	3.563 (4.227)	3.291
4	25	6.814 (53.370)*	3.167 (25.882)*	1.835 (12.941)	1.649 (12.393)	0.032
	50	1 problem*	12 problems*	6.215 (99.785)	4.751 (65.548)	0.132
	100	–	–	12 problems	1.490 (6.804)	0.620
	200	–	–	–	3 problems	3.267

* Times obtained by previous application of reduction procedure IKR.

Table 2
Data sets from Class 2; 30 trials for each entry; average (maximum) times in CDC-6600 seconds

m	*n*	HF	MT	BB	BB*	IKR
2	25	0.107 (0.571)	0.068 (0.402)	0.035 (0.270)	0.057 (0.280)	0.030
	50	0.348 (2.228)	0.134 (0.432)	0.051 (0.502)	0.160 (0.614)	0.123
	100	0.787 (2.353)	0.664 (1.616)	0.073 (0.464)	0.638 (0.970)	0.597
	200	3.457 (13.675)	2.583 (6.122)	0.096 (0.406)	3.254 (3.465)	3.207
3	25	2.198 (28.138)*	4.640 (66.365)*	0.226 (1.880)	0.220 (1.471)	0.031
	50	7 problems*	13 problems*	0.132 (1.931)	0.232 (1.564)	0.126
	100	–	–	0.158 (0.925)	0.763 (1.455)	0.621
	200	–	–	0.111 (0.571)	3.428 (4.241)	3.347
4	25	4 problems*	4 problems*	0.503 (10.804)	0.455 (9.144)	0.031
	50	–	–	0.471 (5.400)	0.539 (5.137)	0.130
	100	–	–	0.329 (1.448)	0.939 (1.841)	0.611
	200	–	–	0.241 (1.224)	3.504 (4.755)	3.278

Times obtained by previous application of reduction procedure IKR.

The Bound and Bound algorithm of Sections 4 and 5 was generally the fastest method. Its performance was clearly better for Class 2 than for Class 1, while both HF and MT presented comparatively small variations between the two classes; this is probably because BB is at an advantage when one of the knapsacks is much greater than the others.

The average times of IKR grow steeply when n grows; consequently BB* was faster than BB in some cases with $n=25$ and in the "hardest" data set (Class 1, $m=4$).

It should be noted that the ratio (maximum time)/(average time) is much greater for BB than for both HF and MT; this probably explains the irregular behaviour of BB when n varies for the same data set.

References

[1] N. Christofides, G. Carpaneto, A. Mingozzi and P. Toth, The loading of liquids into tanks, Imperial College Research Report, London (1976).

[2] M.S. Hung and J.C. Fisk, An algorithm for 0-1 multiple knapsack problems, Naval Res. Logist. Quart. 24 (1978) 571–579.

[3] G. Ingargiola and J.F. Korsh, An algorithm for the solution of 0-1 loading problems, Operations Research 23 (1975) 1110–1119.

[4] S. Martello and P. Toth, An upper bound for the zero-one knapsack problem and a branch and bound algorithm, European J. Operat. Res. 1 (1977) 169–175.

[5] S. Martello and P. Toth, Algorithm for the solution of the 0-1 single knapsack problem, Computing 21 (1978) 81–86.

[6] S. Martello and P. Toth, Solution of the 0-1 multiple knapsack problem, European J. Operat. Res. 4 (1980) 276–283.

[7] A. Neebe and D. Dannenbring, Algorithms for a specialized segregated storage problem, Technical Report No. 77-5, University of North-Carolina (1977).