THE BOUNDED KNAPSACK PROBLEM WITH SETUPS

by

H. SURAL*

L.N. VAN WASSENHOVE**

and

C.N. POTTS†

97/71/TM

* Research Assistant at INSEAD, Boulevard de Constance, Fontainebleau 77305 Cedex, France.

** Professor of Operations Management and Operations Research at INSEAD, Boulevard de Constance, Fontainebleau 77305 Cedex, France.

† Professor at the University of Southampton, UK.

A working paper in the INSEAD Working Paper Series is intended as a means whereby a faculty researcher's thoughts and findings may be communicated to interested readers. The paper should be considered preliminary in nature and may require revision.

Printed at INSEAD, Fontainebleau, France.

The Bounded Knapsack Problem with Setups

Haldun Sural*, Luk N. Van Wassenhove* and Chris N. Potts**

* *Technology Management Area, INSEAD, Fontainebleau, France*
** *Faculty of Mathematical Studies, University of Southampton, U.K.*

Abstract

In the bounded knapsack problem with setups there are a limited number of copies of each item and the inclusion of an item in the knapsack requires a fixed setup capacity. Analysis of special cases of the problem allows us to derive the borderline between hard and easy problems. We develop a branch and bound algorithm for the general problem and present some computational results.

Keywords: knapsack, complexity, branch and bound

1 Introduction

The bounded knapsack problem with setups may be stated as follows. Given are a knapsack of capacity b and n items, where each item i (i=1,...,n) has a limited number of copies m_i. Each copy of item i consumes one unit of capacity and has a contribution c_i. In addition, including item i in the knapsack consumes a fixed setup of a_i units of capacity. The problem is to pack items into the knapsack so that the total contribution is maximized subject to the capacity availability constraint. Mathematically, we can formulate the problem as

$$\begin{aligned}
(P)\qquad & \text{Maximize } z=\sum_{i=1}^{n} c_i x_i \\
& \text{subject to } \sum_{i=1}^{n}\left(x_i+a_i y_i\right)\le b, \\
& \qquad 0\le x_i \le m_i y_i, \qquad\qquad i=1,\ldots,n, \\
& \qquad x_i,\ \text{integer},\ y_i=0 \text{ or } 1, \qquad i=1,\ldots,n.
\end{aligned}$$

where x_i and y_i are variables, and a_i, c_i, m_i and b are nonnegative integers for all i=1,2,...,n. We may assume, without loss of generality, that $\left(m_i+a_i\right)\le b$ for all i, and $\sum_{i=1}^{n}\left(m_i+a_i\right)>b$.

An application of problem P arises in the context of investment planning, especially in capital budgeting. Consider, for instance, a decision maker who wants to allocate his capital among several types of investment alternatives. Each alternative yields a return per unit of money invested. The total amount that can be allocated to each type of investment is limited and investing in it incurs an initial fixed cost, or sunk cost. Problem P maximizes total return.

Another application of P can be found in scheduling. The single machine scheduling problem with setups to minimize the weighted number of late jobs, introduced by Kovalyov, Potts and Van Wassenhove [6], is equivalent to P for the case of unit processing times and a common due date. Hence, problem P also occurs when a large number of jobs belonging to a few classes requiring a setup time have to be prepared for a shipment leaving at a given time.

Problem P also arises as a subproblem in many production planning problems where resources need to be set up before a production run. For instance, Diaby, Bahl, Karwan and Zionts [3], Kirca [5], and Sural [8] describe different versions of P in their algorithms for solving the multi-item lot sizing problem with setup times. Goesmans [4] investigates the polyhedral structure of the constraint set in P with the purpose of generating strong valid inequalities. A problem similar to P, which occurs as a subproblem of a parallel machine scheduling problem with setups, is studied by Chajakis and Guignard [2]. They are interested in a knapsack problem for which an item is packed if the knapsack is set up for its family. One can easily show that P is a special case of their problem.

Finally it is also interesting to state that P can be transformed to the multiple-choice knapsack problem by introducing new variables. The multiple-choice knapsack problem is defined as follows.

$$\begin{aligned}
&\textit{(MCP)}\ \text{Maximize} && z=\sum_{j=1}^{k} p_j x_j && \\
&\text{subject to} && \sum_{j=1}^{k} w_j x_j \le b, && \\
& && \sum_{j\varepsilon N_i} x_j = 1, && i=1,\ldots,n, \\
& && x_j = 0 \text{ or } 1, && \text{for } j\varepsilon N_i \text{ and } i=1,\ldots,n.
\end{aligned}$$

where x_j is a variable and p_j, w_j and b are nonnegative integers for all $j=1,\ldots,k$. Furthermore, all the $N_i^{'}s$ are disjoint and non-empty.

Introducing m_i new binary variables for each i in P, whose (w_j, p_j) values $(j \varepsilon N_i)$ are (a_i+1, c_i), $(a_i+2, 2c_i)$, ..., $(a_i+m_i, m_i c_i)$, respectively, and one variable whose $(w_j, p_j)=(0, 0)$, we transform P to an equivalent MCP having n sets and $k=n+\sum_{i=1}^{n} m_i$ items.

It follows that problem P occurs both in its own right and as a subproblem in several more complex production planning and scheduling problems. In the latter case, problem P needs to be solved repeatedly as a subroutine. It therefore makes sense to study problem P in more detail and to design adequate (*i.e.*, reasonably fast) algorithms.

In this paper, we first develop complexity results for P and its special cases, thereby establishing the boundary between NP-hard and polynomially solvable problems. A simple algorithm for the well-solvable cases is given in Section 2. A branch and bound algorithm and heuristic procedure for problem P are discussed in Section 3. In Section 4 we outline and discuss our computational experiments. Section 5 concludes the article.

2 Complexity analysis of problem *P*

Problem P is (binary) NP-hard as can easily be seen by putting $m_i=1$ for $i=1,...,n$. All setup variables y_i can now be removed from P and the knapsack constraint can be written as $\sum_{i=1}^{n}\left(1+a_i\right)x_i \leq b$ where $x_i = 0$ or 1 for $i=1,...,n$. Hence the problem reduces to the well-known NP-hard regular 0-1 Knapsack problem.

It can easily be verified that the following dynamic programming recursion provides a pseudo-polynomial algorithm for problem P. For $i=1,...,n$, define

$$z_i(d)=\max\left\{\sum_{j=1}^{i} c_j x_j : \sum_{j=1}^{i}\left(x_j + a_j y_j\right) \leq d, x_j \leq m_j y_j, x_j \geq 0, y_j = 0, j=1,...,i.\right\}$$

for $d=0,1,\dots,b$. Then, $z_n(b)=z(b)$ denotes the optimal solution to problem P. Let $z_0(d)=0$ for $d=0,1,\dots,b$. The following recursion is computed for each i, where $i=1,\dots,n-1$:

$$z_i(d)=\begin{cases} z_{i-1}(d) & \text{for } 0\leq d\leq a_i \\ \max_{0<x_i\leq m_i}\{z_{i-1}(d) \quad c_i x_i + z_{i-1}(d-a_i-x_i)\} & \text{for } a_i<d\leq b \end{cases}$$

For $i=n$ we have $z(b)=\max_{0<x_n\leq m_n}\{z_{n-1}(b), c_n x_n + z_{n-1}(b-a_n-x_n)\}$.

For each i, computation of $z_i(d)$ requires $O(b^2)$ time. Hence the time complexity for solving problem P is $O(nb^2)$. The space complexity is $O(nb)$ since storage of the state vector corresponding to $z_i(d)$ for each i would be sufficient.

Special cases

Since problem P is NP-hard, it is worthwhile to consider its special cases in order to clearly define the boundary between easy and hard problems. The first case where $c_i=1$ for all i is equivalent to the single machine scheduling problem with setups to minimize the number of late items with unit processing times and a common due date. Kovalyov, Potts and Van Wassenhove [6] show that the latter problem is NP-hard. For convenience, a direct proof is given below.

Theorem 1 *The bounded knapsack problem with setups where $c_i=1$ for all i is NP-hard.*

Proof. The decision version of our problem is NP-complete by transformation from the NP-complete Partition problem: Given positive integers $\alpha_1,\dots,\alpha_n$, is there a subset S of the index set $\{1, \dots, n\}$, such that $\sum_{i\in S}\alpha_i=A/2$ where $A=\sum_{i=1}^{n}\alpha_i$?

Given any instance of Partition, we can construct the following instance of our problem. There are n items with $c_i=1$, $m_i=\alpha_i$ $a_i=\alpha_i$ for all i, and $b=A$. We show that there exists a set S

for which $\Sigma_{i\varepsilon S}\alpha_i=A/2$ if and only if there is a bounded knapsack solution whose objective function value is at least A/2.

If there is a subset S for which $\Sigma_{i\varepsilon S}\alpha_i=A/2$, then leaving exactly $A/2$ units for setups, we can insert all copies of the items of S into the knapsack without exceeding b. Conversely, if there is a bounded knapsack solution with objective function value at least equal to $A/2$, there must be at least $A/2$ copies in the knapsack. Since $m_i=a_i=\alpha_i$ for all i, the capacity requirement for setups is at least equal to the capacity taken by the copies of items. Thus, in a solution with $A/2$ copies in the knapsack, the total setup capacity is $A/2$ and hence these copies correspond to a set S for which $\Sigma_{i\varepsilon S}\alpha_i=A/2$.

Theorem 2 *The bounded knapsack problem with setups where $a_i=1$ for all i is NP-hard.*

Proof. We show that the decision version of the bounded knapsack problem with unit setups is *NP*-complete by transformation from the Equal Cardinality Partition problem: Given positive integers $\alpha_1,\ldots,\alpha_{2k}$, is there a subset S of the index set $\{1,\ldots,2k\}$, such that $|S|=k$ and $\Sigma_{i\varepsilon S}\alpha_i=A/2$, where $A=\sum_{i=1}^{2k}\alpha_i$?

Given any instance of Equal Cardinality Partition, we can construct the following instance of the bounded knapsack problem with unit setups, where we, for simplicity, assume that c_i can be real. There are $n=2k$ items, $c_i=(A+\alpha_i)/(A+\alpha_i-1)$ and $m_i=A+\alpha_i-1$ for all i, and $b=(n+1)A/2$. We show that there exists a set S for which $|S|=k$ and $\Sigma_{i\varepsilon S}\alpha_i=A/2$ if and only if there is a solution to the bounded knapsack problem with unit setups such that $z\geq(n+1)A/2$.

First, suppose there is a subset S for which $|S|=k$ and $\Sigma_{i\varepsilon S}\alpha_i=A/2$. In the knapsack problem, set $x_i=A+\alpha_i-1$ and $y_i=1$ for $i\varepsilon S$, and $x_i=y_i=0$ for $i\varepsilon\{1,\ldots,2k\}\backslash S$. Since $\Sigma_{i\varepsilon S}(x_i+y_i)=kA+A/2=b$, this solution is feasible. Moreover, $z=\Sigma_{i\varepsilon S}(A+\alpha_i)=(n+1)A/2$.

Conversely, suppose that there is a solution to the bounded knapsack problem with unit setups such that $z\geq(n+1)A/2$. Let $S=\{i: y_i=1\}$. We show first that $|S|\geq k$. If $|S|<k$, then $z=\Sigma_{i\varepsilon S}(1+1/(A+\alpha_i-1))x_i \leq \Sigma_{i\varepsilon S} m_i+|S| \leq (n/2-1)A+A$, which contradicts the assumption that $z\geq(n+1)A/2$.

We now show that $x_i= m_i$ for $i\varepsilon S$. Suppose that this is not the case. Note that this analysis applies when $|S|>k$: since $\Sigma_{i\varepsilon S} m_i>(n/2+1)A>b$, it follows that $x_i<m_i$ for some $i\varepsilon S$. Since the knapsack constraint implies that $\Sigma_{i\varepsilon S} x_i\leq(n+1)A/2-|S|$, we obtain $z=\Sigma_{i\varepsilon S}(1+1/(A+\alpha_i-1))x_i \leq(n+1)A/2-|S|+\Sigma_{i\varepsilon S} x_i/(A+\alpha_i-1)$. Since $x_i<A+\alpha_i-1$ for some $i\varepsilon S$, we deduce that $z<(n+1)A/2$, which again contradicts the assumption that $z\geq(n+1)A/2$. Therefore, $x_i=A+\alpha_i-1$ for $i\varepsilon S$, and $|S|=k$.

The knapsack constraint can be expressed as $\Sigma_{i\varepsilon S}\alpha_i\leq A/2$. Moreover, the inequality $z\geq(n+1)A/2$ can be written as $\Sigma_{i\varepsilon S}\alpha_i\geq A/2$. Thus, $\Sigma_{i\varepsilon S}\alpha_i=A/2$, and S defines a solution to Equal Cardinality Partition.

Efficient algorithms are available if further restrictions are placed on the problem parameters. For instance, if two out of the three parameters are equal for all items, P can be solved by packing the items in an order implied by the following *dominance* concept.

We say that if item i dominates item k, then in an optimal solution *(a)* $y_i \geq y_k$, and *(b)* if $y_i = y_k = 1$, then $x_i = m_i$ and $0 < x_k \leq m_k$.

Lemma 1 *Item i dominates item k if* $c_i m_i \geq c_k \min\{m_i + a_i,\ m_k\}$, $c_i \geq c_k$, $a_i \leq a_k$ *and at least one equality is strict.*

Proof. To show that the first part of the dominance condition is valid, we suppose to the contrary that there exists an optimal solution to P, where $y_k = 1$ and $y_i = 0$. The condition that at least one inequality is strict yields three cases; either *(i)* $c_i m_i > c_k \min\{m_i + a_i,\ m_k\}$, or *(ii)* $c_i > c_k$, or *(iii)* $a_i < a_k$. In each case, one can easily show that setting up for i and increasing x_i, and decreasing x_k until $x_i = m_i$ or $x_k = 0$ (and $y_k = 0$ in turn) would either improve the objective function, or yield an alternate optimum. The proof for the validity of the second part of the dominance condition similarly follows.

By Lemma 1, setting two out of the three sets of parameters c_i, m_i and a_i equal, we are able to find a dominance relation between each pair of items. Thus, all items can be ranked. In case of a tie, of course, items are ranked arbitrarily. It follows that packing items into the knapsack in that rank order solves the special cases where two sets of parameters are equal. If the items are indesced in rank order, an optimal solution to the problem is of the form $\sum_{i=1}^{r-1}\left(m_i + a_i\right) \leq b$ and $\sum_{i=1}^{r}\left(m_i + a_i\right) > b$, and can be characterized by the critical rth item and those items which dominate the rest of n-r items. The crucial step is to determine the first r items rather than their mutual order. It is possible to pick the r items to be included in the knapsack without sorting using an idea by Balas and Zemel [1]. Steps of this algorithm can be found in Martello and Toth [7] and can easily be adapted to our problems by adding an inevitable step that inserts copies of item r into the knapsack until it is completely filled.

Theorem 3 *The Balas and Zemel algorithm which runs in O(n) time solves problem P where for each i* (i=1,...,n) *(a)* $c_i = c$ *and* $m_i = m$, *or (b)* $c_i = c$ *and* $a_i = a$, *or (c)* $m_i = m$ *and* $a_i = a$.

Proof. The proof follows from setting $\lambda_i=1/a_i$ for *(a)*, $\lambda_i=m_i$ for *(b)*, and $\lambda_i=c_i$ for *(c)*, and defining the weights a_i used in the sum sets as $a_i=a_i+m_i$ for all i in Theorem 1 of Balas and Zemel [1].

From the above discussion, it follows that problem P is hard even if one set of parameters (*i.e.*, a_i's, m_i's, or c_i's) are equal. It is easy when two sets of parameters are equal.

3 Branch and bound algorithm for solving P

In the knapsack problem literature, branch and bound algorithms are widely used to find an optimal solution, especially for large n. We describe such an algorithm to solve problem P. It is an extension of the Horowitz-Sahni algorithm for the 0-1 knapsack problem (see, Martello and Toth [7]).

Upper bound

A simple upper bound on the problem is found by solving the *LP* (linear programming) relaxation of P. The *LP* relaxation of the problem can be derived from Dantzig's solution to the continuous knapsack problem. Let $q_i=c_i m_i/(m_i+a_i)$ denote the contribution rate of item i and reorder the items such that $q_1 \geq q_2 \geq, \dots, \geq q_n$. An optimal solution is $y_i=1$ for $i=1,\dots,r-1$, $y_r=\left[b-\sum_{i=1}^{r-1}(m_i+a_i)\right]/(m_r+a_r)$, and $y_i=0$ for $i=r+1,\dots,n$ where $r=\min\left\{k: \sum_{i=1}^{k}(m_i+a_i)>b\right\}$ is called the critical item index. Hence, $x_i=m_i y_i$ for all i and $u=\sum_{i=1}^{r} c_i x_i$ constitutes an upper bound for P. Naturally, an *LP* solution can be determinedg from the critical index in $O(n)$ time.

The *LP* based upper bound can be improved with a little effort. For convenience, suppose that items are still in the same order $q_1 \geq q_2 \geq, \dots, \geq q_n$. If, for example, y_r, the binary variable corresponding to the critical index r, is not integral in the *LP* optimal solution, it can be forced to be integral (*i.e.*, either to be zero or one) by imposing an integrality constraint. Obviously, the resultant value would be at least as good as the initial value.

Let $\bar{u}=\sum_{i=1}^{r-1}c_i m_i$ and $\bar{b}=b\sum_{i=1}^{r-1}(m_i+a_i)$. When we impose y_r=0, the variable(s) $r+1, r+2, \ldots, n$ would take turns at filling the knapsack until the residual capacity is used up. Then the new critical index s would be $s=\min\left\{k:\sum_{i=r+1}^{k}(m_i+a_i)>\bar{b}\right\}$ and the new bound would be equal to $u_0=\bar{u}+\sum_{i=r+1}^{s-1}c_i m_i+c_s m_s y_s$. Note that this corresponds to solving the relaxed problem P with an additional constraint y_r=0 to optimality. In case of imposing y_r=1, however, we use a heuristic to avoid increasing the computational burden. Let $\bar{m}_r$ be the necessary number of copies to be added for item r given that y_r=1. The new upper bound subject to the additional constraint is

$$u_1=\bar{u}+\left(c_r-q_{r-1}\right)\bar{m}_r+q_{r-1}\left(\bar{b}-a_r\right), \text{ where } \bar{m}_r=\begin{cases}1 & \text{if } q_{r-1}\geq c_r \text{ and } \bar{b}\leq a_r\\ \bar{b}-a_r & \text{if } q_{r-1}\geq c_r \text{ and } \bar{b}>a_r\\ m_r & \text{otherwise}\end{cases}$$

Here the idea is simply as follows. Imposing y_r=1 means, in general, that at least one copy of item r should be inserted in the knapsack, or more if it improves, but at most m_r. For instance, when $q_{r-1}\geq c_r$ and $\bar{b}>a_r$, the optimal solution is to fill the residual capacity by item r after its setup capacity is reduced. On the other hand, when $q_{r-1}\geq c_r$ and $\bar{b}\leq a_r$, the maximum necessary copies of item r to be inserted is one and the minimum necessary value to be excluded from the solution is $a_r-\bar{b}$ for its setup. This brings one unit contribution of c_r but reduces $\bar{u}$ at least by $q_{r-1}\left(1+a_r-\bar{b}\right)$.In case of $q_{r-1}<c_r$, inserting all copies of r improves the

solution. In order to add $c_r m_r$ we, however, have to remove at least $q_{r-1}\left(m_r - \bar{b} + a_r\right)$ from the solution. Ultimately, the new upper bound is equal to max $\{u_0, u_1\}$.

The following theorem characterizes the optimal solution to the original problem P and provides a basis for both our heuristic procedure and branch and bound algorithm.

Theorem 4 *There exists an optimal solution to problem P in which every variable x_i, except at most one, is either at its upper bound or zero.*

Proof. Suppose that, in an optimal solution to P, there exist two variables, x_i and x_j, such that $0<x_i<m_i$ and $0<x_j<m_j$, and $c_i \geq c_j$. In such a solution, increasing x_i and decreasing x_j until $x_i=m_i$, or $x_j=0$ (and $y_j=0$ in turn) would improve the objective function if $c_i>c_j$. This contradicts optimality. Clearly, $c_i=c_j$ indicates an alternate optimum.

We deduce the following corollary.

Corollary 1 *The variable, whose value is neither at its upper bound nor zero in an optimal solution, has the smallest contribution of all non-zero variables.*

Lower bound

A continuous LP solution to P can be converted to an integer one by considering the critical item r. For instance, if $b-\sum_{i=1}^{r-1}\left(m_i+a_i\right)<a_r$, then item r can simply be taken out, otherwise some copies can be taken out until its setup becomes feasible. However, in some cases, the resulting feasible solution might be arbitrarily bad. Consider, for instance, a two-item problem where $b=k+1$, $c_1=2$, $c_2=k$, $m_1=1$, $m_2=1$, $a_1=1$, and $a_2=k$ where $k>2$. A feasible solution derived from LP yields $z_{LP}=2$. Item 2 corresponds to item r. On the other hand, the optimal solution is $z_{OPT}=k$. Unfortunately, as k increases the ratio z_{LP}/z_{OPT} approaches zero.

In some circumstances, however, even a small adjustment to the *LP* solution might yield an improvement. We next present such a heuristic procedure.

Let *N* be the index set of all items, $S \subseteq N$, and $P(S)$ an instance of problem *P* defined for a subset *S*. $y^*(i)$ and $x^*(i)$ denote the optimal solution values of $P(S)$ for $i \varepsilon S$. The output *sol* denotes the resulting (feasible) solution value for *P* while *fractional*, *k* and *l* are auxiliary variables. Additionally, *R*, *F* and *A* are auxiliary index sets.

```
procedure heuristic procedure:
begin comment initialization,
    $S:=N$; $R:=\emptyset$; fractional:="yes";
    comment adjustment to the LP solution,
    repeat
            find the LP solution of $P(S)$;
            $F:=\{i:x^*(i)=m(i),\ i\varepsilon S\}$; $b:=b-\Sigma_{i\varepsilon F}m(i)+a(i)$;
            $A:=\{i:m(i)+a(i)>b,\ i\varepsilon S\backslash F\}$; $R:=R\cup F$; $S:=S\backslash\{F\cup A\}$;
    until $S:=\emptyset$ or $b:=0$;
            begin
            $k:=\mathrm{argmin}_{i\varepsilon R}\{c(i)\}$; $b:=b+m(k)+a(k)$;
            $sol:=\Sigma_{i\varepsilon R\backslash\{k\}}x^*(i)c(i)$; $R:=N\backslash R$;
      solve $P(R\cup\{k\})$ with additional constraint $\Sigma_{i\varepsilon R}\,y(i)\leq 1$;
      $sol:=sol+\Sigma_{i\varepsilon R}x^*(i)c(i)+x^*(k)c(k)$;
      for each $i\varepsilon R$ do
            begin
                if $x^*(i):=m(i)$ then fractional:="no" and $l:=i$
            end;
      if $fractional:="no"$ and $y^*(k):=0$ then
            begin
                  $R:=R\setminus\{l\}$; $b:=b-m(l)-a(l)$;
                  solve $P(R)$;
                  $sol:=sol+\Sigma_{i\varepsilon R}x^*(i)c(i)$
            end;
   end
end.
```

In the above procedure we solve both integer *P(S)*'s by total enumeration, which is done in *O(n)* time since at most two items can enter into the solution. The overall time complexity of the procedure is also *O(n)*.

Algorithm

The branch and bound algorithm we propose is based on a depth-first strategy and the setup variables are chosen for branching. Initially, by using the lower and upper bound procedures we compute both of these bounds on the solution at the very first node of the branch and bound tree. If the lower bound is greater than or equal to the upper bound we halt the algorithm since it gives the optimal solution. Otherwise, we save it as a best solution at hand and go through the algorithm.

The algorithm proceeds with two types of moves, *i.e.*, a forward move and backtrack. Based on Theorem 4 and its corollary, we begin with sorting all items in the order of $c_1 \geq c_2 \geq, \ldots, \geq c_n$. At a forward move, we insert an item into the current solution either at its upper bound or zero, except the last one. The last item to be inserted into the solution just before the capacity expires, is inserted at an in-between value determined by the residual capacity. If an item is inserted at zero into the current solution, an upper bound on the current solution is computed by using the upper bound procedure. But the bound is computed just for the last item if more items are successively inserted at zero level. If the bound is less than or equal to the value of the best solution on hand, we fathom this node and backtrack. Clearly, a backtrack move is performed to take out the last item, inserted at a non-zero level, from the current solution. Otherwise, *i.e.*, if the upper bound is not worse than the best solution, a forward move occurs; we thus consider the next item that can be inserted into the current solution. Consideration of the *n*th item completes the current solution and updating of the best solution at hand is checked. We halt the algorithm whenever a further backtracking move is not possible. The steps of the algorithm are given in Appendix.

4 Computational experiments

The computational behaviour of the branch and bound algorithm is analyzed on sets of randomly generated test problems. We consider two types of problem sets, namely,

- uncorrelated *(U)*: c_i, m_i, and a_i are randomly distributed on U[1,101];
- correlated: values of two of the parameters c_i, m_i, and a_i, all of which are randomly distributed on U[1,101], are paired in a special way while those of the remaining parameter are kept as they are. The idea is to increase the expected difficulty of a problem instance. Accordingly, three types of construction are possible;

  Generate *n*-item data where c_i, m_i, and a_i randomly distributed on U[1,101]. To construct a problem instance in *(C)*, sort and renumber m_i and a_i's according to non-increasing order; in *(M)*, sort and renumber c_i and a_i's according to non-increasing order; and in *(A)*, sort and renumber c_i's in non-increasing order and m_i's in non-decreasing order. Note that, in each case we keep the data in the original order just for a single parameter.

The problem sizes are chosen as n=25, 50, 100. We generate 25 test problems for each size and construction, and conduct our experiments for two different values of capacity, $b=0.5\sum_{i=1}^{n}\left(m_i+a_i\right)$ and 400. Note that we have got enough capacity to pack about half of items in the former case while very few items might enter into the knapsack in the latter case.

The computational experiments are performed on a PC-Pentium with 120 MHz for 600 problem instances in total. The sorting needed by the algorithm is done with a subroutine given in Martello and Toth [7]. We compare the Fortran implementation of the algorithm with Lindo. Results are reported in Table 1 for each problem class, *U*, *C*, *M*, and *A*, and each value of b and n. The average execution times, including sorting times, are expressed in seconds in the table (*cpu(a)* and *cpu(l)*, respectively) but neither the input nor the output times are included in these times. In addition, the average number of nodes and backtracks are given for our algorithm. Furthermore, we provide two types of results for the heuristic procedure; the average percentage error and the number of times that the procedure has found the optimal solution, which are denoted by "*% error*"and "#", respectively. The expression of the average percentage error is 100*(optimal-lower bound)/optimal.

As shown by results in Table 1, both methods solve the problems relatively quickly, but our algorithm clearly outperforms Lindo. For all n, the algorithm yields considerably smaller execution times for b=400 than for $b=0.5\sum_{i=1}^{n}\left(m_i+a_i\right)$, as expected, because a small knapsack is packed by very few items. Furthermore, as n grows larger the average execution times grow fairly rapidly.

Table 1 Computational results

case	*capacity*	*n*	*node(backtrack)*	*% error(#)*	*cpu (a)*	*cpu (l)*
C	*400*	*25*	27(12)	1.736(10)	0.000	0.322
		50	67(30)	0.853(10)	0.002	1.450
		100	133(59)	0.977(7)	0.002	3.608
	$0.5\sum_{i=1}^{n}(m_i+a_i)$	*25*	79(38)	0.653(8)	0.000	0.282
		50	147(71)	0.228(6)	0.004	0.979
		100	534(254)	0.090(5)	0.008	3.445
U	*400*	*25*	45(22)	1.378(13)	0.002	0.196
		50	99(46)	1.497(8)	0.005	0.340
		100	160(74)	0.971(10)	0.004	1.044
	$0.5\sum_{i=1}^{n}(m_i+a_i)$	*25*	100(49)	0.313(11)	0.002	0.211
		50	571(279)	0.336(6)	0.006	0.650
		100	2584(1268)	0.093(3)	0.042	3.730
M	*400*	*25*	82(38)	0.717(14)	0.000	0.627
		50	294(135)	0.705(13)	0.002	2.854
		100	375(170)	0.119(14)	0.004	10.774
	$0.5\sum_{i=1}^{n}(m_i+a_i)$	*25*	156(76)	0.391(7)	0.002	0.427
		50	945(459)	0.216(6)	0.004	1.925
		100	10023(4885)	0.072(2)	0.121	20.976
A	*400*	*25*	140(66)	1.306(13)	0.000	0.291
		50	430(199)	1.332(11)	0.004	0.952
		100	1164(537)	0.855(6)	0.009	2.395
	$0.5\sum_{i=1}^{n}(m_i+a_i)$	*25*	259(129)	0.499(10)	0.004	0.309
		50	1420(703)	0.266(6)	0.022	1.081
		100	5242(2598)	0.110(9)	0.123	4.720*

* Average solution time for Lindo is given for 24 problems; one problem could not be solved within the pivot limit 10 million we set.

The heuristic procedure does reasonably well. Both the average percentage error and the number of times that the optimal solution is found decrease as n grows. The average percentage error over 600 problems is 0.654.

Apart from the above experiments, we have tested the performance of the algorithm on certain large-size instances of P. We only considered the A and M type test problems, which appear to be the "harder" ones. We set n=400 and $b = 0.5\sum_{i=1}^{n}(m_i + a_i)$). The average solution times over 25 test problems are 0.28 and 1.64 minutes while the average percentage errors of the heuristic procedure are 0.016 and 0.045 for A and M type test problems, respectively.

5 Concluding remarks

We have introduced the bounded knapsack problem with setups. To our knowledge, this is the first attempt to examine the problem although a variety of knapsack problems have been well studied in the literature.

The boundary between easy and hard problems is established by examining special cases of the problem. It is shown that P remains NP-hard for the special cases where *(i)* all the upper bounds, or *(ii)* all the contributions, or *(iii)* all the setups are equal to 1. We provide an $O(n)$ algorithm for special cases of P where two sets of parameters are equal. We develop a branch and bound algorithm to solve P to optimality and propose a lower bound procedure to find a heuristic solution. Our computational experiments indicate that both the algorithm and the heuristic perform quite well.

Acknowledgement

The research by the first author was partially supported by the Scientific and Technical Research Council of Turkey, NATO Advance Fellowship. The authors wish to thank

Professor Omer Kirca of the Middle East Technical University, Turkey for his helpful comments on an earlier phase of the research.

Appendix

The steps of the branch and bound algorithm to solve *P* are as follows. Here *z* denotes the optimal objective function value while *y*(i)* and *x*(i)* denote the optimal values of the current best solution at hand for i=1,...,n. *zstar, ch, y(i)* and *x(i)* (i=1,...,n), *k,* and *j* are auxiliary variables.

```
procedure branch and bound algorithm:
begin comment initialization and bounding;
    compute a lower bound z and an upper bound $\xi$ by using procedures;
      if $z \geq \xi$ then return;
      zstar:=0; ch:=b; c(n+1):=0; a(n+1):=$\infty$; m(n+1):=0; i=1;
   comment performing forward move;
   L1: while $a(i)+1 \leq ch$ do
            begin
              if $m(i)+a(i) \leq ch$ then
                    begin
                      y(i):=1; x(i):=m(i);
                      ch:=ch-a(i)-x(i); zstar:=zstar+c(i)*x(i);
                      i:=i+1
                    end
              else
```

```
                    begin
                      $y(i):=1$; $ch:=ch-a(i)$; $x(i):=ch$;
                      $ch:=0$; $zstar:=zstar+c(i)*x(i)$; $i:=i+1$;
                      go to $L3$
                    end
      end;
L2: while $i \leq n$ and $a(i)+1>ch$ do
      begin
        $y(i):=0$; $x(i):=0$; $i:=i+1$
      end;
    if $i=n$ then go to $L1$;
comment updating the best solution so far;
L3: if $i>n$ or $ch:=0$ and $z<zstar$ then
                      begin
                        $z:=zstar$; $j:=\min\{i-1, n\}$;
                        for $k:=1$ to $j$ do $y^*(k):=y(k)$; $x^*(k):=x(k)$;
                        for $k:=j+1$ to $n$ do $y^*(k):=0$; $x^*(k):=0$;
                        $i:=n$; go to $L5$
                      end;
L4: begin
      compute an upper bound $\xi$ by using upper bound procedure for
      free items on the tree if $ch>0$, o/w go to L5;
      if $z \geq zstar+\xi$ then go to $L5$ else go to $L1$
    end;
comment backtracking;
```

```
    L5: find j:=max{k<i: y(k)=1};
    if no such j then return;
    ch:=ch+x(j)+a(j); zstar:=zstar-c(j)*x(j); y(j):=0; x(j):=0; i:=j+1;
    go to L2;
end.
```

References

[1] Balas, E. and E. Zemel, "An Algorithm for Large Zero-One Knapsack Problems", *Operations Research 28(5)*, 1130-1154 (1980).

[2] Chajakis, E., D. and M. Guignard, "Exact Algorithms for the Setup Knapsack Problem", *INFOR* 32 (3), 124-142 (1994).

[3] Diaby, M., H.C. Bahl, M.H. Karwan, and S. Zionts, "Capacitated Lot Sizing and Scheduling by Lagrangean Relaxation", *European Journal of Operational Research 59*, 444-458 (1992).

[4] Goesmans, S., "Valid Inequalities and Separation for Mixed 0-1 Constraints with Variable Upper Bounds", *Operations Research Letters* 8, 315-322 (1989).

[5] Kirca, O., "A Heuristic Procedure for the Dynamic Lot Size Problem with Setup Time", Technical Report 89-13, METU, Ankara (1989).

[6] Kovalyov, M.Y., C.N. Potts, and L.N. Van Wassenhove, "Single Machine Scheduling with Set-ups to Minimize the Number of Late Items: Algorithms, Complexity and Approximation", Working Paper 96//TM, INSEAD, Fontainebleau (1996).

[7] Martello, S. and P. Toth, *Knapsack Problems, Algorithms and Computer Implementations*, John Wiley and Sons, New York (1990).

[8] Sural, H., *Multi-Item Lot Sizing Problem with Setup Times*, Ph.D. thesis, METU, Ankara (1996).