Parabola Volume 18, Issue 1 (1982)

THE KNAPSACK PROBLEM

BY

ELVIN J. MOORE

The knapsack problem is a simple example of a type of "integer programming" problem which is frequently met in the field of mathematics known as Operations Research. Roughly speaking, Operations Research deals with mathematical problems arising in such areas as factory operation, resource allocation, airline scheduling, school timetabling, and business management, among others.

A typical example of a knapsack problem is the following. Consider a hiker who is going bushwalking carrying a backpack (knapsack). He can carry a maximum weight of 40 kg, and he has available a total of six different items (e.g. sleeping bag, tins of food) which he would like to take if possible. Assume that he can assign a value to each of the items as shown in table 1. His problem is then to select items to pack so that the value carried in his knapsack is the maximum possible.

Item	Value	Weight
1	\$76	16kg
2	\$40	10kg
3	\$20	6kg
4	\$30	6kg
5	\$24	8kg
6	\$96	16kg

Table 1. Values and weights of items.

We let n be the total number of items, W be the maximum weight that can be carried, v_i be the value of item i, w_i be the weight of item i, and x_i be the number of units of item i which are packed in a load. In the above example, $n = 6$, $W = 40$, $v_4 = 30$, $w_4 = 6$. Further, $x_4 = 0$ means item 4 is not packed while $x_4 = 1$ means item 4 is packed. The general knapsack problem is then to select values of the x_i's so that the total value

$$V = \sum_{i=1}^{n} v_i x_i$$

is a maximum subject to the constraints

$$\sum_{i=1}^{n} w_i x_i \leq W \quad \text{and} \quad x_i = 0 \text{ or } 1 \text{ for } i = 1,2, \ldots, n.$$

Problems of this form are called integer linear programming problems.

It has been shown that the knapsack problem belongs to a class of very hard problems called the NP-complete problems. However, if the restriction that only complete items can be packed ($x_i = 0$ or 1) is replaced by the condition that parts of items can be packed ($0 \leq x_i \leq 1$), then the solution is very simple. To obtain the solution when the x_i's can be real numbers, we first calculate the "value density" (value per unit weight) of each item and then write down the items in order of decreasing value density. This gives us table 2.

Item	Value	Weight	Value density
6	\$96	16kg	6 \$/kg
4	\$30	6kg	5 \$/kg
1	\$76	16kg	4.75 \$/kg
2	\$40	10kg	4 \$/kg
3	\$20	6kg	3.33 \$/kg
5	\$24	8kg	3 \$/kg

Table 2. Items rearranged in order of decreasing value density.

To obtain the knapsack load of maximum value, we just pack items or parts of items starting with the top line of the table and continue until the maximum allowed weight of 40kg is reached. The solution is to pack items 6, 4, 1 and 0.2 of item 2 ($x_6 = x_4 = x_1 = 1$, $x_2 = 0.2$, $x_3 = x_5 = 0$) for a maximum value of \$210.

This property of the knapsack problem that it is much more difficult to solve with integers than with real numbers is in fact typical of many problems. As a simple illustration compare the following: Find real numbers x and y such that $x^2 + y^2 = 299$ with find integers x and y such that $x^2 + y^2 = 299$. In the real number case an infinite number of solutions exist, and any one of them can be immediately written down simply by selecting a value of x and then putting $y = \pm \sqrt{299 - x^2}$. In the integer case it is not obvious if any solutions exist (none does), and to search for a solution it would be necessary to use an "exhaustive search" procedure of selecting each possible integer value for x and checking if $\sqrt{299 - x^2}$ is an integer.

The use of such an exhaustive search algorithm of considering all possible combinations of values of the six x_i's, computing the weights and values for each

combination, and selecting the combination of maximum value with weight not exceeding 40kg, is a possible method of solving the integer knapsack problem. As there are two possibilities (0 or 1) for each x_i there are only a total of $2^6 = 64$ combinations to check. However, the procedure quickly becomes impractical as the number of items is increased. If there are n items, then there are a total of 2^n possible combinations to be checked. This exhaustive search method is an example of an "exponential time" algorithm, i.e., the time required to compute a solution using the algorithm increases exponentially with the number of items considered. If we assume that 10^7 combinations can be checked per second (which is not unreasonable for present computers), we obtain table 3.

Number of items n	Time required for solution
2	4×10^{-7} secs.
10	10^{-4} secs.
20	.1 sec.
30	1.8 mins.
40	1.3 days
50	3.6 years

Table 3. Computation times for the exhaustive search algorithm.

Our earlier statement that the knapsack problem is an NP-complete problem means in effect that no algorithm can exist which will always find a solution in less than exponential time. Fortunately, however, there exist algorithms which find a solution very quickly for almost all knapsack problems. In fact, it turns out to be difficult to actually construct an example of a knapsack problem which the algorithms would take an exponential time to solve.

One efficient algorithm for solving the knapsack problem is based on the "branch and bound" method. It is usual to represent this method by a "tree" in which each branch represents a decision about the loading of some item. In figure 1 some of this tree is shown. The top circle is the root of the tree, and represents the initial stage of the loading process in which no decision about the loading of items has yet been made. This root then branches to the first set of circles (which are called nodes), with each branch representing a possible choice about the loading of the item of greatest value density (item 6). The 6 in the left node means item 6 is loaded ($x_6 = 1$), while the 6* in the right node means item 6 is not loaded ($x_6 = 0$).

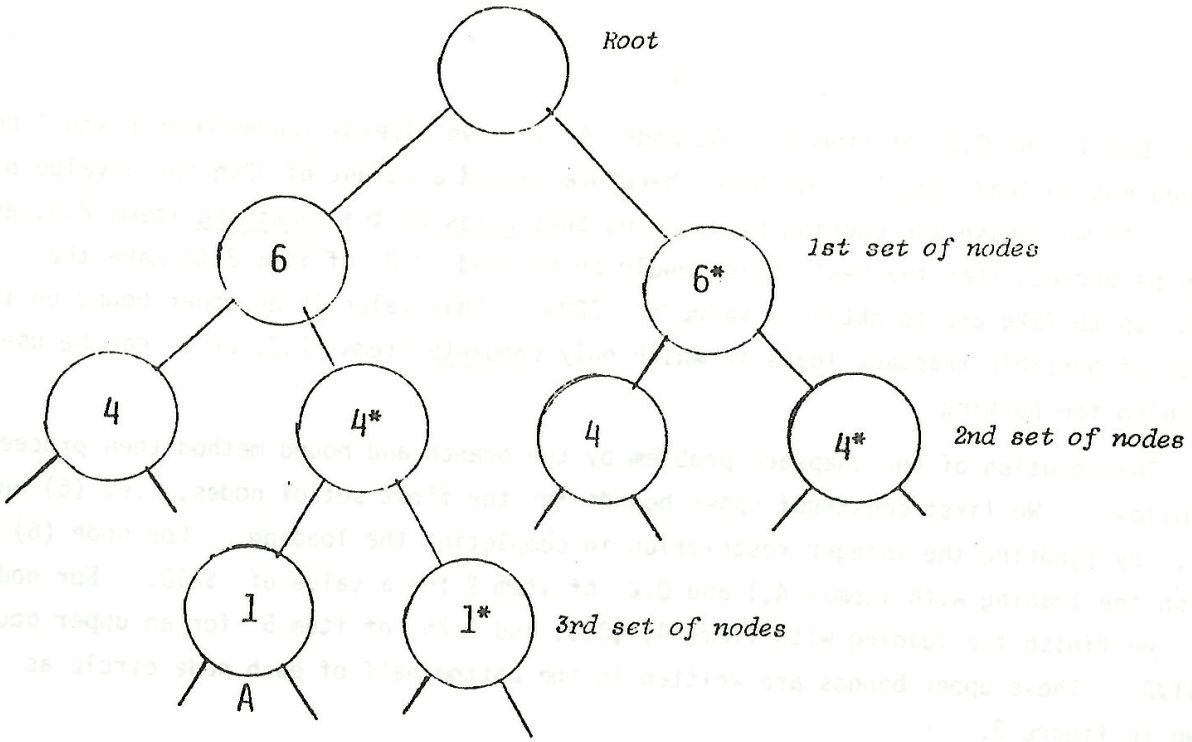

Figure 1. Partial tree of solutions illustrating the branching scheme.

Each of these two nodes has two branches representing possible choices about the loading of the item of second highest value density (item 4). The four nodes in the second set represent the four possible choices for the loading of items 6 and 4. The branching process is continued until the sixth set of nodes is reached when decisions have been made about the loading of each of the six items. A node in the sixth set represents a complete knapsack load and is called a "terminal node".

Each node represents a sequence of choices which have been made about items appearing on the path from the node back to the root node (empty knapsack). In figure 1, the node labelled A in the third set can be interpreted by writing down any numbers appearing in nodes on the path from the root node to node A. For this node we can write down (6,4*,1); which means item 6 is loaded, item 4 is not loaded, item 1 is loaded, and that no decisions have yet been made about items 2, 3, and 5.

The second basic idea in the branch and bound method is to construct upper bounds on the values of all possible knapsack loads which could be obtained if the packing were continued from the partial load represented by a node. We shall illustrate the process by constructing the upper bounds for the root node and node A. At the root node the knapsack is empty. Starting with an empty knapsack, we cannot possibly obtain a value for a load greater than the real value solution of $210 obtained by packing

items 6,4,1, and 0.2 of item 2. At node A we have already loaded items 6 and 1 and decided not to load item 4. We have therefore packed a weight of 32kg for a value of \$172. If we finish the packing by assuming that parts of the remaining items 2,3, and 5 can be packed, then the best choice would be to load 0.8 of item 2 to make the weight up to 40kg and to obtain a value of \$204. This value is an upper bound on the values of possible knapsack loads in which only complete items 2,3, or 5 can be used to finish the packing.

The solution of the knapsack problem by the branch and bound method then proceeds as follows. We first construct upper bounds for the first set of nodes, i.e. (6) and (6*), by ignoring the integer restriction in completing the loading. For node (6) we finish the loading with items 4,1 and 0.2 of item 2 for a value of \$210. For node (6*) we finish the loading with items 4,1,2,3, and 0.25 of item 5 for an upper bound of \$172. These upper bounds are written in the bottom half of each node circle as shown in figure 2.

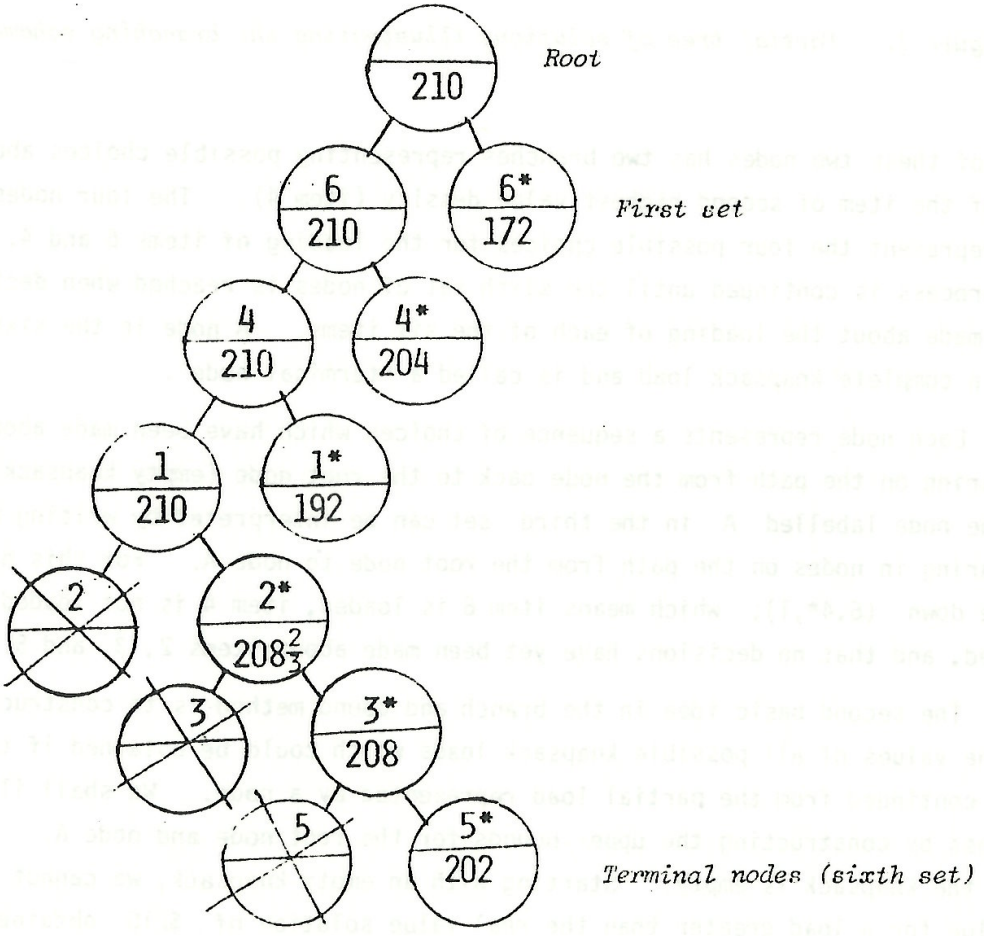

Figure 2. Tree for solution of the knapsack problem after terminal node (6, 4, 1, 2, 3*, 5*) has been reached.*

As the upper bound for (6) is higher than that for (6*) we choose to branch to the second set of nodes from (6). The upper bounds for (6,4) and (6,4*) are then calculated, and their values written in the bottom half of each node circle as before. As the (6,4) bound is higher, we branch to the third set from it and again construct upper bounds. As (6,4,1) has the higher bound we branch to the fourth set from it. In figure 2, the node (6,4,1,2) can be crossed out as the weight of items 6,4,1 and 2 exceeds the weight of 40kg. We therefore branch from (6,4,1,2*). The node (6,4,1,2*,3) can again be crossed out as items 6,4,1,3 are too heavy. The branching from (6,4,1,2*,3*) represents a decision about the last item (item 5). The only choice for item 5 which remains within the weight limit is 5*. The corresponding node (6,4,1,2*,3*,5*) is a terminal node (i.e. a full knapsack load) of value $202. This value of $202 for a full load is a lower bound on the maximum value solution; any full load whose value is less than $202 is clearly not the maximum value load.

The solution now continues as shown in figure 3.

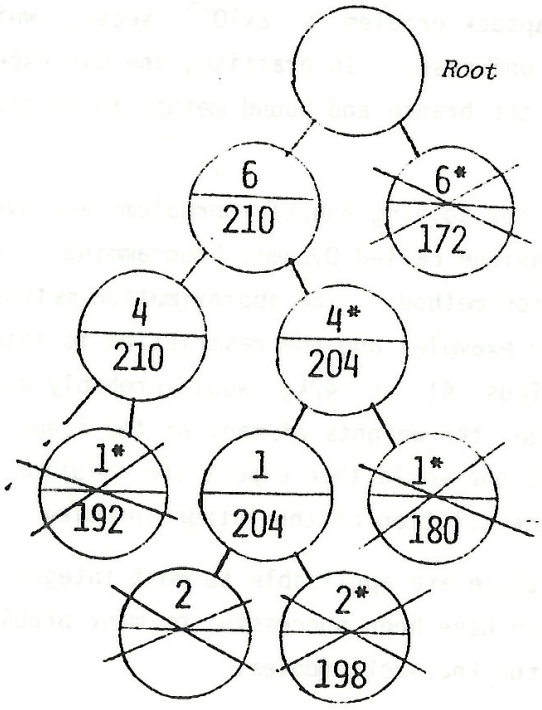

Figure 3. Continuation of the tree for solution of the knapsack problem. Note that some nodes from figure 2 have been repeated.

We can cross out node (6,4,1*); as we have arleady found that no possible load containing this selection can have a value in excess of $192. Similarly, node (6*) can be crossed out. However, node (6,4*) with upper bound of $204 cannot be crossed out; as it is possible that a load containing this selection could have a value greater than $202. We therefore branch from (6,4*) until either all nodes can be crossed out

(weight exceeds 40kg or upper bound is less than $202) or a terminal node is reached. The nodes which must be examined are shown in figure 3.

The solution of the knapsack problem is complete when all nodes have been crossed out except for one terminal node, which then represents the load of maximum value. In our example, the surviving terminal node is (6,4,1,2*,3*,5*). The maximum value load therefore contains items 6,4, and 1, has a weight of 38kg, and is of value $202.

To obtain the maximum solution we have considered a total of 16 nodes (the value for the root node is not actually required). It is possible that we could have obtained a solution for a 6-item knapsack problem by examining a total of 12 nodes (minimum of two nodes for each set), but it is also possible that we might have had to consider a total of 126 nodes (2 + 4 + 8 + ... + 64). For an n-item knapsack problem, the minimum number of nodes which must be considered is 2n, whereas the maximum number which might have to be considered is $2 + 4 + 8 + \ldots + 2^n = 2^{n+1} - 2$. Thus, for example, the n = 100, and with the same assumptions about computer speed as in table 3, the minimum time to solve the knapsack problem is 2×10^{-5} secs., while the maximum time is greater than the age of the universe. In practise, one can expect the time taken to solve knapsack problems by the branch and bound method to be close to the minimum time.

Alternative efficient methods for solving knapsack problems are available. One popular algorithm is based on a technique called Dynamic Programming. A second popular algorithm is based on an approximation method. The approximation method recognizes that the weight limit (40kg in our example) and the restriction to integer values of the x_j's can often be relaxed. Thus 41 or 42kg would probably be acceptable. Also, if the number of items is large, the weights of many of the items will usually be much less than the weight limit, and one would then expect the solution of the real value problem to be very close to the solution of the integer problem.

The ideas outlined in this article are applicable to most integer programming problems, and the methods of solution have been successful on many problems which are considerably more complicated than the knapsack problem.