



---

An Efficient Algorithm for the 0-1 Knapsack Problem

Author(s): Robert M. Nauss

Source: *Management Science*, Sep., 1976, Vol. 23, No. 1 (Sep., 1976), pp. 27-31

Published by: INFORMS

Stable URL: <https://www.jstor.org/stable/2629752>

---

JSTOR is a not-for-profit service that helps scholars, researchers, and students discover, use, and build upon a wide range of content in a trusted digital archive. We use information technology and tools to increase productivity and facilitate new forms of scholarship. For more information about JSTOR, please contact [support@jstor.org](mailto:support@jstor.org).

Your use of the JSTOR archive indicates your acceptance of the Terms & Conditions of Use, available at <https://about.jstor.org/terms>



INFORMS is collaborating with JSTOR to digitize, preserve and extend access to *Management Science*

JSTOR

## AN EFFICIENT ALGORITHM FOR THE 0-1 KNAPSACK PROBLEM\*†

ROBERT M. NAUSS

*University of Missouri-St. Louis*

In this note we present an efficient algorithm for the 0-1 knapsack problem and announce the availability of a callable FORTRAN subroutine which solves this problem. Computational results show that 50 variable problems can be solved in an average of 4 milliseconds and 200 variable problems in an average of 7 milliseconds on an IBM 360/91.

The 0-1 knapsack is a very simple model. The interested reader is referred to the special chapter on the knapsack problem in the excellent integer programming (IP) book by Garfinkel and Nemhauser [2]. Ross and Soland [10] and Nauss [9] give examples where the 0-1 knapsack serves as a subproblem in the solution of the generalized assignment problem and the capacitated facility location problem respectively. Other applications abound, and an efficient 0-1 knapsack code could potentially be used in many IP codes. The purpose of this note is to make such a code available to interested researchers and practitioners. Copies of the card deck along with documentation are available from the author for a handling charge of \$5.00.<sup>1</sup>

Consider the 0-1 knapsack problem:

$$\begin{aligned} \max \quad & cx \\ x = 0, 1 \quad & wx \leq B \end{aligned} \tag{P}$$

where  $x$  is an  $n$ -vector and  $B$  is a scalar. Without loss of generality we assume  $c > 0$ ,  $w > 0$ ,  $B > 0$ , and we define  $v(\cdot)$  to be the optimal solution value of  $(\cdot)$ . By capitalizing on the simplicity of this problem, it is possible to achieve substantial savings in computation time over, say, a general 0-1 IP code. There are two properties of (P) which may be exploited successfully. First we assume that the variables have been ordered by decreasing "bang-for-buck" ratios so that  $c_1/w_1 \geq c_2/w_2 \geq \dots \geq c_n/w_n$ . With this ordering the solution to the linear program  $(\bar{P})$  (replacing  $x = 0, 1$  by  $0 \leq x \leq 1$ ) becomes analytic. That is, variables with the largest bang-for-buck are placed in the knapsack at their upper bounds of 1 until no more room remains in the knapsack. At this point the variable which could not fit is placed in the knapsack at a fractional level such that the knapsack is filled. It is clear that all variables, with the possible exception of one, have value 0 or 1 in an optimal solution to  $(\bar{P})$ . It follows that by setting the fractional variable to 0, feasibility in (P) is achieved, so that a lower bound on  $v(P)$  is readily available. It is these two properties (an analytic solution to  $(\bar{P})$  and a simple feasible solution generator) which are exploited in our algorithm.

Korsh and Ingaragiola (KI) [8] have developed an algorithm for 0-1 knapsacks which has proved to be very effective in reducing computation times. Basically they employ an inexpensive LP test which, if passed, allows a variable to be fixed (pegged) to 0 or 1 at the root (initial) node of a branch and bound tree. Computational results

\* Processed by Professor Arthur M. Geoffrion, Departmental Editor for Integer and Large-Scale Programming; received February 5, 1975, revised September 17, 1975. This paper has been with the author 1 month for revision.

† This research was partially supported by the National Science Foundation under Grant GP-36090X and by the Office of Naval Research under Contract N00014-69-A-200-4042. Reproduction in whole or in part is permitted for any purpose of the United States Government.

<sup>1</sup> Requests for copies should be sent to the author at: School of Business Administration, University of Missouri-St. Louis, 8001 Natural Bridge Road, St. Louis, Missouri, 63121.

show that upwards of 80% of the variables may be pegged to 0 or 1. Once the pegging tests are completed, the "reduced" knapsack problem consisting of the unpegged variables is solved by any available knapsack algorithm. Since computation time for the pegging test is linearly proportional to the number of variables, and a branch and bound approach is generally exponentially proportional, such a device would appear to be quite attractive. This is borne out in the computational results of KI.

Dembo [1] has noted that the concept of Lagrangean relaxation may be used in carrying out the KI pegging tests. While his test is slightly weaker than the KI test, the computation time required for the pegging phase only is about 2/3 less than for the KI method.

Horowitz and Sahni (HS) [7] have developed a branch and bound algorithm which dominates the well-known Greenberg and Hegerich [5] algorithm. We present a variant of the HS algorithm (Steps 7–17 below) which has decreased computation times (in the branch and bound phase) by approximately  $1/3$  over the original HS algorithm.

In order to make the presentation of the algorithm clear, we shall appeal to the general branch and bound framework given in Geoffrion and Marsten [4]. An explanation of the finer points of the algorithm will be deferred until later.

### Algorithm A

1. Order the variables by decreasing bang-for-buck so that  $c_1/w_1 \geq c_2/w_2 \geq \dots c_n/w_n$ . Set  $I_1 = I_0 = \emptyset$ .
2. Solve  $(\bar{P})$  getting an optimal solution  $\bar{x}$  and an optimal dual multiplier  $\bar{\lambda}$  associated with the budget constraint. If  $\bar{x}$  is feasible in  $(P)$ , stop: the solution is optimal. Otherwise denote the index of the fractional variable by  $r$ .
3. Find a lower bound  $z^*$  for  $v(P)$  by setting  $\bar{x}_r = 0$  in the solution to  $(\bar{P})$ . Let  $x^* = \bar{x}$ . Try to improve  $z^*$  by certain heuristics.
4. For  $\forall i = 1, \dots, r-1$ , if  $v(\bar{P}) - c_i + \bar{\lambda}w_i \leq z^*$ , set  $I_1 = I_1 \cup \{i\}$  ( $x_i$  is pegged to 1).
5. For  $\forall i = r+1, \dots, n$ , if  $v(\bar{P}) + c_i - \bar{\lambda}w_i \leq z^*$ , set  $I_0 = I_0 \cup \{i\}$  ( $x_i$  is pegged to 0).
6. Solve the remaining knapsack problem:

$$\text{Max } \sum_{i \in I_1} c_i + \sum_{i \notin I_1 \cup I_0} c_i x_i \quad (R)$$

$$\sum_{i \notin I_1 \cup I_0} w_i x_i \leq B - \sum_{i \in I_1} w_i$$

by using the branch and bound procedure in Steps 7–17.

7. Initialize the candidate list to consist of  $(R)$  and let the incumbent value be  $z^*$ .
8. If the candidate list is empty, stop:  $x^*$  is an optimal solution to  $(P)$  and  $z^*$  is the optimal value.
9. Select a candidate problem  $(CP)$  from the candidate list by a LIFO rule.
10. Solve  $(\overline{CP})$  getting an optimal solution  $\bar{x}$ .
11. If  $(\overline{CP})$  is infeasible, go to 8.
12. If  $v(\overline{CP}) \leq z^*$ , go to 8.
13. If an optimal solution of  $(\overline{CP})$  is feasible in  $(CP)$ , go to 17.
14. Choose that  $x_j$  which is the free variable with the largest bang-for-buck.
15. If  $w_j \leq B - \sum_{i: x_i \text{ set to } 1} w_i$ , then add only  $(CP \mid x_j = 0)$  to the candidate list, add the restriction  $x_j = 1$  to  $(CP)$ , and go to 14. Otherwise go to 16.
16. If  $w_j > B - \sum_{i: x_i \text{ set to } 1} w_i$ , add the restriction  $x_j = 0$  to  $(CP)$ , choose that  $x_j$  which is the free variable with the largest bang-for-buck, and return to the beginning of this step. Otherwise go to 10.

17. A feasible solution to (P) has been found. Set  $z^* = v(\overline{CP})$ ,  $x^* = \bar{x}$ , and go to 8.

In Step 2 an optimal dual multiplier  $\bar{\lambda}$  for  $(\bar{P})$  can be shown to be equal to  $c_r/w_r$ . In Step 3 two heuristics are used in an attempt to improve the value of  $z^*$ . First, set  $\tilde{x} = \bar{x}$  and  $\tilde{x}_r = 0$ . The solution  $\tilde{x}$  then has a slack in the constraint with value  $s = \bar{x}_r \cdot w_r$ . Now, for  $i = r + 1, \dots, n$  the following is done: if  $w_i \leq s$ , set  $\tilde{x}_i = 1$  and  $s = s - w_i$ . If  $s > 0$ , repeat this step for  $i = i + 1$ . If  $c\tilde{x} > cx^*$ , set  $z^* = c\tilde{x}$  and  $x^* = \tilde{x}$ . Basically, this heuristic puts extra variables in the knapsack until no more variables fit. The second heuristic begins by setting  $\tilde{x} = \bar{x}$  and  $\tilde{x}_r = 1$ . This overfills the knapsack by  $s = (1 - \bar{x}_r) \cdot w_r$ . Then for  $i = r - 1, r - 2, \dots, 1$  the following is done: set  $\tilde{x}_i = 0$ ,  $s = s - w_i$ , and if  $s > 0$  repeat this step for  $i = i - 1$ . When  $s \leq 0$ , set  $s = -s$  and return to the test loop in the first heuristic. Thus this heuristic begins by overfilling the knapsack, and then variables are withdrawn until feasibility is obtained. At this point the test loop in the first heuristic is employed. The pegging tests in Steps 4 and 5 utilize the notion of Lagrangean relaxation (LGR). Consider the relaxation:

$$\max_{x=0,1} cx + \bar{\lambda}(B - wx). \quad (\text{LGR}_{\bar{\lambda}})$$

It is easily seen (Geoffrion [3]) that  $v(\text{LGR}_{\bar{\lambda}}) = v(\bar{P})$  when  $\bar{\lambda}$  is an optimal dual multiplier for  $(\bar{P})$ , and that the solution to  $(\text{LGR}_{\bar{\lambda}})$  is analytic. That is,

$$\begin{aligned} \hat{x}_i &= 1 && \text{if } c_i - \bar{\lambda}w_i > 0, \\ &= 0 && \text{if } c_i - \bar{\lambda}w_i \leq 0. \end{aligned}$$

Thus we have  $v(\text{LGR}_{\bar{\lambda}} | x_i = 1) = v(\bar{P}) - c_i + \bar{\lambda}w_i$  if  $\hat{x}_i = 0$ , and  $v(\text{LGR}_{\bar{\lambda}} | x_i = 0) = v(\bar{P}) - c_i + \bar{\lambda}w_i$  if  $\hat{x}_i = 1$ . Since  $v(\text{LGR}_{\bar{\lambda}}) \leq v(P) \forall \bar{\lambda} \geq 0$ , it follows that the pegging tests are valid. Steps 4 and 5 may be enhanced by adding the following tests if the LGR tests fail:

4b. If  $v(\bar{P} | x_i = 0) \leq z^*$ , then set  $I_1 = I_1 \cup \{i\}$ .

5b. If  $v(\bar{P} | x_i = 1) \leq z^*$ , then set  $I_0 = I_0 \cup \{i\}$ .

The branch and bound algorithm of Steps 7–17 is straightforward, but a few comments may make it clearer. In Step 9 a LIFO selection rule is used, thus

TABLE 1  
*Comparison of HS Algorithm and Algorithm A Omitting Steps 1–6*

(Time in milliseconds excluding I/O and sort time on an IBM 360/91)			
Problem Identifier	Number of Variables	H&S	Algorithm A Omitting Steps 1–6
1365	50	6	4
1397	50	8	5
1398	50	8	6
1326	50	12	8
1406	50	7	5
1366	50	16	11
1282	50	7	5
1340	50	15	9
1288	50	9	7
Total for 9 problems		98	67
Average for 9 problems		10	7

guaranteeing linear storage and minimal setup costs in a computer implementation. Step 13 is an addition to the HS implementation which recognizes that if a relaxation has an optimal integer feasible solution, then the current candidate problem may be fathomed. The branching strategy in Steps 14–16 is done as follows. The most attractive free variable (in terms of largest bang-for-buck) is chosen as the branch variable. However, since this variable has value 1 in  $(\overline{CP})$ , we have  $v(\overline{CP} \mid x_j = 1) = v(\overline{CP})$ . Thus, no repotimization is required, and the next branch variable may be chosen. This continues until the next variable chosen,  $x_j$ , cannot fit in the knapsack at

TABLE 2  
*Computation Time in Milliseconds for Algorithm A (excluding sort<sup>2</sup> and I/O time on an IBM 360/91)*

Problem	Number of Variables	% Reduction in Steps 1–6	Time for Steps 2–6	Time for Steps 7–17	Total Time
1365	50	88	2	0	2
1397	50	82	2	1	3
1348	50	62	2	2	4
1326	50	56	2	4	6
1406	50	82	2	1	3
1366	50	58	2	6	8
1282	50	80	3	1	4
1340	50	82	2	3	5
1288	50	78	2	3	5
1468	50	86	2	1	3
	Total Average	75	21 2	22 2	43 4
2823	100	89	3	2	5
2772	100	65	3	2	5
2763	100	67	3	5	8
2945	100	88	3	1	4
2795	100	99	2	0	2
2706	100	85	3	5	8
2589	100	99	3	0	3
2992	100	95	3	1	4
2447	100	94	3	0	3
2771	100	86	3	1	4
	Total Average	87	29 3	17 2	46 5
5531	200	86	4	3	7
5790	200	79	4	10	14
5423	200	94	4	2	6
5641	200	99	5	0	5
5614	200	90	4	3	7
5536	200	94	4	2	6
5108	200	93	4	2	6
5274	200	95	5	1	6
5448	200	92	4	4	8
5734	200	93	4	4	8
	Total Average	92	42 4	31 3	73 7

<sup>2</sup> Sort time averages about 2.5 milliseconds for 50 variable problems and 6 milliseconds for 100 variable problems.

TABLE 3  
*Relative Increases in Computation Time as a  
 Function of the Number of Variables in a Knapsack*

Number of Variables	KI Algorithm	Algorithm A
50	1.0	1.0
100	1.7	1.1
200	4.2	1.7

a level of 1. But this implies that  $x_j$  may be pegged to 0 due to feasibility considerations. Pegging variables to 0 continues until no longer possible. At this point control is returned to Step 10, and the relaxation of the current candidate problem is solved. In actuality, a group of variables (contiguous by index) are committed to 1 until this is no longer possible, and then a group of variables (contiguous by index) are pegged to 0 until this is no longer possible. This allows the LP in Step 10 to be bypassed after most branching operations, which in turn reduces computation time measurably. This is seen in Table 1 where the HS algorithm (as coded by HS) is compared to Steps 7–17 of Algorithm A. Note that Steps 1–6 were omitted in these runs. Both algorithms were coded in FORTRAN H and run on an IBM 360/91. Random  $c_i$  and  $w_i$  were generated from a uniform distribution,  $U[10, 100]$ , and  $B$  was set to  $0.5 \cdot (\sum_{i=1}^n w_i)$ . Results show that Steps 7–17 of Algorithm A reduced computation time by 33% when compared with the HS algorithm.

Results for all of Algorithm A (Steps 1–17) are given in Table 2. These results clearly dominate KI's results, even when machine differences are taken into account. Table 3 portrays this dominance. The trend, as the number of variables increases, definitely is in favor of Algorithm A. In fact, quadrupling the number of variables increases computation time by a factor of 1.7, while for KI this factor is 4.2. Finally, we mention that inclusion of Steps 4b and 5b in Algorithm A was ineffective, and in fact increased computation times slightly.

#### References

1. DEMBO, R. S., private communication, May 1974.
2. GARFINKEL, R. AND NEMHAUSER, G., *Integer Programming*, John Wiley and Sons, 1972.
3. GEOFFRION, A. M., "Lagrangian Relaxation for Integer Programming," *Mathematical Programming Study* 2, (1974), pp. 82–114.
4. ——— AND MARSTEN, R. E., "Integer Programming Algorithms: A Framework and State-of-the-Art Survey," *Management Science*, Vol. 18, No. 9 (May 1972), pp. 465–491.
5. GREENBERG, H. AND HEGERICH, R., "A Branch Search Algorithm for the Knapsack Problem," *Management Science*, Vol. 16, No. 5 (January 1970), pp. 327–332.
6. HAMMER, P. L. AND NGUYEN, S., "A Partial Order in the Solution Space of Bivalent Programs," presented at the 41st Meeting of ORSA, New Orleans, Louisiana, April 1972.
7. HOROWITZ, E. AND SAHNI, S., "Computing Partitions with Applications to the Knapsack Problem," *Journal of the Association for Computing Machinery*, Vol. 21, No. 2 (April 1974), pp. 277–292.
8. INGARAGIOLA, G. P. AND KORSH, J. F., "Reduction Algorithm for Zero-One Single Knapsack Problems," *Management Science*, Vol. 20, No. 4 (December 1973), pp. 460–463.
9. NAUSS, R. M., *Parametric Integer Programming*, Ph.D. Dissertation, University of California, Los Angeles, 1974.
10. ROSS, G. T. AND SOLAND, R. M., "A Branch and Bound Algorithm for the Generalized Assignment Problem," *Mathematical Programming*, Vol. 8, No. 1 (1975), pp. 91–103.