BASEMENT

LIBRARY
OF THE
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

HD28
.M414
no.177-66

WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

METHODS FOR THE SOLUTION OF THE MULTI-DIMENSIONAL 0/1 KNAPSACK PROBLEM*

177-66

H. Martin Weingartner and David N. Ness

April 1966

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

METHODS FOR THE SOLUTION OF THE MULTI-DIMENSIONAL 0/1 KNAPSACK PROBLEM*

177-66

H. Martin Weingartner and David N. Ness

April 1966

*Work on this study was supported by the Ford Foundation's Grant to the Sloan School of Management for Research in business finance. It was also supported in part by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense under the Office of Naval Research Contract Number NONR-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government. This paper was prepared for presentation at the 29th Annual Meeting of the Operations Research Society of America in Santa Monica, California, May 18, 1966.

HD28
.M44
[illegible]

RECEIVED
JUN 23 1966
M. I. T. LIBRARIES

METHODS FOR THE SOLUTION OF THE MULTI-DIMENSIONAL 0/1 KNAPSACK PROBLEM

H. Martin Weingartner and David N. Ness

ABSTRACT

In the knapsack problem, given the desirability of each of a number of items, one seeks to find that subset which satisfies a constraint on total weight. The multi-dimensional variant imposes constraints on additional variables of the items; the 0 1 specification means that an item is either taken or not, i.e., multiples of the same item are not considered, except possibly indirectly. Traditionally the 1-dimensional knapsack problem is solved by means of dynamic programming. The multi-dimensional problem is usually reduced to a one-dimensional one by use of Lagrangian Multipliers which, however, do not generally yield the exact solution to the problem posed. The present paper considers methods for obtaining the exact solution to the problem, and not an approximate one. Additional algorithms are developed which are applied within a dynamic programming framework. Given these, the object is to obtain solutions efficiently, and in attaining this goal heuristic methods are employed. Efficiency of the methods is based upon the use of an interactive computer system in which the heuristics of the problem solver are applied and changed as the character of the solution process evolves. The project was conducted with the Compatible Time Sharing System of M.I.T.'s Project MAC. The problem arises in the context of capital budgeting, but has obvious applications in a variety of other areas. The methods have been employed for solving numerical problems with as many as 105 items, the parameters having been obtained from industrial applications.

INTRODUCTION

The knapsack problem is a familiar one in Operations Research. In its simplest form, the problem is to find the most desirable set of articles a camper should pack in his knapsack given a measure of the desirability of each item and its weight, as well as the maximum weight which the knapsack (or hiker) can carry. In the multi-dimensional knapsack problem, additional limitations, e.g., on volume or height, width and length are also imposed. The 0/1 qualification to the problem, which is the one to be discussed here, disregards the possibility (except perhaps indirectly) of

taking along more than one of a given article. Thus the question is simply that of inclusion or exclusion of an item.

The general knapsack problem has been discussed in the literature in a variety of places [3]. Motivation for the present approach was provided by certain capital budgeting problems in which investment projects are to be selected subject to expenditure limitations in several time periods, or to limitations on several inputs. The propriety of the model for budgeting has been fully discussed in [6]. It should also be born in mind that in many budgeting applications in which the constant terms of the constraints are not entirely rigid, a different formulation, based on linear programming, may be more appropriate, as was analyzed in [7]. A mixture of the two formulations, rigid and "flexible" limitations also occurs in practical problems.

The purpose of this paper is to describe computational methods, based on dynamic programming, for the solution of allocation with rigid constraints. These methods all seek to attain the optimum solution, not merely good solutions, for sizable problems and are not based on the use of heuristic methods. They make essential use of an interactive computer system in which the course of the computations is controlled by the user, who responds to the status of the problem solution.

THE BASIC MODEL

For descriptive purposes (if not for computation) the problem may be stated as an integer programming problem. Let b_i be the payoff from inclusion of project i, c_{ti} the outlay required in period t on project i, C_t the maximum to be spent in period t. Further, let x_i be the "fraction"

of project i accepted, so that $x_i = 1$ means acceptance, $x_i = 0$ rejection. The basic model then is

$$\text{(a)} \quad \text{Maximize} \qquad \sum_{i=1}^{n} b_i x_i$$

$$\text{(b)} \quad \text{Subject to} \qquad \sum_{i=1}^{n} c_{ti} x_i \leq C_t, \qquad t=1,\ldots,T$$

(1)

$$\text{(c)} \qquad 0 \leq x_i \leq 1, \qquad i=1,\ldots,n$$

$$\text{(d)} \qquad x_i \text{ integer}$$

Since, in the present interpretation of the problem,[1] all c_{ti} are nonnegative the model in (1) may be solved, in principle, as a dynamic programming problem as follows:

$$\text{(2)} \quad f_i(C_1', C_2', \ldots, C_T') = \max_{x_i=0,1} [b_i x_i + f_{i-1}(C_1'-c_{1i}x_i, C_2'-c_{2i}x_i, \ldots, C_T'-c_{Ti}x_i)], \qquad i=1,\ldots,n$$

for

$$C_t' - c_{ti}x_i \geq 0; \qquad t=1,\ldots,T; \; i=1,\ldots,n,$$

and

$$f_0(C') = 0$$

where $f_i(C')$ is the total value of the optimally selected projects with projects i+1, i+2, ..., n still to be considered, and the unallocated funds are given by the vector $C' = (C'_1, C'_2, \ldots, C'_T)$. As stated above, the stages of the dynamic programming calculations are the computation of undominated feasible strategies (vectors of values for the x_j, $j=1,\ldots,i$) which result from consideration of an additional project. Undominated here means that the strategy is not simultaneously lower in payoff and higher in outlay in all of the budget years.[2] Feasibility means simply that the strategy does not violate any of the budget constraints.

The algorithm given in (2) may be thought of as using either backward or forward optimization or neither of these. The reason is that, as far as the algorithm is concerned, the order is arbitrary. Projects may be arranged in any way, although some ways may be better than others. However, the arrangement of projects is more properly regarded as a preprocessing operation and as such will be discussed in a separate section, below.

Straightforward application of dynamic programming to this problem, as reported in [6], proved inadequate to the task when more difficult problems were attempted.[3] For this reason further computational methods were devised and the computer programs were rewritten using entirely different concepts of data handling.

THE COMPLEMENT PROBLEM AND ALGORITHM

The difficulty of growth in the length of the list of undominated feasible strategies, possibly beyond the high-speed memory capacity of the computer, or the requirement of an uneconomical amount of computer time

can arise in even modest sized problems.[4] When the budget constraints are relatively loose, making possible the acceptance of most, though not all, projects, it is possible to speed up the calculations by solving the complement" problem in which it is desired to eliminate projects, rather than to accept them. The model corresponding to (2) is obtained as follows. Let

$$\bar{C}_t = \sum_{i=1}^{n} c_{ti}, \qquad t=1,\ldots,T \tag{3}$$

and

$$\underline{C}_t = \bar{C}_t - C_t, \qquad t=1,\ldots,T. \tag{4}$$

Further, let

$$y_i = 1 - x_i \tag{5}$$

so that if $y_i = 1$, project i is excluded; if $y_i = 0$, it is included. Then $\underline{C}_t$ is the smallest amount to be eliminated from each budget starting with an amount equal to the total of outlays required for inclusion of all projects. This is to be achieved by the elimination of the least aggregate payoff consistent with this requirement. The model then becomes

$$\text{(a)}\quad f_i(C_1'', C_2'', \ldots, C_T'') = \min_{y_i=0,1} \left[b_i y_i + f_{i-1}(C_1''-c_{1i}y_i, C_2''-c_{2i}y_i, \ldots, C_T''-c_{Ti}y_i) \right]$$

$$i=1,\ldots,n$$

for

$$\text{(b)}\quad C_t' - c_{ti}y_i \geq \min\,(C_t'', 0), \qquad t=1,\ldots,T;\ i=1,\ldots,n. \tag{6}$$

and

$$\text{(c)}\quad f_0(C'') = 0$$

where $C'' = (C_1'', C_2'', \ldots, C_T'')$ is the vector of budgeted amounts still to be eliminated, and $f_i(C'')$ is the optimal amount of payoff deducted from the total possible payoff without constraints with C" still to be eliminated from the budgets and with projects i+1, i+2, ..., n still to be considered.

The complement problem appears to bear a certain resemblance to the dual method of linear programming. A minimization has been stubstituted for the original maximization, and the side condition now is to ascertain when a feasible solution has been obtained, as opposed to the requirement that the solutions remain feasible. The complement algorithm may take more or less time than the direct algorithm, and also more or less internal memory. Before, however, entering into a discussion of its relative merits, we turn instead to different devices which are useful in speeding up the computations.

"LOOK-AHEAD" METHODS

Some small refinements can lead to substantial decreases in the number of branches of the tree of potential strategies which must be examined. Experience with a variety of problems has demonstrated dramatic improvements in running time and maximum list length from employment of these additional algorithms. The simplest of these which has been used in this study applies especially when budgets are loose, the situation mentioned in connection with the use of the complement algorithm.

At any stage i, define a strategy called "Rest" in which $x_j=1$ for $j=i,\ldots,n$, i.e., the projects remaining to be considered are included, and $x_j=0$ for $j=1,\ldots,i-1$, the projects already considered. At this stage, test to see whether it is possible to add the strategy "Rest" to a given strategy on the list of strategies so that the resulting strategy satisfies

the original constraints. Specifically, for the strategy represented by $f_{i-1}(C')$, if

$$(7) \qquad \sum_{j=i}^{n} c_{tj} \leq C'_t, \qquad t=1,\ldots,T$$

then the new strategy consisting of the given one plus all subsequent projects is optimal for the total amount of budgets consumed thereby. Hence this strategy at stage i needs not be considered at subsequent stages. Instead, the combined strategy can be put directly on the output list for final ranking of the $f_n(C')$ at the end, or otherwise used.

As can readily be seen, with loose budgets in which but few projects must be rejected, once some projects have been eliminated, the subsequent stages for the given strategy would lead to the inclusion of all remaining projects.[5]

The Look-ahead computations can also be adapted to the complement problem. However, instead of testing for the possibility of nonoptimality, the test is for the possibility of nonfeasibility. Let C_t be the constant of constraint t, as before, and define $\overline{C}_t$ and $\underline{C}_t$ as in (3) and (4) above, respectively. Then, if at stage i for a given strategy

$$(8) \qquad \sum_{j=i}^{n} c_{tj} < \underline{C}_t - \sum_{j=1}^{i-1} c_{tj} y_j, \qquad t=1,\ldots,T$$

then this strategy can never be part of a <u>feasible</u> one. Excluding all remaining projects will not sufficiently increase the amounts that must be eliminated to attain a solution within the original constraints. Hence, this strategy can be disregarded for further computation.

No doubt more complex Look-ahead methods can be devised. The simplicity of these, however, makes them advantageous for rapid calculations.

LOWER BOUND ELIMINATION METHOD

A device similar to the Look-ahead is the following. At stage i eliminate any strategy from further consideration which, if it were augmented by all projects remaining to be considered, would yield a payoff less than the highest payoff already achieved. That is, see if for a given strategy (or alternatively, for given C') at stage i

$$f_{i-1}(C') + \sum_{j=i}^{n} b_j \tag{9}$$

is less than the highest payoff of a feasible strategy already obtained. This method requires noting the best feasible strategy. It can be extremely useful, especially when combined with the Upper Bound method, to be described below. It does, however, have the limitations of driving toward the single best strategy without making the next best strategies available.

One method of obtaining a good feasible solution to use as a Lower Bound is to solve (1) as a linear programming problem, but ignoring the fractional projects which result.[6] This method is costly if general purpose LP programs must be used. It is quite evident, however, that the simple structure of the problem can lead to the linear programming solution speedily by other methods. The first of these is especially simple if there are only two constraints.

The following procedure[7] is based on the dual to (1) regarded as a linear programming problem, i.e., ignoring constraints (1d).[8] The projects

are ranked by the quantity

$$(10) \qquad \alpha \frac{c_{1i}}{b_i} + (1 - \alpha) \frac{c_{2i}}{b_i}$$

from the lowest to highest, for a given trial value[9] of α. The payoff as well as the outlays in each period are cumulated, and the solution desired is that which, while including only integral projects, comes closest to exhausting the allotted budgets without violating any of them. The process is repeated in principle for all possible values of $0 \leq \alpha \leq 1$.[10]

UPPER BOUNDS

The Lower Bound Elimination Method requires finding a feasible solution to begin the branch-paring process. The higher the lower bound, the more branches may be discarded, and the faster the computations will reach the optimum. It is often worthwhile to find an upper bound to the solution, for several purposes. First, if the computations become too long or require more high-speed memory than is available, it is useful to know how far the best feasible solution already obtained differs from an absolute maximum. Secondly, it is sometimes helpful to inject a trial lower bound into the computational procedure in the hope that it will yield a better value, in fact, the optimal one, more quickly than would the use of feasible solution lower bound. For this purpose one needs to know what the theoretical maximum is.

An obvious upper bound for these purposes is the linear programming solution <u>including</u> the payoff from fractional projects. It is clear that no integral solution can exceed this payoff since it imposes additional conditions on the problem. Again, the linear programming solution as

required here may be obtained by short-cut methods as outlined above. In the two constraint case for each trial value of α, after the projects have been ranked and the integral projects which fall within the constraints have been identified, the indices of the next two projects in the rankings are noted. A simple two-equation two-unknown problem is then solved to find the fractions of these projects which are to be accepted for the given value of α.

Let C_t' be the remaining amounts in the budgets after the integral projects have been taken out. Letting x_j and x_k be the fractions of the two projects sought, they are obtained by solving

$$c_{1j}x_j + c_{1k}x_k = C_1'$$

(11)

$$c_{2j}x_j + c_{2k}x_k = C_2'.$$

The value of the payoff from the fractional projects, i.e., $b_jx_j + b_kx_k$ is added to the payoff from the integral projects, and the total is noted for comparison with the payoff obtained from other values of α. Since the optimum value of α requires that the fractional projects satisfy (11) in the non-degenerate case[11] and they must also be the next in this ranking scheme, the procedure will locate the LP optimum. Although the optimal dual values are unique in the general case, there is generally a range of α over which this procedure yields the optimal solution.[12] Further, since this is a continuous problem, there are no "gaps" and finding the optimum is guaranteed.[13]

USE OF AN INTERACTIVE COMPUTER SYSTEM

Initially the use of an interactive facility such as M.I.T.'s Project MAC[14] was simply for aid in debugging and fast turn-around. Early results indicated, however, that the order in which projects were taken up affected the time and storage requirements dramatically. The need for finding good ordering methods suggested that a more fruitful approach would be to utilize the interactive feature to respond to the status of the computations. Making it simple for the user to interpose his heuristics on the solution process requires two components. First, the progress being made must be displayed conveniently, on request, and second the decisions made by the user must easily be implemented.

Perhaps most important is the additional requirement that the user be able to aid his judgement by obtaining side information about the alternatives from which he must choose before making his decisions. Ultimately, all of these components might be collapsed into a machine-learning procedure in which the computer takes over the tasks performed by the user.[15] In complex problems this need not be the best way of accomplishing the goal, and, in any case, it was not the method followed here.

PROJECT ORDERING

As will be detailed below, either during or preceding the computations, it may be important to change the order in which the dynamic programming algorithm processes additional projects. At different times, various alternative methods for ordering the list of remaining projects may be called on. The most important of these are the following.

1. Payoff. This ranks the projects in decreasing order of the payoff, b_i.

2. Consumption. This is in decreasing order of the sum of the project's outlays in each period, $\sum_{t=1}^{T} c_{ti}$.

3. Weighted Consumption (a). Here ranking is done by weighting the consumption by a project in a given budget by the ratio of the total consumption required for all projects remaining to be considered to the original budget ceiling, then summing over the budgets; i.e., by

$$(12) \qquad \sum_{t=1}^{T} c_{ti} \left[\frac{\sum_{j=i+1}^{n} c_{tj}}{C_t} \right],$$

taken in decreasing order.

4. Weighted Consumption (b). In this ordering the remaining projects are ranked after weighting the consumption for a project in each budget by a measure of the tightness of that budget ceiling at this stage. Specifically, projects are ranked in decreasing order of

$$(13) \qquad \sum_{t=1}^{T} \left[\max \left(\sum_{j=i+1}^{n} c_{tj} - C_t, 0 \right) \right] c_{ti}.$$

5. Consumption t. The ranking here is in decreasing order of consumption in the given budget t, c_{ti}.

6. Best. This is in decreasing order of payoff divided by Weighted Consumption (a),

$$(14) \qquad \frac{b_i}{\sum_{t=1}^{T} c_{ti}\left(\sum_{j=i+1}^{n} c_{tj}/C_t\right)}$$

7. Worst. This inverses the order of Best. It is used for purposes of comparison, and to eliminate this project from the Look-ahead solution.[16]

8. Inverse Payoff. This is the inverse order of payoff and is used with the Backward solution--the solution of the complement problem.

9. Inverse Weighted Consumption. This is the inverse order of Weighted Consumption (a) for use with the Backward solution method.

10. Projects may be selected by the following criterion. At any stage i it may be determined what the maximum value of any solution, feasible or infeasible, would be if a given project not yet considered were somehow unavailable. Denoting the set of project already considered by J, for any project k, $k \notin J$, this quantity is

$$\gamma_k \equiv \sum_{j \notin J} b_j - b_k.$$

Let ϕ be the best payoff already obtained, or the payoff corresponding to a Lower Bound or to a guess, as in an Upper Bound. Then, if

$$\gamma_k + f_{i-1}(C') < \phi$$

then the strategy corresponding to $f_{i-1}(C')$ would be deleted by Lower Bound Elmination from the list if project k were brought in at stage i. The number of potential deletions by this method may be obtained for any project not yet considred, and this number is reported for some or all of the remaining projects.

Although Payoff ranking will be the same as ranking by this method, the minimum number of deletions obtained in this way may be combined with estimates of the number of deletions produced through the application of other criteria. These deletions may be in addition to those resulting from dominance. The chief characteristic of such a selection device is that it requires only one pass through the list of feasible strategies at this stage. It differs from incorporating a project in that the latter requires dominance checking, which involves an order of magnitude more computation.

THE BASIC PROGRAM

It may be useful to outline the basic computing method[17] by contrasting it with the approach followed in [6] in which a detailed flow chart is given. In simplest terms, a project is considered alone and in combinations with other projects. In either case it must lead to a feasible

solution, i.e., one which satisfies the original constraints. In the program utilized before, a new project is first tested for feasibility. By maintaining the list of strategies (or combinations as they have been defined before) in increasing order of payoff, the new project is tentatively inserted into the list in its appropriate place. It is then compared with all preceding items to see whether one or more strategies are dominated by the new one, and next comparing it with items following it to see whether it, itself, is dominated. Dominance here means that either the payoff of one is higher than the other and the outlays in each budget are the same or less, or the payoff is the same but the outlays are less in at least one budget. If undominated the new item is added to the list in its appropriate place. Then it is added to each item in the previously existing list, and the dominance tests are repeated.

This program requires both a temporary and a permanent list of strategies at each stage. Further, as an item is inserted, all succeeding items of the list must be moved down one line; as an item is deleted, all succeeding items on the list must be moved up one line. These procedures require a substantial proportion of the time used by the program.

The chief departure from this earlier approach that was followed here pertains to the utilization of a more flexible storage scheme. Instead of ordering the list of strategies by payoff, items are appended to the list as they are generated. Each item now contains the address of the next item on the list. Deletion of an item is accomplished by changing the address contained in the preceeding item to that of the following one. Thus all references to the deleted item have been obliterated, and the storage space corresponding to it may be made available for later use.

In addition, the file of projects is kept up-to-date by flagging those projects which have already been considered.

Dominance testing may be thought of in terms of traversing the current list of strategies and the separate step of testing individual items on this list. Taking the latter first, it is necessary to decide whether the new item dominates, is dominated by, or neither dominates nor is dominated by, the item from the list. Let J be the set of projects included in the given strategy from the list, i.e., for which $x_j=1$. Such an item may then be represented by the vector

$$(15) \qquad \left(\sum_{j\in J} b_j,\ C_1 - \sum_{j\in J} c_{1j},\ C_2 - \sum c_{2j},\ \ldots,\ C_T - \sum_{j\in J} c_{Tj}\right).$$

A new item, which may be a single project the first time it is considered or a project being considered in combination with others, resembles (15). For present purposes, call the new item A and the item from the existing list of strategies B. Comparison of A and B proceeds term by term. When the first component of A exceeds the first term of B, if all subsequent components of A are <u>not</u> exceeded by the corresponding components of B, then A dominates B. If, on the other hand, at least one subsequent component of B exceeds the corresponding one from A, then neither dominates the other. As soon as this condition is discovered, comparisons are terminated. A is dominated by B when, in the above, the conclusions follow after interchanging the roles of B and A. Ties are similarly resolved lexicographically.

In traversing the list of feasible strategies at the given stage, if an item of the strategy list is dominated, it is marked for deletion. If the new item is dominated, consideration of it ceases and a new combination

is generated. If the new item is not dominated by any item on the list it is appended to the list, and consideration shifts to the creation of a new combination. For reasons of computational efficiency the first new item appended to the list at any stage is flagged. This flag is used to terminate creation of new combinations and dominance checking.[18]

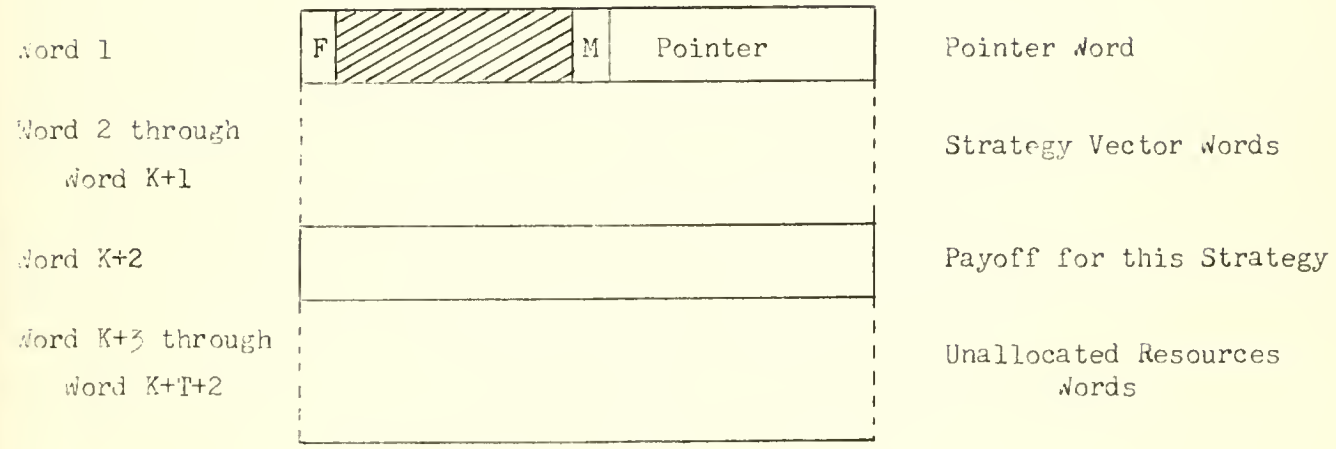
The programs for this study were written, for the most part, in assembly language. Further, in order to improve computational efficiency without detracting unnecessarily from flexibility, certain important variables[19] were made parameters of the assembly. By writing some recursive macro-definitions, it was thus possible to generate a program which would handle any number of projects and constraints, subject only to the limits of core storage. This program has very few "red-tape" instructions; e.g., it avoids loop-counting by making a string out of the most frequently executed loops.[20]

PROGRAM OPERATION

The different test problem are kept in "files" which are available on demand from a large volume disc store. When the program is started, it first requests the name of a block of data and a "direction" of approach, i.e., original (Forward) or Complement (Backward) as explained earlier. The program then reads in this data and puts each project into a block[21] of cells which are strung together to comprise the File of Projects. A null project, one with zero payoff and unallocated resources equal to the constraints, is placed on the List of Strategies. The program then turns control over to the user who may do any number of different things:

Figure 1

SCHEMATIC REPRESENTATION OF AN ELEMENT FROM THE LIST OF STRATEGIES



$K = [\frac{n+35}{36}]$, i.e., the integer part of $(n+35)/36$, for a 36 bit/word machine

n, T are as defined earlier

Pointer has the address of the next block of words on the list, or 0 if this is the last block on the list.

M is 1 if this block is to be deleted during the next deletion pass, 0 otherwise.

F is 1 if the block is "flagged" and 0 otherwise

The K words, no.s 2 through K+1, contain a bit-pattern representing this strategy vector.

Word K+2 contains the amount of the payoff for this strategy.

The T words, no.s K+3 through K+T+2, contain the amount of the budgets unallocated by this strategy in budgets 1,2,...,T, respectively.

1. "Guess" a solution--e.g., the Linear Programming Lower Bound solution--for use in Lower Bound Elimination.
2. Select a project from the File of Projects which best satisfies any one of a number of preprogrammed criteria and directly incorporate it, or obtain a print-out of the selected project's characteristics.
3. Define a new criterion, by programming it on-line, and apply it as in 2. above.
4. Directly incorporate any designated project which has not been incorporated previously into the List of Strategies according to the algorithm being used, i.e., Forward or Backward.
5. Copy the current list on a secondary, bulk-storage medium for later runs.
6. Restore the status of the lists from secondary storage.
7. Print out any portion of either the List of Strategies or the File of Projects.
8. Ask the program to make a "guess" about what will happen if a certain project is incorporated next (to be explained below).
9. Print out summary information about all of the projects not yet considered.

STATUS INFORMATION

After each project has been considered the program prints the project number, the time of day, the number of additions to the List of Strategies, the number of deletions, the new list length, the number of feasible combinations generated, the amount of computer time (in seconds) consumed during this pass, the total amount of time consumed so far in the run, the percentage change in list length, the value of the best solution to date, and an indication of how this best solution was obtained (by a "guess," by Look-ahead or directly). The program maintains a list of free space, i.e., memory available for blocks but at present not being used, and whenever this list is exhausted the program attempts to find space for 100 blocks at the top of the currently used portion of memory. As this is done it informsthe user by a message on his console.

This information along with the optional printout described above give a reasonable indication of the status at any given point. It is employed to help decide on a course of action.

LARGE PROBLEM COMPUTING STRATEGIES USED

During the course of this study a large problem was solved in several different ways. Some experience was also gained with a problem which has not, as yet, yielded to these methods. This experience has led to some heuristics which, while not claimed to be optimal, nevertheless seem to produce reasonable computational efficiency.

The usual procedure is as follows. First, the Linear Programming Lower Bound solution to the problem is obtained,[22] and this solution is

set as the minimum for Lower Bound Elimination. At first the projects are considered either by the Payoff or by the Weighted Consumption (a) ordering. In the early stages, until the Look-ahead produces a feasible solution, it appears that Payoff keeps the list length as small, on the average as does Weighted Consumption (a), while the other ordering methods seem to behave significantly worse. With tight constraints the Payoff ordering is superior to the others. If a feasible Look-ahead solution is produced, ordering is by Payoff from this point on.

Should the list length appear to "blow-up," the question is raised whether one may be able to attain a solution at all. It is possible to obtain some indication about the likelihood that this will take place by establishing an artificially high minimum as, for example, something near the sum of all project payoffs. This Upper Bound trims the list of strategies by Lower Bound Elimination very heavily, and unless one is lucky, it will not produce a feasible solution at all. Nevertheless, it yields some information about the magnitude of the problem at the cost of relatively little computation time, as follows. Another run is made next with a somewhat lower minimum, which while still not yielding a feasible solution, does give an indication of how the size of the list of strategies is increasing. Since it is known that a solution exists which is at least as good as the Linear Programming Lower Bound solution, some reasonable guesses about the amount of time which would be required to obtain the optimum is not too difficult.

On occasion when applying these procedures of estimating the size of the problem, certain projects seem to eliminate relatively large numbers of strategies from the list. When this occurs such projects are

designated for early direct incorporation in subsequent runs. Generally, however, the Payoff ordering method is utilized without variation.

THE COMPUTING ENVIRONMENT

Before presenting the experience gained with these programs some comments on the method of timing computer runs are in order. First, all references to time are to central processor time used. The time-sharing system within which this work was conducted makes a distinction between the time that a program is actively operated on for the production of the user's results, and the time spent moving that program into and out of the central memory. Central processor time bears little relation to the amount of time the user actually spends at his console. The latter, of course, depends not only on system load, which determines the speed with which requests from the user's console are executed, but also on the time the user takes to decide on a course of action in exploiting the interactive facility of the program.

Second, if a course of action were predetermined for a problem it would be possible to implement a batch processing version of these procedures. Such programs would obviate the need for some intermediate output and its generation, and further they would not require facility for gaining control of the computing process during a run. The resulting batch-processing times would be expected to improve on the results obtained here.

It is important to note that this research was carried on over the course of several months and on an experimental time-sharing system. During this time the computer environment was undergoing a continual

evolution.[23] While most of the system changes did not affect the operation of these programs, the timing figures presented are subject to some small (certainly less than 1 per cent) variation, since the time consumption characteristics of certain system subroutines were not invariant. It was observed, for example, that the time required to assemble the main program apparently increased from approximately 30 to 45 seconds. Since, however, by far the greatest portion of the timing figures given was actually consumed in programs directly under user control, it is clear that this necessary and important evolution in no way detracts from the results.

EXPERIMENTAL RESULTS

Two two-constraint problems were used to develop and test the program described above.[24] These problems, derived from actual applications, involved 28 and 105 projects, and are given in Tables 1 and 2, respectively. Each was run several times using different heuristics regarding order of consideration of the projects during the computations.

The results of the 28-project problem runs are summarized in Table 3. The first column of this table gives the budget ceilings, while the second presents the optimal payoff, the third column states the method of solution, Forward or Backward. The lower bound, if any, used in lower Bound Elimination is given in the fourth column. The fifth column specifies the maximum number of feasible strategies which was required during the entire run. Time of run is given in column six, and the final column marked "Comment" gives more qualitative information about the chief method for ordering project consideration.[25]

Table 1

PAYOFF AND OUTLAYS FOR 28-PROJECT PROBLEM

Project Number	Payoff	First Budget Consumption	Second Budget Consumption	Project Number	Payoff	First Budget Consumption	Second Budget Consumption
1	1898	45	30	15	479	0	10
2	440	0	20	16	440	25	12
3	22507	85	125	17	490	0	10
4	270	150	5	18	330	0	9
5	14148	65	80	19	110	25	0
6	3100	95	25	20	560	0	20
7	4650	30	35	21	24355	165	60
8	30800	0	73	22	2885	0	40
9	615	170	12	23	11748	85	50
10	4975	0	15	24	4550	0	36
11	1160	40	15	25	750	0	49
12	4225	25	40	26	3720	0	40
13	510	20	5	27	1950	0	19
14	11880	0	10	28	10500	100	150

Table 2

PAYOFF AND OUTLAYS FOR 105-PROJECT PROBLEM

Project Number	Payoff	First Budget Consumption	Second Budget Consumption	Project Number	Payoff	First Budget Consumption	Second Budget Consumption
1	41850	75	0	15	47910	0	10
2	38261	40	0	16	30250	12	10
3	23800	365	0	17	107200	0	50
4	21697	95	0	18	4235	10	2
5	7074	25	0	19	9835	0	5
6	5587	17	0	20	9262	50	5
7	5560	125	0	21	15000	0	10
8	5500	20	0	22	6399	0	5
9	3450	22	0	23	6155	10	6
10	2391	84	0	24	10874	0	11
11	761	75	0	25	37100	0	41
12	460	50	0	26	27040	50	30
13	367	15	0	27	4117	60	5
14	24785	0	5	28	32240	150	40

(continued)

Table 2 (continued)

PAYOFF AND OUTLAYS FOR 105-PROJECT PROBLEM

Project Number	Payoff	First Budget Consumption	Second Budget Consumption
29	1600	0	2
30	4500	0	6
31	70610	75	100
32	6570	0	10
33	15290	102	25
34	23840	0	39
35	16500	0	30
36	7010	40	13
37	16020	60	30
38	8000	0	15
39	31026	165	60
40	2568	0	5
41	2365	0	5
42	4350	0	10
43	1972	45	5
44	4975	0	15
45	29400	0	91
46	7471	0	24
47	2700	25	10
48	3840	0	15
49	22400	150	90
50	3575	0	15
51	13500	0	60
52	1125	0	5
53	11950	158	55
54	12753	0	60
55	10568	85	50
56	15600	95	75
57	20652	0	100
58	13150	89	65
59	2900	20	15
60	1790	0	10
61	4970	0	30
62	5770	0	35
63	8180	0	50
64	2450	0	15
65	7140	0	45
66	12470	80	80
67	6010	0	40
68	16000	110	110
69	11100	0	80
70	11093	15	80
71	4685	0	36
72	2590	60	20
73	11500	5	90
74	5820	135	50
75	2842	0	25
76	5000	0	50
77	3300	25	35
78	2800	0	30
79	5420	300	60
80	900	35	10
81	13300	100	150
82	8450	0	110
83	5300	0	70
84	750	25	10
85	1435	0	20
86	2100	0	30
87	7215	225	104
88	2605	25	40
89	2422	0	40
90	5500	0	94
91	8550	0	150
92	2700	0	50
93	540	0	10
94	2550	0	50
95	2450	0	50
96	725	5	16
97	445	0	10
98	700	60	20
99	1720	0	50
100	2675	100	90
101	220	0	10
102	300	0	15
103	405	0	39
104	150	0	20
105	70	0	20

Table 3

RESULTS OF COMPUTER RUNS WITH THE 28-PROJECT PROBLEM

Budget Ceilings	Optimal Payoff	Solution Method	Payoff in Lower Bound Elimination Method	Maximum Length of List of Strategies	Time# (sec.s)	Comment
$600, $600	$141,278	Forward	0	2443	731.5	Without Special devices. See Text.
$600, $600	$141,278	Forward	139508*	869	213.7	Worst criterion.
$600, $600	$141,278	Forward	139508*	447	36.3	Natural order.
$600, $600	$141,278	Forward	139508*	40	2.4	Weighted Consumption (a). criterion.
$600, $600	$141,278	Forward	139508*	25	1.8	Payoff criterion.
$600, $600	$141,278	Forward	0	66	3.5	Weighted Consumption (a) criterion.
$600, $600	$141,278	Forward	141277+	21	1.8	Payoff.
$600, $600	$141,278	Backward	0	112	5.3	Weighted Consumption (a) criterion.
$600, $600	$141,278	Backward	0	266	19.9	Reverse payoff.
$600, $600	$141,278	Backward	139508*	1153	174.8	Reverse Weighted Consumption criterion.
$500, $500	$130,883	Forward	0	24	2.1	Payoff
$500, $500	$130,883	Forward	129723*	20	1.6	Payoff.
$300, $300	$ 95,677	Forward	0	27	2.4	Mixed.
$300, $600	$119,337	Forward	0	32	2.2	Payoff.
$600, $300	$98,796	Forward	0	26	2.1	Payoff.
$562, $497	$130,623	Forward	0	24	1.8	Payoff.

Optimum set with ceilings of $600, $600 includes 14 projects: Nos. 3, 5, 6, 7, 8, 10, 12, 13, 14, 19, 21, 23, 24, 26.

#Although times are reported to tenths of seconds, they must be regarded as accurate only to ±.5 seconds.

*Payoff of the Linear Programming Lower Bound solution.

+One less than the optimum payoff.

An early run of this problem, without user intercession during the computations and before the Lower Bound Elimination Method and the Look-ahead were implemented, required 731 seconds.[26]

The fastest run, to date, required only 1.8 seconds, of which at least one second may be attributed to "overhead."[27] It is particularly interesting to note that even when using the Linear Programming Lower Bound solution, Lower Bound Elimination and Look-ahead, one run took 214 seconds. This provides an indication of the effect of order of consideration.

The results of runs made with the larger problem are summarized in Table 4. This problem was large enough so that it was necessary to stop runs which would have required vast amounts of computer time. The best run one could reasonably expect for this problem took 31 seconds. Using a different ordering the same problem required 317 seconds. Several runs which would have required much more time were abandoned after 100 to 200 seconds had been consumed.

SOME OBSERVATIONS ON THE COMPUTATIONS

The program presented here has solved a large problem. It should be noted, however, that the constraints were either "loose"[28] or "tight." Neither of these circumstances presented any difficulties. When the constraints are reasonably tight a large number of strategies are infeasible, and the list length is kept within manageable limits. When the constraints are loose the Look-ahead and Lower Bound Elimination Methods eliminate a substantial number of feasible strategies. In an intermediate range, as, for example, when both budget ceilings were set at $2,000, the lists produced grew sufficiently long to have necessitated terminating the

Table 4

RESULTS OF COMPUTER RUNS WITH THE 105-PROJECT PROBLEM

Budget Ceilings	Optimal Payoff	Solution Method	Payoff in Lower Bound Elimination Method	Maximum Length of List of Strategies	Time# (sec.s)	Comment
$3,000, $3,000	$1,095,444	Forward	1094591*	644	316.7	Weighted Consumption (a) criterion.
$3,000, $3,000	$1,095,444	Forward	1094591*	167	30.9	Payoff criterion.
$3,000, $3,000	$1,095,444	Forward	0	752	240.0	Payoff criterion.
$3,000, $3,000	$1,095,444	Backward	0	1054	392.7	Payoff and other criteria.
$3,000, $3,000	$1,095,444	Forward	0	379	149.4	Mixed criteria.
$3,000, $3,000	$1,095,444	Forward	1095444+	157	23.7	Payoff criterion.
$ 500, $ 500	$ 624,319	Forward	605477 ++	405	120.2	Mixed Payoff and Weighted Consumption (a).

The optimal set of projects with ceilings of $3,000 and $3,000 excludes Nos. 36, 72, 79, 80, 84, 87, 93-105.

#Although times are reported to tenths of seconds, they must be regarded as accurate only to ±.5 seconds.

*Payoff of the Linear Programming Lower Bound solution.

+One less than the optimum payoff.

++Payoff of a feasible solution from an aborted run.

run. Evidently, more powerful devices are still needed to bring the computing time within the range of experience with loose or tight budgets.

Consideration has been given to the possibility of allowing the user to bring a project in and to back-up and bring in a different project in its place if desired. Since this facility was provided easily by the time sharing system being used, it was not directly incorporated into the program. It is possible that such a technique would be quite helpful in certain situations, but not enough experience with it was obtained to charactertize the circumstances under which it would be appropriate.

The results obtained, especially with respect to the time for computing large problems, are not necessarily the best obtainable. They depend not only on the efficiency of the computer programs themselves, but more directly on the heuristics employed. These were limited by the insights the authors were able to generate. An important inference seems to be warranted. This is that in the computation of large combinatorial problems, the separation of algorithm construction and computer programming may be considerably less efficient than bringing the applied mathematician into a close partnership with the computer. The combination may obviate the necessity for devoting substantial effort to the solution of the meta-problem--the rules for applying the algorithms. The mathematician may be able to develop insights into the nature of the process and to respond to characteristics of the particular numerical problem at hand without specifying ad hoc how he wishes the computations to react to such variations.

FOOTNOTES

1. Alternative formulations, allowing for negative c_{ti} as implied by the generation of funds or other resources, are most appropriate to a different class of models. These have been treated in [7] Chapters 8 and 9 and will not be taken up here.

2. This verbal description is not entirely accurate, leaving out "degenerate" cases. Strictly, one strategy dominates another if the payoffs are identical and so are the outlays in all but one period, in which the dominated one requires more, or the outlays are all identical and the payoff of the dominated one is lower. See also below.

3. It should be pointed out here that Bellman's method [1] employing a LaGrange multiplier for one or more constraints cannot be relied upon to produce the integer solution sought here, and will not be taken up further. See also [4] and [6].

4. The maximum number of strategies that might have to be considered is, of course, 2^n, where n is the number of projects. For n = 25, this is approximately 3,356,000.

5. The examples discussed in the final sections will help to illustrate these remarks.

6. The maximal number of fractional projects is given by T, the number of constraints, as simple extreme point arguments will prove. See [7], Sec. 3.8.

7. This was first suggested by W. L. Eastman.

8. See [7], Sec. 3.5.

9. If the dual variables corresponding to the two budget constraints (1b) are denoted by ρ_1 and ρ_2, then $\alpha = \dfrac{\rho_1}{\rho_1 + \rho_2}$.

10. With the availability of an interactive computer system, it is possible to remain flexible, i.e., not to fix a search procedure.

11. Degeneracy in the primal solution here means that either x_j or x_k or both are zero in (11). Degeneracy in the dual means that either $\alpha = 0$, or 1, i.e., not all budgeted quantities are utilized in the optimal solution. Neither of these introduces any difficulties.

12. It is important to note that the value of α which yields the LP Upper Bound is not necessarily the same as that which produces the LP Lower Bound, above, which is the best solution ignoring fractional projects.

13. See [4].

14. At the time of writing, this consists of an IBM 7094 Computer set up for time-shared use on remote Teletype and IBM 1050 consoles.

15. See, for example, machine learning in programs for playing checkers, [5].

16. That is, this Worst project is removed from the list of projects still to be considered, and its is probably not combined with strategies on the list because such combinations are dominated.

17. The basic program discussed here omits reference to Look-ahead methods and other devices.

18. That dominance testing may be suspended at this point follows from the theorem that any new strategy cannot dominate or be dominated by another new strategy.

19. The number of constraints, T, and the maximum number of projects, n.

20. As in the earlier program in [6], the strategy vectors were condensed into binary strings. See Figure 1.

21. A block is a set of contiguous memory locations as defined in Figure 1.

22. This is done by a program separate from the one described above.

23. The major effect of this evolution on the research reported here was the frequent change in system software.

24. This is in addition to a 25-project problem of Cord [2] discussed in detail in [6]. The computing time reported there, using the program referred to earlier, was 63 seconds. Using the same computer system, this time was reduced to 1.6 seconds here.

25. The terms used there are defined in the section on Project Ordering.

26. This problem never yielded to the program reported in [6].

27. I.e., to a virtually irreducible set of housekeeping instructions.

28. The ceilings of $3,000 in each budget came from the actual application.

REFERENCES

[1] Bellman, R., "Comment on Dantzig's Paper on Discrete-Variable Extremum Problems," Operations Research, October 1957, pp. 723-24.

[2] Cord, J., "A Method for Allocating Funds to Investment Projects when Returns are Subject to Uncertainty," Management Science, January 1964, pp. 335-41.

[3] Dantzig, G., "Discrete-Variable Extremum Problems," Operations Research April 1957, pp. 266-77.

[4] Everett, H., III, "Generalized LaGrange Multiplier Method for Solving Problems of Optimum Allocation of Resources," Operations Research, May-June 1963, pp. 399-417.

[5] Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers," IBM Journal of Research and Development, III, No. 3 (July 1959), pp. 210-229.

[6] H. M. Weingartner, "Capital Budgeting of Interrelated Projects: Survey and Synthesis," Management Science, XII, No. 2 (March 1966), pp. 485-516.

[7] _____, Mathematical Programming and the Analysis of Capital Budgeting Problems, Englewood Cliffs: Prentice-Hall, Inc., 1963.

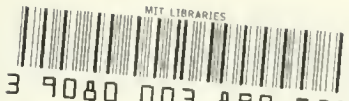
AUG 2 1 1968

Date Due

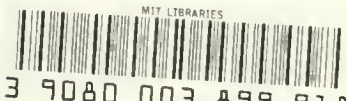
MAR 09 '76
APR 28 '76
NOV 16
APR 17 1985
MAR 19 1992
APR 05

4/11

Lib-26-67



173-66



175-66

176-66

177-66

178-66

180-66

181-66

182-66

183-66

184-66

185-66

186-66