Reprinted from OPERATIONS RESEARCH
Vol. 14, No. 6, November-December, 1966
Printed in U.S.A.

THE THEORY AND COMPUTATION OF KNAPSACK FUNCTIONS†

P. C. Gilmore and R. E. Gomory

International Business Machines Corporation, Yorktown Heights, New York

(Received March 21, 1966)

In earlier papers on the cutting stock problem we indicated the desirability of developing fast methods for computing knapsack functions. A one-dimensional knapsack function is defined by:

$$f(x) = \max\{\Pi_1 Z_1 + \cdots + \Pi_m Z_m;\ l_1 Z_1 + \cdots + l_m Z_m \leqq x,\ Z_i \geqq 0,\ Z_i \text{ integer}\}$$

where Π_i and l_i are given constants, $i=1, \cdots, m$. Two-dimensional knapsack functions can also be defined. In this paper we give a characterization of knapsack functions and then use the characterization to develop more efficient methods of computation. For one-dimensional knapsack functions we describe certain periodic properties and give computational results.

KNAPSACK problems of the most general type can arise directly in two ways. If a portion of space is being packed with objects, each having a value, the knapsack problem is to find the most valuable packing. Alternatively and equivalently if a portion of space is being cut into pieces of different values, the knapsack problem is to find the most valuable way of cutting.

An example of the first viewpoint is the problem of loading a ship or box car with the most valuable cargo as discussed, for example, in reference 1; an example of the second kind would be the cutting of a sheet of glass into the most valuable assortment of pieces for windows, windshields, etc.

In addition to arising in these direct ways, knapsack problems also arise as column generating subproblems in very large linear programs. In reference 3 we showed that much of the difficulty of the one-dimensional cutting stock problem could be overcome by solving a one-dimensional knapsack subproblem.

The one-dimensional knapsack problem is this: given positive lengths $l_1, \cdots, l_m$, and corresponding nonnegative values $\Pi_1, \cdots, \Pi_m$, find for a given x the value $F(x)$, where $F(x)$ is the maximum of $\Pi_1 Z_1 + \cdots \Pi_m Z_m$ with $Z_1, \cdots, Z_m$ nonnegative integers satisfying $l_1 Z_1 + \cdots l_m Z_m \leqq x$. Intuitively one is fitting lengths l_i into a box of length x. The function $F(x)$ is called the *knapsack function.*

† This research was supported in part by the Office of Naval Research under Contract No. Nonr 3775(00).

In reference 5 we showed that higher dimensional cutting stock problems could be solved by linear programming if certain higher dimensional knapsack functions could be calculated for specified values of their arguments. We also discussed in some cases methods of calculation. For a general two-dimensional cutting stock problem for example, a two-dimensional knapsack function G is defined as follows: One is given rectangles of positive dimensions (l_i, w_i), $i=1, \cdots, m$ that have nonnegative values $\Pi_1, \cdots, \Pi_m$ associated with them; then $G(x, y)$ is the maximum of $\Pi_1 Z_1 + \cdots + \Pi_m Z_m$, where $Z_1, \cdots, Z_m$ are nonnegative integers such that

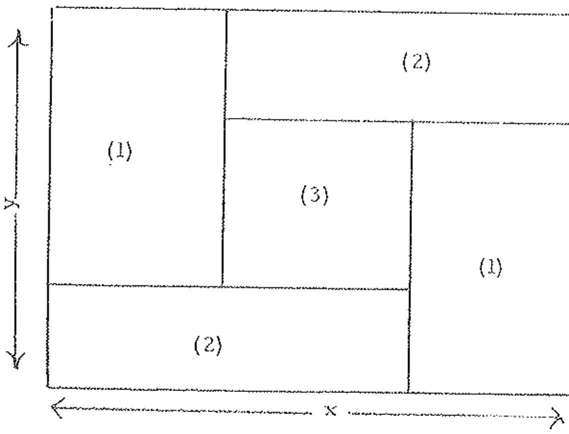

Figure 1

there exists a way of dividing a rectangle (x, y) into Z_i rectangles (l_i, w_i), for $i=1, \cdots, m$. For example, in Fig. 1 a permitted division of the rectangle (x, y) into two rectangles marked (1), two rectangles marked (2), and one rectangle marked (3) is illustrated. In that case, $G(x, y)$ is known to be at least $2\Pi_1+2\Pi_2+\Pi_3$, if Π_i is the value of a rectangle marked (i). The calculation of $G(x, y)$ for given x and y is not an easy task. Fortunately, for many practical cutting stock problems, the calculation of another knapsack function F suffices. F is defined like G except that in dividing a rectangle (x, y) into Z_i rectangles (l_i, w_i) for $i=1, \cdots, m$, the following restriction is imposed: The division must take place by a series of straight lines that extend from one edge of a rectangle to an opposite edge, parallel to the other two edges; we will call them 'guillotine cuts.' Here by 'a rectangle' is meant either the original rectangle (x, y) or a rectangle obtained from it by one or more guillotine cuts. In Fig. 2 a permitted division of the rectangle (x, y) in the calculation of $F(x, y)$ is illustrated. In that case, $F(x, y)$ is known to be at least $\Pi_1+\Pi_2+2\Pi_3+2\Pi_4$.

Because of the way F has been defined in terms of guillotine cuts it satisfies two fundamental divide-in-two inequalities: $F(x_1+x_2, y) \geqq F(x_1, y)+F(x_2, y)$ and $F(x, y_1+y_2) \geqq F(x, y_1)+F(x, y_2)$. The one-dimensional knapsack function satisfies a divide-in-two inequality also. In fact, the one dimensional knapsack function can be regarded as a special case of the two dimensional function. For, given l_i, $i=1, \cdots, m$, with associated values Π_i, let $w_i=1$; then $F(x)=F(x, 1)$.

In this paper, we wish to study in greater depth than we did in references 4 or 5 the theory and computation of knapsack functions that satisfy the basic divide-in-two inequalities. Only briefly in Sections 3 and 5 will we

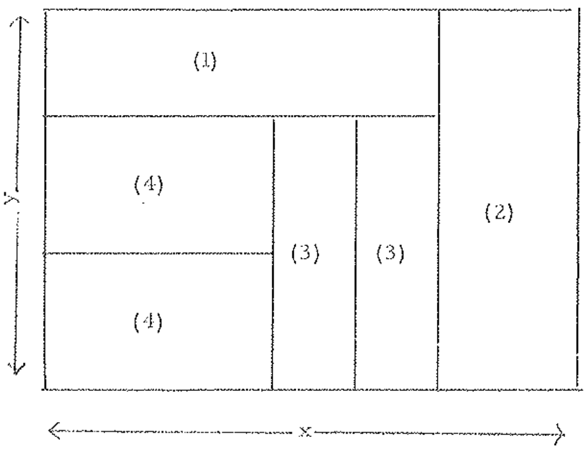

Figure 2

study functions that do not satisfy these inequalities, and only in the case of some one-dimensional knapsack functions having special practical importance. We restrict our attention to knapsack functions of two dimensions and less because the theory and computation of higher dimensional knapsack functions is sufficiently well illustrated in two dimensions.

In Section 1 we look at two-dimensional knapsack functions more abstractly than we do elsewhere in order to lay foundations for the remainder of the paper. In that section one-dimensional knapsack functions are to be regarded as defined by $F(x, 1)$ as just indicated. In fact, almost the entire paper could have been written about two-dimensional knapsack functions with one-dimensional knapsack functions regarded as special cases. However, the gain in elegance would result in a loss in intelligibility. Instead, therefore, with the exception of Section 1 we will treat one-dimensional knapsack functions separately from two-dimensional knapsack functions using the results concerning the former to provide introductory insights into the study of the latter.

In Section 2 we take up the study of one-dimensional knapsack functions to provide a basis for their computation as described in Section 4 as well as to prove some theorems interesting for themselves. For example, we prove that knapsack functions have periodic properties that can be exploited for their computation. In Section 3, as said earlier, some one-dimensional knapsack functions not satisfying the basic divide-in-two inequality are studied, and in Section 5 methods of computation for them are discussed.

In Section 6 a simple method of computing two-dimensional knapsack functions is described in detail to provide a basis for the description in Section 7 of a more efficient method. Finally in Section 8 we discuss two-dimensional knapsack functions arising from what we called in reference 5 staged guillotine cutting. Sections 6 and 7 provide the foundations for this discussion.

Throughout the paper, the lengths l_i and w_i, $i=1, \cdots, m$ are regarded as integer and the variables x and y are regarded as integer valued. In practical problems the lengths l_i and w_i are rarely integer but they are always rational. Consequently, it is always possible to convert them into integers by choosing $\delta>0$ and replacing l_i by $[l_i/\delta]$ and w_i by $[w_i/\delta]$. Here as elsewhere in the paper $[x]$ denotes the greatest integer not exceeding x.

1. THEORETICAL FOUNDATIONS

IN THIS section we wish to look at multidimensional knapsack functions from a more abstract view than we do elsewhere in the paper to provide a foundation for the main part of the paper. Because all of the difficulties of higher dimensional knapsack functions already occur for two-dimensional functions, we restrict our attention to those functions. However, we do regard one-dimensional functions as a special case for the theory developed in this section.

From the discussion in the introduction, it is clear that when the rectangles (l_i, w_i) of worths Π_i, $i=1, \cdots, m$, are given, the knapsack function $F(x, y)$ defined from them satisfies the following three sets of inequalities:

$$F(x, y) \geqq 0, \tag{1}$$

$$\begin{aligned} F(x_1+x_2, y) &\geqq F(x_1, y)+F(x_2, y), \\ F(x, y_1+y_2) &\geqq F(x, y_1)+F(x, y_2), \end{aligned} \tag{2}$$

$$F(l_i, w_i) \geqq \Pi_i. \qquad (i=1, \cdots, m) \tag{3}$$

Inequalities (1) and (3) speak for themselves. The inequalities (2) are a consequence of the permitted method of cutting a large rectangle (x, y) into the smaller rectangles (l_i, w_i). F is not the only function to satisfy these inequalities, although it is the minimal function in the sense of the following theorem.

THEOREM 1. *F is a knapsack function defined from the rectangles (l_i, w_i) with values Π_i, $i=1, \cdots, m$, if and only if F satisfies* (1), (2), (3), *and* (4): *For any G satisfying* (1) *to* (3), *$F(x, y) \leqq G(x, y)$ for all x and y.*

The value of the knapsack function F for arguments x and y is the maximum value that one can obtain from a rectangle (x, y) by dividing it into the smaller rectangles (l_i, w_i) of values Π_i by a series of guillotine cuts. F satisfies (1) to (3). There remains therefore to prove that F satisfies (4) also. This proof will be carried out by induction on the total number k of cuts needed to achieve the value $F(x, y)$ from the rectangle (x, y).

If $F(x, y)$ is achieved by no cutting, that is $k=0$, then necessarily either $x=l_i$, $y=w_i$, and $F(x, y)=\Pi_i$ for some i, or $x=y=0$. In the former case since G satisfies (3), $G(x, y) \geqq F(x, y)$. In the latter case $F(0, 0)=0$ because of (1) and (2) and therefore $G(0, 0) \geqq F(0, 0)$ by (1).

Assume therefore that (4) is true for those x and y for which $F(x, y)$ is achieved with k or fewer cuts. We will prove it true also for those achieved with $k+1$ cuts. Let $F(x, y)=Z_1 \Pi_1+\cdots+Z_m \Pi_m$, where Z_i is the number of rectangles (l_i, w_i) used to achieve $F(x, y)$. At least one of the $k+1$ cuts divides (x, y) into two rectangles and without loss in generality we can assume that the cut produces two rectangles (x_1, y) and (x_2, y), where $x_1+x_2=x$. Within the rectangle (x_1, y) we will assume that for each i there are contained Z_i^1 of the ith rectangle and within (x_2, y), Z_i^2 of the ith rectangle, where $Z_i=Z_i^1+Z_i^2$. By (2), $F(x_1+x_2, y) \geqq F(x_1, y)+F(x_2, y)$ and by the definition of F, $F(x_1, y) \geqq Z_1^1 \Pi_1+\cdots+Z_m^1 \Pi_m$ and $F(x_2, y) \geqq Z_1^2 \Pi_1+\cdots+Z_m^2 \Pi_m$. But $F(x_1+x_2, y)=Z_1 \Pi_1+\cdots+Z_m \Pi_m$ so that necessarily $F(x_1, y)=Z_1^1 \Pi_1+\cdots+Z_m^1 \Pi_m$ and $F(x_2, y)=Z_1^2 \Pi_1+\cdots+Z_m^2 \Pi_m$. By the induction assumption $G(x_1, y) \geqq F(x_1, y)$ and $G(x_2, y) \geqq F(x_2, y)$ for any G satisfying (1) to (3). But since such a G satisfies (2) it follows that $G(x_1+x_2, y) \geqq G(x_1, y)+G(x_2, y) \geqq F(x_1, y)+F(x_2, y)=F(x_1+x_2, y)$ and therefore $G(x, y) \geqq F(x, y)$ since $x_1+x_2=x$.

Since a minimal function F [in the sense of (4)] satisfying (1) to (3) is necessarily unique, we have established Theorem 1.†

The characterization of a knapsack function given in Theorem 1 is a

† Theorem 1 can be regarded as a statement about the dependence on their right-hand sides x of solution values $F(x)$ to 1-inequality integer programs. The theorem generalizes without any difficulty to the m-inequality case. It is only necessary to change the induction from the sum of cuts to the sum of the integer variables.

characterization of the function for given rectangles (l_i, w_i) of values Π_i. More abstractly, however, we can ask: given any function G defined for all x and y, $x, y \geqq 0$, how can it be recognized as a knapsack function, and if it is a knapsack function, then what are the rectangles (l_i, w_i) and values Π_i defining it? To prepare for the theorem answering these questions it is first necessary to give a definition.

A point (x, y) will be said to be a source point for a function G if and only if there exist no x_1 and x_2, $x_1>0$, $x_2>0$ and $x=x_1+x_2$, such that $G(x, y)=G(x_1, y)+G(x_2, y)$, and there exist no y_1 and y_2, $y_1>0$, $y_2>0$, and $y=y_1+y_2$, such that $G(x, y)=G(x, y_1)+G(x, y_2)$.

THEOREM 2. *A necessary and sufficient condition for a function F to be a knapsack function is that it satisfy* (1) *and* (2). *If (l_1, w_1), (l_2, w_2), $\cdots$, (l_m, w_m) are all the source points (x, y) for which $x \leqq L$ and $y \leqq W$ then F is the knapsack function defined from rectangles (l_i, w_i) of values $F(l_i, w_i)$, $i=1, \cdots, m$, in the range $0 \leqq x \leqq L$ and $0 \leqq y \leqq W$.*

That conditions (1) and (2) are necessary follows directly from Theorem 1. Now let F satisfy (1) and (2). Then F certainly satifies (3) also when (l_i, w_i), $i=1, \cdots, m$, are the source points of F and $\Pi_i=F(l_i, w_i)$. To show that F is a knapsack function defined from rectangles (l_i, w_i) of values $F(l_i, w_i)$ it is therefore sufficient by Theorem 1 to prove that if G satisfies (1) to (3) then $F(x, y) \leqq G(x, y)$ for all x and y, $0 \leqq x \leqq L$, $0 \leqq y \leqq W$. But let G be any function satisfying (1) to (3) and assume $G(x, y)<F(x, y)$ for some x and y. Let (x_0, y_0) be such that $G(x_0, y_0)<F(x_0, y_0)$, while for every x, $x<x_0$, $G(x, y_0) \geqq F(x, y_0)$ and every y, $y<y_0$, $G(x_0, y) \geqq F(x_0, y)$. (x_0, y_0) cannot be a source point of F since then G could not satisfy (3). Therefore either there exists x_1 and x_2, $x_1>0$, $x_2>0$ and $x_0=x_1+x_2$ such that $F(x_0, y_0)=F(x_1, y_0)+F(x_2, y_0)$ or there exists y_1 and y_2, $y_1>0$, $y_2>0$ and $y_0=y_1+y_2$ such that $F(x_0, y_0)=F(x_0, y_1)+F(x_0, y_2)$. Without loss of generality we can assume the latter. From the definition of x_0 and y_0 it follows that $F(x_0, y_1) \leqq G(x_0, y_1)$ and $F(x_0, y_2) \leqq G(x_0, y_2)$ while $G(x_0, y_0)<F(x_0, y_0)$. But then $G(x_0, y_0)<F(x_0, y_0)=F(x_0, y_1)+F(x_0, y_2) \leqq G(x_0, y_1)+G(x_0, y_2)$ contradicting (2).

One remark is relevant about our definition of knapsack function and the necessary and sufficient conditions of this theorem. The number m of given lengths l_i in the one-dimensional case or of given rectangles (l_i, w_i) has been assumed to be finite. When a knapsack function is defined over an infinite range this limitation is no longer necessary or desirable. The statement of Theorem 2 makes this clear: the number m is a function of the bounds L and W and m could increase without bound as L and/or W increased without bound. However for the sake of convenience we will continue to regard m as always finite.

Theorem 1 does have as an interesting consequence that the problem of computing the knapsack function F, for given (l_i, w_i) and Π_i and for a finite range of its argument is a linear programming problem. For let $X(x, y)$ denote a variable for each (x, y) in the range for which F is to be computed. Since by Theorem 1 necessarily $F(x, y) \leqq X(x, y)$ when $X(x, y)$ satisfies (1) to (3), it follows that $\sum_x \sum_y F(x, y) \leqq \sum_x \sum_y X(x, y)$. F is therefore the solution to the linear programming problem of minimizing $\sum_x \sum_y X(x, y)$ subject to the inequalities obtained by substituting X for F in (1) to (3). This is an interesting conclusion considering the origin of our interests in knapsack functions.

An efficient method for computing F is by a modified dynamic programming technique that is based upon the functional equation:

$$F(x, y) = \max\{F_0(x, y), F(x_1, y) + F(x_2, y), F(x, y_1) + F(x, y_2); \\ x \geqq x_1 + x_2, \quad 0 < x_1 \leqq x_2, y \geqq y_1 + y_2, \text{ and } 0 < y_1 \leqq y_2\} \tag{5}$$

where $$F_0(x, y) = \max\{0, \Pi_j; l_j \leqq x \text{ and } w_j \leqq y\}. \tag{6}$$

To justify such a method we want therefore to prove Theorem 3.

THEOREM 3. *The functional equation (5) is satisfied only by the knapsack function.*

That there is a function satisfying (5) can be quickly established by induction on x and y. That a function F satisfying (5) satisfies (1) and (2) can also be quickly verified. Applying Theorem 2 we know therefore that any function F satisfying (5) is a knapsack function on the rectangles defined by its source points in any finite range of its arguments. Clearly any source point (x_0, y_0) of F is a point (l_i, w_i) since for such a point $F(x_0, y_0) = F_0(x_0, y_0)$. But not all points (l_i, w_i) are necessarily source points. By Theorem 1, F is then the minimum function satisfying (1), (2), and (3) for those i for which (l_i, w_i) is a source point. But if (l_i, w_i) is not a source point $F(l_i, w_i) \geqq \Pi_i$ and therefore clearly F satisfies (3) for all i and it remains minimum with respect to the enlarged set of inequalities.

2. THEORETICAL DISCUSSION OF ONE-DIMENSIONAL KNAPSACK FUNCTIONS

IN THIS SECTION we will devote our attention to one-dimensional knapsack functions only; that is knapsack functions $F(x)$ defined from lengths l_1, $\cdots$, l_m of given values Π_1, $\cdots$, Π_m by the equation:

$$F(x) = \max\{Z_1 \Pi_1 + \cdots + Z_m \Pi_m; Z_i \text{ nonnegative integers}, \\ \text{and } Z_1 l_1 + \cdots + Z_m l_m \leqq x\}. \tag{7}$$

Clearly this function satisfies the one-dimensional form of (5), namely:

$$\begin{aligned} F_0(x) &= \max\{0,\ \Pi_i;\ l_i \leqq x\}, \\ F(x) &= \max\{F_0(x),\ F(x_1)+F(x_2);\ x \geqq x_1+x_2; \\ &\qquad\qquad 0<x_1\leqq x_2\}. \end{aligned} \tag{8}$$

Therefore by Theorem 3 (7) can be taken as another definition of the one-dimensional knapsack function and we will do so for the remainder of the paper. However we will return to (8) in Section 4 where algorithms for computing F are described.

The purpose of this section is to provide motivation and justification for the knapsack algorithms described in Section 4. To that end it will be necessary not only to know the value $F(x)$ achieved for a particular length x but also *how* that value is achieved. It would appear to be enough to know the number Z_i of times the length l_i is used to achieve $F(x)$ for $i=1, \cdots, m$; that is nonnegative integers Z_i for which $F(x)=\sum_{i=1}^{i=m} Z_i\,\Pi_i$ where $x\geqq \sum_{i=1}^{i=m} Z_i\,l_i$. But more detail is useful.

Actually a specification of how to cut a length x is what we will call a partition of x: a sequence of numbers $x_1, x_2, \cdots, x_r$, $0<x_1<x_2<\cdots<x_r=x$, where each x_j indicates a cut to be made. We will describe a partition by specifying a fixed ordering $i_1, i_2, \cdots, i_m$ of the indices $1, 2, \cdots, m$; then if Z_i pieces of length l_i are to be cut for $i=1, \cdots, m$, the sequence of x_j's is defined: $l_{i_1}, 2l_{i_1}, \cdots, Z_{i_1}\,l_{i_1}, Z_{i_1}\,l_{i_1}+l_{i_2}, Z_{i_1}\,l_{i_1}+2l_{i_2}, \cdots, Z_{i_1}\,l_{i_1}+Z_{i_2}\,l_{i_2}, \cdots, Z_{i_1}l_{i_1}+Z_{i_2}l_{i_2}+\cdots+Z_{i_m}\,l_{i_m}$. However the last number in that sequence is not necessarily x. But we can without loss add a length $l_{m+1}=1$ for which $\Pi_{m+1}=0$, since such a length simply permits a utilization of 'waste' at zero value, and then have for some Z_{m+1}, $x=Z_{i_1}\,l_{i_1}+\cdots+Z_{i_m}\,l_{i_m}+Z_{m+1}\,l_{m+1}$. Consequently a partition of x is defined by $Z_1, Z_2, \cdots, Z_{m+1}$.

We will assume that an ordering $i_1, \cdots, i_m$ has been chosen and that the lengths $l_1, \cdots, l_m$ have been reindexed so that $l_1, l_2, \cdots, l_m$ is the desired ordering of the lengths. In addition we assume that $l_{m+1}=1$ and $\Pi_{m+1}=0$.

A consequence of our assumptions is that for any given x, $x\geqq 0$, there exists one or more partitions $Z_1, Z_2, \cdots, Z_{m+1}$ realizing $F(x)$; that is such that $F(x)=Z_1\,\Pi_1+\cdots+Z_{m+1}\,\Pi_{m+1}$. We would like to associate with each x a unique partition realizing $F(x)$. To do this we choose from among those partitions realizing $F(x)$ the reverse lexicographically largest one. Precisely we choose that partition $Z_1^*, \cdots, Z_{m+1}^*$ realizing $F(x)$ such that if $Z_1, \cdots, Z_{m+1}$ is any partition realizing $F(x)$ and $Z_{m+1}=Z_{m+1}^*, \cdots, Z_{r+1}=Z_{r+1}^*$, then $Z_r\leqq Z_r^*$. We can now then define an integer valued function $l(x)$ as follows: For $x\leqq 0$ define $l(x)=1$. For $x>0$ a unique partition $Z_1, \cdots, Z_{m+1}$ is associated with x for which $Z_i\neq 0$ for some i.

Define then $l(x)=\max\{i; Z_i\neq 0\}$. Alternatively $l(x)$ can be defined directly in terms of F as follows:

$$l(x)=\max\{1, i; F(x)=F(x-l_i)+\Pi_i\}. \tag{9}$$

With this introduction we can prove at once:

THEOREM 4. *The knapsack function $F(x)$ is defined by the equation:*

$$F(x)=\max\{0, F(x-l_i)+\Pi_i; i\geqq l(x-l_i), x\geqq l_i\}, \tag{10}$$

where l is defined by (9).

Let $Z_1, \cdots, Z_{m+1}$ be the partition associated with x and let $r=l(x)$ so that $F(x)=Z_1\Pi_1+\cdots+Z_r\Pi_r$ and $Z_r>0$. Then $Z_1, \cdots, (Z_r-1)$ is a partition realizing $F(x-l_r)$. For by (2) $F(x)\geqq F(x-l_r)+F(l_r)\geqq F(x-l_r)+\Pi_r$ and hence $Z_1\Pi_1+\cdots+Z_r\Pi_r\geqq F(x-l_r)+\Pi_r$. Consequently $Z_1\Pi_1+\cdots+(Z_r-1)\Pi_r\geqq F(x-l_r)$ from which follows $F(x-l_r)=Z_1+\cdots+(Z_r-1)\Pi_r$. Further $Z_1, \cdots, Z_r-1$ is the reverse lexicographically largest partition realizing $F(x-l_r)$ since from a larger one, say $Z_1^*, \cdots, Z_r^*, \cdots, Z_m^*$, a partition of x larger than $Z_1, \cdots, Z_r, \cdots, Z_m$ is obtained; namely, $Z_1^*, \cdots, Z_r^*+1, \cdots, Z_m^*$. Consequently $r\geqq l(x-l_r)$ and we have $F(x)=F(x-l_r)+\Pi_r$.

The recursion (10) with the associated function l defined in (9) is the basis for a method of computation of F described in Section 4. Certain refinements of this method become important when F has to be computed for a range of x large relative to the sizes of $l_1, \cdots, l_m$. In the remainder of this section we wish to provide a basis for the refinements as well as prove some theorems interesting in themselves.

No assumption has been made so far about the particular ordering chosen for the lengths $l_1, l_2, \cdots, l_m$. For the remainder of this section we will assume a particular ordering dependant upon the relative value densities $\rho_i=\Pi_i/l_i$, $i=1, \cdots, m$ of the lengths. We will assume that the lengths $l_1, l_2, \cdots, l_m$ have been so ordered that

$$\rho_1\leqq\rho_2\leqq\cdots\leqq\rho_m.\dagger \tag{11}$$

The additional length $l_{m+1}=1$ with $\Pi_{m+1}=0$ is needed again only at the end of this section.

Before proving two theorems it is necessary to introduce some notation. For any r and s, $1\leqq r\leqq s\leqq m$, $F_{(r,s)}$ will denote the knapsack function defined from the lengths $l_r, l_{r+1}, \cdots, l_s$ with the values $\Pi_r, \Pi_{r+1}, \cdots, \Pi_s$.

Since the sequence (11) of densities is increasing then one would expect

† For those readers familiar with the knapsack algorithm described in reference 4, note that the ordering we assume here is just the reverse of the ordering assumed there. For equally dense lengths no special order is implied in (11) nor necessary for any of the results derived in this paper. However practical benefits can result from ordering the lengths of equal density in decreasing length.

that the lengths of higher density would have a better chance of being employed since for a given length x they can potentially contribute more to the total value $F(x)$. These expectations can be made precise in the following theorem:

THEOREM 5. *For any x there exists for each r, $1 \leqq r \leqq m$, an x_0 such that*

$$F(x) = F_{(1,r)}(x_2) + F_{(r+1,m)}(x_1). \tag{12}$$

for some $x_2 \leqq x_0$ and $x_1 + x_2 \leqq x$.

By (11) $p_s \leqq p_{r+1}$ for all s, $s \leqq r$. Consequently a replacement of any combination of lengths $l_1, \cdots, l_r$ of total length x' by some multiple of l_{r+1} of the same total length x' will not decrease the value obtained for x'. The proof of the theorem requires only a precise use of this device.

For each i, $1 \leqq i < r$, let a_i and b_i be integers such that $a_i l_i = b_i l_{r+1}$, and let $x_0 = a_1 l_1 + \cdots + a_r l_r$. We wish to show that this latter x_0 is suitable for the theorem. Let x_1 and x_2 be such that $x_1 + x_2 \leqq x$ and such that (12) is satisfied. For the given x let x_2 be chosen as small as possible so that if $F_{(1,r)}(x_2) = Z_1 \Pi_1 + \cdots + Z_r \Pi_r$ then $x_2 = Z_1 l_1 + \cdots + Z_r l_r$. If $x_2 \leqq x_0$ then the theorem has been established for the given x. Assume therefore that $x_2 > x_0$. Then for some i, $Z_i \geqq a_i$, say $i = r$. Because $a_r l_r = b_r l_{r+1}$ and because $p_r \leqq p_{r+1}$ it follows that $a_r \Pi_r \leqq b_r \Pi_{r+1}$. But then if $x_2' = x_2 - a_r l_r$ and $x_1' = x_1 + b_r l_{r+1}$ one has $F(x) = F(x_2') + F(x_1')$ and (12) is satisfied for a smaller x_2 contrary to an assumption made earlier. Consequently $x_2 \leqq x_0$ and the theorem is established.

The method of calculation of x_0 described in the proof is clumsy. In practice x_0 can be computed while F is being computed by Algorithm 1.B described in Section 4. Results of calculations indicate that for any r, x_0 is much smaller than the value for it computed in the proof. *See*, for example, the table in Section 4 where results of calculations on a series of knapsack problems are given.

Of special interest is the case when $r = m-1$. Then the theorem asserts that there exists an x_0 with the following property: for any x, $x \geqq x_0$, there is an x_2, $x_2 \leqq x_0$, for which $F(x) = F_{(1,m-1)}(x_2) + [(x-x_2)/l_m]\Pi_m$. Consequently $F(x) = F(x-l_m) + \Pi_m$ for $x > x_0$. Then if we define

$$f(x) = F(x) - [x/l_m]\Pi_m, \tag{13}$$

we have that $f(x) = F(x-l_m) + \Pi_m - [x/l_m]\Pi_m = F(x-l_m) + \Pi_m - \{[(x-l_m)/l_m]\Pi_m + \Pi_m\} = F(x-l_m) - [(x-l_m)/l_m]\Pi_m = f(x-l_m)$. We have proven therefore the following interesting theorem:

THEOREM 6. *There exists an x_0 such that for all x, $x > x_0$, $f(x) = f(x-l_m)$; that is $f(x)$ is periodic of period l_m.*†

† This theorem has since been extended in reference 7 to the case when F is defined with a discount factor α. In this case a factor α^{l_i} multiplies $F(x-l_i)$ in (10).

The function f is defined for all x but is most interesting to us in the range in which it is periodic. That is we are really interested in the function defined for all l, $0 \leqq l < l_m$ as follows:

Let N be an integer such that $Nl_m \geqq x_0$ then:

$$\varphi(l) = f(Nl_m + l). \tag{14}$$

φ can of course be computed by first computing $F(x)$ and $[x/l_m]\Pi_m$ for all x, $0 \leqq x \leqq (N+1)l_m$, that is by computing it directly from (13). It is interesting to note, however, that φ can be computed independently of F, in much the same way as F is computed from the equations (9) and (10). To prepare for the description of the procedure given in the next section we will substitute for F in (10) by means of (13). After transposing $[x/l_m]\Pi_m$ equation (10) becomes:

$$f(x) = \max\{0, f(x-l_i) + \Pi_i - ([x/l_m] - [(x-l_i)/l_m])\Pi_m;\ i \geqq l(x-l_i),\ x \geqq l_i\}.$$

The term 0 may be used instead of $-[x/l_m]$ since when $0 \leqq x < l_m - [x/l_m] = 0$ while when $x \geqq l_m$, the second term in the maximization is nonnegative. Letting $\Pi_i' = [l_i/l_m]\Pi_m$, letting $r(x/y)$ be the remainder upon dividing x by y and letting $l_i' = r(l_i/l_m)$ the equation can be written:

$$f(x) = \max\Big\{0, f(x-l_i) + \Pi_i' - \big[\big(r((x-l_i)/l_m) + l_i'\big)/l_m\big]\Pi_m;$$
$$x \geqq l_i,\quad i \geqq l(x-l_i)\Big\}.$$

For x sufficiently large $f(x) = \varphi(r(x/l_m))$ and $f(x) \geqq 0$ so the equation can be written

$$\begin{aligned}\varphi(r(x/l_m)) = \max\{&\varphi(r((x-l_i)/l_m)) + \Pi_i' \\ &- [(r((x-l_i)/l_m) + l_i')/l_m]\Pi_m;\quad i \geqq l(x-l_i),\quad x \geqq l_i\}.\end{aligned} \tag{15}$$

This equation will be used as the basis for Algorithm 1.C described in the fourth section.

A slightly different development along the lines of (13)–(15) is also possible. Instead of $f(x)$ we define $\bar{f}(x)$ by

$$\bar{f}(x) = F(x) - (x/l_m)\Pi_m. \tag{13$'$}$$

$\bar{f}(x)$ is periodic, just as $f(x)$ is and we can introduce $\bar{\varphi}(l) = \bar{f}(Nl_m + l)$. To obtain the recursion for $\bar{\varphi}$ we substitute in (10) as before obtaining

$$\bar{f}(x) = \max\{-(x/l_m)\Pi_m,\ \ \bar{f}(x-l_i) + \Pi_i';\quad i \geqq l(x-l_i),\quad x \geqq l_i\}, \tag{14$'$}$$

where $\Pi_i' = \Pi_i - (l_i/l_m)\Pi_m$. Note that because of the assumed ordering (11), $\Pi_i' \leqq 0$ for all i; nevertheless for sufficiently large x the second term in the maximization dominates the first so that we can obtain as before

$$\bar{\varphi}(r(x/l_m)) = \max\{\bar{\varphi}(r(x-l_i)l_m)) + \Pi_i';\quad i \geqq l(x-l_i),\quad x \geqq l_i\}. \tag{15$'$}$$

Each of the specialized knapsack functions can be defined by (7) with an additional restricting condition placed on the variables $Z_1, \cdots, Z_m$ in the maximization. The first specialization arises because of the cutting-knife limit, which was discussed in reference 4. It arises when only a limited number K of pieces may be cut from any given stock piece. The additional restriction to be placed on the variables is therefore:

$$\sum_{i=1}^{i=m} Z_i \leqq K. \tag{16}$$

The cutting-knife limitation (16) places a bound on the sum of the variables Z_i. Another limitation that can occur independently of (16) and also simultaneously with (16) is an individual bound b_i on each variable Z_i:

$$Z_i \leqq b_i. \qquad (i=1, \cdots, m) \tag{17}$$

This, for example permits one to define the specialized knapsack function discussed by DANTZIG[2] by taking $b_i=1$, $i=1, \cdots, m$.

The last specialization we will discuss places a bound on the number of variables Z_i that may be nonzero. If N is that bound then this last limitation can be expressed:

$$\sum_i \delta(Z_i) \leqq N, \tag{18}$$

where $\delta(Z)=1$ if $Z>0$ and $=0$ otherwise.

Whenever the restrictions (16) to (18) are inserted as additional restrictions on the variables $Z_1, \cdots, Z_m$ in the equation (7), either alone or in any combination, the resulting function no longer satisfies (2). For example let $G(K, x)$ be the value of the function defined by (7) when (16) is added as a restriction for a given K. If $G(K, x_1)$ and $G(K, x_2)$ are attained by using K_1 and K_2 pieces respectively with $K_1+K_2>K$, then obviously it is possible for $G(K, x_1+x_2)<G(K, x_1)+G(K, x_2)$. Any one of the other restrictions operates similarly. But it is possible to write a functional equation for G nevertheless:

$$G(0, x)=0, \text{ for all } x,$$

$$G(K+1, x)=\max\{G(K, x), G(K, x-l_i)+\Pi_i; \\ x \geqq l_i, i \geqq l(K, x-l_i)\}, \tag{19}$$

where l is defined

$$l(K, x)=\max\{1, i; G(K, x)=G(K-1, x-l_i)+\Pi_i\}. \tag{20}$$

Consider now the specialized knapsack function defined when condition (17) is inserted into (7). Let it be $G(b_1, \cdots, b_m, x)$ which can be defined:

$$G(0,\cdots,0,x)=0,\ \text{for all } x,$$

$$G(b_1,\cdots,b_{i+1},0,\cdots,0,x) = \max\{G(b_1,\cdots,b_i,0,\cdots,0,x-nl_{i+1})+n\Pi_{i+1};\ x\geqq nl_{i+1},\ b_{i+1}\geqq n\geqq 0\}. \qquad (21)$$

Consider lastly the specialized knapsack function defined when condition (18) is inserted into (16). If $G(N,x)$ is the function so defined then it satisfies

$$G(0,x)=0,$$
$$G(N+1,x)=\max\{G(N,x-nl_i)+n\Pi_i;\ x\geqq nl_i\geqq 0\}. \qquad (22)$$

For both the functions defined in (21) and (22) a function l can be defined like the one defined in (20). Equations comparable to (19) can be derived but this will not be done here.

In some applications two or even all three of the conditions (16) to (18) must apply simultaneously. For example in the corrugated-box cutting-stock problem discussed in reference 5 all three conditions generally apply: N is usually 2 but for a few special corrugators where it is 1 or 3, K is generally 5 or 6 and the bounds b_i are needed to deal with special situations. Let now $G_{(1,r)}(K,N,x)$ be the knapsack function defined by (7) with the restrictions (16) to (18) added and with $Z_i=0$ for $r\leqq i$. This function then satisfies:

$$G_{(1,0)}(K,N,x)=0 \quad \text{for all} \quad K,N,\text{ and } x,$$

$$G_{(1,r+1)}(K,N,x)=\max\{G_{(1,r)}(K,N-n^*,x-nl_{r+1})+n\Pi_{r+1};\ x\geqq nl_{r+1},\ K,b_{r+1}\geqq n\geqq 0\}, \qquad (23)$$

where $n^*=1$ if $n>0$ and $=0$ otherwise.

4. COMPUTATION OF THE KNAPSACK FUNCTIONS OF SECTION 2.

To APPRECIATE the significance of Theorem 4 for the construction of an algorithm to compute F it is worthwhile to describe an algorithm to compute F based on (8) alone. The concepts we develop in this discussion will then prove to be important in discussing algorithms for both one and two dimensional knapsack functions.

An algorithm based on (8) to compute $F(x)$ for a range $0\leqq x\leqq L$, will require two memory grids $F^*(x)$ and $l^*(x)$. When the computation is completed $F^*(x)$ will be $F(x)$ and $l^*(x)$ will be used to determine how $F(x)$ can be achieved for any given x.

Algorithm 1.A (Basic Step-off)

```
I. Initialize $F^*(x)=F_0(x)$ and $l^*(x)=x$ for $0\leqq x\leqq L$. Let $x_2=1$.
II. 1. Let $x_1=1$.
    2. If $x_1+x_2\leqq L$ then let $V=F^*(x_1)+F^*(x_2)$ and go to II.3. Otherwise go to II.4.
    3. If $V>F^*(x_1+x_2)$, then let $F^*(x_1+x_2)=V$, let $l^*(x_1+x_2)=x_1$, and go to II.4. Otherwise go to II. 4.
    4. If $x_1<x_2$ then let $x_1=x_1+1$ and go to II.2. Otherwise go to III.
III. If $x_2<L$ then let $x_2=x_2+1$ and go to II.1. Otherwise stop.
```

The algorithm is extremely simple. In the main part of the algorithm two fundamental operations take place. First a 'step-off' point x_2 is either given initially in I or is determined later in III. The determination in Algorithm 1.A is of the utmost simplicity. At the moment x_2 is used as a step-off point one has already calculated $F(x)$ for $0\leqq x\leqq x_2$. Secondly one 'steps-off' from x_2 with a number of lengths x_1 determined in this case also with the utmost simplicity in II.1 and II.4. At a step-off from x_2 with a length x_1 one determines whether the existing value $F^*(x_1+x_2)$ of the length x_1+x_2 is exceeded by the achievable value $F^*(x_1)+F^*(x_2)$ in which case $F^*(x_1+x_2)$ is replaced by $F^*(x_1)+F^*(x_2)$ and $l^*(x_1+x_2)=x_1$, the step-off length. For any given x, $l^*(x)$ is to be used to determine how $F^*(x)$ is to be achieved as follows: the partition of x achieving $F^*(x)$ is given by $x_1, x_2, \cdots, x_r=x$, with $x_{j-1}=x_j-l^*(x_j)$ for $j\geqq 2$. Naturally one could equally well have $l^*(x)$ record the step-off points by initializing it to be identically 0 in I; and then, when updating has to be done in II.3, setting $l^*(x_1+x_2)=x_2$. For again the partition achieving $F^*(x)$ is given by $x_1, x_2, \cdots, x_r=x$ with $x_{j-1}=l^*(x_j)$ for $j\geqq 2$.

The first obvious change to be made in Algorithm 1.A is to limit x_1 to being only one of the lengths $l_1, l_2, \cdots, l_m$. This simple change alone results in an algorithm that is an improvement upon the algorithm described in reference 1 for solving the knapsack problem—called there the load-problem. But additional limitations on both x_1 and x_2 will be introduced with each change justified by Theorem 4 and further resulting economies.

This approach generalizes to the methods of reference 6. While (15)′ is more transparent than (14), (15) has computational advantages. In both (15) and (15)′ all reference to variables other than remainders can be dropped, so that the following simple relation holds from (15)′.

$$\phi(r(x/l_m))=\max\{0, \phi(r((x-l_i)/l_m))+\Pi_i'\}. \qquad (15)''$$

3. ONE-DIMENSIONAL KNAPSACK FUNCTIONS THAT DO NOT SATISFY THE DIVIDE-IN-TWO INEQUALITY

IN THIS section we will provide functional equations for three specialized one-dimensional knapsack functions that do not satisfy the basic divide-in-two inequality (2) and cannot, therefore, be computed by any of the equations developed in Section 2. Each of the specializations is studied because it has arisen in significant applications of the cutting stock methods of references 3 and 4. However we will no longer justify the equations of this section in the manner in which we justified (10) because such justification would be very similar to the proof of Theorem 4.

In (10) the step-off points are the points $x-l_i$ while the lengths l_i are the step-off lengths. Not all the lengths l_i need be used as step-off lengths from a step-off point x_2 but only those lengths l_i for which $i \geqq l(x_2)$. In particular, therefore, if $l(x_2)=m+1$, one need only step-off from x_2 with the 'waste length' $l_{m+1}=1$ for which $\Pi_{m+1}=0$. But stepping-off with such a length merely ensures that $F^*(x)$ remains monotone increasing; i.e., one could have $F^*(x_2-1)>F^*(x_2)$ without the possibility of a waste length. One is justified, therefore in skipping over the point x_2+1 in III.2 as a step-off point if $F^*(x_2+1) \leqq F^*(x_2)$ by setting $F^*(x_2+1)=F^*(x_2)$. This is exactly what Algorithm B below does in III.2. Note that now instead of $l^*(x)$ recordng the step-off length it records instead the index of the step-off length used in reaching x. Also that $F_0(x)$ is computed in the main part of the algorithm rather than being assumed available for initialization as in Algorithm 1.A The change in II.3 from $>$ in Algorithm 1.A to $\geqq$ in Algorithm 1.B is necessary in order that after computation l^* is l as defined in (9).

Recall that $F^*(x)$ and $l^*(x)$ refer merely to two memory grids employed by the algorithm.

Algorithm 1.B (Ordered Step-off)

```
I. Reorder the lengths so that $\Pi_1/l_1 \leqq \Pi_2/l_2 \leqq \cdots \leqq \Pi_m/l_m$. Initialize
   $F^*(x)=0$, for $0 \leqq x \leqq L$, $l^*(0)=1$ and $x_2=0$.
II. 1. Let $j=l^*(x_2)$.
    2. If $x_2+l_j \leqq L$ then let $V=\Pi_j+F^*(x_2)$ and go to II.3. Otherwise
       go to II.4.
    3. If $V \geqq F^*(l_j+x_2)$ then let $F^*(l_j+x_2)=V$, let $l^*(l_j+x_2)=j$, and go
       to II.4. Otherwise go to II.4.
    4. If $j<m$ then let $j=j+1$ and go to II.2. Otherwise go to III.1.
III. 1. If $x_2<L$, let $x_2=x_2+1$ and go to III.2. Otherwise stop.
     2. If $F^*(x_2)>F^*(x_2-1)$ go to II.1. Otherwise let $F^*(x_2)=F^*(x_2-1)$,
        let $l^*(x_2)=m+1$, and go to III.1.
```

We wish to show that F^* and l^* as calculated by the algorithm are F and l as defined by (9) and (10). By Theorem 4 it is sufficient to show that they satisfy

$$l^*(x)=\max\{1,\, i;\, F^*(x)=F^*(x-l_i)+\Pi_i\}, \qquad (24)$$

$$F^*(x)=\max\{0,\, F^*(x-l_i)+\Pi_i;\, i \geqq l^*(x-l_i),\, x \geqq l_i\}. \qquad (25)$$

For $x=0$ it is immediate that F^* and l^* satisfy (24) and (25). Assuming this true for $x \leqq x_0$ we will prove it true for x_0+1. Recall, however, that the formulas (9) and (10), and therefore, (24) and (25), allowed for an extra length $l_{m+1}=1$ for which $\Pi_{m+1}=0$.

Only in the steps II.3 and III.2 can $l^*(x_0+1)$ have been given any value. Necessarily it has been given a value since every point x_2 is considered in the test of III.2. If it has been given a value by III.2, then $l^*(x_0+1)=m+1$, while if it has been given a value by II.3, $l^*(x_0+1)=j$ for some j, $1 \leqq j \leqq m$. By the induction assumption $l^*(x)$ for $x \leqq x_0$ is defined by (24). Consequently, all permissible values $F^*(x_0+1-l_i)+\Pi_i$ given in (25) for $F^*(x_0+1)$ have been tried as long as $i \leqq m$, and for the case $i=m+1$, step III.2 is equivalent. Steps II.3 and III.2, therefore, carry out the maximization of (25), and (25) must define $F^*(x)$ for $x=x_0+1$. And so must (24) define $l^*(x)$ for $x=x_0+1$.

We have, therefore, that $F^*(x)=F(x)$ and $l^*(x)=l(x)$ for all x, $0 \leqq x \leqq L$, after the completion of the algorithm. We can, therefore, revert to l and F again, without the added asterisks.

Earlier in this section, we saw how l was used to determine how the value $F(x)$ was achieved for any given x. With respect to the Algorithm 1.B, l can play another role. It also provides a direct measure of the amount of computation completed by the algorithm, since, for each step-off point x_2 one steps-off with the lengths l_j, $j=l(x_2)$, $l(x_2)+1, \cdots, m$, $m+1$. If one counts the computation involved in III.2 like step-off with the length l_{m+1}, then the total number of step-offs is

$$\sum_{x=0}^{L-1} [m+2-l(x)]. \tag{26}$$

At worst, $l(x)=1$ for all x, $0 \leqq x \leqq L-1$ so that an absolute upper bound on the number of step-offs is $Lx(m+1)$. However, that bound is never attained when $m>1$ since for any point $x=nl_i$, where $i>1$, $l(x)>1$ because $\rho_i \geqq \rho_1$. Indeed by Theorem 5, for any r there exists an x_0 such that if $x>x_0$ then $l(x) \geqq r+1$. Consequently, for L sufficiently large the number of step-offs can be very much less than $Lx(m+1)$. Actually, if L is sufficiently large, we know by Theorem 5 that for some x_0, $l(x)=m$ for all x, $x_0<x \leqq L$. For such x, $F(x)$ need not be computed by the algorithm since $F(x)=F(x-nl_m)+n\Pi_m$ where $n=[(x-x_0)/l_m]+1$.

Since l is computed at the same time as F by the algorithm 1.B, an x_0 can be computed during the calculation and the calculation terminated when $x=x_0$ in III.1. We will describe the changes needed in Algorithm 1.B to accomplish this.

Let there be an x^* such that $l^*(x) \geqq m$ for every x for which $x^*<x \leqq x^*+\lambda$, where $\lambda=\max\{l_i;\ i \geqq l^*(x^*)\}$. Then $x^*+\lambda$ would be a suitable x_0. The modifications necessary in Algorithm 1.B to compute x^* are simple.

Initially in I, x^* is set to 0. Then III is replaced by

```
III′. 1. If $x_2<x^*+\lambda$ and $x_2<L$, let $x_2=x_2+1$ and go to III′.2. Otherwise stop.
      2. If $l^*(x_2)<m$ let $x^*=x_2$ let $\lambda=\max\{l_i,\ i\geqq l^*(x_2)\}$ and go to III′.3. Otherwise go to III′.3.
      3. If $F^*(x_2)>F^*(x_2-1)$ go to II.1. Otherwise let $F^*(x_2)=F^*(x_2-1)$, let $l^*(x_2)=m+1$ and go to III′.1.
```

We could then call the changed algorithm 1.B, the terminating step-off algorithm.

We now turn to the problem of computing φ directly using the equation (15) rather than via the indirect method of computing F first and using (13) and (14). The algorithm requires two memory grids, one for φ^* of length l_m and one for l^* of length that will be determined during the calculation; the length however will not exceed x_0 as it is computed by the modification to Algorithm 1.B. The major differences between 1.B and 1.C is that in 1.C x will be taken modulo l_m for φ^*; that is $\varphi^*(x)$ will be recorded at the point $r(x/l_m)$ on the grid φ^* although $l^*(x)$ will be recorded at the point x on the grid l^*. Further, the term $[(r((x-l_i)/l_m]+l_i')/l_m)\Pi_m$ in (15) will require that Π_m be subtracted from Π_i' in those cases when $r[(x-l_i)/l_m]+l_i'\geqq l_m$. Recall that $l_i'=r(l_i/l_m)$ and that $\Pi_i'=\Pi_i-[l_i/l_m]\Pi_m$.

Algorithm 1.C (Periodic Step-off)

```
I. Reorder the lengths so that $\Pi_1/l_1\leqq\Pi_2/l_2\leqq\cdots\leqq\Pi_m/l_m$. Initialize $\varphi^*(x)=0$, for $0\leqq x\leqq l_m$, $l^*(0)=1$, and $x_2=0$. Let $x^*=0$ and $\lambda=\max\{l_i\}$.
II. 1. Let $j=l^*(x_2)$.
    2. If $r(x_2/l_m)+l_j'<l_m$, then let $V=\Pi_j'+\varphi^*(r(x_2/l_m))$ and go to II.3. Otherwise let $V=\Pi_j'-\Pi_m+\varphi^*(r(x_2/l_m))$ and go to II.3.
    3. If $V\geqq\varphi^*\{r((l_j+x_2)/l_m)\}$ then let $\varphi^*\{r((l_j+x_2)/l_m)\}=V$, let $l^*(l_j+x_2)=j$ and go to II.4. Otherwise go to II.4.
    4. If $j<m$ then let $j=j+1$ and go to II.2. Otherwise go to III.1.
III. 1. If $x_2<x^*+\lambda$, let $x_2=x_2+1$ and go to III.2. Otherwise stop.
     2. If $l^*(x_2)<m$ let $X^*=x_2$ let $\lambda=\max\{l_i;\ i\geqq l^*(x_2)\}$ and go to III.3. Otherwise go to III.3.
     3. If $r(x_2/l_m)>0$ let $V=\varphi^*\{r((x_2-1)/l_m)\}$ and go to III.4. Otherwise let $V=\varphi^*\{r((x_2-1)/l_m)\}-\Pi_m$ and go to III.4.
     4. If $\varphi^*(r(x_2/l_m))>V$ go to II.1. Otherwise let $\varphi^*(r(x_2/l_m))=V$, let $l^*(x_2)=m+1$, and go to III.1.
```

It is worth remarking that the technique employed to compute φ by computations modulo l_m can also be employed to compute F with less memory than either Algorithm 1.A or 1.B if one needs to know F for only a small range of x. For now let $\lambda = \max\{l_i;\ 1 \leqq i \leqq m\}$. Then one can always get by with at most λ memory locations for the F^* grid if one needs to know x for at most some range $x_0 < x \leqq x_0 + \lambda$. One simply treats x modulo λ when recording $F^*(x)$. However, one does still need a full range of the grid l^*.

The Algorithm 1.C records the values of F on a grid $F^*(x)$ of length l_m, while it can require for l a grid of greater length. In reference 6 an algorithm is described that requires only a grid of length l_m in both cases.

To give some indication of the efficiency of Algorithms 1.B and 1.C, we append in Table I results of a series of computations of knapsack functions. The functions were obtained as follows. A cutting-stock problem was solved as a linear programming problem. At each pivot step the knapsack function to be computed was computed by Algorithm 1.B and the values of the functions l and F printed out. The cutting-stock problem chosen was Al-5 of the test problems of reference 4. In that paper we described two devices used for speeding up the calculation of knapsack functions: first what we called the median method and second the recognition of identical prices. Both of these devices result in the selection of a subset of all the possible lengths $l_1, \cdots, l_m$ for the calculation of F. The effect of these devices is indicated in the column "no. of lengths" in Table I; instead of this number always being 20, it fluctuates from pass to pass. Hence the knapsack functions of the test series are all defined by a subset of 20 lengths varying in size from 182 to 549, and by prices determined from a linear programming solution to a cutting stock problem. The method of selection of lengths however always included one or more of the longer lengths; the minimum of the longest lengths selected was 470 and that minimum was realized only on pass 2, while from pass 9 on the minimum was 542. The length L over which the calculation was carried out was 1777, thus not very large relative to the maximum of the lengths used to define the knapsack functions

In the column "no. of true step-offs" is listed

$$N_1 = -L + \sum_{x=0}^{L-1} [m+2-l(x)],$$

that is the total number of step-offs with lengths other than l_{m+1}. To obtain the sum (26) from this column it is necessary therefore to add 1777. A measure of the efficiency of Algorithm 1.B. for a given knapsack function can be obtained by calculating $E = (N_1 + L)/N_2$ where $N_2 = Lx(m+1)$, the number of step-offs that would have to be made were no step-off points and no step-off lengths skipped over. A small E indicates an efficient calculation. For the results of Table I E is never less than 0.065 and never more than 0.149. For the later passes, say from pass number 40 on

TABLE I

Pass no.	No. of lengths	No. of true step offs	Confirmed x_0 if known	Pass no.	No. of lengths	No. of true step offs	Confirmed x_0 if known
1	6	87		46	10	1043	
2	5	72	1286	47	20	857	1430
3	8	81	1010	48	20	1278	
4	6	74		49	10	1043	
5	7	111	1372	50	20	1748	
6	6	81	1428	51	20	416	
7	10	136		52	10	755	
8	8	106	1551	53	20	742	
9	11	129		54	10	936	
10	9	147		55	20	1267	
11	11	233		56	10	937	
12	9	144		57	20	857	1444
13	14	84	1076	58	18	1000	1412
14	9	128		59	10	438	
15	15	338		60	20	1423	
16	18	136	1672	61	10	646	
17	17	364		62	20	1363	
18	10	166	1606	63	19	764	1214
19	19	405		64	10	646	
20	10	281	1504	65	20	1336	
21	18	232	1266	66	20	1349	
22	9	245		67	10	646	
23	19	553	1369	68	20	1495	
24	10	292		69	20	1482	
25	20	1298		70	10	646	
26	10	379		71	20	1492	
27	20	898	1678	72	19	379	906
28	10	354		73	10	756	
29	19	1036		74	19	1479	
30	9	341		75	10	626	
31	19	902	1764	76	19	1364	
32	10	691		77	20	1516	
33	20	997		78	10	708	
34	10	445		79	20	1490	
35	19	801	1422	80	19	775	
36	10	844		81	10	840	
37	19	1143		82	20	1838	
38	10	816		83	10	891	
39	19	1434		84	20	1652	
40	19	776	1489	85	10	905	
41	10	864		86	20	1737	
42	20	1412		87	10	772	
43	10	1039		88	20	1543	
44	20	997					
45	20	976					

E is close to 0.08 when the number of lengths is 19 or 20 and close to 0.14 when the number of lengths is 10

The last column "confirmed x_0 if known," gives the value of x_0 calculated by the modified Algorithm 1.B when that value can be confirmed by calculating $F(x)$ for x not greater than $L=1777$. Despite the fact that L is only about three times the largest length, x_0 could be calculated in a number of instances. The number was reduced during the later passes when the variation in the densities (Π_i/l_i) is likely to be less. However the table does confirm the importance of Theorem 5 for a class of practical knapsack functions.

5. COMPUTATION OF THE KNAPSACK FUNCTIONS OF SECTION 3.

AFTER THE very thorough discussion of Section 4 we can be very brief about the computation of the functions of Section 3 by algorithms like those of Section 4. Basically the improvements of these algorithms can still be applied, but now instead of having a single memory grid for F^*, two or more must be defined. By way of illustration we will describe an algorithm for computing $G(K, x)$ defined in (19) and (20). We assume in this case that G is to be computed for $K=K_0$ and $0\leqq x\leqq L$. We will assume $2(K_0+1)$ different memory grids $G^*(K, x)$ and $l^*(K, x)$, $K=0, \cdots, K_0$ although in practice this can be reduced to 4 since only $G(K-1, x)$ and $l(K-1, x)$ need be known to compute $G(K, x)$ and $l(K, x)$.

Algorithm 1.D (Cutting Knife Step-off)

```
I. Reorder the lengths so that $\pi_1/l_1\leqq\pi_2/l_2\leqq\cdots\leqq\pi_m/l_m$. Initialize
   $G^*(K, x)=0$ for $0\leqq x\leqq L$ and $1\leqq K\leqq K_0$. Let $l^*(0, 0)=1$, $K=0$,
   and $x_2=0$.
II.  1. Let $j=l^*(K, x_2)$.
     2. If $x_2+l_j\leqq L$ then let $V=\pi_j+G^*(K, x_2)$ and go to II.3. Otherwise
        go to II.4.
     3. If $V\geqq G^*(K+1, l_j+x_2)$ then let $G^*(K+1, l_j+x_2)=V$, let $l^*(K+1,
        l_j+x_2)=j$ and go to II.4. Otherwise go to II.4.
     4. If $j<m$ then let $j=j+1$ and go to II.2. Otherwise go to III.1.
III. 1. If $G(K+1, x_2)\geqq G(K, x_2)$ go to II.2. Otherwise let $G(K+1, x_2)=$
        $G(K, x_2)$ and let $l^*(K+1, x_2)=l^*(K, x_2)$ and go to II.2.
     2. If $x_2<L$, let $x_2=x_2+1$ and go to III.3. Otherwise go to III.4.
     3. If $G^*(K, x_2)>G^*(K, x_2-1)$ go to II.1. Otherwise let $G^*(K, x_2)=$
        $G^*(K, x_2-1)$, let $l^*(K, x_2)=m+1$ and go to III.1
     4. If $K<K_0$ let $K=K+1$, let $x_2=0$ and go to II.1. Otherwise stop.
```

In reference 4 we described an algorithm for computing $F(L)$ alone rather than $F(x)$, $0\leqq x\leqq L$. The algorithm has the advantage that it requires very little memory. It further has the advantage that it can be readily modified to calculate any of the functions of Section 3 as was shown in reference 4 for the cutting knife limitation. It has the disadvantage, however, of sometimes requiring long calculations. Nevertheless, it has proved practical for the cutting knife limitation.

6. A SIMPLE ALGORITHM FOR COMPUTING TWO-DIMENSIONAL KNAPSACK FUNCTIONS

THE ALGORITHM 1.A described in Section 4, was based simply upon (8), the one-dimensional form of (5). We then showed how Theorem 4 could be applied to modify the algorithm to the computationally more efficient Algorithm 1.B. We now want to discuss two-dimensional computations based on (5), alone. But before doing so we would like to derive two simple consequences of that equation in order to indicate how the two-dimensional computations are reduced to a series of one-dimensional step-off computations.

Consider the two auxiliary functions F_1 and F_2 defined as follows:

$$F_1(x, 0)=0,$$

$$F_1(x, y)=\max\{F(x, y-1), F(x_1, y)+F(x_2, y); \quad x\geqq x_1+x_2,\ 0<x_1\leqq x_2\}, \tag{27}$$

$$F_2(0, y)=0,$$

$$F_2(x, y)=\max\{F(x-1, y), F(x, y_1)+F(x, y_2); \quad y\geqq y_1+y_2,\ 0<y_1\leqq y_2\}. \tag{28}$$

It follows immediately from (5) that F also satisfies

$$F(x, y)=\max\{F_0(x, y), F_1(x, y), F_2(x, y)\}. \tag{29}$$

In Section 4 we introduced the concepts of step-off point and step-off length. These concepts can now be simply generalized to two-dimensions. In (27) any point (x_2, y) is a step-off point and any x_1, $0<x_1\leqq x_2$, a step-off length. Similarly in (28) any point (x, y_2) is a step-off point and any y_1, $0<y_1\leqq y_2$, a step-off length.

In order to make these ideas completely precise, we will now describe an algorithm for the calculation of $F(x, y)$, $0\leqq x\leqq L$, $0\leqq y\leqq W$. Like Algorithm 1.A it will be of utmost simplicity and will be later modified into a computationally more efficient algorithm. The algorithm will require three memory grids $F^*(x, y)$, $l^*(x, y)$, and $w^*(x, y)$ although frequently in practice $l^*(x, y)$ and $w^*(x, y)$ can be combined into one. The grids $l^*(x, y)$ and $w^*(x, y)$ are used, like $l^*(x)$ for $F^*(x)$, to record how the value $F^*(x, y)$ is achieved. When the computation is completed $F^*(x, y)=$

$F(x, y)$ while the memory grids l^* and w^* will have computed two functions l and w which we now wish to define.

The function $l(x)$ defined in (9) is directly dependent upon the particular order chosen for the lengths $l_1, l_2, \cdots, l_m$, the possible step-off lengths in the one-dimensional case. In the two-dimensional case the set of possible step-off lengths is a function of the step-off point and the direction in which the step-off is to take place, that is whether the step-off is to take place along a line of constant y or constant x. Nevertheless, different orderings of the step-off lengths can be considered; for example, in decreasing size or in increasing price density; that is in increasing size of $F(x_1, y)/x_1$ for a step-off length x_1 or of increasing size of $F(x, y_1)/y_1$ for a step-off length y_1. To simplify the presentation of the algorithms, we will choose the order of decreasing size although it should be emphasized that other orders can be considered.

If the value $F(x, y)$ for a point (x, y) has been achieved by stepping-off with a length x_1 from a step-off point (x_2, y), where $0<x_1 \leqq x_2$, $x_1+x_2=x$, and $F(x, y)=F(x_1,y)+F(x_2,y)$, we will choose for the value of l for the point (x, y) the smallest step-off length used in achieving $F(x, y)$. Otherwise it is x. Formally, therefore, we define for x, $y \geqq 1$,

$$l(x, y)=\min\{x_1, x;\ 0<x_1 \leqq x-x_1,\quad F(x, y)=F(x_1, y)+F(x-x_1, y)\}; \tag{30}$$

and similarly

$$w(x, y)=\min\{y_1, y;\ 0<y_1 \leqq y-y_1,\quad F(x, y)=F(x, y_1)+F(x, y-y_1)\}. \tag{31}$$

It is these functions that are computed as l^* and w^* in the Algorithm 2.A to be introduced below. We now turn to a description of the algorithm.

At the moment x_2 is used as a step-off point in the Algorithm 1.A or 1.B, $F^*(x)=F(x)$, for $0 \leqq x \leqq x_2$, while for $x>x_2$, all that is known is that $F^*(x) \leqq F(x)$. After x_2 has been used as a step-off point $F^*(x_2+1)=F(x_2)$. The situation in Algorithm 2.A (as well as 2.B) is a little more complicated.

Prior to (x_2, y_2) being used as a step-off point the situation is as follows: $F^*(x, y)=F(x, y)$ for all x and y, $0 \leqq x \leqq x_2$ and $0 \leqq y \leqq y_2$, as well as $x_2< x \leqq L$ and $0 \leqq y < y_2-1$. Further $F^*(x, y_2+1)=F_2(x, y_2+1)$ for all x, $0 \leqq x < x_2$ where F_2 is defined in (28). The value $F^*(x, y_2)$ for $x_2 < x \leqq L$ is the maximum of $F_2(x, y_2)$ and some incompletely computed value $F_1(x, y_2)$, as F_1 is defined in (27). This situation is illustrated in Fig. 3. During the step-offs from (x_2, y_2) along $y=y_2$, $F^*(x_2+1, y_2)$ becomes $F(x_2+1, y_2)$ while during the step-offs along $x=x_2$ $F^*(x_2, y_2+1)$ becomes

$F_2(x_2, y_2+1)$. Along the line $y=y_2$, one steps-off with all lengths x_1, $0<x_1\leqq x_2$ and along the line $x=x_2$ with all lengths y_1, $0<y_1\leqq y_2$.

In the Algorithm 1.A, x_2+1 is the step-off point from which step-offs are made after x_2 has been used. In the Algorithm 2.A, (x_2+1, y_2) follows (x_2, y_2) as a step-off point when $x_2+1\leqq L$; otherwise $(1, y_2+1)$ follows (x_2, y_2).

We now will give a description of the Algorithm 2.A. The algorithm

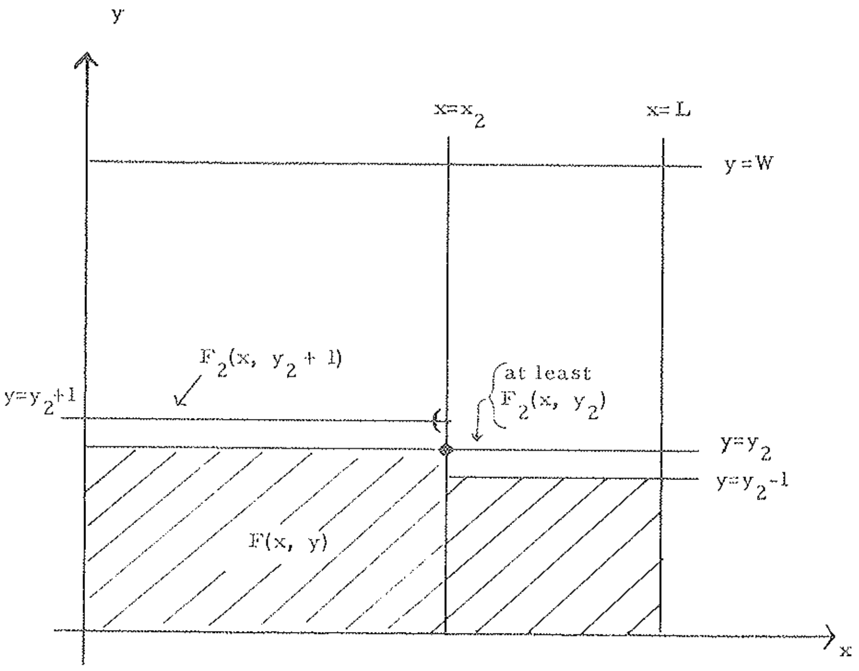

Figure 3

has four major parts. In part I initialization takes place. In parts II and III, it is assumed that a step-off point (x_2,y_2) has been defined. In part II step-offs from (x_2, y_2) take place along the line $y=y_2$ while in part III step-offs take place along the line $x=x_2$, the former step-offs are taken first. Finally in part IV a new step-off point is determined in the order discussed above. Recall that F_0 is defined in (6).

Algorithm 2.A (Basic Two-dimensional Step-off)

I. Let $F^*(x, y)=F_0(x, y)$ for $0\leqq x\leqq L$ and $0\leqq y\leqq W$. Let $l^*(l_i, w_i)=l_i$ and $w^*(l_i, w_i)=w_i$ for $i=1, \cdots, m$, and let $l^*(0, 0)=w^*(0, 0)=0$. Let $x_2=y_2=1$ and go to II.1

II. 1. Let $x_1=1$.
2. If $x_1+x_2 \leqq L$ then let $V=F^*(x_1, y_2)+F^*(x_2, y_2)$ and go to II.3. Otherwise go to III.1.
3. If $V>F^*(x_1+x_2, y_2)$ then let $F^*(x_1+x_2, y_2)=V$, let $l^*(x_1+x_2, y_2)=x_1$, and let $w^*(x_1+x_2, y_2)=y_2$ and go to II.4. If $V=F^*(x_1+x_2, y_2)$ then let $l^*(x_1+x_2, y_2)=x_1$ and go to II.4. Otherwise go to II.4.
4. If $x_1<x_2$ then let $x_1=x_1+1$ and go to II.2. Otherwise go to III.1.

III. 1. Let $y_1=1$.
2. If $y_1+y_2 \leqq W$ then let $V=F^*(x_2, y_1)+F^*(x_2, y_2)$ and go to III.3. Otherwise go to IV.1.
3. If $V>F^*(x_2, y_1+y_2)$ then $F^*(x_2, y_1+y_2)=V$, let $w^*(x_2, y_1+y_2)=y_1$,

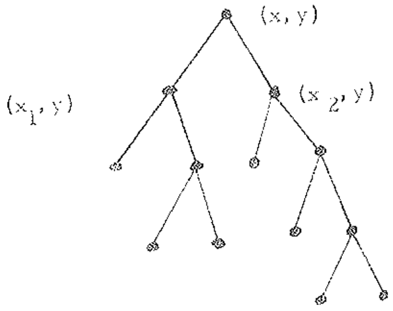

Figure 4

let $l^*(x_2, y_1+y_2)=x_2$ and go to III.4. If $V=F^*(x_2, y_1+y_2)$ then let $w^*(x_2, y_1+y_2)=y_1$ and go to III.4. Otherwise go to III.4.

IV. 1. If $x_2<L$ then let $x_2=x_2+1$ and go to II.1. Otherwise go to III.2.
2. If $y_2<W$ then let $y_2=y_2+1$ and $x_2=1$ and go to II.1. Otherwise stop.

When the computation is completed, l^* and w^* can be used to determine how $F^*(x, y)$ is achieved for any x and y. The 'backtracking' in this case is a little more complicated than in the one-dimensional case as described in Section 2. In the one-dimensional case, one must only determine a partition of a line segment from 0 to x. In the two-dimensional case one is determining a special tree structure as illustrated in Fig. 4, in which each of the nodes is labeled with the dimensions of a rectangle.

Each node of the tree has either no other nodes below it in the tree (a 'dangling' node) or has exactly two nodes immediately below it. A node labeled (x, y) with nodes below it labeled (x_1, y) and (x_2, y) means that the rectangle (x, y) should be divided into two rectangles (x_1, y) and (x_2, y) so that necessarily $x=x_1+x_2$. Similarly if a node labeled (x, y) has nodes labeled (x, y_1) and (x, y_2) immediately below it then the rectangle (x, y)

should be divided into two rectangles (x, y_1) and (x, y_2) so that necessarily $y = y_1 + y_2$.

For given functions l^* and w^* we will associate with every point (x, y), $x, y \geqq 1$, a unique tree structure realizing the value $F^*(x, y)$ just as in the one-dimensional case l^* associated with every x a unique partition realizing the value $F^*(x)$. A node labeled (x, y) will be a dangling node if and only if $l^*(x, y) = x$ and $w^*(x, y) = y$. If $l^*(x, y) < x$ then the nodes below the node labeled (x, y) are labeled $(l^*(x, y), y)$ and $(x - l^*(x, y), y)$; otherwise they are labeled $(x, l^*(x, y))$ and $(x, y - l^*(x, y))$.

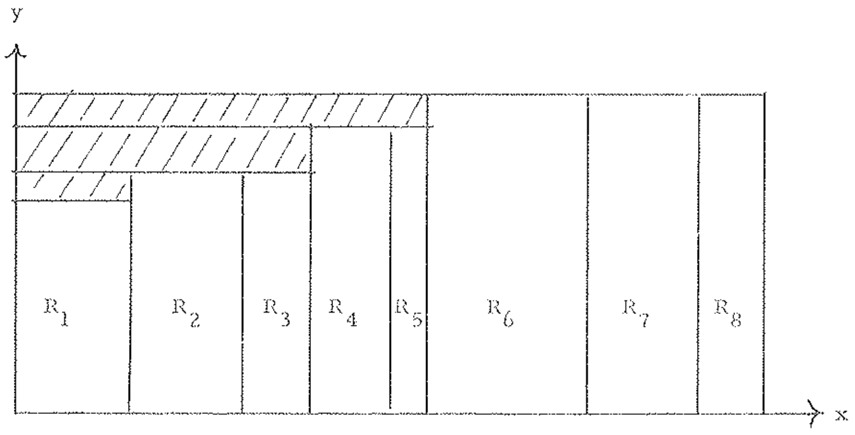

Figure 5

Because of the manner in which l^* and w^* have been computed the patterns of cuts for a rectangle (x, y) determined by them takes a characteristic form that can be illustrated with one special case. Consider the pattern of cuts illustrated in Fig. 5 where the cross-hatched area is waste. The first rectangle R_1 in the pattern reading from left to right has the smallest width (height) of any rectangle in the pattern, and among rectangles of the same width it will have the largest length. The next two rectangles R_2 and R_3 are the next smallest in width and are in order of decreasing length, and so on throughout the pattern. Similarly, if the cuts were in the opposite direction, the pattern of cuts would be as illustrated in Fig. 6.

These kinds of patterns of cuts will be repeated throughout an entire pattern. For example, if rectangle R_2 in the pattern of Fig. 5 is cut, then it must be cut in the manner of the pattern of Fig. 6, while if a rectangle of Fig. 6 is cut, then it is cut in the manner of the pattern of Fig. 5.

Had the Algorithm 2.A computed functions l and w different from those

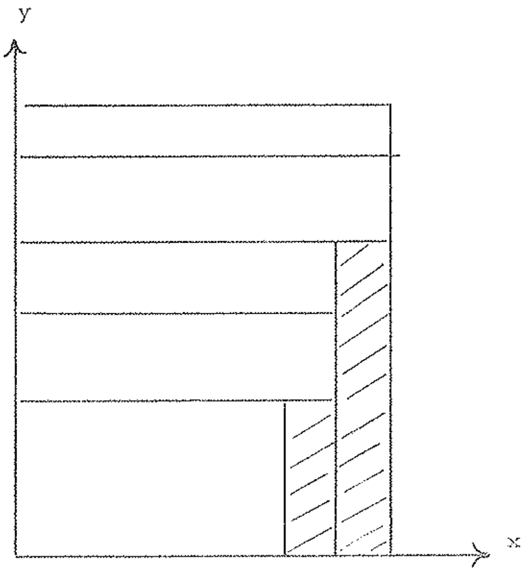

Figure 6

defined in (30) and (31) then the pattern corresponding to the tree structure realizing $F(x, y)$ would have exhibited different properties. For example, if $l(x, y)$ took as its value the x_1 for which $F(x_1, y)/x_1$ was as large as possible, then, for example, the order of the rectangles R_1, R_2, $\cdots$, R_8 in Fig. 3 would differ. They would be primarily ordered by width but those of the same width would then be ordered differently. As we remarked earlier, there may be advantages to considering differently defined functions l and w.

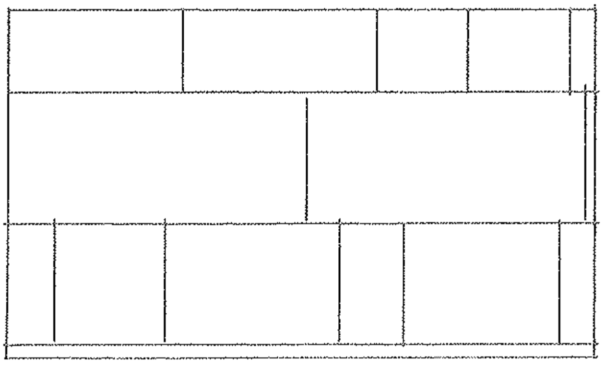

Figure 7

7. IMPROVEMENTS FOR THE ALGORITHM 2.A.

WE WANT now to introduce into Algorithm 2.A some of the improvements added to Algorithm 1.B. These were of two kinds. First, some of the points x_2 could be skipped over as step-off points. Here some of the points (x_2, y_2) may be skipped over as step-off points for lengths along the line $y=y_2$ and some of the points (x_2, y_2) (perhaps the same ones) may be skipped over as step-off points for lengths along the line $x=x_2$. Second, it was recognized that x_1 had only to be one of the lengths $l_1, l_2, \cdots, l_m$ and in addition, for a given x_2, had only to be a length l_i for which $i \geqq l(x_2)$. In the two-dimensional case, we will find that in stepping-off from a step-off point (x_2, y_2) along the line $y=y_2$ only some of the lengths x_1, $0<x_1 \leqq x_2$, will have to be used and that further they will satisfy $x_1 \leqq l(x_2, y_2)$, where l will be redefined below; similarly for stepping-off along the line $x=x_2$. The restrictions will be made fully precise and the functions l and w will be redefined below.

As a first step we would like to introduce a minor change into the definitions of l and w. In the one-dimensional case, a special 'waste' length $l_{m+1}=1$ with $\Pi_{m+1}=0$ was introduced. It had the effect that when $F(x)=F(x-1)$, then $l(x)=m+1$. We would now like to modify the definitions of l and w so as to introduce a comparable effect. The modifications necessary in definitions (30) and (31) are simple. The new definitions read:

$$\text{If } F(x, y)=F(x-1, y) \text{ then } l(x, y)=0. \tag{32}$$

$$\text{Otherwise, } l(x, y)=\min\{x_1, x; 0<x_1 \leqq x-x_1, w(x_1, y)>0, \\ F(x, y)=F(x_1, y)+F(x-x_1, y)\}.$$

$$\text{If } F(x, y)=F(x, y-1) \text{ then } w(x, y)=0. \tag{33}$$

$$\text{Otherwise, } w(x, y)=\min\{y_1, y; 0<y_1 \leqq y-y_1, l(x, y_1)>0, \\ F(x, y)=F(x, y_1)+F(x, y-y_1)\}.$$

Definition (32) has the following consequences: $l(x, y)=0$ if and only if $F(x, y)=F(x-1, y)$. Also, if $x_1=l(x, y)$ and $x_1>0$ then $x_1=l(x_1, y)$ and $w(x_1, y)>0$. In particular, if $x_1=l(x_1, y)$ and $x_1>0$ then $F(x_1, y)>F(x_1-1, y)$ and $F(x_1, y)>F(x_1, y-1)$. The effect of definition (32) will be to limit step-off lengths along a line of constant y to lengths x_1 for which $x_1=l(x_1, y)$, $x_1>0$. Definition (33) has a similar effect for step-off lengths along a line of constant x.

These definitions can now be combined with an equation arising from (5) to provide a new definition of F analogous to the definition in the one-dimensional case provided by (9) and (10). For $x, y \geqq 1$:

$$\begin{aligned} F(x, y) = \max\{&F_0(x, y), F(x-1, y), F(x, y-1), F(x_1, y) \\ &+F(x-x_1, y), F(x, y_1)+F(x, y-y_1); \\ &0<x_1=l(x_1, y)\leqq l(x-x_1, y), 0<l(x, y_1), \\ &0<y_1=w(x, y_1)\leqq w(x, y-y_1), 0<w(x_1, y)\}. \end{aligned} \tag{34}$$

We can now prove:

THEOREM 7. *The knapsack function is defined by equation* (34) *for* $x, y \geqq 1$, *where* l *and* w *are defined by* (32) *and* (33) *and* $F(x, 0)=F(0, y)=0$.

(34) certainly defines $F(1, 1)$ since the choice of values is then simply $F_0(1, 1)$. Assume that it defines $F(x, y)$ for all x and y, $0\leqq x\leqq x_0$ and $0\leqq y\leqq y_0$. We will prove that it defines $F(x_0+1, y_0)$ also. If $F(x_0+1, y_0)$ is one of $F_0(x_0+1, y_0)$, $F(x_0, y_0)$ or $F(x_0+1, y_0-1)$, then clearly it is defined by (34). If it is not, then for some $x_1, 0<x_1\leqq x_0+1-x_1$, $F(x_0+1, y_0)=F(x_1, y_0)+F(x_0+1-x_1, y_0)$, or for som $y_1, 0<y_1\leqq y_0-y_1$, $F(x_0+1, y_0)=F(x_0+1, y_1)+F(x_0+1, y_0-y_1)$. Let the former be the case. The argument for the latter is analogous. Since $F(x_0+1, y_0)>F(x_0+1, y_0-1)$ there must be an x_1 for which also $w(x_1, y_0)>0$. Among *such* x_1 choose the smallest. Then necessarily $x_1\leqq l(x_0+1-x_1, y_0)$ since if $l(x_0+1-x_0, y_0)=0$ then $F(x_0+1, y_0)=F(x_0, y_0)$, while $l(x_0+1-x_0, y_0)<x_1$ would contradict the assumption that x_1 was the smallest possible step-off length. Hence, $F(x_0+1, y_0)$ is defined by (34). One can prove analogously that $F(x_0, y_0+1)$ is also defined by (34). Consequently the theorem has been proved.

The changes to be brought to Algorithm 2.A are now evident. First, a point (x_2, y_2) may be skipped as a step-off point along the line $y=y_2$ when $l(x_2, y_2)=0$, and along the line $x=x_2$ when $w(x_2, y_2)=0$. Second, for a given step-off point (x_2, y_2) only those x_1 need be used as step-off lengths for which $0<x_1\leqq l(x_2, y_2)$ and $w(x_1, y_2)>0$, and only those y_1 need be used as step-off lengths for which $0<y_1\leqq w(x_2, y_2)$ and $l(x_2, y_1)>0$.

Algorithm 2.B (Ordered Two-dimensional Step-off)

```
I. Let $F^*(x, y)=F_0(x, y)$ for $0\leqq x\leqq L$ and $0\leqq y\leqq W$. Let $x_2=y_2=1$,
   let $l^*(l_i, w_i)=l_i$, and $w^*(l_i, w_i)=w_i$ for $i=1, \cdots, m$ and go to IV.3.
II. 1. Let $x_1=1$.
    2. If $x_1+x_2\leqq L$ then go to II.3. Otherwise go to IV.4.
    3. If $w^*(x_1, y_2)>0$ and $x_1=l^*(x_1, y_2)$ then let $V=F(x_1, y_2)+$
       $F^*(x_2, y_2)$ and go to II.4. Otherwise go to II.5.
    4. If $V>F^*(x_1+x_2, y_2)$ then let $F^*(x_1+x_2, y_2)=V$, let $l^*(x_1+x_2, y_2)=$
       $x_1$ and $w^*(x_1+x_2, y_2)=y_2$ and go to II.5. If $V=F^*(x_1+x_2, y_2)$
       then let $l^*(x_1+x_2, y_2)=x_1$ and go to II.5. Otherwise go to II.5.
    5. If $x_1<l^*(x_2, y_2)$ then let $x_1=x_1+1$ and go to II.2. Otherwise go
       to IV.4.
```

```
III. 1. Let $y_1 = 1$.
     2. If $y_1 + y_2 \leqq W$ then go to III.1. Otherwise go to IV.1.
     3. If $l^*(x_2, y_1) > 0$ and $y_1 = w^*(x_2, y_1)$ then let $V = F^*(x_2, y_1) + F^*(x_2, y_2)$
        and go to III.4. Otherwise go to III.5.
     4. If $V > F^*(x_2, y_1 + y_2)$ then let $F^*(x_2, y_1 + y_2) = V$, let $w^*(x_2, y_1 + y_2) =$
        $y_1$ and $l^*(x_2, y_1 + y_2) = x_2$ and go to III.5. If $V = F^*(x_2, y_1 + y_2)$
        then let $w^*(x_2, y_1 + y_2) = y_1$ and go to III.5. Otherwise go to III.5.
     5. If $y_1 < w^*(x_2, y_2)$ then let $y_1 = y_1 + 1$ and go to III.2. Otherwise
        go to IV.1.
IV.  1. If $x_2 < L$ then let $x_2 = x_2 + 1$ and go to IV.3. Otherwise go to IV.2.
     2. If $y_2 < W$ then let $y_2 = y_2 + 1$, let $x_2 = 1$ and go to IV.3. Otherwise
        stop.
     3. If $F^*(x_2, y_2) > F^*(x_2 - 1, y_2)$ go to II.1. Otherwise let $F^*(x_2, y_2) =$
        $F^*(x_2 - 1, y_2)$, let $l^*(x_2, y_2) = 0$ and go to IV.4.
     4. If $F^*(x_2, y_2) > F^*(x_2, y_2 - 1)$ go to III.1. Otherwise let $F^*(x_2, y_2) =$
        $F^*(x_2, y_2 - 1)$, let $w^*(x_2, y_2) = 0$ and go to IV.1.
```

The algorithm requires memory space for l^*, w^*, and F^* in order to be implemented. If it were the case that $l^*(x, y)w^*(x, y) = 0$ for all x and y then memory requirements could be reduced by assigning both l^* and w^* to the same signed ($\pm$) memory space. As l and w have been defined however, it does not follow that $l(x, y)w(x, y) = 0$; for example it might be that $l(l_i, w_i) = l_i$ and $w(l_i, w_i) = w_i$. However l and w could be redefined to give for example l always the priority for nonzero value over w and so to permit a reduction of the memory requirements for the algorithm.

8. STAGED TWO-DIMENSIONAL KNAPSACK FUNCTIONS

The improvements brought to Algorithm 2.B are also relevant for the computation of knapsack functions defined by what we called in reference 5 staged cutting. In staged cutting of a rectangle (L, W), say 2-stage cutting, the rectangle is first cut down its length into strips and then these strips are cut across their width into rectangles as illustrated in Fig. 7. The value of a small rectangle (x, y) cut in this way is $F_0(x, y)$ as defined in (6); that is the value Π_i of the largest value rectangle (l_i, w_i) for which $l_i \leqq x$ and $w_i \leqq y$.

Formally the value $F_k(x, y)$ obtained for a rectangle (x, y) in k-stage cutting can be defined for any k:

$$F_0(x, y) = \max\{0, \Pi_i; l_i \leqq x \text{ and } w_i \leqq y\}.$$

For $r \geqq 0$,

$$F_{2r+1}(x, y) = \max\{F_{2r}(x, y), F_{2r+1}(x_1, y) + F_{2r+1}(x_2, y); \quad x \geqq x_1 + x_2, 0 < x_1 \leqq x_2\}. \quad (35)$$

For $r \geqq 1$,

$$F_{2r}(x, y) = \max\{F_{2r-1}(x, y), F_{2r}(x, y_1) + F_{2r}(x, y_2); \quad y \geqq y_1 + y_2, 0 < y_1 \leqq y_2\}.$$

To indicate how the improvements brought to Algorithm 2.B could also be used in the computation of the functions $F_k(x, y)$ we could proceed as we did in Section 7; that is we could first define functions $l_k(x, y)$ and $w_k(x, y)$ then modify the definitions (35). However, at this point we will simply outline the changes to be made in Algorithm 2.B in order for it to be used to calculate $F_k(x, y)$ for any k, $k \geqq 1$.

In Algorithm 2.A or 2.B, each point (x_2, y_2), $1 \leqq x_2 \leqq L$ and $1 \leqq y_2 \leqq W$, is considered once for use as a step-off point. It may be used as a step-off point for step-offs both along the line $y = y_2$ and along the line $x = x_2$, but having once been so used it will never be considered for use again. When calculating $F_k(x, y)$ by means of a modified Algorithm 2.A or 2.B, a point (x_2, y_2) will be considered k times for use as a step-off point but at the rth time it is being considered it may be used as a step-off point for step-offs only along the line $y = y_2$ when r is odd, and only along the line $x = x_2$ when r is even. No other modifications are necessary in the algorithm.

9. CONCLUSIONS

In this paper we first approached knapsack functions abstractly in terms of a characterization and then went on to develop efficient methods of computing these functions. The methods are all modified dynamic programming algorithms. We have also studied briefly in Sections 3 and 5 knapsack functions of a more general kind than those characterized in Section 1. These topics can be extended further in that direction by pursuing some of the subjects mentioned in reference 5. For example, a study of stock with defects, or more generally position-dependant values for rectangles within the stock, would have led to two-dimensional functions not satisfying the basic divide-in-two inequality. A study of what we called in reference 5 a k-group staged cutting also leads to more general knapsack functions. Much work remains to be done in those directions.

REFERENCES

1. Richard Bellman and Stuart Dreyfus, *Applied Dynamic Programming*, Princeton University Press, Princeton, New Jersey, 1962.
2. George B. Dantzig, "Discrete Variable Extremum Problems," *Opns. Res.* **5**, 266–277 (1957).
3. P. C. Gilmore and R. E. Gomory, "A Linear Programming Approach to the Cutting Stock Problem," *Opns. Res.* **9**, 849–859 (1961).
4. ——— and ———, "A Linear Programming Approach to the Cutting Stock Problem—Part II," *Opns. Res.* **11**, 863–888 (1963).
5. ——— and R. E. Gomory, "Multi-Stage Cutting Stock Problems of Two and More Dimensions," *Opns. Res.* **13**, 94–120 (1965).
6. R. E. Gomory, "On the Relation Between Integer and Non-Integer Solutions to Linear Programs," *Proc. Nat. Acad.* **53**, 260–265 (1965).
7. Jeremy F. Shapiro and Harvey M. Wagner, "A Finite Renewal Algorithm for the Knapsack and Turnpike Models," to be published in *Opns. Res.*