An Algorithm for Modular Partitioning

HAROLD S. STONE*

Stanford Research Institute, Menlo Park, California

ABSTRACT. An algorithm is described for partitioning the elements of a digital logic design into physical packages where each package contains components of one type and where connected components of the same type are assigned to the same package. The problem is a special case of the one-dimensional cutting-stock problem.

KEY WORDS AND PHRASES: design automation, partitioning algorithms, cutting-stock problem, modular design

CR CATEGORIES: 5.40, 6.0

1. *Introduction*

The impact of new device technologies on computers has been felt through an orders-of-magnitude increase in computer speed and complexity in the last decade. The impact of large scale integrated technology has not yet been felt, but it is likely that it will bring another order-of-magnitude increase in speed and complexity. To realize the potential of the present and newly emerging device technologies, it has become increasingly important to use the power of present generation computers for the automated design and construction of the next generation of machines. A problem is treated here that is associated with automated methods for the implementation of logic designs with modular components.

Although the problem is stated in the context of implementation of digital logic, the problem is a special case of the more general cutting-stock problem that has been treated elsewhere [1, 2, 3]. Also related are the knapsack problem [4] and the machine-sequencing problem [5]. The algorithm derived in the present paper, however, is believed to be new in the sense that it cannot be obtained directly from the algorithms for the more general problems by specializing those algorithms to the restricted context.

2. *Background*

Prior to the emergence of large and medium scale integrated circuits, logic designs were implemented largely with modular components. In this way, a relatively small variety of modules can be used to realize logic designs. Logic modules normally contain a number of identical gates, and all connections to gates are accessible outside the module. For example, a module might contain 20 NANDS, 10 trigger flip-flops, or 8 set/reset flip-flops. Although the costs of large scale integrated circuits

The research reported in this paper was sponsored by, but does not necessarily constitute the opinion of, the Air Force Cambridge Research Laboratories, Office of Aerospace Research, under Contract F19628-68-C-0120.

* On leave at Stanford University, 1968–1969.

make it attractive to dispense with modular techniques at the device level and move toward the use of "functional" modules, the design and manufacture of functional, large scale integrated circuits rests heavily on the use of modular-design techniques within the integrated package. Thus methods for automating modular design are important for current technology and will undoubtedly continue to be important to support the manufacture of newer functional devices as large scale integration advances in development.

Automatic modular implementation techniques have historically followed a three-step process—partitioning, placement, and conductor interconnection. Partitioning is the process of assigning logic elements to modules. Placement is the process of positioning modules in a physical environment, such as on a backplane or a printed-circuit board. The process of conductor interconnection refers to the calculation of masks for printed wiring, or sequencing and routing for discrete conductor wiring. Of the three processes, we concentrate only on partitioning here.

The problem that we attempt to solve is characterized as follows. A logic design is specified in terms of a number of different primitive components and the interconnections between them. The design is to be implemented with homogeneous modules; i.e. they each contain only one type of primitive component, and for each type of component there is a fixed number of components per module. A partitioning algorithm must determine an assignment of components in the logic design to modules; the assignment should tend to use as few modules as possible, yet also tend to lead to short interconnections.[1] An assignment with the minimum number of modules may require excessively long interconnections between modules, while an assignment that results in relatively short interconnections may be too lavish in the use of modules. A good partitioning algorithm is one that achieves a balance between the two criteria.

The absolute minimum number of modules required for an implementation can be determined easily from the number of components of each type that are used in the design and the maximum number of components of each type that are available in a single module. In fact, if there are n components of a given type, and at most m components of this type can be placed in a single module, then $\langle n/m \rangle$ modules of this type are required, and there are approximately $n!/(m!)^{\langle n/m \rangle}$ different assignments that achieve the minimum.[2] This is simply the number of different ways of selecting $\langle n/m \rangle$ subsets, each containing m objects, from a set of n objects.

Since we are interested in short interconnections as well as a minimum or near minimum number of modules, it is necessary to consider connectivity of components in a partitioning algorithm. We do not consider the connectivity of components of different types, for these components must necessarily be placed in different modules, and it is the function of a module placement algorithm to arrange for short interconnections between these components. On the other hand, it is a desirable characteristic of a partitioning algorithm to place two connected components of the same type in the same module. This characteristic implies that a connected cluster of identical components should be placed in a single module, provided that the cluster size does not exceed the size of a module. The problem, then, can be stated as fol-

[1] The partitioning problem for which modules need not be homogeneous has been treated by Lawler [7], who showed the problem can be formulated as an integer linear program. Lawler, Levitt, and Turner [8] have recently proposed a graph-theoretical approach to partitioning.

[2] $\langle x \rangle$ denotes the smallest integer not less than x.

lows. Find a partitioning algorithm that assigns connected clusters of identical components to the same module and that requires as few modules as possible. As stated here, the sets of components of each different type can be treated independently by the partitioning algorithm. Other formulations are possible in which the interdependence of the different types of components is an important factor, but these are not considered here.

In Section 3 we reformulate the modular-design problem abstractly and cast it as an integer linear program. Following that, in Section 4 we consider a set of techniques for reducing the integer program in order to decrease the computation required. Although standard techniques exist for the solution of integer linear programs, it appears more attractive in the case at hand to use branch-and-bound techniques; this is discussed in Section 5. Some concluding observations are presented in Section 6.

3. *Formulation of the Partitioning Problem as an Integer Linear Program*

Given a logic design composed of components of several different types, we partition the components into homogeneous modules by considering each set of components of a single type as a separate problem. The subproblems are formulated by partitioning a logic network into disjoint subnetworks such that each subnetwork contains elements of only one type and there is one subnetwork for each element type. Each subnetwork is then partitioned into disjoint collections of components such that each set in the refined partition can be assigned to a single module. To obtain the refined partition for each component type, one essentially solves an integer program. In the following discussion, the prototype program is developed.

The subnetworks obtained in the coarse partition are generally not connected in the graph-theoretical sense. A cluster of connected components in a subnetwork is called an *atom*, and the number of components in the cluster is called the *size* of the atom. For example, let the subnetwork consist of the trigger flip-flops in the logic design. Then, an atom of size 1 is a single trigger flip-flop that is isolated from all other similar flip-flops, and an atom of size 3 is a collection of three trigger flip-flops that are interconnected in some way but are otherwise isolated from all other flip-flops of the same type. A given subnetwork is characterized by the vectors $\mathbf{n} = (n_1, n_2, \cdots, n_t)$ and $\mathbf{s} = (s_1, s_2, \cdots, s_t)$, where the subnetwork contains n_i atoms of size s_i. The problem is to partition the collection of atoms into the fewest possible sets, each containing atoms whose total size is no greater than m. By assumption, $s_{\max} \leq m$ where $s_{\max}$ is the size of the largest atom. If any atom is found to have a size larger than m, then the atom must be divided into two or more smaller atoms so that no atomic size after division is larger than m. The problem of dividing atoms is not treated in this paper because such algorithms necessarily depend on the constraints of the technology in which the design is implemented.

A set of atoms is said to be *admissible* if the sum of sizes of the atoms in the set is less than m and the set contains n_i or fewer atoms of size s_i. The number of admissible sets is a function of m, $\mathbf{s}$, and $\mathbf{n}$, and it is finite since m, $\mathbf{s}$, and $\mathbf{n}$ are finite. Let the number of admissible sets be denoted by k, and let $A = [a_{ij}]$ be a matrix of dimension $t \times k$ that is a natural representation of the admissible sets. That is, a_{ij} is equal to the number of atoms of size s_i in the jth admissible set.

The partitioning problem can now be formulated as the following integer linear program. Minimize

$$\sum_{i=1}^{k} x_i \tag{1}$$

subject to

$$A\mathbf{x}^T \geq \mathbf{n}^T \tag{2}$$

where $\mathbf{x} = (x_1, x_2, \cdots, x_k)$ and each x_i is a nonnegative integer. In eq. (2), both $\mathbf{x}$ and $\mathbf{n}$ are treated as column vectors. The components of $\mathbf{x}$ correspond to the admissible sets of atoms that form a partition of the logic subnetwork. The value of x_j in the final solution is the number of logic modules which have atoms whose sizes are distributed as specified by the jth admissible set.

It is noteworthy that this problem is a special case of the cutting-stock problem for one-dimensional stock that has been studied by Gilmore and Gomory [1, 2]. The key difference between their approach and the approach taken here is that the special character of the logic partitioning problem leads to possible reductions that it may not be possible to make in the more general case. Gilmore and Gomory have studied the relationship of the cutting-stock problem to the knapsack problem [2, 4] and have developed algorithms for it that are similar but not identical to the algorithm presented here. It is also noteworthy that the machine-sequencing problem studied by Held and Karp [5] is identical to the problem at hand if there are no precedence constraints. In that problem, one is given a set of jobs to perform where n_i instances of the ith task must be performed and it takes t_i time units to perform each instance at a machine station. If the entire set of tasks must be performed in m units of time, then the problem is to find the minimum number of machine stations that can be used to assure completion of the tasks within the allotted time.

The inequality in eq. (2) can be treated as an equality because of a property of admissible sets. Given any $\mathbf{x}$ that minimizes eq. (1) and does not satisfy eq. (2) with equality, we can find another solution $\mathbf{x}'$ by the following process. Because $\mathbf{x}$ fails to satisfy eq. (2) with equality, there must be at least one component of $A\,\mathbf{x}^T$, say the ith component, which is greater than n_i. This means that the selection of sets corresponding to $\mathbf{x}$ results in the selection of too many of the ith atoms. To obtain the desired solution, one must find any selected set that contains the ith atom and delete at least one occurrence of that atom from the set. The resulting subset is also admissible.

We may repeat the process of deleting atoms from selected sets until we have precisely n_i atoms of size s_i for all i. The collection of subsets that results corresponds to that selected by some vector $\mathbf{x}'$ which satisfies the appropriate constraints. We have not increased the number of selected sets, and since $\mathbf{x}$ is a minimal solution, $\mathbf{x}'$ does not reduce the number of selected sets. The vector $\mathbf{x}'$ therefore is a minimal solution that satisfies eq. (2) with equality.

By way of example, let us consider the problem of partitioning atoms of sizes $\mathbf{s} = (1,3,4,5)$ with distribution $\mathbf{n} = (1,1,2,4)$ into modules of size 11. Admissible subsets include $\{1,3,4\}$, $\{1,4,5\}$, and $\{1,5,5\}$, among others, but $\{1,1,4,4\}$ is inadmissible because there is only one atom of size 1, and $\{3,4,5\}$ is inadmissible because its size is greater than 11. All together there are 18 admissible subsets, and the A

ATOMS																		n
1	1	1	1	1	1	1	1	1	1									1
3		1			1	1				1	1	1	1					1
4			1		1		1	2			1		2	1	1	2		2
5				1		1	1		2			1			1		1	2	4

A MATRIX

FIG. 1. The A matrix for a sample problem

matrix that corresponds to them is shown in Figure 1. The unique minimal solution to eq. (1) that satisfies eq. (2) with equality corresponds to the selection of subsets {3,4,4}, {1,5,5}, and {5,5}. This solution corresponds to the case $x_9 = x_{13} = x_{18} = 1$, where all other $x_i = 0$.

Although several methods are available for the solution of integer programs,[3] it is quite naive to assume that the dimensions of the problem are small enough to fit into present-day computers. With moderate-sized problems it is likely that the A matrix could contain 10^6 or more columns. In order to reduce the size of the problem and the amount of computation, it is both necessary and desirable to make use of information that is peculiar to modular partitioning problems. The type of algorithm acceptable for this problem is one which allows us to focus attention on a small subset of columns of the A matrix. To guard against the possibility of even this subset exceeding the capacity of the computer, we develop a method for generating each column of the subset one-at-a-time when needed. The reduction techniques are described in Section 4.

4. *Reduction Techniques*

Reduction techniques in integer programs involve operations on the rows or columns of the A matrix that reduce its size, and are usually accompanied by the determination of values for one or more components of the solution vector $\mathbf{x}$. In our case, the reductions take the form of the deletion of columns and rows of the A matrix. Suppose, for example, that it can be determined that the value of x_j is c. Then the jth column of A can be deleted and $\mathbf{n}$ can be replaced by $\mathbf{n}'$ where [4] $\mathbf{n}'^T = n^T - c\ A\ \mathbf{e}_j$. If any component of $\mathbf{n}'$ is 0, then all atoms of the corresponding size have been assigned to sets, and a row of the A matrix can be deleted. All columns with nonzero entries in the row to be deleted correspond to sets that cannot be selected in the reduced problem. Hence these columns can also be deleted.

Figure 2 contains an example that serves to clarify the explanation of the reduction techniques. Figure 2(a) shows an A matrix, a $\mathbf{n}$ vector appearing as a column to the left of the A matrix, and the solution vector $\mathbf{x}$ appearing as a row above the matrix. Examination of Figure 2(a) shows that we must select the fourth set five times in a minimal solution, because no other set contains the first atom. Then 5 is added to the present value of x_4, and $\mathbf{n}$ is reduced by 5 times the fourth column of A. The 0 in the first coordinate of n permits us to delete the first row from the matrix as well as all columns with nonzero entries in the first

[3] See Lawler and Wood [6] for a survey of integer programming techniques.

[4] $\mathbf{e}_j$ is the unit column vector with a 1 in the jth coordinate and 0's elsewhere.

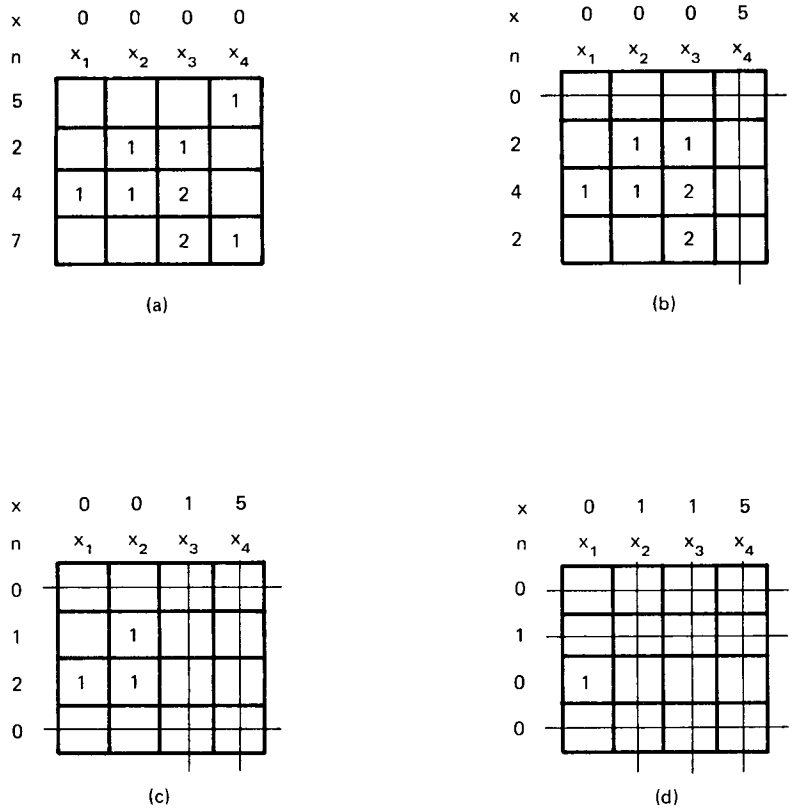

FIG. 2. A sequence of reductions applied to a sample partitioning problem

row. This is shown symbolically in Figure 2(b) by the lines passing through row 1 and column 4. Figures 2(c) and 2(d) show the results of two more reductions, and since the matrix in this example can be reduced completely, a minimal solution is $\mathbf{x} = (1,1,1,5)$.

For the remainder of this section we concentrate on rules for reductions. The example points out a trivial case.

Reduction 1. If all atoms of a particular size can be assigned to one and only one subset, then a minimal solution must contain enough copies of the subset to account for all the atoms of that size. In matrix terms, if the ith row of A contains a single nonzero entry, say in the jth column, then the value of x_j must be at least n_i/a_{ij}. (The value of x_j is actually exactly n_i/a_{ij}, and moreover, $a_{ij} \mid n_i$, because we can always satisfy eq. (2) with equality.)

A less obvious reduction can be made when a set contains two atoms whose sizes sum to m.

Reduction 2. If there exists a set, say the jth set, that contains precisely two atoms whose sizes sum to m, then this set can be selected c times, where c is the largest integer such that $\mathbf{n} - c\,A\,\mathbf{e}_j \geq 0$.

PROOF. We have to show that a solution to the reduced problem contains no more modules than a solution to the original problem. Let the atoms in the jth set have sizes y_1 and y_2. Consider any minimal solution to the original problem

in which y_1 and y_2 are not paired as frequently as they would be if Reduction 2 had been performed. Then the minimal solution must contain at least one subset with a y_1 atom and no y_2 atom, and a subset with a y_2 atom and no y_1 atom. Consider the atoms associated with y_1 in its subset. The sum of their sizes must be less than or equal to y_2, because $y_1 + y_2 = m$, and the size of any subset is limited by m. Then y_2 and the associates of y_1 can be exchanged in their respective subsets to form a new minimal solution in which y_1 and y_2 are paired together. The process can be repeated until all possible pairs of y_1 and y_2 atoms are placed together, but this solution is of the form produced by Reduction 2. Therefore, the minimal solution to a program to which Reduction 2 has been applied is also a minimal solution to the unreduced program.

For the next reduction we introduce the notion of *slack*. The *slack* of a set of atoms is by definition the difference between m and the sum of the atomic sizes. For $m = 5$, the slack of the sets {1,2} and {2,2} is 2 and 1, respectively. Reduction 2 generalizes in the following manner.

Reduction 2′. If an admissible set contains two atoms of sizes y_1 and y_2 whose sum is strictly less than m, and if there is no admissible set containing either y_1 or y_2 with less slack than the slack of the set containing just y_1 and y_2, then Reduction 2 applies to the set containing y_1 and y_2.

PROOF. The proof follows the proof of Reduction 2. In any minimal solution, either y_1 and y_2 are paired or they can be paired by interchanging y_2 with the associates of y_1.

Up to this point, we have consistently assumed that all admissible sets must be considered when applying Reductions 1, 2, and 2′. In fact, this is not true. It is possible to perform reductions by considering a small subset of the admissible sets. To show this we introduce two new definitions.

A set of atoms A is said to *cover* a set B if the following conditions hold:

(a) B can be partitioned into disjoint subsets of atoms such that the total size of the atoms in each subset is less than or equal to the size of an atom of A;

(b) each atom of A covers at most one subset of B for (a) to hold.

Intuitively, if A covers B then B can be obtained from A by splitting the atoms of A and possibly discarding some atoms. The set {3,4} covers the sets {1,2,4}, {3,2,2}, and {2,2,2}. It does not cover {3,4,1} or {5,2}. Note that the covering property is transitive.

An admissible set is said to be *maximal* with respect to the collection of admissible sets if it is not covered by another admissible set.

THEOREM 1. *Let y be the size of the largest atom. If there exists a unique maximal set containing atoms of size y among the collection of maximal sets, then Reduction 1 can be applied to that set.*

PROOF. Since y is the size of the largest atom, all sets containing atoms of size y must be covered by the unique maximal set containing atoms of size y. Following the method of proof of Reduction 2, we see that if in a minimal solution a y atom did not appear in a maximal set, then its set could be transformed into a maximal set by a series of exchanges of atoms. This is true because atoms of a nonmaximal set can be exchanged with other atoms to form a maximal set. Thus the minimal solution of the reduced problem has no more modules than the minimal solution of the unreduced problem.

Theorem 1 and Reductions 1, 2, and 2′ form the basis of the following algorithm.

Step 1. Find all pairs of atoms whose sizes sum to m. Reduction 2 can be applied to the doubleton sets containing these atoms.

Step 2. Find all atoms that cannot be paired with other atoms. Reduction 1 can be applied to the singleton sets containing these atoms.

Step 3. Find the atom of the largest size and generate the maximal sets that contain atoms of this size. If there is a unique such set, apply Theorem 1 and Reduction 1. If any reductions occur, return to step 2; otherwise, continue.

Step 4. Generate the doubleton sets, if any, that satisfy Reduction 2′ and apply the reduction. If any reductions occur, then return to Step 2; otherwise, terminate this part of the process.

It is rather interesting to note that all the steps can be carried out without constructing the A matrix. An algorithm for generating the maximal admissible sets is given in the Appendix.

The following example illustrates the use of the algorithm above. Suppose that we are to partition atoms of size $\mathbf{s} = (2,4,5,6,7,8,9)$ into sets of size 9 with $\mathbf{n} = (8,4,5,2,4,3,3)$.

We apply step 1 and find that a minimal solution contains 4 copies of each of the doubleton sets $\{7,2\}$ and $\{5,4\}$. The reduction leaves $\mathbf{n} = (4,0,1,2,0,3,3)$ and only atoms of size 2, 5, 6, 8, and 9 remain.

Application of step 2 shows that atoms of size 8 and size 9 must be assigned to singleton sets. Therefore, a minimal solution contains three sets with atoms of size 8 and three sets with atoms of size 9. After reduction, $\mathbf{n} = (4,0,1,2,0,0,0)$.

In step 3 we note that the largest remaining atoms can be assigned to the unique maximal set $\{6,2\}$. We place two copies of this set in the minimal solution and reduce again, leaving $\mathbf{n} = (2,0,1,0,0,0,0)$ and only atoms of size 2 and 5 remaining.

Step 3 can be reapplied, since a new atom has maximal size. The second application of step 3 assigns the set $\{5,2,2\}$ to the minimal solution and the problem is complete.

Although the example does not illustrate the application of step 4 explicitly, the mechanics involved are similar to those in steps 1, 2, and 3.

This completes the development of the reduction algorithm. In Section 5 we consider techniques for solving the reduced program.

5. *Branch-and-Bound Techniques for Solving Reduced Modular Partitioning Problems*

In the most general case, it is not possible to solve partitioning problems by reduction techniques alone. While it is true that standard linear program techniques can be used on the reduced program, the peculiarities of partitioning problems make it more attractive to devise special techniques for their solution.

In the present situation, given a reduced program, we can create a sequence of smaller programs such that a minimal solution of the original program is the minimal solution of one program in the sequence. To create a sequence of programs we can select an arbitrary variable, say x_j, and solve the programs that result from setting $x_j = 0, 1, \cdots$, up to the largest possible value of x_j. One need not solve every program in the sequence in order to find the minimal solution. As each new program is created, a lower bound on the number of modules in its solution is calculated. The programs that are treated in successive steps are those with the lowest lower bounds.

Suppose then that we find that the programs with $x_j = 0$ and 1 have the lowest bounds on the minimum number of modules required. Then to each of these programs we apply the algorithm of Section 4 to obtain a further reduction. If the reduction results in a complete solution that attains the lower bound, then we are finished. If a solution is found that attains a number higher than the lowest lower bound, then we can ignore all programs whose lower bounds are equal to or greater than this current solution, but we still must check those programs with better lower bounds. If a subprogram cannot be reduced completely by the reduction algorithm, then its reduction must be placed in the list of subprograms that require further analysis.

The branch-and-bound approach described above is well known [6]. For the class of modular partitioning problems, the interesting aspects of the problem concern the calculation of the lower bound and the choice of variable for the branching operation.

As mentioned earlier, a lower bound for a solution to a program is $\langle n/m \rangle$ where n is the sum of the atoms to be assigned and m is the module size. If in a subprogram we select a set c times, and that set has slack v, then a new lower bound is $\langle (n + cv)/m \rangle$. Thus lower bounds are straightforward to compute for reduced programs. The problem of selecting a variable for branching is a much more difficult problem.

It is wise to proceed along branches that tend to arrive at the minimal solution with the least amount of processing. To do this, select variables that correspond to admissible subsets that are very likely to be in the minimal solution. The following theorem guides us in this choice.

THEOREM 2. *Let y be the atom of maximal size. Then the minimum solution which is constrained to place as many y atoms as possible in maximal admissible sets has no more sets than the unconstrained minimal solution.*

PROOF. In any solution in which a y atom is not placed in a maximal admissible set, the atoms associated with y can be exchanged with atoms of other sets so that the set containing y becomes maximal.

The implication of Theorem 2 is that we should branch on variables corresponding to maximal admissible sets that contain the largest atom. A complete partitioning algorithm, therefore, is composed of the reduction techniques outlined in Section 4, the algorithm for generating maximal sets described in the Appendix, standard branch-and-bound techniques, and a criterion imposed by Theorem 2 on the selection of the branch variables.

To show how the branch-and-bound algorithm works, consider the following example. Let atoms have sizes $\mathbf{s} = (1,3,4,5,7)$, $\mathbf{n} = (1,2,1,4,1)$, and let $m = 13$. The problem cannot be reduced by any of the reductions of Section 4; thus it is necessary to branch and bound. Since the atomic sizes sum to 38, the initial lower bound on the number of modules is 3.

To obtain the first branch variable we have to generate the maximal sets that contain atoms of size 7. These are found to be the sets $\{7,3,3\}$ and $\{7,5,1\}$. We arbitrarily select the variable corresponding to $\{7,3,3\}$ to be the first branch variable. We can select this set either once or not at all, so that two new subproblems are candidates for solution. In this case there is no slack in the subset, and therefore both subproblems have the same lower bound for a minimal solution, 3 modules.

Let us consider the subproblem in which $\{7,3,3\}$ is selected once. The $\mathbf{n}$ vector

for this problem is (1,0,1,4,0). The largest atom has size 5, and it is a member of the unique maximal set {5,5,1} We apply Theorem 1 and note that we must select {5,5,1}. The problem reduces to a problem with $\mathbf{n} = (0,0,1,2,0)$. Further reductions apply and force us to select sets {5,5} and {4} for our solution. Since there are four modules in this solution and the lower bound on the unchecked branch is 3, we must backtrack.

In the branch with lower bound 3, we do not select set {7,3,3}, which in turn forces us to select {7,5,1}, by Theorem 2. Its selection reduces to the problem in which $\mathbf{n} = (0,1,2,3,0)$. In this problem the largest atomic size is 5, and it is a member of the unique maximal set {5,5,3}. We apply Theorem 1 once, reduce, and find that the remaining elements form the set {5,4,3}, which satisfies Theorem 1. In taking the current branch, we have selected the three sets {5,5,3}, {5,4,3}, and {7,5,1} and have found a solution that satisfies the lower bound for a minimal solution. This solution is therefore minimal.

6. *Some Final Remarks*

The partitioning process described here is guaranteed to find a minimal solution to the modular partitioning problem, but the minimal solution is not necessarily a unique minimal solution. In a design-automation environment, the partitioning algorithm ought to be used to find a first-order approximation to the assignment of gates to modules. The first-order approximation might then be improved by algorithms that take into consideration the constraints peculiar to a particular technology. If the partition found by the algorithm is to be used as a first-order approximation, then it may be worthwhile to produce suboptimal partitions in order to decrease computation. One convenient way of finding good suboptimal solutions quickly is to apply the algorithm given in this paper with limited backtracking.

It has been mentioned in passing that the algorithm solves a special case of the cutting-stock problem. The algorithm can be applied to one-dimensional cutting-stock problems provided that the cost function depends linearly on the waste. An important facet of cutting-stock problems is that there may be several stock lengths from which to select, whereas we have considered that class in which there is only a single length. It may be the case that the algorithm presented here can be generalized to solve the multiple-stock-length problem and similar variants of the cutting-stock problem with less computation than with presently known algorithms.

ACKNOWLEDGMENT. The stimulus for the problem treated in this paper was provided by Drs. Karl Levitt and Abraham Waksman of Stanford Research Institute. The author is particularly indebted to Mr. Charles L. Jackson of Signatron, Inc., Lexington, Mass., for many suggestions and comments that contributed to the development of the partitioning algorithm.

APPENDIX. An Algorithm for Generating Maximal Admissible Sets

Let each maximal admissible set be ordered so that $s_i < s_j$ if $i < j$. Then we can define a lexicographic ordering of the maximal admissible sets in one of two natural ways, ascending or descending order. The algorithm described in this appendix generates the maximal admissible sets in ascending lexicographic order.

The algorithm given in the text of the paper requires the generation of maximal admissible sets that contain the largest atom. If the largest element has size s_t, then a suitable algorithm is one which generates all the maximal admissible sets of size $m - s_t$ under the constraint that one fewer atom of size s_t is available than is available in the original problem. This gives the desired result because all maximal admissible sets that contain at least one atom of size s_t are maximal admissible sets of size $m - s_t$ in the reduced problem, and conversely. (Any set that covers a set in the reduced problem also covers it in the unreduced problem, and conversely.)

The technique used to generate the maximal admissible sets is a backtracking technique that is similar to the column-generation procedures of Gilmore and Gomory [2]. The method involves generating a maximal admissible set of total size r where $r < m$. Then an atom of size $s \leq m - r$ can be added to the set to make a maximal admissible set of size $r + s$ if and only if the addition of the atom of size s does not create a subset of atoms whose size is precisely the size of an unassigned atom. Special processing is required in the last step of the algorithm if the slack is nonzero.

The algorithm requires that there be two vectors N and L of dimension t, where t is the number of distinct atomic sizes. N is a vector whose ith component N_i contains the number of unassigned atoms of size s_i. The vector L records in its ith position the number of atoms of size s_i that have been selected. The indices are arranged so that $s_i < s_j$ if $i < j$. Hence the largest atomic size is s_t. A third vector P is used to record the sizes of subsets of selected atoms in order to guarantee that the size of no subset is exactly equal to that of an unselected item. The sth entry of P is set to a positive number if there is at least one subset of selected atoms whose size totals s, and to 0 otherwise. The dimension of P is s_t, the size of the largest atom.

The algorithm proceeds as follows.

Step 1. Initialize the entries of L and P to 0, and set the values of N to the components of the vector $\mathbf{n} = (n_1, n_2, \cdots, n_t)$.

Sets are formed by adding atoms of successively smaller sizes. The parameter i is used to index in descending order through the atomic sizes starting with s_t and ending with s_1. The parameter q represents the amount of slack in the current assignment. In general, after inspecting atoms of size s_i and possibly making an assignment of one or more atoms of this size to a subset, we compute a new q and attempt to find maximal admissible assignments for atoms of size s_{i-1} or less to subsets of size q. To control this process we initialize i and q in step 2.

Step 2. Initialize i to the value t, and q to the value m.

Step 3. In this step we generate new maximal admissible sets that contain 0, 1, 2 $\cdots$ atoms of size s_i, in addition to the distribution of atoms described by the present entries of L. The maximum number of atoms of size s_i to be assigned is the smaller of N_i and $[q/s_i]$.

There are two special cases to be considered. If q is exactly equal to s_i, then at most one atom of size s_i can be assigned to the current maximal admissible set. Since this assignment covers all those assignments for which 0 atoms of size s_i are added to the current set, it is not necessary to investigate further assignments in which 0 atoms of s_i are added to the current set. This case is recognized in step 3b.

The second special case occurs when $i = 1$. For this case the assignment that assigns the maximum number of atoms of size s_1 to the subset covers all assignments which assign fewer atoms of size s_1. Moreover, the exact slack in the admissible

assignment is known at this point and can be taken into account. If after assigning atoms of size s_1, the resulting set is found to be maximal, it is one of the maximal admissible sets to be reported by this algorithm. Processing for the second special case occurs in steps 3i–3k.

The explicit processes for step 3 are outlined as follows:

Step 3a. If $i = 1$, then go to step 3i.

Step 3b. (Generate sets with atoms smaller than s_i.) If $q \neq s_i$ then apply step 3 with i decreased by 1. In the recursive application implied here we assume that control returns to this point with the value of i unchanged.

Step 3c. (Test to see if no more atoms of size s_i can be added.) If $N_i = 0$ or $q < s_i$, then go to step 3h for backtracking.

Step 3d. (Add one more atom of size s_i to the current set of atoms.) Increase L_i by 1, decrease N_i by 1, and decrease q by s_i. L_i now contains the number of selected atoms of size s_i, and N_i contains the number of unassigned atoms of size s_i.

Step 3e. (Update P vector.) Scan the P vector starting at position $s_t - s_i$, indexing downward to $s_i + 1$. If a nonzero entry is encountered at index j, then some subset of previously selected atoms has total size j. Since we have added an atom of size s_i, there is a new subset of atoms of size $j + s_i$. To reflect this fact, inspect P_{j+s_i} and set it nonzero if it is 0. To simplify the backtracking process, the quantity entered in P_{j+s_i} is the number "$+i$." After completing the scan of the P vector, "$+i$" is entered at P_{s_i} if $L_i = 1$, to reflect that a singleton set of size s_i has been created for the first time.

At the completion of this step there is a nonzero quantity in P_j for $s_i \leq j \leq s_t$ if and only if a subset of selected atoms has total size j.

Step 3f. (Guarantee that current subset is maximal.) If some subset of atoms has been created that is precisely equal in total size to the size of an unassigned atom, then the current assignment is not maximal. To check this condition, scan the N vector from N_{i+1} to N_t. If N_j is greater than 0 for some j, check P_{s_j}. If it is nonzero, then go to step 3h for backtracking. The reason is that there is a subset of atoms of total size s_j and at least one unassigned atom of size s_j. Hence, the current subset is not maximal.

Step 3g. If $q = 0$, a maximal set with 0 slack has been found. In this case, go to step 3k. Otherwise, go to step 3b.

Step 3h. (Backtrack.) If $i = t$ then terminate. Set N_i to $N_i + L_i$. Set q to $q + L_i \cdot S_i$. Set L_i to 0. Scan the P vector from s_i to t. If P_j is "$+i$," set P_j to 0. This returns P to its state prior to the selection of atoms of size s_i. Increase the index of i and return control to the point from which step 3 was applied.

Step 3i. (Process for $i = 1$.) Set L_1 to the minimum of N_1 and $[q/s_1]$. Set N_1 to $N_1 - L_1$. Set q to $q - L_1 \cdot s_1$. We have now assigned L_1 atoms of size s_1 to the current set.

Step 3j. (Guarantee that subset is maximal.) The current assignment is maximal if and only if there is no subset of atoms whose total size is equal to the size of an unassigned atom or is no more than q units less than the size of an unassigned atom. To determine this condition we update the P vector. First we scan P backwards from $P_{s_t - s_1}$ to P_{s_1+1}. If $P_j \neq 0$ then for $1 \leq k \leq L_1$, examine $P_{j+k \cdot s_1}$ and set each one to 1 if it is 0. After scanning the P vector, examine $P_{k \cdot s_1}$ for $1 \leq k \leq L_1$ and set $P_{k \cdot s_1}$ to 1 if it is 0.

Now scan the N vector from N_2 to N_t searching for nonzero entries. For each nonzero N_j in the N vector, check the entries $P_{s_j - k}$, $1 \leq k \leq q$. If there exists a P_{s_j-k} that is nonzero, then there is an unassigned atom of size s_j that covers the subset of size P_{s_j-k} within the allowed slack q. If this is the case, go to step 3h for backtracking, since the current subset is not maximal.

Step 3k. Output the current state of the vector L. It is a maximal assignment. Go to step 3h. (This completes the algorithm.)

The use of the algorithm is illustrated by an example from the text. Here we have $\mathbf{s} = (1,3,4,5,7)$, $\mathbf{n} = (1,2,1,4,1)$, and $m = 13$. The problem is to generate all maximal subsets that contain at least one atom of size 7. To solve the problem

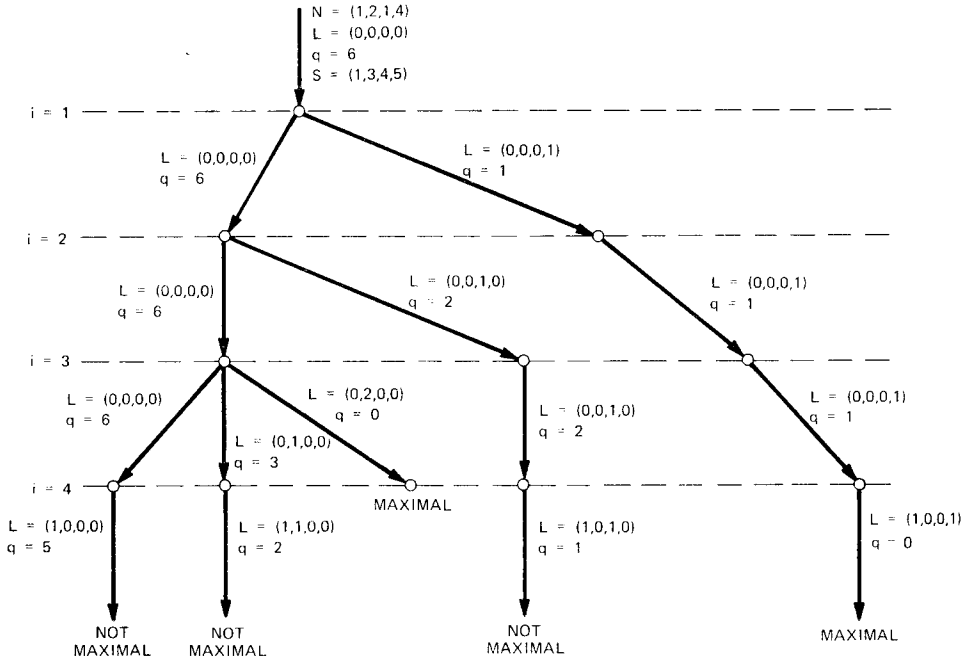

FIG. 3. The decision tree resulting from the application of the maximal admissible algorithm

we generate the maximal sets of size 6 using $\mathbf{s} = (1,3,4,5)$ and $\mathbf{n} = (1,2,1,4)$. In steps 1 and 2 of the algorithm we initialize P to (0,0,0,0,0) and L to (0,0,0,0).

The decision tree that results from following the algorithm is shown in Figure 3.

It is worthwhile to remark on the efficiency of the maximal subset algorithm. The backtrack nature of the algorithm leads to a natural upper bound on the computational complexity of the algorithm. This bound is exponential in the number of atomic sizes. In spite of the pessimistic upper bound, the algorithm can lead to efficient implementations for two reasons. First, the pruning characteristic of backtrack algorithms often leads to substantially less computation than is indicated by the overly pessimistic bounds. Second, for the modular partitioning application, the problem parameters are sufficiently small that the exponential upper bound on complexity is not a serious matter. This is particularly true when the atomic sizes are nearly as large as the module size. In the cutting-stock situation, quite the opposite may be true because the problems become much larger, and the exponential nature of the algorithm is a severe obstacle to practical implementation.

REFERENCES

1. GILMORE, P. C., AND GOMORY, R. E. A Linear programming approach to the cutting-stock problem. *Oper. Res. 9*, 6 (Nov.–Dec. 1961), 849–859.
2. —— AND ——. A linear programming approach to the cutting-stock problem—Pt. II. *Op. Res. 11*, 6 (May–June 1963), 863–888.
3. —— AND ——. Multi-stage cutting stock problems of two and more dimensions. *Oper. Res. 13*, 1 (Jan.–Feb. 1965), 94–120.
4. —— AND ——. The theory and computation of knapsack functions. *Oper. Res. 14*, 6 (Nov.–Dec. 1966), 1045–1074.
5. HELD, M., AND KARP, R. M. A dynamic programming approach to sequencing problems. *SIAM J. Numer. Anal. 10*, 1 (Mar. 1962), 196–210.

6. LAWLER, E. L., AND WOOD, D. E. Branch and bound methods: A survey. *Oper. Res. 14*, 4 (July–Aug. 1966), 699–719.
7. ——. Electrical assemblies with a minimum number of interconnections. *IRE Trans. EC-11* (Feb. 1962), 86–88.
8. ——, LEVITT, K. N., AND TURNER, J. B. Module clustering to minimize maximum delay in digital networks. *IEEE Trans. EC-17* (Feb. 1969).

RECEIVED OCTOBER, 1968