J.A. Lukes

Combinatorial Solution to the Partitioning of General Graphs

Abstract: This paper reviews a dynamic programming procedure for the partitioning of connected graphs with integer-weighted nodes and positive valued edges. The upper bound on the number of feasible partitions generated using this technique is shown to grow factorially in the number of graph nodes. The use of graph properties is then introduced to reduce the number of feasible partitions generated in the determination of the optimal partition. Depending upon the structure of the graph, the use of these properties can cause a significant reduction in the computation time and storage space required to partition the graph.

Introduction

Given a graph G with integer-weighted nodes and nonnegative edge values, the *partitioning* of G consists of the allocation of the nodes in G to clusters such that the node weights of each cluster do not exceed a given maximum (i.e., a weight constraint). The objective of such a partitioning is to assign nodes to clusters so that the sum of the edge values joining nodes in different clusters is minimal.

Partitioning problems of this type are encountered in the assignment of logic blocks to circuit cards in computer hardware design [1, 2] and in the assignment of computer information to physical blocks of storage [3, 4, 5].

In this paper we review a dynamic programming technique for generating the optimal partition of an n-node connected graph. This technique consists of first assigning the labels $1, 2, \cdots, n$ to the graph nodes. On the jth step of the procedure, the feasible partitions of the subgraph with nodes $1, 2, \cdots, j$ are generated from the feasible partitions of the subgraph containing nodes $1, 2, \cdots, j-1$. The optimal graph partition is then a feasible partition of the entire graph.

We show that an upper bound on the number of feasible partitions generated on the jth step is of order $j!$. We then describe a concept that reduces this upper bound from order $j!$ to $x_j\ (x_j!)\ W^{x_j}$ where x_j is the number of nodes with labels less than j connected to nodes with labels j or greater and W is the weight constraint of a cluster. For a graph with values of x_j and $W \ll j$, the reduction in the number of feasible partitions that must be kept on the completion of the jth step of the dynamic programming procedure is significant.

We further prove that a graph with cutpoints (hence two or more blocks) can be partitioned by generating the feasible partitions of each block independently, then merging these partitions to find the optimal partition of the entire graph. This procedure reduces the upper bound in the number of partitions generated to find the optimal from $n!$ to $n_k!$, where the entire graph has n nodes and the kth block n_k nodes.

The importance of these results lies not only in the technique but in the fact that reference [6] contains an implementation of the procedure whose growth in computation time and storage space on the jth step is of the order $p_j\ (\log_2 p_j)$, where p_j represents the number of feasible partitions generated on the jth step. Consequently, we have derived an upper bound on computation time for the partitioning of connected graphs.

Problem definition and restrictions

Given a graph $G = (V, E)$ with node set V and edge set E as in Fig. 1. A *partition* of G is a collection of k clusters of nodes $\{c_i\}$ $(i = 1, 2, \cdots, k)$ such that

$$\bigcup_{i=1}^{k} c_i = V,$$

$$c_i \bigcap c_j = \varnothing \text{ for all } i \neq j.$$

Each node x has an integer weight w_x and each edge (x, y) a positive value v_{xy}. As indicated we impose a weight constraint W on each cluster [5].

An *optimal partition* is defined as some partition of G, $P_G\ (\text{opt}) = \{c_1, c_2, \cdots, c_k\}$, with the property that each cluster c_i satisfies the weight constraint,

$$\sum_{x \in c_i} w_x \leq W,$$

and in which

$$\sum_{i=1}^{k} \sum_{x,y \in c_i} v_{xy} \text{ is maximal.}$$

We impose several restrictions on the problem investigated here. The nodes of the graph must have nonnegative integer weights, and the values of the edges are positive. Another restriction is that the graph be connected (i.e., a path exists from any node in the graph to all other nodes in the graph). Given a disconnected graph G, each connected subgraph of G is partitioned independently, i.e., each cluster consists of nodes from the same connected subgraph.

The final restriction is that a multigraph must be transformed into a graph by the following modification. If more than one edge exists between two nodes x and y, then the several edges joining x and y are replaced by one with a value v_{xy} equal to the sum of the values of those edges.

General partitioning algorithm

In this section we review [3] a partitioning technique that has as its basis a dynamic programming procedure similar to that used in the solution of the one-dimensional knapsack problem [7]. The similarity between that problem and the partitioning problem becomes apparent when their properties are compared.

The one-dimensional knapsack problem is that faced by a mountain climber who has a knapsack that can carry a maximum weight of W pounds and a number of different items he wishes to carry in the knapsack. Each item has, as well as a weight, a value associated with it, and the sum of the weights of the items exceeds W. A mathematical statement of this problem is the following:

One-dimensional knapsack problem

Let w_x = weight of item x $(x = 1, 2, \cdots, n)$,
v_x = value of item x,
W = capacity of knapsack.

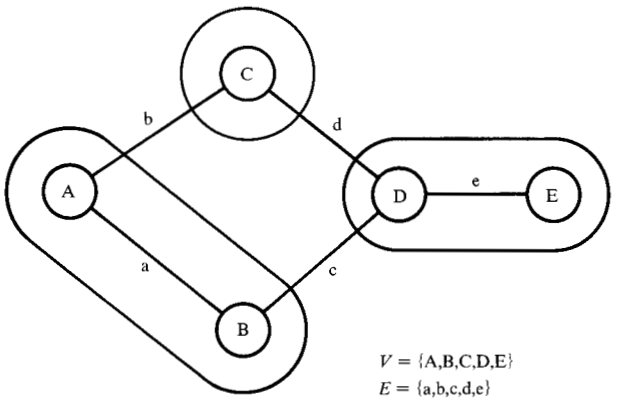
Maximize $\sum_{x=1}^{n} v_x \Gamma_x$ subject to the constraint

$$\sum_{x=1}^{n} w_x \Gamma_x \leq W$$

$$\Gamma_x = \begin{cases} 1 & \text{if item } x \text{ is in the knapsack} \\ 0 & \text{otherwise.} \end{cases}$$

The mathematical statement of the partitioning problem given below is seen to be an extension of the one-dimensional knapsack problem to the distribution of interconnected, weighted items into many knapsacks or clusters, each of capacity W:

Let w_x = the weight of node x
v_{xy} = the value of edge (x, y)
n = the number of graph nodes
N = the number of clusters in the partition
W = the weight constraint.



Partition = $\{c_1, c_2, c_3\}$
c_1 = (A,B)
c_2 = (C)
c_3 = (D,E)

Figure 1 A partition of graph $G = (V, E)$ with a weight constraint of 2 and all nodes having unit weight.

Then maximize

$$\sum_{i=1}^{N} \sum_{x=1}^{n} \sum_{y=1}^{n} v_{xy} \Gamma_{xi} \Gamma_{yi}$$

subject to the constraints

$$\sum_{x=1}^{n} w_x \Gamma_{xi} \leq W \text{ for } i = 1, 2, \cdots, N$$

$\Gamma_{xi} = 1$ if node x is in cluster i and 0 otherwise.

In order to pose the partitioning problem as one suitable for solution by dynamic programming, the graph is first labeled [5]. The jth step, or stage, of the partitioning process generates the feasible partitions of the subgraph consisting of those nodes with labels $\leq j$. These partitions correspond to the states of the jth stage. The partitions of the subgraph consisting of those nodes with labels no greater than j are created from the partitions of the $(j-1)$th by adding node j to these partitions within the limitations imposed by the weight constraint. The policy decision is the determination of which partitions of step $j-1$ can have node j added to one of their clusters to generate feasible partitions of step j.

The following definitions are useful in describing the basic partitioning process.

A *k-adjacency of node j* is defined as a feasible partition generated on step j with a cluster containing node j whose weight is k. An example of a 2-adjacency of node 4 is shown in Fig. 2. A k-adjacency of a node is not unique, as demonstrated in Fig. 2.

We have already imposed a weight constraint on each cluster of a partition. A further constraint [5, 6] imposed on each cluster is that the nodes contained within it form a connected subgraph.

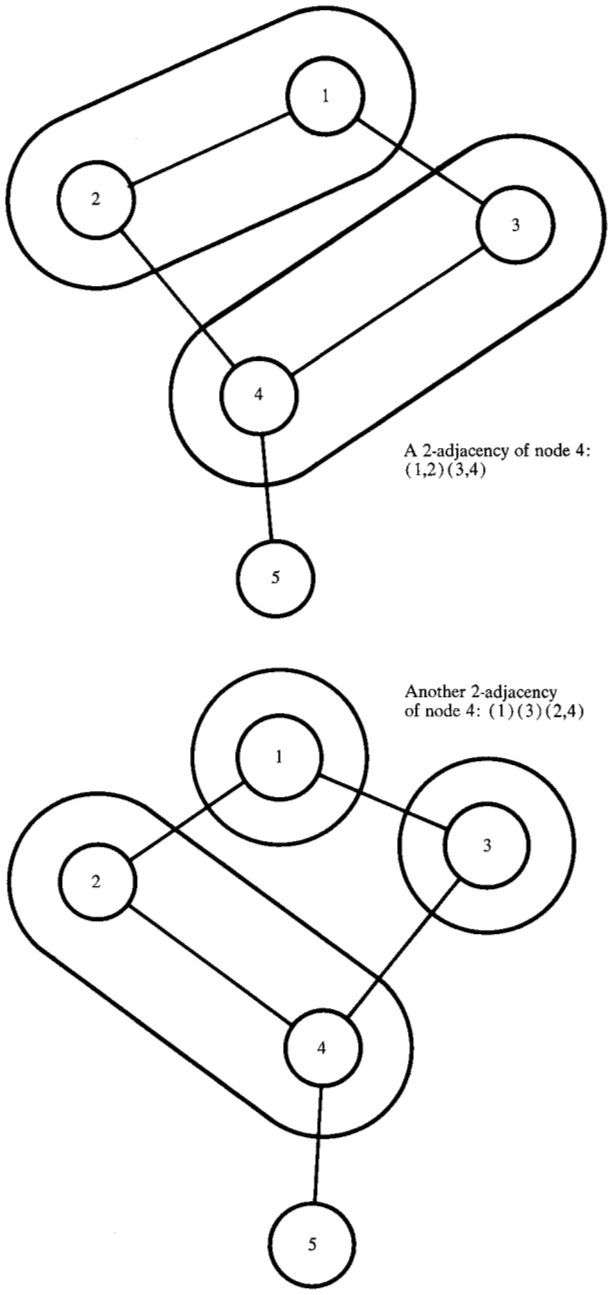


Figure 2 A k-adjacency of node j.

A *feasible partition* of a graph G is therefore defined as a partition whose clusters each satisfy the following properties:

1. The sum of the weights of the nodes in a cluster must not exceed W, the weight constraint.
2. The nodes in a cluster must form a connected subgraph of G.

In the process of partitioning a connected graph G the only partitions that need to be generated are those whose clusters have a weight not exceeding the weight constraint and that contain nodes that may form a connected subgraph of G. In generating the set of feasible partitions on step j, the weight constraint is easily tested by adding node j to each cluster of some partition generated on step $j-1$ and rejecting the resulting partitions with a cluster whose weight exceeds W. A newly created feasible partition must not only have clusters that satisfy the weight constraint, but its clusters must also contain nodes that presently form a connected subgraph, or form a connected subgraph with the addition of one or more nodes with labels greater than j. Let this restriction be called the *connectivity constraint*. In order to recognize some cluster of a feasible partition of step $j-1$ to which node j can be added without violating the connectivity constraint, we introduce the concept of the connected set.

The *connected set* for a node j is defined as that set of nodes that, if one or more of them appears in a cluster of a partition generated on step $j-1$, guarantees that the addition of node j to that cluster may on some step $m > j$ form a connected subgraph. The properties of a node i in the connected set for node j, denoted by CONN (j), are:

1. $i < j$;
2. node i
   (a) is adjacent to node j, or
   (b) lies on a path $i, x_1, x_2, \cdots, x_r, j$,

where

$$\sum w_x \leq W$$

and

$$x \in \{i, x_1, \cdots, x_r, j\}$$
$$x_1 > j,$$
$$x_2 > j,$$
$$\vdots$$
$$x_r > j.$$

The second property guarantees that a partition with a cluster containing two nodes i and j that are presently disconnected, but become connected if nodes $x_1, x_2, \cdots, x_r$ are added to that cluster, is generated on step j.

An illustration of the connected set associated with each node of the given graph is shown in Fig. 3.

• *Dynamic programming procedure*

We now describe the dynamic programming procedure to form the optimal partition of the graph. The graph is assumed to have been labeled. It should be noted that the particular labeling used affects the partitioning process; however, no general labeling technique is known that yields the minimal computation time.

The $(j-1)$th step of the partitioning algorithm has as its states the feasible partitions of the subgraph consisting

of those nodes with labels $<j$, denoted by P_{j-1}. We then add node j to all partitions in P_{j-1} with a cluster satisfying the criteria:

1. The addition of node j does not cause the cluster weight to exceed the weight constraint;
2. There exists a node in CONN (j), the connected set for node j, in the cluster.

The resulting partitions are the states of step j, P_j.

The value of each partition equals the summation of the values of the edges within the clusters of the partition.

The dynamic programming process is outlined below:

Step 1
For each node j find the connected set CONN (j).

Step 2
$j = 0$, $P_0 = \varnothing$

Step 3
$j = j + 1$

Let the weight of node j be denoted by w_j. Set P_j consists of the following partitions:

(a) Form the k-adjacencies for $k = w_j$. Each such k-adjacency is formed by adding a cluster containing node j alone to the set of clusters of a partition in P_{j-1}.
(b) For $k = w_j + 1, w_j + 2, \cdots, W$ form the k-adjacencies of node j. Only those partitions in P_{j-1} with at least one cluster containing a node in CONN (j) can be used in the generation of these partitions.

Step 4
Go to Step 3 until $j = n$ for an n-node graph.

Step 5
Select the maximal-valued feasible partition in P_n. This is the optimal-valued partition of the graph.

An example of the use of this algorithm is given in Fig. 4. The results of each step of the algorithm are shown in tabular form. Each row of this table corresponds to a step of the procedure; the kth column and jth row of the table contain the k-adjacencies of node j.

• *Growth rate for dynamic programming procedure*
Although the dynamic programming procedure just described generates an optimal partition of a graph without resorting to total enumeration, the question arises as to the number of feasible partitions possible for a connected graph. Reference [6] shows the growth in computation time to vary as

$$\sum_{j=1}^{n} np_j(\log_2 p_j), \text{ where } p_j = |P_j|,$$

and the storage requirements vary as np_j, where n equals the number of graph nodes.

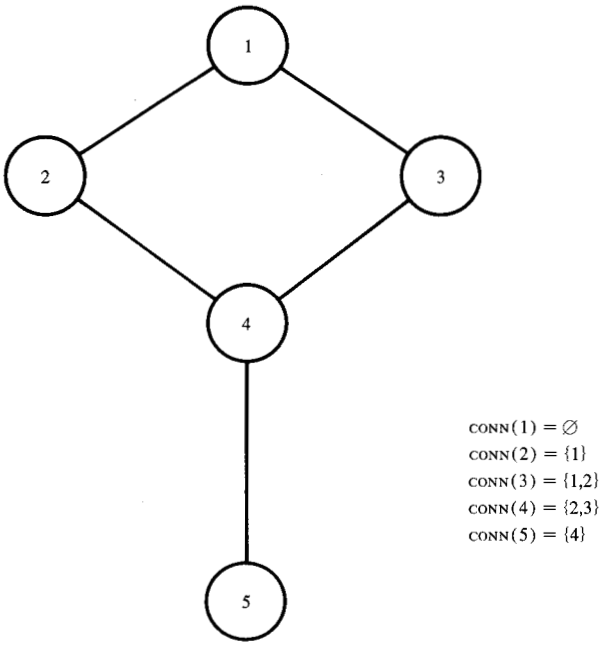


Figure 3 Connected set for a graph in which all nodes have unit weight and $W = 3$.

Consider first the growth in the cardinality of P_j for total enumeration. To generate this number we assume that the graph is complete (every pair of its nodes are adjacent) so that no combination of nodes in some cluster is disconnected. Also, no weight constraint is imposed upon the clusters. The upper bound on the size of P_j is the number of ways in which j distinct objects can be distributed in i nondistinct cells, where i varies from one to j and is given by the Stirling number of the second kind, $S(j, i)$. Thus

$$p_j < \sum_{i=1}^{j} S(j, i), \text{ where } p_j = |P_j|.$$

A closed form for this summation does not appear to exist, but an upper bound results from the recurrence relationship:

$$p_j < (1 + c_j)\, p_{j-1},$$

where $p_j = |P_j|$ and $c_j = |\text{CONN } (j)|$.

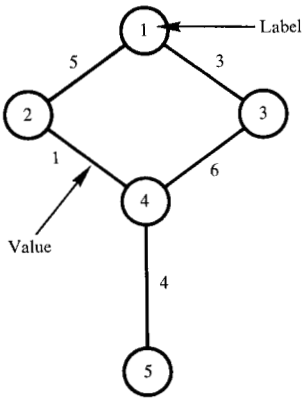
This relationship is derived from the fact that P_j is made up of two subsets:

1. The 1-adjacencies of P_j, of which there are p_{j-1},
2. The k-adjacencies of P_j, where $k > 1$.

The size of the latter set is bounded by $c_j\, p_{j-1}$, because each node in CONN (j) can generate no more than p_{j-1} partitions of p_j:

For the complete graph $|\text{CONN}|(j)| = j - 1$; therefore

$$p_j < j!.$$



Node	Connected set
1	∅
2	{1}
3	{1,2}
4	{2,3}
5	{4}

Step	k-adjacencies 1	2	3
1	(1) = 0		
2	(1)(2) = 0	(1, 2) = 5	
3	(1)(2)(3) = 0 (1, 2)(3) = 5	(1, 3)(2) = 3 (1)(2, 3) = 0	(1, 2, 3) = 8
4	(1)(2)(3)(4) = 0 (1, 2)(3)(4) = 5 (1, 3)(2)(4) = 3 (1, 2, 3)(4) = 8	(1)(2, 4)(3) = 1 (1)(2)(3, 4) = 6 (1, 3)(2, 4) = 4 (1, 2)(3, 4) = 11	(1, 2, 4)(3) = 6 (1, 3, 4)(2) = 9 (1)(2, 3, 4) = 7
5	(1)(2)(3)(4)(5) = 0 (1, 2)(3)(4)(5) = 5 (1, 3)(2)(4)(5) = 3 (1, 2, 3)(4)(5) = 8 (1)(2, 4)(3)(5) = 1 (1)(2)(3, 4)(5) = 6 (1, 3)(2, 4)(5) = 4 (1, 2)(3, 4)(5) = 11 (1, 2, 4)(3)(5) = 6 (1, 3, 4)(2)(5) = 9 (1)(2, 3, 4)(5) = 7	(1)(2)(3)(4, 5) = 4 (1, 2)(3)(4, 5) = 9 (1, 3)(2)(4, 5) = 7 (1, 2, 3)(4, 5) = 12 thus optimal partition is (1, 2)(3, 4, 5) value = 15	(1)(2, 4, 5)(3) = 5 (1)(2)(3, 4, 5) = 10 (1, 3)(2, 4, 5) = 8 (1, 2)(3, 4, 5) = 15

Figure 4 Dynamic programming procedure for graph in which $W = 3$ and all nodes have unit weight.

In reference [6] it is shown that a lower bound on the number of feasible partitions grows as f^j, where $1 < f < 2$.

Use of graph properties in partitioning

The computation and storage requirements of the dynamic programming procedure grow factorially in the number of graph nodes, limiting the utility of this procedure should it simply generate all feasible partitions. In this section we introduce several concepts that take advantage of properties of graphs to significantly reduce the computation time and storage requirements for certain classes of graphs by reducing the number of feasible partitions generated.

The first concept discussed is that of the *isolated set*. Using this concept we show that the growth in the number of partitions generated on step j of the partitioning process is dependent only on the number of nodes with labels less than j connected to nodes with labels greater than or equal to j and not on the number of nodes j. The second concept takes into account the existence of cutpoints and blocks in a graph.

• *The isolated set*

A node i is defined to be an element of the isolated set for node j, denoted ISOL (j), if it satisfies the following properties:

1. The label i is less than j.
2. Node i is not adjacent to any node with label $\geq j$.

Figure 5 illustrates this definition.

Several properties of the isolated set result from this definition.

1. The size of ISOL (j) is independent of the weight constraint.
2. The connected set and the isolated set for any node j are mutually exclusive. This property follows from the definition of each set.
3. Let CONN $(j)_{max}$ denote the set of nodes with labels less than j that are not elements of ISOL (j). Then CONN $(j)_{max} = \{i | i = 1, 2, \cdots, j-1\} -$ ISOL (j) and is independent of the weight constraint.
4. CONN $(j)_{max} \supseteq$ CONN (j) for every weight constraint W. Note that CONN (j) is a function of W and CONN $(j)_{max}$ is not.

The growth in the size of ISOL (j) is a nondecreasing function of k, as we show in the following theorem:

Theorem ISOL $(j) \subseteq$ ISOL $(j+1)$.

Proof Assume that ISOL $(j) \supset$ ISOL $(j+1)$. Then there exists at least one node i that is in ISOL (j) but not in ISOL $(j+1)$. By definition i is adjacent to no node with label greater than j; consequently it is adjacent to no node with label greater than $j+1$, contrary to the assumption. Therefore, ISOL $(j) \subseteq$ ISOL $(j+1)$.

We now show that the concept of the isolated set can be used to modify the partitioning process so that only a subset of the feasible partitions of a step of the process must be generated on that step.

We have defined the set CONN $(j)_{max}$ as those nodes in the set $\{1, 2, \cdots, j-1\}$ not in ISOL (j). Those partitions generated on step $j-1$ can then be separated into disjoint subsets where each partition in a subset has the same distribution of the nodes from CONN$(j)_{max}$ in its clusters as the other partitions in that subset. Furthermore, any cluster in the partition that contains one or more nodes from CONN $(j)_{max}$ has the same weight as the comparable cluster in each of the other partitions in the subset. An example of two partitions in the same subset is

(1, 2) (4) (3, 5) and (1) (3) (4) (2, 5),

where CONN $(6)_{max} = \{4, 5\}$ and each node has unit weight.

Any two partitions in the same subset are defined as *similar partitions*. We define the *dominant partition* of a set of similar partitions as that partition of maximal value. If two or more partitions are similar and have equal maximal values, then one is arbitrarily chosen as the dominant partition.

The reason for separating the set of feasible partitions of step $j-1$, P_{j-1}, into sets of similar partitions is that all but the dominant partition can be deleted from each subset of P_{j-1}. We show in the appendix that this result reduces the upper bound on the number of feasible partitions generated on step j from $j!$ to

$$x_j(x_j!)(W^{x_j})$$

where $x_j = |\text{CONN}(j)_{max}|$ and W is the weight constraint.

For small values of W and x_j this result represents a significant reduction in the number of feasible partitions that must be generated on the jth step of the partitioning process. In the appendix we prove the isolated set theorem, showing that all but the dominant partitions of step $j-1$ can be deleted from P_{j-1}.

An illustration of the results of this theorem is given in Fig. 6.

We note that the size of the isolated set for the nodes of a graph is a function of the labeling assigned to the graph. No labeling technique is yet known for minimizing the number of feasible partitions, although [6] describes heuristic techniques for labeling.

• *Cutpoints*

A *cutpoint* of a connected graph $G = (V, E)$ is defined as a node c such that $V - \{c\}$ is the node set of nontrivial disconnected graph G'. A nonseparable graph is connected, nontrivial, and has no cutpoints. A *block* of a graph G is a maximal nonseparable subgraph of G. An illustration of these definitions is given in Fig. 7.

If a connected graph G has more than one block, the following theorem proves that it is valid to find the opti-

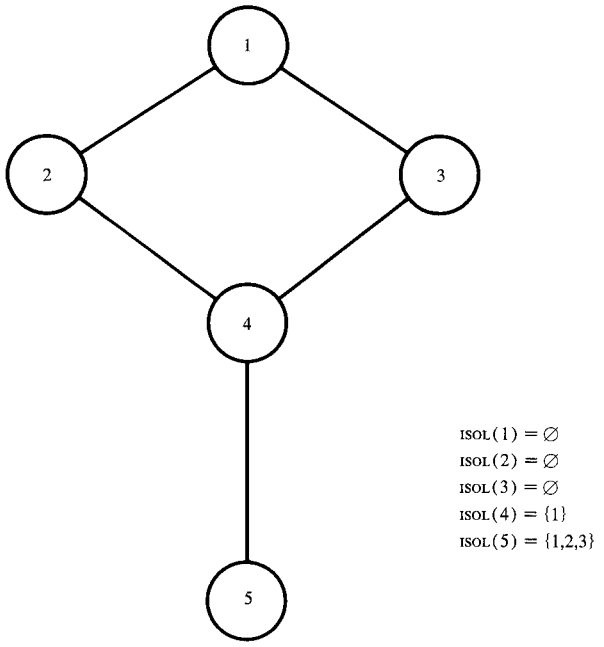
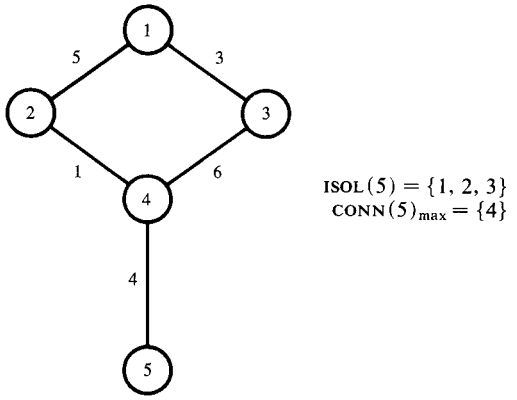


Figure 5 Isolated set.

Figure 6 Application of isolated set theorem.



From Fig. 4 the sets of similar partitions in P_4 are:

S_1		S_2	
(1)(2)(3)(4)	value = 0	(1)(3)(2, 4)	value = 1
(1, 2)(3)(4)	value = 5	(1)(2)(3, 4)	value = 6
(1, 3)(2)(4)	value = 3	(1, 3)(2, 4)	value = 4
(1, 2, 3)(4)	value = 8	(1, 2)(3, 4)	value = 11

S_3	
(3)(1, 2, 4)	value = 6
(2)(1, 3, 4)	value = 9
(1)(2, 3, 4)	value = 7

Dominant partitions of P_4 are:

Set	*Dominant partition*	
S_1	(1, 2, 3)(4)	value = 8
S_2	(1, 2)(3, 4)	value = 11
S_3	(2)(1, 3, 4)	value = 9

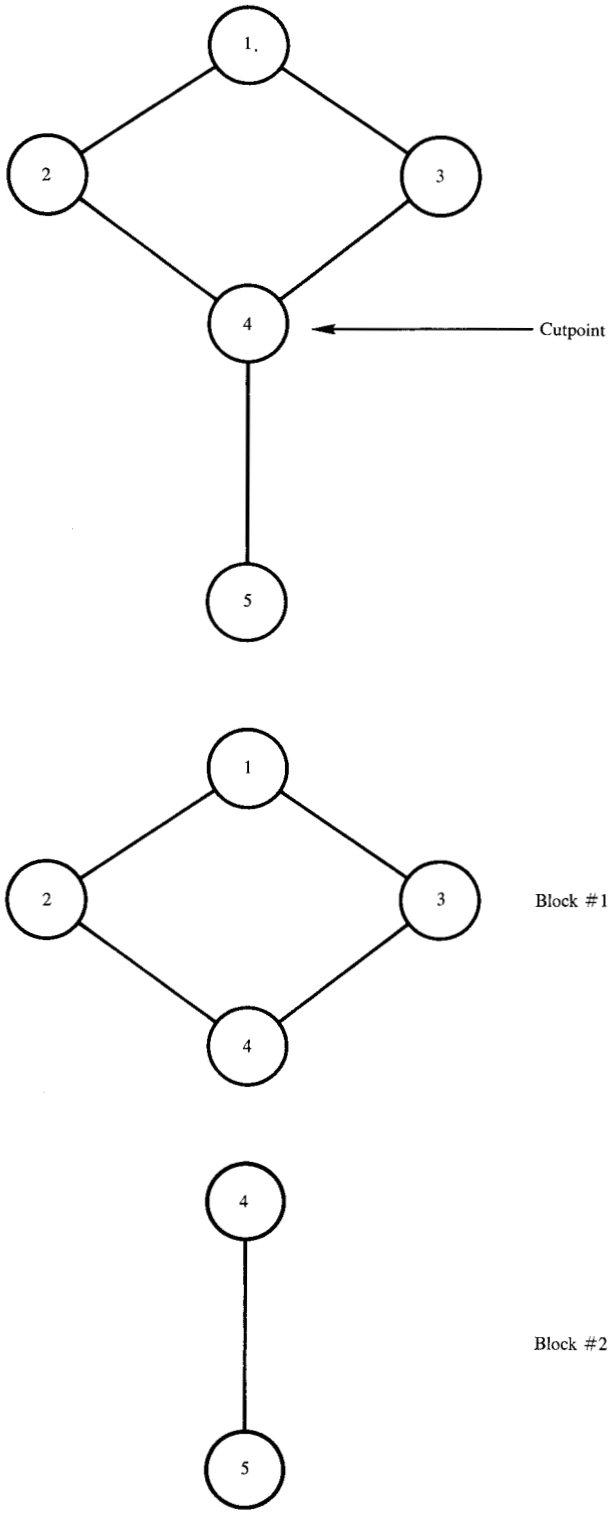


Figure 7 Graph with a cutpoint. Splitting node results in blocks #1 and #2.

mal partitions of each block in any order and then combine these partitions to generate an optimal partition of G. A proof of this theorem is given in the appendix.

Block independence theorem If a graph G has q blocks, where $q > 1$, then the optimal partition of G, $p(\text{opt})$, can be created by first partitioning the blocks independently, then combining the resulting partitions.

The existence of cutpoints (hence blocks) in a graph reduces the number of partitions generated on any step to a number directly proportional to the number of partitions generated if each block were partitioned independently. A brief summary of one method of partitioning such a graph follows.

Assume that a graph G has q blocks, $B_1, B_2, \cdots, B_q$. Let block B_k have c_k cutpoints. We then partition each block B_k independently, omitting the cutpoints c_k from any isolated set for the nodes in B_k, and keep the optimal 1-adjacencies, 2-adjacencies, $\cdots$, W-adjacencies for each cutpoint in B_k. Therefore, no more than $c_k W$ need be kept for block B_k.

When all blocks are partitioned, we then find some block with only one cutpoint in its node set and combine it with a block containing the same cutpoint. We do so by merging the clusters of the partitions of the two blocks containing their common cutpoint, leaving the other clusters unchanged. This operation takes no more than $W(W+1)/2$ steps.

The merger of blocks may reduce the number of unmerged cutpoints in the resulting subgraph containing these two blocks by one. We then find another block with one unmerged cutpoint and merge it with a block containing their common cutpoint. This procedure continues until the optimal graph partition is found.

This technique is correct because the blocks and their cutpoints form a tree called the block-cutpoint graph [8]. Consequently the method of merging block partitions is essentially a modification of the tree partitioning algorithm described in [5].

• *Examples*

To illustrate the effectiveness of the isolated set on the existence of blocks in reducing the partitions generated on each step of the partitioning process, we now examine several graph types that readily lend themselves to analysis.

A dramatic example of the reduction in computation time and storage is the following. Reference [6] shows that the minimum number of partitions generated on the jth step for the simple k-node tree of Fig. 8(a) is greater than 1.6^j. Using the analysis above this bound is reduced to the following:

$$|P_j| < x_j(x_j!)W^{x_j},$$

where $x_j = |\text{CONN}(j)_{\max}| = 1$ for all $j > 1$. Thus

$|P_j| < W$, where W is the weight constraint.

Another graph whose value of x_j is independent of j is also shown in Fig. 8(b). For a width parameter h, ISOL $(j) = \{i | i$ has label $< j - h\}$. Thus

$x_j = h$ for all j, and $|P_j| < h\ (h!)\ W^h$.

A more careful analysis results in the upper bound

$|P_j| < W^h$.

An example of the effectiveness of the use of block independence in partitioning is given in [5].

An example of the use of the partitioning algorithm taking advantage of the isolated set concept is given in Fig. 6. It is instructive to compare the number of partitions generated here with the number generated using the dynamic programming procedure alone (Fig. 4). We see that significantly fewer partitions are generated on each step by the general partitioning algorithm. We have, however, not made use of the block independence theorem, although the graph has two blocks.

Conclusion

We have described a modified dynamic programming procedure for the partitioning of connected graphs with integer weighted nodes and edges whose values are positive. The algorithm employs the concept of the isolated set to reduce the upper bound in partitioning a subgraph of the given graph from a number growing factorially in the number of subgraph nodes to one that is a function of the graph connectivity and graph labeling.

A further reduction in computation time is afforded for a graph with cutpoints, because, as we have shown, such a graph can be partitioned by block and the block partitions merged.

Acknowledgments

This paper is part of the author's Ph.D. Thesis at Stanford University, based on work which was supported there by an IBM Resident Fellowship. The author thanks Professor H. S. Stone for his continued advice and encouragement during the development of the work.

References

1. E. L. Lawler, "Electrical Assemblies with a Minimum Number of Interconnections," *IRE Trans. Elec. Comput.* **11,** 86, (1962).
2. F. Luccio and M. Sami, "On the Decomposition of Networks in Minimally Interconnected Subnetworks," *IEEE Trans. Circuit Theory* **CT-16,** 184 (1969).
3. B. W. Kernighan, "Some Graph Partitioning Problems Related to Program Segmentation," Ph.D. Thesis, Princeton University, Princeton, New Jersey, 1969.
4. B. W. Kernighan, "Optimal Sequential Partitions of Graphs," *J. Assoc. Comput. Mach.* **18,** 34 (1962).
5. J. A. Lukes, "Efficient Algorithm for the Partitioning of Trees," *IBM J. Res. Develop.* **18,** 217 (1974).

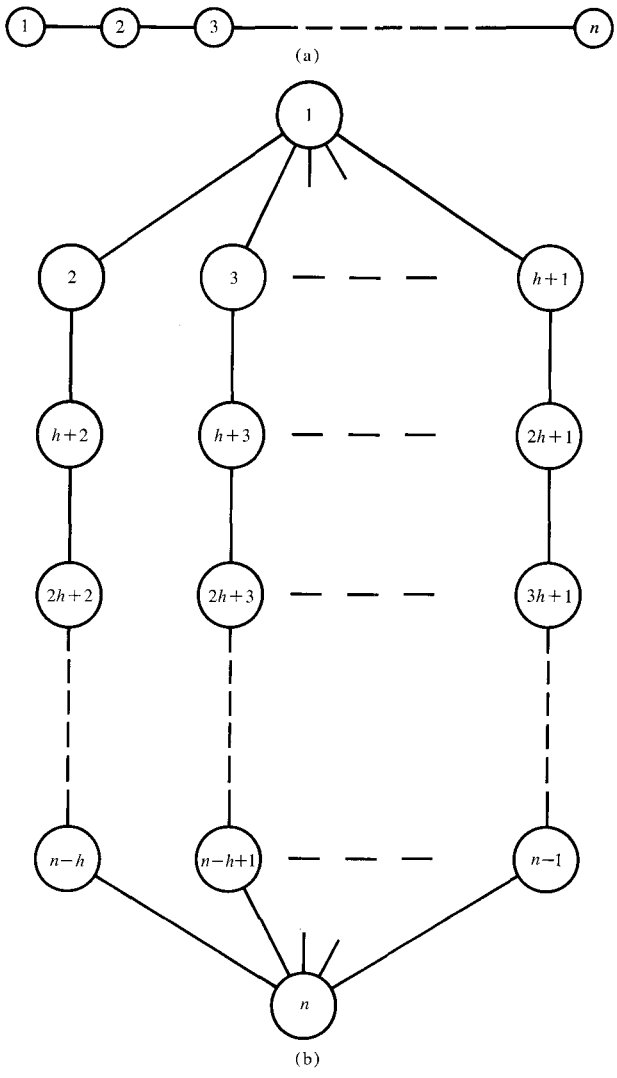


Figure 8 Maximum level k-node tree (a), and graph with constant-size connected set (b), where h is the width parameter.

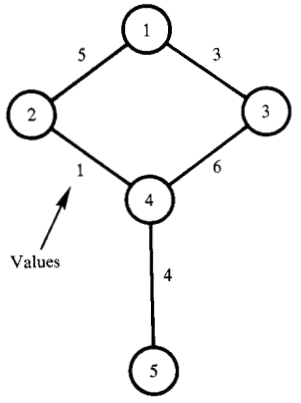
6. J. A. Lukes, "Combinatorial Solutions to Partitioning Problems," Ph.D. Thesis, Stanford University, Stanford, California, 1972.
7. P. C. Gilmore and R. E. Gomory, "The Theory and Computation of Knapsack Functions," *Oper. Res.* **14,** 1045 (1966).
8. F. Harary, *Graph Theory*, Addison-Wesley Publishing Co., Reading, Mass., 1969, p. 36.

Appendix: Proof of theorems

Isolated Set Theorem The only partitions of step k-1 necessary in generating the partitions of step k are the dominant partitions.

Proof Let G be an n-node graph. A partition p generated on some step k in the process of partitioning G can be represented by a sequence of pairs

$[1, (\ )], [2, c_2], [3, c_3], \cdots, [k, c_k],$



k	CONN(k)	ISOL(k)
1	$\varnothing$	$\varnothing$
2	{1}	$\varnothing$
3	{1, 2}	$\varnothing$
4	{2, 3}	{1}
5	{4}	{1, 2, 3}

	k-adjacencies		
Step	1	2	3
1	(1) = 0		
2	(1)(2) = 0	(1, 2) = 5	
3	(1)(2)(3) = 0 (1, 2)(3) = 5	(1, 3)(2) = 3 (1)(2, 3) = 0	(1, 2, 3) = 8
4	(1, 2, 3)(4) = 8	(1)(2, 4)(3) = 1 (1)(2)(3, 4) = 6 (1, 2)(3, 4) = 11	(1, 2, 4)(3) = 6 (1, 3, 4)(2) = 9 (1)(2, 3, 4) = 7
5	(1, 2)(3, 4)(5) = 11	(1, 2, 3)(4, 5) = 12	(1, 2)(3, 4, 5) = 15

Optimal partition is (1, 2)(3, 4, 5) with value = 15.

Figure 9 Example of graph partitioning algorithm application. All nodes have unit weight and $W = 3$.

where the first entry of a pair represents the node with label i, the second entry the cluster to which node i is added on step i, and ( ) denotes the empty cluster. The advantage of this notation over the nodal representation is that it describes precisely how p is generated. An example of this notation is [1, ( )], [2, ( )], [3, (2)], [4, (1)], [5, (2, 3)]. This representation is equivalent to the nodal representation $p = (1, 4)\ (2, 3, 5)$.

Let P_i be the set of partitions generated on step i of the partitioning process. We then define a *derivation* of a partition p from a partition q, where p is in P_k and q is in $P_j (j < k)$, as the sequence

$$[j+1, c_{j+1}], [j+2, c_{j+2}], \cdots, [k, c_k].$$

This notation is a variation of the above representation of p that ignores the steps leading up to the generation of partition q.

Let two partitions f and g generated on step k-1 be similar, and let f dominate g. Assume that there exists a partition of G, g_n, derived from g that has a greater value than any partition of G derived from f. We now show that this assumption is false.

Let a derivation of g_n from g be $[k, c_k]$, $[k+1, c_{k+1}]$, $\cdots$, $[n, c_n]$. Since f and g are similar, there is a partition f_n derived from f with the derivation $[k, \bar{c}_k]$, $[k+1, \bar{c}_{k+1}]$, $\cdots$, $[n, \bar{c}_n]$ such that for $i = k, k+1, \cdots, n$, c_i and $\bar{c}_i$ have the same weight and the nodes in c_i differ from those in $\bar{c}_i$ only if they are in ISOL (k). Note that the nodes in the isolated set of node k share no edge with a node whose label is greater than $k-1$. As a consequence, the values of partitions generated on steps $k, k+1, \cdots, n$ are independent of the nodes in ISOL (k) that appear in a cluster together with nodes in CONN $(k)_{\max}$.

Since clusters c_i and $\bar{c}_i$ $(i = k, k+1, \cdots, n)$ have nodes that differ only if they are in ISOL (k), the sum of the values of the edges in c_i and $\bar{c}_i$ can differ by the sum of the values of those edges between nodes in ISOL (k) contained in each cluster. Since f dominates g, the sum of the values of edges in $\bar{c}_i$ is equal to or greater than the sum of the edges in c_i and f_n dominates g_n. Consequently, the value of f_n is greater than or equal to the value of g_n, contrary to the assumption made above. It is therefore not contrary to an optimal policy to delete all partitions of p_{k-1} dominated by another partition.

Block Independence Theorem If a graph G has q blocks, where $q > 1$, then the optimal partition of G, $p(\text{opt})$, can be created by first partitioning the blocks independently, then combining the resulting partitions.

Proof Consider the nodal representation of $p(\text{opt})$:

$$\underbrace{[(\,)\cdots(\,)]}_{NC_1}\underbrace{[(\,)\cdots(\,)]}_{NC_2}\cdots\underbrace{[(\,)\cdots(\,)]}_{NC_q}\underbrace{[(\,)\cdots(\,)]}_{C}.$$

Here, NC_i represents a (possibly empty) set of clusters whose nodes are not cutpoints and are all from the same block B_i. The set C consists of clusters each of which contains at least one cutpoint.

The nodal representation of $p(\text{opt})$ assumes this form because of the special properties of a graph with one or more cutpoints. Since the only node in a block B_i adjacent to nodes not in B_i is a cutpoint, a cluster that contains nodes from B_i, but no cutpoint, must only contain nodes from B_i as a result of the connectivity constraint. This property justifies the collection of clusters into sets NC_i for block B_i in the nodal representation above.

Each cluster $c \in C$ contains two types of nodes;

1. A set of cutpoints $\{k_1, k_2, \cdots, k_x\}$;
2. A set of nodes $\{i_1, i_2, \cdots, i_a, j_1, j_2, \cdots\}$ none of which are cutpoints.

The latter set can be partitioned into subsets by the equivalence relationship BLOCK, where u BLOCK v if u

and v are nodes in the same block B_i. If the restriction on duplication of nodes implicit in the partitioning problem is removed, then the cluster c can be replaced by a set of clusters $\{c_1, c_2, \cdots, c_z\}$, where these clusters have the following properties:

1. Each cluster c_i contains the union of the set of nodes of c from some block B_j created by the equivalence relation BLOCK and the set of cutpoints of c also in block B_j;
2. $\Sigma_{i=1}^{z}$ VALUE$[c_i]$ = VALUE$[c]$, where VALUE$[c_i]$ equals the sum of the values of edges contained in cluster c_i.

Note that some cutpoint k may appear in several of the clusters making up the set $\{c_1, c_2, \cdots, c_z\}$.

When we perform the process above on each cluster in C, the nodal representation of $p(\text{opt})$ is transformed to

$$\underbrace{[(\,)\cdots(\,)]}_{NC_1}\cdots\underbrace{[(\,)\cdots(\,)]}_{NC_q}\underbrace{[(\,)\cdots(\,)]}_{C_1}\cdots\underbrace{[(\,)\cdots(\,)]}_{C_q},$$

where C_i = a set of clusters of nodes from block B_i including at least one cutpoint of B_i in each cluster. The value of the cover $p(\text{opt})'$ given by this nodal representation equals that of $p(\text{opt})$ and is made up of sets of clusters (NC_i, C_i) representing a partition of block B_i. No edge exists from a cluster in the set (NC_i, C_i) to a cluster in the set (NC_j, C_j) for $i \neq j$ because of the duplication of cutpoints.

In conclusion we note that one can reverse the process of decomposing $p(\text{opt})$ into the cover $p(\text{opt})'$ and generate $p(\text{opt})$ by first finding the partitions of each block and the combining these partitions.

The following theorem develops an upper bound on the number of feasible partitions generated on the kth step of the partitioning process when modified to include the concept of the isolated set.

Theorem Let CONN $(k)_{\max}$ = the set of nodes with labels less than k not in ISOL(k), i.e.

$$\text{CONN}(k)_{\max} = \{1, 2, \cdots, k-1\} - \text{ISOL}(k).$$

and let

$$x_k = |\text{CONN}(k)_{\max}|.$$

For a weight constraint of W there are no more than

$$x_k(x_k!)(W^{x_k})$$

partitions generated on step k of the partitioning process.

Proof The partitions of step $k-1$ can be separated into disjoint subsets by the property that all partitions in a given subset have the same distribution of the nodes in CONN(k) in their clusters. If, for example, the set of partitions of step 4 is $P_4 = \{(1)(2)(3,4), (1,2)(3,4), (1,3)(2,4)$, and $(1,2,3)(4)\}$ and CONN$(5) = \{3, 4\}$, then the subsets of P_4 satisfying the above property are $\{(1,2)(3,4), (1)(2)(3,4)\}$ and $\{(1,3)(2,4), (1,2,3)(4)\}$. Note that no limitation is placed on the nodes in ISOL(k) in a cluster. We now show that any subset of P_{k-1} so formed has no more than W^{x_k} partitions in it, where W is the weight constraint and x_k is the maximum size of CONN(k) for any weight constraint.

Let P'_{k-1} be a set of partitions of step $k-1$ each of which has the same distribution of nodes in CONN(k) in its clusters. If a partition in P'_{k-1} has a cluster containing nodes $i_1, i_2, \cdots, i_m$ that are in CONN(k), then every other partition in P'_{k-1} also has a cluster containing nodes $i_1, i_2, \cdots, i_m$. No restriction is placed, however, on the nodes in ISOL(k) in a cluster containing this subset of CONN(k). Consequently the weight of a cluster of a partition in P'_{k-1} containing nodes $i_1, i_2, \cdots, i_m$ need not be the same for each partition in P'_{k-1}. There are a maximum of x_k nodes in CONN(k); consequently we can distribute the nodes of CONN(k) into no more than x_k distinct clusters. Any given cluster can assume a weight that varies from one to W. Assume then that every partition in P'_{k-1} has x_k clusters that contain a node in CONN(k) and that every such cluster can have a weight that varies from one to W. The number of partitions in P'_{k-1} is then no greater than W^{x_k}, because this number represents the number of different combinations of x_k clusters, where each cluster can assume a weight from one to W. This result follows from the isolated set theorem, as we now show.

Assume that two partitions in P'_{k-1}, p and q, have clusters such that for every cluster of p containing a set of nodes in CONN (k), the cluster of q containing the same set of nodes in CONN (k) has equal weight. Also, assume that the value of p is greater than or equal to that of q. The isolated set theorem then proves that q can be deleted from P'_{k-1}.

We now prove that an upper bound on the number of partitions of step k generated from the set P'_{k-1} is given by

$$x_k W^{x_k}$$

where for simplicity we assume that $W \leq x_k$.

Assume that each partition is the set P'_{k-1} has r clusters that contain at least one node in the set CONN (k). Also, let each node have unit weight. Node k can then be added to each of the r clusters of a partition in P'_{k-1} if the weight of the cluster to which k is added is less than W. Let $P(i)$ denote the set of partitions in P'_{k-1} whose ith cluster, of those clusters that contain a node in CONN (k), has weight less than W. The number of feasible partitions of step k generated by adding node k to a cluster of a partition of P'_{k-1} is then given by

$$\sum_{i=1}^{r} |P(i)|.$$

The upper bound on $|P(i)|$ is given by

$$|P(i)| \leq W^{r-1}(W-1),$$

and the maximum value of r is x_k; therefore no more than

$$x_k W^{x_k-1}(W-1)$$

partitions of step k can result by adding node k to the clusters of the partitions in P'_{k-1}. There are W^{x_k} 1-adjacencies of step k derived from the partitions in P'_{k-1}; hence

$$x_k(W-1)W^{x_k-1} + W^{x_k}$$

partitions are generated from the subset P'_{k-1}. If we assume that $W \leq x_k$, then

$$x_k(W-1)W^{x_k-1} + W^{x_k} \leq x_k W^{x_k}.$$

From the third section there are fewer than $x_k!$ possible ways to distribute the nodes in CONN(k) in clusters; hence the set P_{k-1} can be separated into no more than $x_k!$ subsets. Therefore the upper bound on the number of partitions generated on step k of the partitioning algorithm is

$$x_k(x_k!)W^{x_k},$$

where x_k is independent of the weight constraint.

Received July 25, 1974; revised November 11, 1974

The author is located at the IBM System Development Division Laboratory, 1512 Page Mill Rd., Palo Alto, California 94304.