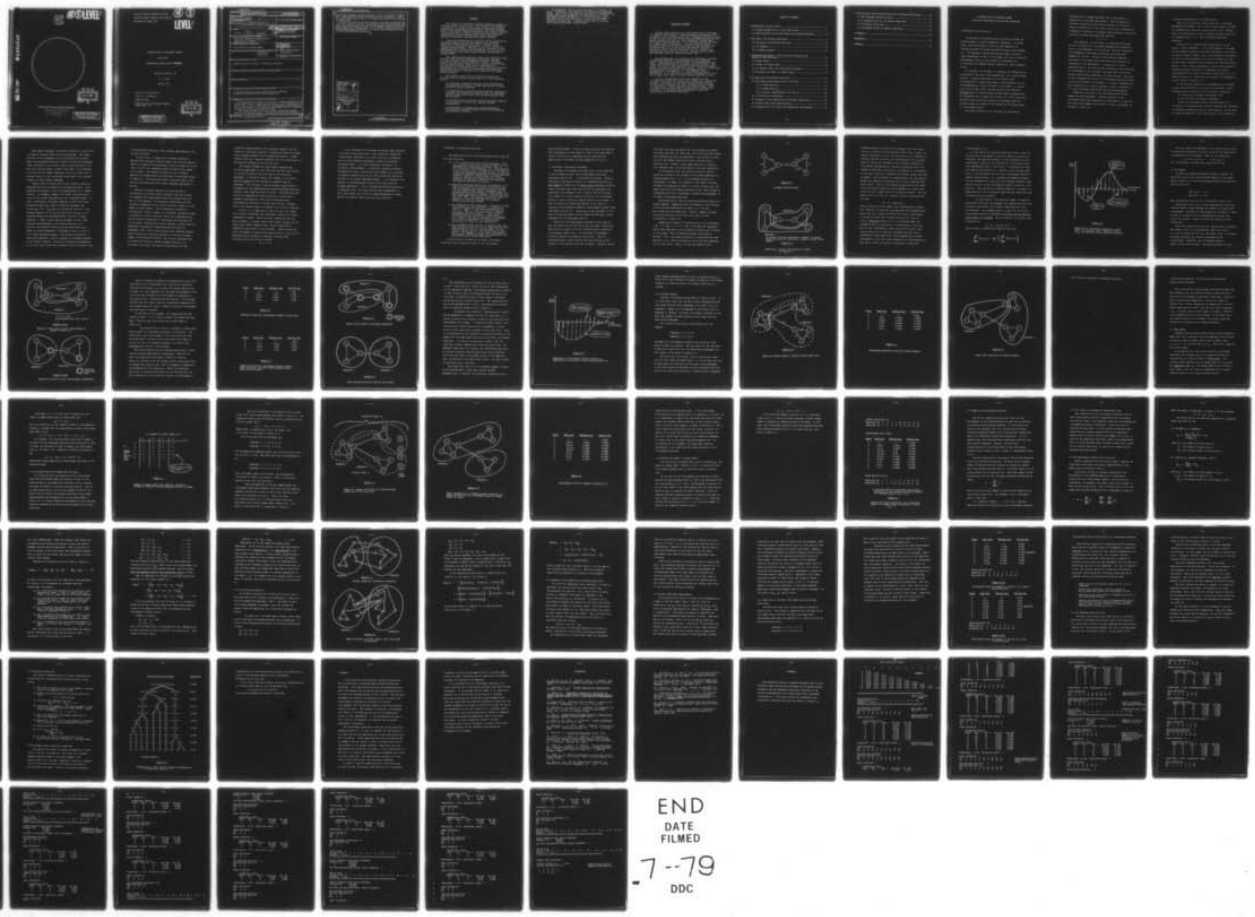
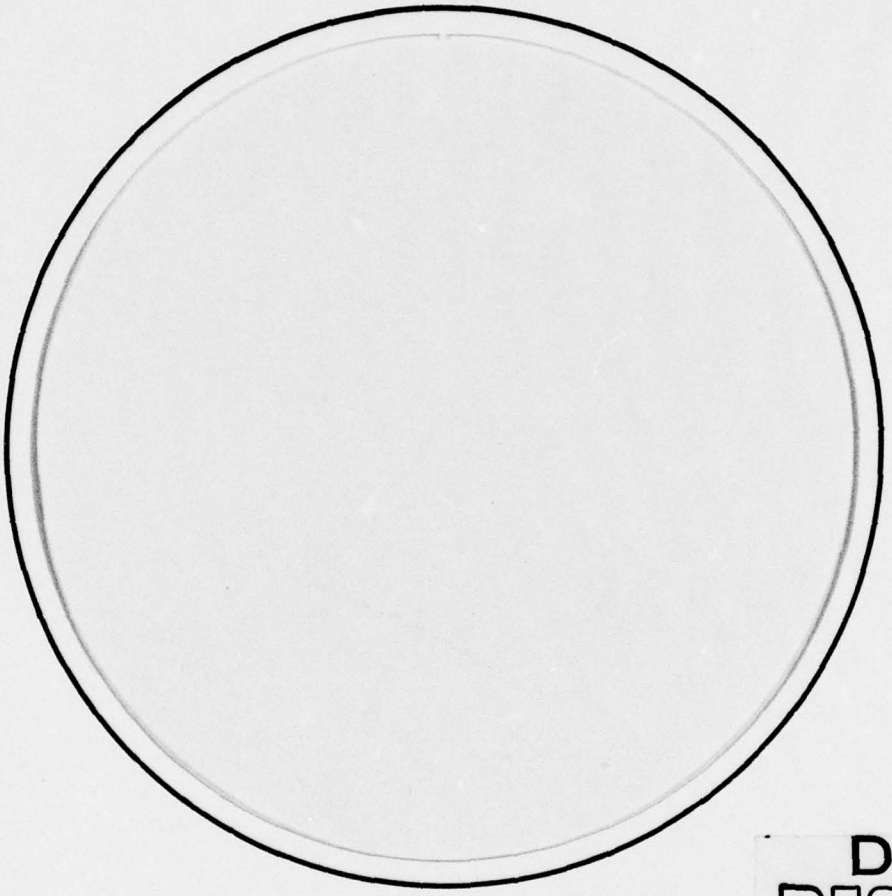
AD-A069 549 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MASS C--ETC F/G 9/2
DECOMPOSITION OF WEIGHTED GRAPHS USING THE INTERCHANGE PARTITIO--ETC(U)
MAR 79 S L HUFF N00039-78-G-0160
UNCLASSIFIED CISR-P010-7903-08 NL

1 OF 1
AD
A069549



END
DATE
FILMED
7-79
DDC

LEVEL II



Center for Information Systems Research
Massachusetts Institute of Technology
Alfred P. Sloan School of Management
50 Memorial Drive
Cambridge, Massachusetts, 02139
617 253-1000

Contract Number N00039-78-G-0160

Internal Report Number P010-7903-08

Deliverable Number 004

(12) (8)

LEVEL II

DECOMPOSITION OF WEIGHTED GRAPHS

USING THE

INTERCHANGE PARTITIONING TECHNIQUE

Technical Report #8

S. L. Huff

March 1979

Principal Investigator:

Prof. S. E. Madnick

Prepared for:

Naval Electronic Systems Command
Washington, D.C.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: (9) Technical Report, #8
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): (6) Decomposition of Weighted Graphs Using the Interchange Partitioning Technique
5. TYPE OF REPORT & PERIOD COVERED
6. PERFORMING ORG. REPORT NUMBER: (14) CISR-P010-7903-08, CISR-TR-8
7. AUTHOR(s): Sid Huff (10) Sid L. Huff
8. CONTRACT OR GRANT NUMBER(s): (15) N0039-78-G-0160
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Center for Information Systems Research, Sloan School of Management, M.I.T., Cambridge, Mass., 02139
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Electronic Systems Command
12. REPORT DATE: (11) March 1979
13. NUMBER OF PAGES: 85
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): (12) 94p.
15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report): Approved for Public Release - distribution unlimited.
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
18. SUPPLEMENTARY NOTES
19. KEY WORDS (Continue on reverse side if necessary and identify by block number): Software architectural design; problem design structuring; mathematical graph modelling; graph decomposition.
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The objective of this study is to develop a systematic approach to the architectural design of complex software systems. This contract builds on earlier work, in which a graph modelling and decomposition methodology was used to operate upon a set of functional requirements and their interrelationships to generate an architectural design.

This report introduces a new algorithm for partitioning weighted graphs in a hierarchical manner. The algorithm, termed the interchange

next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409 590 LB

partitioning technique, has been developed to aid in the analysis of requirements graphs generated through the Systematic Design Methodology. However, it is sufficiently general to be of use in many other types of graph analysis problems as well.

As well as describing and giving examples of the basic interchange technique, this report also includes a discussion of certain simplifications that may be made to the algorithm in order to significantly improve its efficiency without hampering its effectiveness. Also, a "master control" algorithm is presented for guiding the execution of a complete graph decomposition using the interchange partitioning technique.

PREFACE

The Center for Information Systems Research (CISR) is a research center of the M.I.T. Sloan School of Management. It consists of a group of management information systems specialists, including faculty members, full-time research staff, and student research assistants. The Center's general research thrust is to devise better means for designing, implementing, and maintaining application software, information systems, and decision support systems.

Within the context of the research effort sponsored by the Naval Electronics Systems Command under contract N00039-78-G-0160, CISR has proposed to conduct basic research on a systematic approach to the early phases of complex systems design. The main goal of this work is the development of a well-defined methodology to fill the gap between system requirements specification and detailed system design.

The research being performed under this contract builds directly upon results stemming from previous research carried out under contract N00039-77-C-0255. The main results of that work include a basic scheme for modelling a set of design problem requirements, techniques for decomposing the requirements set to form a design structure, and guidelines for using the methodology developed from experience gained in testing it on a specific, realistic design problem.

The present study aims to extend and enhance the previous work, primarily through efforts in the following areas:

1) additional testing of both the basic methodology, and proposed extensions, through application to other realistic design problems;

2) investigation of alternative methods for effectively coupling this methodology together with the preceding and following activities in the systems analysis and design cycle;

3) extensions of the earlier representational scheme to allow modelling of additional design-relevant information;

4) development of appropriate graph decomposition techniques and software support tools for testing out the proposed extensions.

This document, which relates primarily to category (4) above, introduces a new algorithm for partitioning weighted graphs. While this algorithm was developed to accomplish the requirements graph decomposition task particular to the Systematic Design Methodology, it is sufficiently general and powerful to be of use in many other types of graph analysis problems also. Examples of the use of this new algorithm are included in this report.

EXECUTIVE SUMMARY

Complex design problems are characterized by a multitude of competing requirements. System designers frequently find the scope of the problem beyond their conceptual abilities, and attempt to cope with this difficulty by decomposing the original design problem into smaller, more manageable sub-problems. Functional requirements form a key interface between the users of a system and its designers. In this research effort, a systematic approach has been proposed for the decomposition of the overall set of functional requirements into sub-problems to form a design structure that will exhibit the key characteristics of good design: strong coupling within sub-problems, and weak coupling between them.

The decomposition is carried out by modelling a system's requirements and their interdependencies as a weighted graph: requirements are graph nodes, interdependencies are links, and interdependency strengths are link weights. This report introduces a new algorithm for partitioning weighted graphs in a top-down hierarchical manner. While this algorithm was developed to accomplish the requirements graph decomposition tasks particular to the Systematic Design Methodology, it is sufficiently general and powerful to be of use in many other types of graph analysis problems as well.

As well as describing and giving examples of the technique upon which the algorithm is based, this report also includes a discussion of certain simplifications that may be made to the algorithm in order to significantly improve its efficiency, without hampering its effectiveness. Also, a "master control" algorithm is presented for guiding the execution of a complete graph decomposition using the interchange partitioning technique.

TABLE OF CONTENTS

DECOMPOSITION OF WEIGHTED GRAPHS USING THE INTERCHANGE PARTITIONING ALGORITHM.

1 Background and Motivation.

The problem of designing quality software systems has existed almost as long as computers themselves. In recent years, continued growth in the cost and complexity of software has made the design problem even more acute (Boehm 73). The Systematic Design Methodology (SDM), currently under development at MIT, is an attempt to bring a structured methodology to bear on the problem of developing the architecture for complex systems (Andreu 78; Huff & Madnick 78a,b).

At the heart of the SDM is a technique for modelling the requirements, and their interdependencies, for a given design problem. A graph representation, with graph nodes corresponding to system requirements, and (weighted) arcs to interdependencies, forms the basis of this model. A central analytical problem within the SDM concerns the effective decomposition of a given design problem graph representation. By "decomposition" is meant the creation of a set of mutually exclusive and collectively exhaustive subgraphs from the original graph, according to some guideline that makes sense in the problem context.

This paper presents a new technique for performing

decomposition of weighted graphs that is particularly appropriate to the SDM requirements. The new approach, a top-down hierarchical partitioning heuristic, is both an effective method in its own right, as well as an appropriate "companion" technique to accompany other bottom-up clustering heuristics already developed (see Huff 79).

* * * * *

The remainder of this report is organized as follows. Further introduction to the graph decomposition problem is given in the remainder of this section. Section 2 presents the fundamental interchange algorithm, and gives some examples of its use. Section 3 discusses two important extensions to the basic algorithm that greatly extend its usefulness in the SDM context. In Section 4, certain approximations to the calculations performed in the algorithm are introduced so as to enhance the algorithm's efficiency, without significantly weakening its effectiveness. The final section discusses the use of the interchange algorithm within a controlling "master" environment - in effect, imbedding the basic algorithm in a control program in order to implement a full-scale top-down hierarchical partitioning procedure.

Examples of the use of the interchange partitioning scheme are used throughout. Further analysis of the interchange algorithm, comparing its performance in terms of efficiency and decomposition quality to other techniques, is included in (Huff 79).

1.1 Graph Decomposition in the SDM Context.

There are two distinct issues involved in the graph decomposition problem. First, there is the question of what the decomposition objective function ought to be. Second, there is the question of how to go about actually effecting the decomposition - i.e., how to identify and select subgraphs.

The notion of a "good" graph decomposition is, of course, context dependent. In the developement of the SDM, we have adopted a measure that operationalizes a key concept of software design, namely, that a good software architecture is one which both maximizes the internal strength of each system module, and which also minimizes the coupling between modules. Thus a graph decomposion in which each subgraph is densely connected, and pairs of subgraphs are loosely interconnected, will have a relatively high value of M, the objective function. This objective function is discussed in greater detail in Section 4.1.

As for actually effecting the decomposition of graphs, we have experimented with a variety of techniques. These techniques may be classified into two major categories:

(1) clustering techniques (bottom-up), and

(2) partitioning techniques (top-down).

Clustering techniques require the generation of a similarity (or dissimilarity) matrix to express the closeness (or distance) between all pairs of data points. In the case of graph decomposition, the "data points" are the nodes of

the graph. Once a (dis)similarity matrix has been defined, various hierarchial clustering heuristics may be applied to successively lump together individual nodes into subsets, subsets into larger subsets, etc., until the entire set is generated. While hierarchical clustering techniques as such have no inherent stopping rule, the goodness measure, M, may be calculated after each subset merger, and the particular decomposition exhibiting the highest M "remembered" as the best decomposition.

Since the focus of this report is partitioning algorithms, not clustering techniques, further details on the latter will not be discussed here. For additional information on the use of clustering methods in SDM graph decomposition, see (Andreu 78) or (Huff 79).

Partitioning techniques take a variety of forms, but have the common property of dealing directly with the graph structure, rather than a similarity matrix defined out of the graph structure. Typically, partitioning techniques seek to successfully break up a graph into subgraphs until either some stopping criterion is reached, or until each subgraph contains a single node.

Partitioning techniques are especially useful in SDM decomposition for two different reasons. First, since they are fundamentally different from the various types of clustering techniques also used for this function, they provide an effective cross-check on the validity of the results achieved by the other methods. Also, they tend to

locate a best decomposition with a smaller number of steps than clustering techniques, because optimal SDM decompositions occur fairly near the top of the decomposition "tree" (i.e., fairly near to the single subgraph state - see Figure 5.2).

While the graph theory literature reports a variety of approaches to the partitioning problem, none of these schemes are directly appropriate for our class of problems. For example, one common partitioning technique involves identifying complete subgraphs, then using these subgraphs as "leader" subsets, and effecting a final partition by assigning the remaining nodes to one of the leader subgraphs. However, in the SDM context, design problem representations yield graph structures with few if any complete subgraphs of non-trivial size. Also, when complete subgraphs do exist, they often overlap, and there is no obvious way of dealing with this problem in the SDM context.

1.2 Shortcomings of Previous SDM Partitioning Techniques.

In his work on the development of the phase-1 SDM, Andreu (Andreu 78) identified and tested two different partitioning techniques - one a "leader" identification technique similar to the complete subgraph method mentioned above, the other an iterative technique for factoring out subgraphs. These two partitioning techniques exhibit certain difficulties and shortcomings, both inherently and with respect to the present SDM model.

The leader technique, while quite efficient, serves only to identify certain "good" starting subgraphs. The number and size of the subgraphs can be controlled only grossly. Also, the technique generally results in numerous unassigned nodes, and there is no obvious means of deciding what to do with the leftover nodes: whether they ought to be assigned to certain of the leader subgraphs, and if so, which one, or whether certain of the unassigned nodes ought to be grouped together to form another subgraph.

Andreu chose to assign each of the leftover nodes to a subgraph such that the measure M increased the most. This resolution technique is not as straightforward as it sounds, however. For example, suppose the leader technique gives rise to five leader subgraphs plus ten unassigned nodes. In order to determine the leader subgraph to which the first unassigned node (say, node x) should be assigned, the values $\Delta M_{ix} = M' - M_{ix}$ must be calculated, where M' is the goodness measure of the initial partition, and M_{ix} is the goodness measure for unassigned node x placed in leader subgraph i. Node x would then be assigned to that leader subgraph corresponding to the maximum ΔM_{ix} over all i. However, the values M' and M_{ix} must be calculated with a number of nodes unassigned. A problem arises as to how to treat these unassigned nodes: they could be treated as individual subgraphs, could be ignored, or possibly handled in yet another fashion. Each such treatment has drawbacks: as the final node resolution problem is akin to solving a set

of simultaneous equations, such stepwise approaches are not directly suitable.

In particular, a simple hill-climbing resolution approach for assigning leftover nodes, such as that used by Andreu, is especially questionable, being so sensitive to the order in which the unassigned nodes are dealt with, among other things. The ineffectiveness of a hill-climbing approach in decomposition by clustering is discussed further in (Huff 79). The foregoing points illustrate some of the difficulties associated with leader subgraph approaches in general.

Andreu's iterative approach is novel and quite interesting, but suffers from a drawback even more severe: its computational requirements grow very rapidly with the size of the subgraph. The procedure effectively involves iterative recomputation of a matrix of size n (where n = the number of graph nodes). Each recomputation requires n calculations. After a number of iterations, the matrix tends to stabilize in such a way that one or more subgraphs with high internal strength may be identified and "factored out" of the graph. The process is then repeated, as many times as necessary, successively identifying and factoring out subgraphs, until only a single unfactorable subgraph remains. Andreu tested the iterative approach on a few fairly small graphs, with promising results. But he also indicated that the calculation barrier rapidly becomes large as n is increased, since both the amount of calculation at each

iteration (proportional to n^2) increases rapidly, and the number of iterations required to completely factor the graph also increases rapidly (by an unspecified amount). Andreu himself found the iterative approach to be too inefficient to be used for problems of non-trivial size.

Yet another drawback to both leader subgraph and iterative approaches is that they give the user very little in the way of control over the size of the resulting subgraphs. It would be useful, for example, to be able to specify a priori certain minimum or maximum sizes on subgraphs. For instance, a system designer might prefer a decomposition with approximately balanced subgraph sizes and slightly lower objective function value. Providing him with control on subgraph size would allow him to make such a tradeoff intelligently. The leader technique could conceivably be modified to include such control, although there is no obvious way to so modify the iterative technique.

Finally, it is important to note that both the leader and iterative techniques were developed to partition binary (unweighted) graphs. Since the basic graph model has now been extended to include, among other things, weights on the graph links (Huff & Madnick 78b), neither algorithm is immediately applicable to the new model. Modifications for these techniques to extend their applicability to the current SDM model may be possible to develop, although such extensions have not yet been studied.

* * * * * *

In the remainder of this paper we present and illustrate a new partitioning algorithm. This algorithm, termed the "interchange" algorithm (as it functions be interchanging pairs of nodes between subgraphs), avoids the problems discussed above. It is quite efficient (we show it is polynomial time bounded); it generally reaches an optimal decomposition after a relatively small number of iterations; it produces complete partitions (thus avoids the resolution problem of the leader subgraph method); it is designed to be applied to weighted graphs (hence can be used with the extended SDM model); and it included bounds-setting control on subgraph size. The algorithm has been implemented in PL/1, and tested on a variety of graph decomposition cases. Results of some of these tests are also reported.

2 The Basic Interchange Algorithm.

The graph partitioning algorithm presented here has the following features:

(1) It is a hierarchical partitioning approach. That is, it seeks to partition the given graph into two subgraphs in an appropriate fashion, then to partition one of the subgraphs, etc. This approach is the inverse of that followed by the hierarchical clustering techniques mentioned earlier, in that the clustering techniques progressively group subgraphs (subsets) together, forming fewer and larger clusters, whereas the hierarchical partitioning technique successively subdivides the subgraphs, forming more and smaller clusters.

(2) The basic mechanism underlying the technique is to begin with an arbitrary initial factoring of the current subgraph (the subgraph to be partitioned) into two equal-sized, smaller subgraphs, then to perform pairwise interchanges of nodes between the two halves according to a simple criterion until no more improvement can be made in a certain objective function. The resulting two partitions then replace the original subgraph in the global factoring of the given graph.

(3) The algorithm is quite efficient. To factor a graph of 2n nodes into two subgraphs of n nodes each requires a number of operations proportional to $(n^2 + n)$. Thus it is of the class of n-squared procedures, the most efficient class of general graph-manipulation procedures. (Some graph procedures belong to even more efficient classes, e.g., order n, but they are generally oriented to very specific problems or are significantly constrained in terms of their generality.)

(4) The technique allows the user to specify upper and lower bounds on the sizes of the subgraphs to result from the partitioning of a given graph. In the software design context, this feature may be quite useful - for example, in cases where the designer has a priori information about the appropriate size range for a given design subproblem.

In this section we describe the basic interchange algorithm, which forms the heart of the interchange

partitioning scheme. In the following sections, we describe both extensions to, and simplifications of the algorithm, a control structure for implementing the algorithm for application in the SDM, and some examples of its use.

2.1 The Basic Interchange Technique.

The basic interchange technique used in this algorithm is partly based on early work by Kernighan and others (Kernighan and Lin 70). It functions as follows. Given an arbitrary initial partitioning of a weighted graph with an even number of nodes into two equal sized subgraphs, pairs of nodes, one node taken from each of the two subgraphs, are repeatedly interchanged so as to improve the partitioning with respect to a given quality criterion (to be discussed shortly), until a certain stopping condition is reached. A test is required to determine whether or not there is anything to be gained by repeating the procedure another time (Kernighan and Lin 70). Extensions of the procedure to the case of arbitrary-sized initial graphs and non-equal sized subgraphs will be discussed shortly.

Central to the interchange technique is the issue of what the quality criterion (the criterion used to determine which pairs of nodes, out of all possible pairs, should be interchanged) ought to be. In Kernighan's development, interest focused on minimum-cut partitions. That is, Kernighan sought a partitioning such that the sum of the weights on the severed links was a minimum. Actually, quite

efficient heuristic procedures for the minimum-cut problem have been available for some time, one of the earliest having been presented by Ford and Fulkerson in the context of their proof of the famous "max-flow min-cut" theorem (Ford & Fulkerson 60). However, these approaches gave no control over subgraph size, one of Kernighan's (and our) objectives.

In the present case, the quality criterion clearly should be related to our concept of the global goodness of a graph decomposition - i.e., to the goodness measure, M. Accordingly, we adopt, as a measure of the "gain" realized through interchanging two nodes in a given bi-partition, the impact that the interchange would have on M. This concept of interchange gain will be employed to guide the development of the basic interchange algorithm.

The discussion to follow is best motivated by means of a simple example. Suppose we wish to determine a partitioning of the six-node graph given in Figure 2.1, into two partitions of three nodes each. Further, suppose we begin with some arbitrary initial partition, say, the one illustrated in Figure 2.2.

Inspection of Figure 2.2 indicates that the interchange (① -> 1; ⑥ -> 2) (i.e., node 1 switched into subgraph 1, node 6 into subgraph 2) would produce an optimal partitioning of this graph. The interchange procedure functions as follows. First, the quantity $\Delta M(x,y)$ is calculated for each node pair (x,y), where x belongs to one of the two subgraphs and y to the other. $\Delta M(x,y)$ is the net impact on the

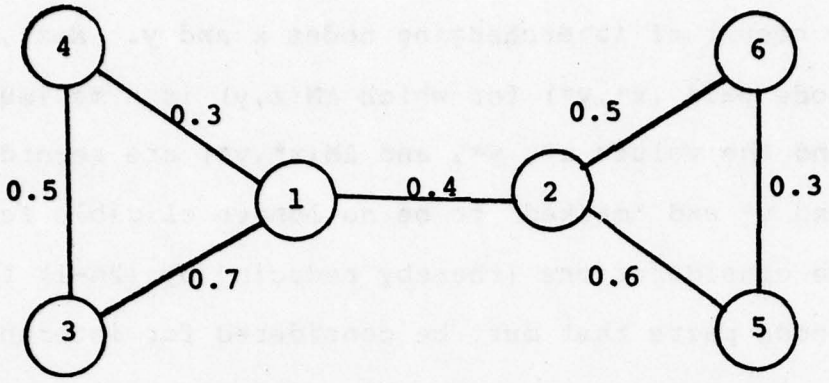
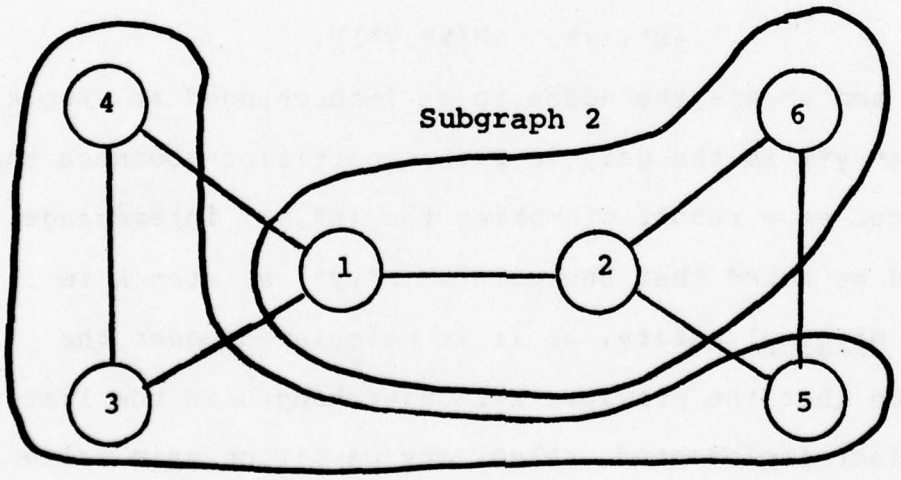


Figure 2.1

A simple weighted graph



Subgraph 1

(to minimize diagram complexity, weights on graph links will only be shown when a graph is first introduced).

Figure 2.2

(Arbitrary) initial partitioning of graph of Figure 2.1

goodness measure M of the entire decomposition that would occur as a result of interchanging nodes x and y. Next, the specific node pair (x*,y*) for which $\Delta M(x,y)$ is a maximum is located, and the values x*, y*, and $\Delta M(x^*,y^*)$ are recorded. Nodes x* and y* and "marked" to be no longer eligible for interchange considerations (thereby reducing by (2n-1) the number of node pairs that must be considered for interchange at the next iteration - assuming the original graph contained 2n nodes). This procedure is repeated again for the reduced graph, and repetition continues until no more unmarked node pairs remain. Thus if the original graph contains 2n nodes, arbitrarily partitioned into two subgraphs of n nodes each, there will be n repetitions of the above procedure. The end result will be a list of n triples, the kth triple being of the form

$$(x^*,\ y^*,\ \Delta M(x^*,y^*)),$$

where x* and y* are the nodes to be interchanged at step k, and $\Delta M(x^*,y^*)$ is the gain in global partition goodness that would occur as a result of making the (x*,y*) interchange. It should be noted that the gain $\Delta M(x^*,y^*)$ at step k is really a <u>marginal</u> entity, as it is calculated under the assumption that the previous k-1 interchanges in the list were in fact implemented. Also, any partition gain value could be positive or negative, although the sum of all n values must always equal zero, since summing all n values corresponds to the total gain achieved from interchanging all node pairs, which is logically equivalent to making no

interchanges at all.

At this point the following question arises: which of the pairwise interchanges in the list ought to actually be effected? One approach would be to only implement the kth interchange if both (1) the previous k-1 interchanges have been implemented, and (2) the first k interchanges all have non-negative gains $\Delta M(x^*,y^*)$. However, this strategy would fail in the case where the first few interchanges produced a (small) negative gain, i.e., worsened the goodness measure for the decomposition, but later interchanges produced larger positive gains, so that the combined impact turned out to be positive. This situation is shown graphically in Figure 2.3. Experience with the interchange algorithm has shown that such situations are not uncommon.

In this figure, the cumulative impact is negative during the first four interchanges, but becomes positive with the fifth interchange, and reaches a maximum after eight interchanges. This example makes it clear that the first k interchanges should be implemented <u>such that the kth partial sum of gains is a maximum</u>. More precisely, if the kth triple is denoted as

$$(x_k, y_k, \Delta M_k(x_k, y_k)),$$

then we seek to identify the index k* such that

$$\sum_{\ell=1}^{k^*} \Delta M_\ell(i_\ell, j_\ell) = \max_q \left\{ \sum_{\ell=1}^{q} \Delta M_\ell(i_\ell, j_\ell) \right\}$$

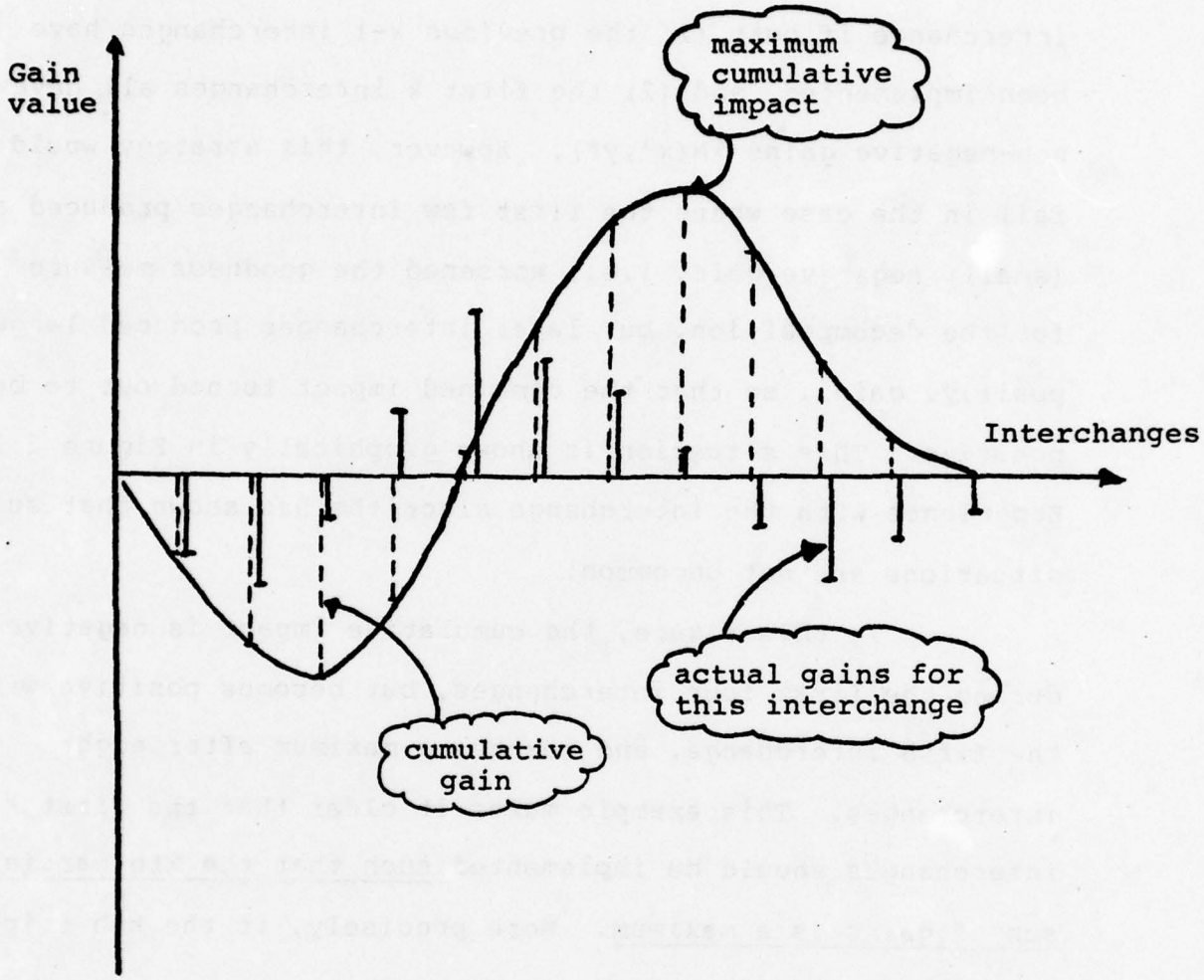


Figure 2.3

Depiction of interchange sequence in which early interchanges have a negative effect, but a net positive effect eventually accrues.

Once the index corresponding to the largest partial sum, k*, has been determined, the final action of the algorithm is to implement the interchanges. That is, the node pairs

$$\{(i_1, j_1), (i_2, j_2), \ldots, (i_{k^*}, j_{k^*})\}$$

are interchanged to produce the terminal partition.

2.2 An Example.

To make the foregoing arguments clearer, consider the execution trace of the algorithm when applied to the graph shown in Figure 2.1, with the following arbitrarily selected initial partition:

partition 1: 1 2 6

partition 2: 3 4 5

This initial partition is that illustrated in Figure 2.2.

During the first cycle of the algorithm there are three eligible nodes in each subgraph (all nodes are eligible for interchange; none have yet been marked as ineligible). Thus we want to calculate 3*3 or 9 possible gain values, and select the largest. Table 2.1 gives the results of this calculation.

From this table we see that the largest gain is achieved when nodes 5 and 1 are interchanged. Thus the triple (5,1,.66) is placed on the list, and nodes 5 and 1 are marked as being ineligible for further consideration for interchanges. Logically, the interchange shown in Figure 2.4(a) has been made. The above procedure is then

Original Goodness Measure M = -0.367

Subgraph #1 nodes	Subgraph #2 nodes: 1	2	6
3	-0.23	-0.25	0.011
4	-0.30	-0.19	-0.02
5	0.65	-0.21	-0.02

Optimal gain value ΔM^* occurs when nodes 1 and 5 are interchanged

Table entries are gain values, $\Delta M(i,j)$.

If nodes 1 and 5 are interchanged, the resulting goodness measure will be M' = -0.367 + 0.65 = 0.283.

Table 2.1

Gain matrix for deciding on initial interchange of graph shown in Figure 2.2.

repeated, with a reduced graph containing four nodes, two in each subgraph.

For the next cycle, since we have decided after the first cycle to interchange nodes 5 and 1, the starting partition is that shown in Figure 2.4(b).

The objective function for this initial partition is M' = 0.29. Only nodes 2,3,4, and 6 are eligible for interchange. Calculations show that the largest gain value is -0.64, corresponding to the interchange of nodes 4 and 2. Since the largest gain is negative, the best possible interchange still makes the partition worse with respect to M. This observation is in line with intuition, since the starting partition at cycle 2 (shown in Figure 2.4(b)) is clearly the overall optimal partition for the network; any interchange at this point will only serve to reduce M.

To complete the second cycle, we add the triple (4,2,-0.64) to the list, and mark nodes 4 and 2 as ineligible for further interchanges. The list now contains two entries:

(5,1,0.66)

(4,2,-0.64)

After the second cycle, we have logically interchanged nodes (5,1) and (4,2), so the third cycle is begun with the starting partition shown in Figure 2.5.

The initial measure for this starting partition is M' = -0.35. Since there is only one pair of nodes eligible for interchanging - node pair (3,6) - a single calculation shows the resulting impact on M to be -0.02.

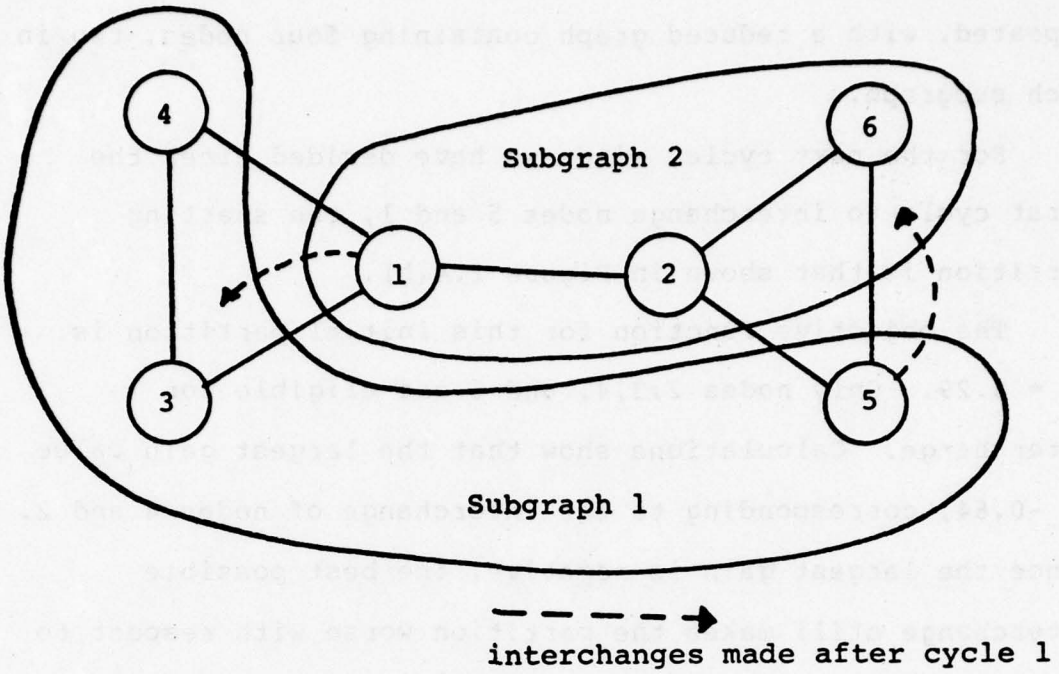


Figure 2.4(a)

Effect of first pair of node interchanges on graph of Figure 2.2.

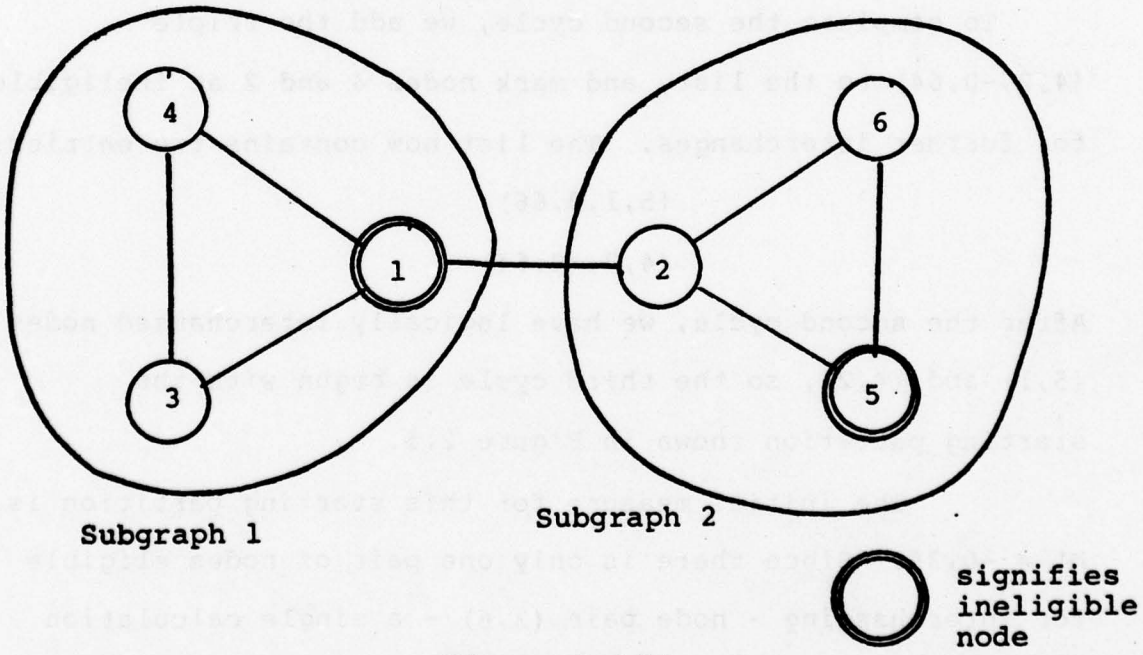


Figure 2.4(b)

Resulting situation after interchanges implemented.

Table 2.2 shows the sequence of maximum gain values and node pairs to be interchanged that resulted at each step. The task at this point is to locate the largest partial sum in Table 2.2, then implement all the node interchanges in the list up to and including the interchange corresponding to that partial sum. As was pointed out earlier, this strategy insures that the trap of ignoring interchange sequences that only become beneficial after a number of initial interchanges have been made is avoided.

In the case of this example, the largest partial sum occurs at step 1 - i.e., after a single interchange has been made. Thus the best final partition is that shown in Figure 2.6.

The question now arises as to whether a single such pass through the interchange procedure is sufficient. Put differently, what would happen if the final partition that was generated at the end of the first pass (that shown in Figure 2.6, for the foregoing example), was used as the starting partition for a second pass?

The second question may be answered empirically in this case by actually performing a second pass. When this is done, the execution trace shown in Table 2.3 results. Table 2.3 indicates that the largest partial sum occurs after the final step, and is 0.00. This corresponds to completely interchanging all the node pairs. Since the resulting partition is logically equivalent to the starting one, the best alternative in this case is to make no interchanges at

Cycle	Node pair	Maximum Gain	Partial sum
0	---	---	0.00
1	(5,1)	0.66	0.66
2	(4,2)	-0.64	0.02
3	(3,6)	-0.02	0.00

Table 2.2

Execution trace for interchange example (first pass)

Cycle	Node pair	Maximum Gain	Partial Sum
0	---	---	0.00
1	(4,2)	-0.64	-0.64
2	(3,6)	-0.01	-0.65
3	(5,1)	0.65	0.00

Table 2.3

Execution trace for interchange example (second pass, starting with best identified partition from the first pass).

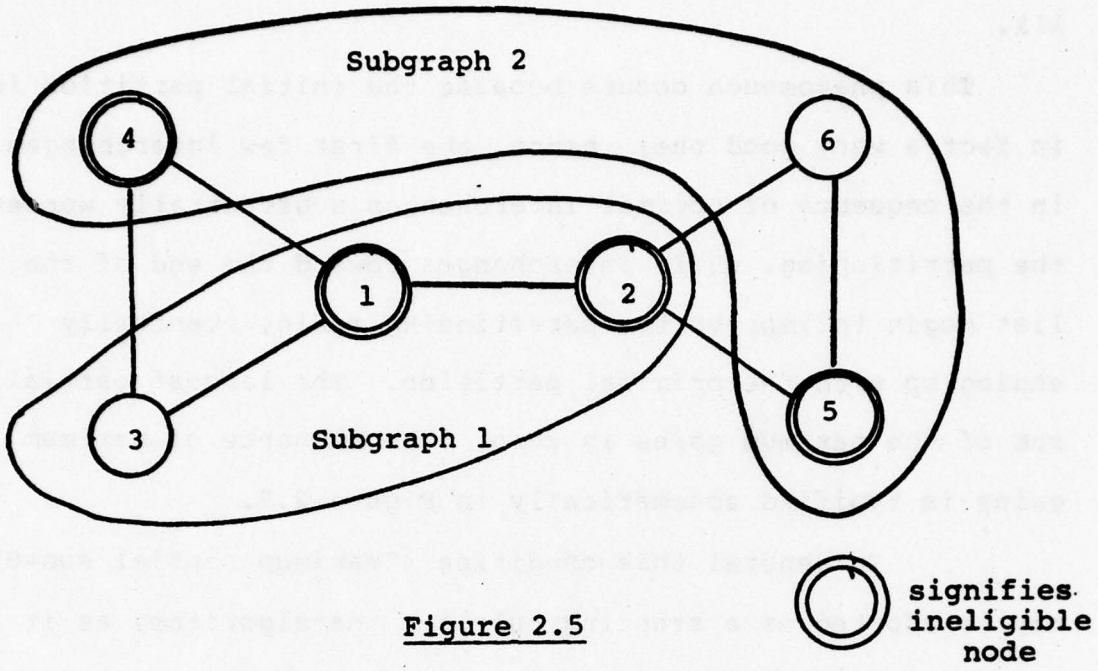


Figure 2.5

Result after second interchange completed.

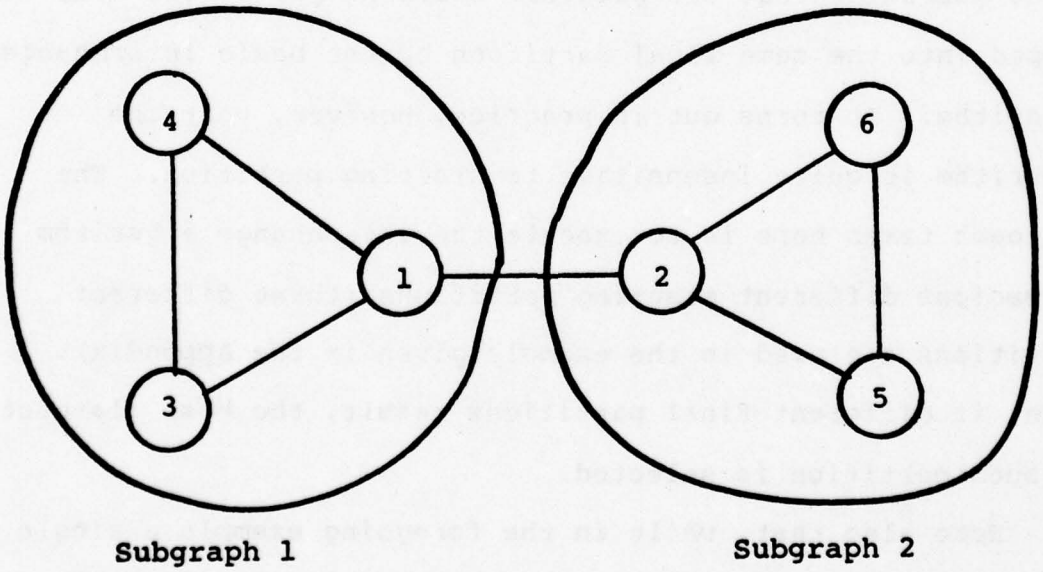


Figure 2.6

Best obtained partition during first pass.

all.

This phenomenon occurs because the initial partition is in fact a very good one; hence, the first few interchanges in the sequence of optimal interchanges substantially worsens the partitioning, while interchanges toward the end of the list begin to improve the partitioning again, eventually ending up with the original partition. The largest partial sum of the maximum gains is zero. The sequence of maximum gains is typified schematically in Figure 2.7.

In general this condition ("maximum partial sum=0") may be adopted as a stopping rule for the algorithm, as it serves to define the situation in which no further partition improvements can be made. It should be noted, however, that the particular resulting partition is dependent, to some extent, on the arbitrarily selected initial partition. There is no guarantee that all possible starting partitions will be mapped into the same final partition by the basic interchange algorithm. It turns out in practice, however, that the algorithm is quite insensitive to starting partition. The approach taken here is to execute the interchange algorithm on various different starting partitions (three different partitions are used in the example given in the Appendix). Then, if different final partitions result, the best (largest M) such partition is selected.

Note also that, while in the foregoing example a single cycle produced both a local and a global optimum decomposition, in general this would not necessarily occur.

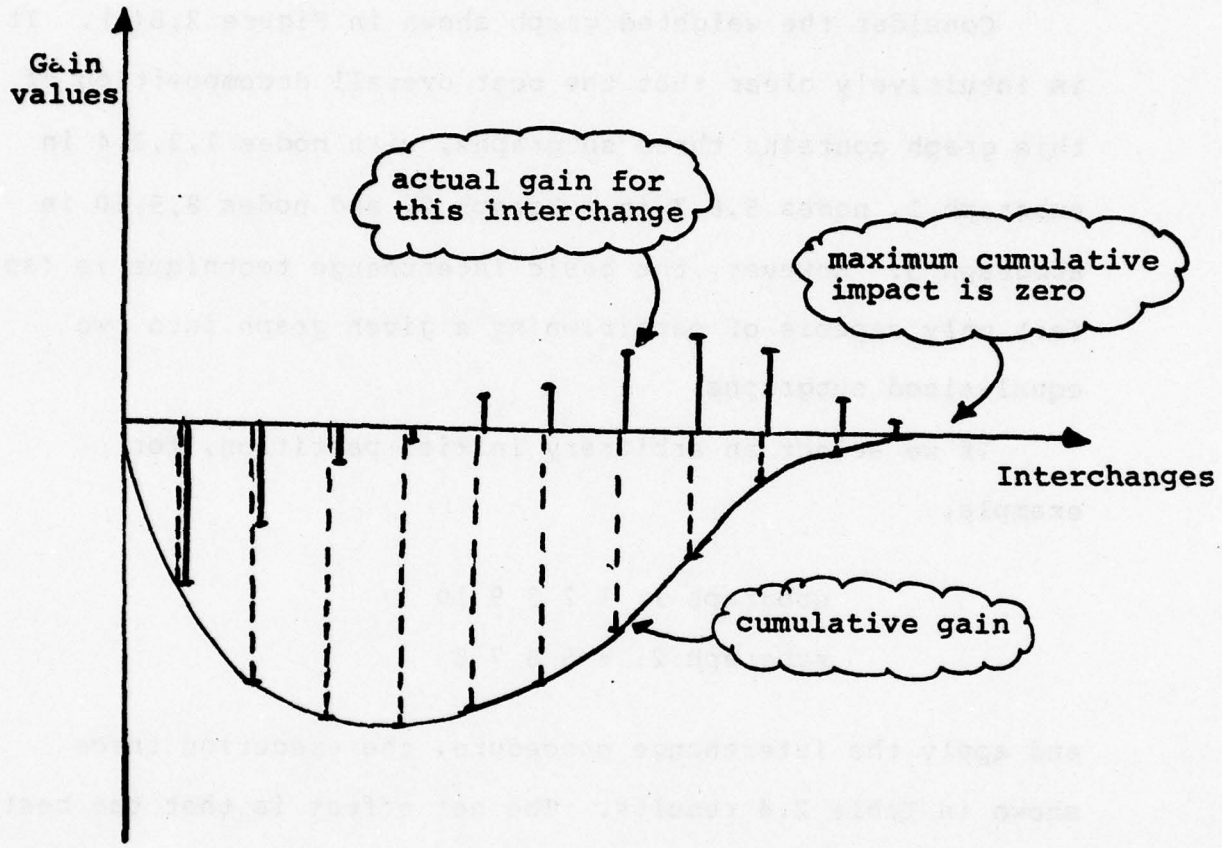


Figure 2.7

Depiction of interchange sequence wherein no improvement can be made in the starting partition.

Other example decompositions will be introduced shortly in which two or more "productive" passes through the interchange sequence are required before the stopping condition is reached.

2.3 A Second Example.

Consider the weighted graph shown in Figure 2.8(a). It is intuitively clear that the best overall decomposition of this graph contains three subgraphs, with nodes 1,2,3,4 in subgraph 1, nodes 5,6,7 in subgraph 2, and nodes 8,9,10 in subgraph 3. However, the basic interchange technique is (so far) only capable of partitioning a given graph into two equal-sized subgraphs.

If we select an arbitrary initial partition, for example,

subgraph 1: 1 2 3 9 10
subgraph 2: 4 5 6 7 8

and apply the interchange procedure, the execution trace shown in Table 2.4 results. The net effect is that the best bi-partition (partition into two equal-sized subgraphs) for this graph is that shown in Figure 2.9.

This partition, of course, is not a particularly good one, for the reason discussed above. It is the best that can be done under the constraint of two equal-sized subgraphs. In the next section a technique will be introduced that will allow us to relax this constraint, thereby greatly extending

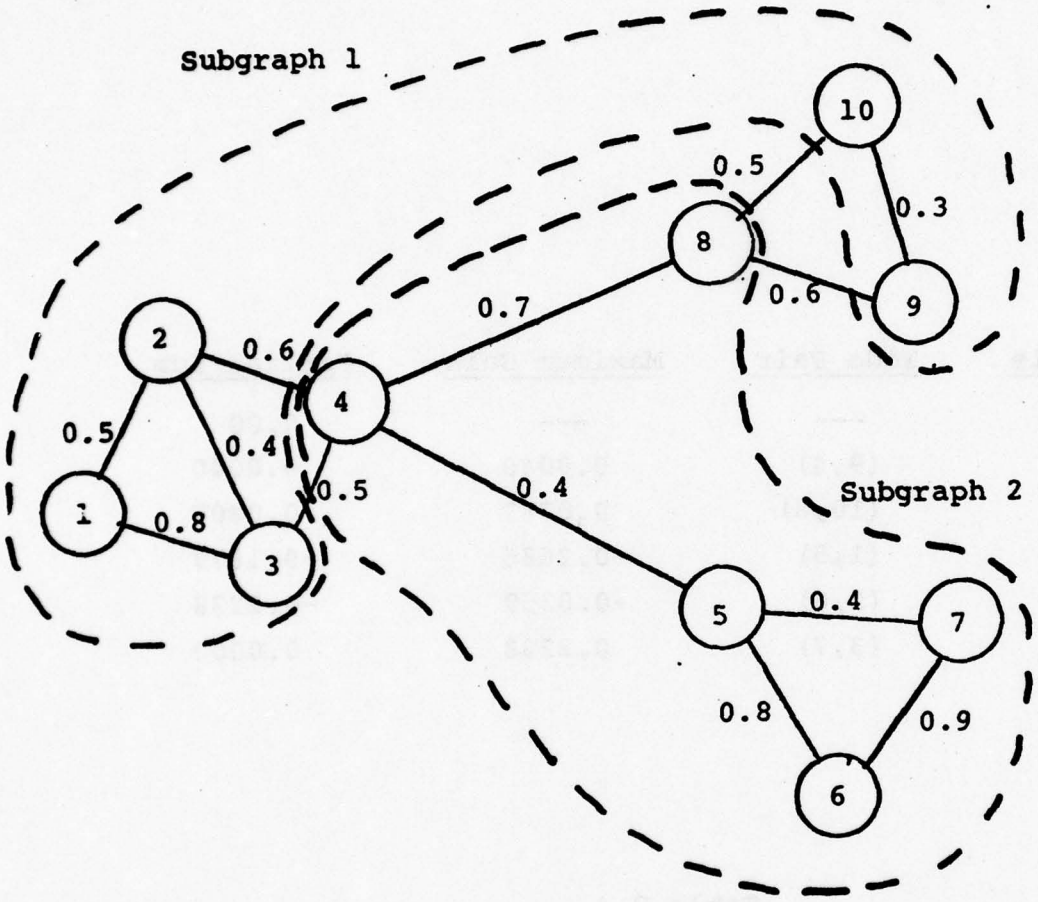


Figure 2.8

Graph for second example, showing initial partition.

Cycle	Node Pair	Maximum Gain	Partial Sum
0	---	---	0.00
1	(9,4)	0.0040	0.0040
2	(10,8)	0.0767	0.0807
3	(1,5)	-0.2686	-0.1879
4	(2,6)	-0.0359	-0.2238
5	(3,7)	0.2238	0.0000

Table 2.4

Interchange execution trace for second example.

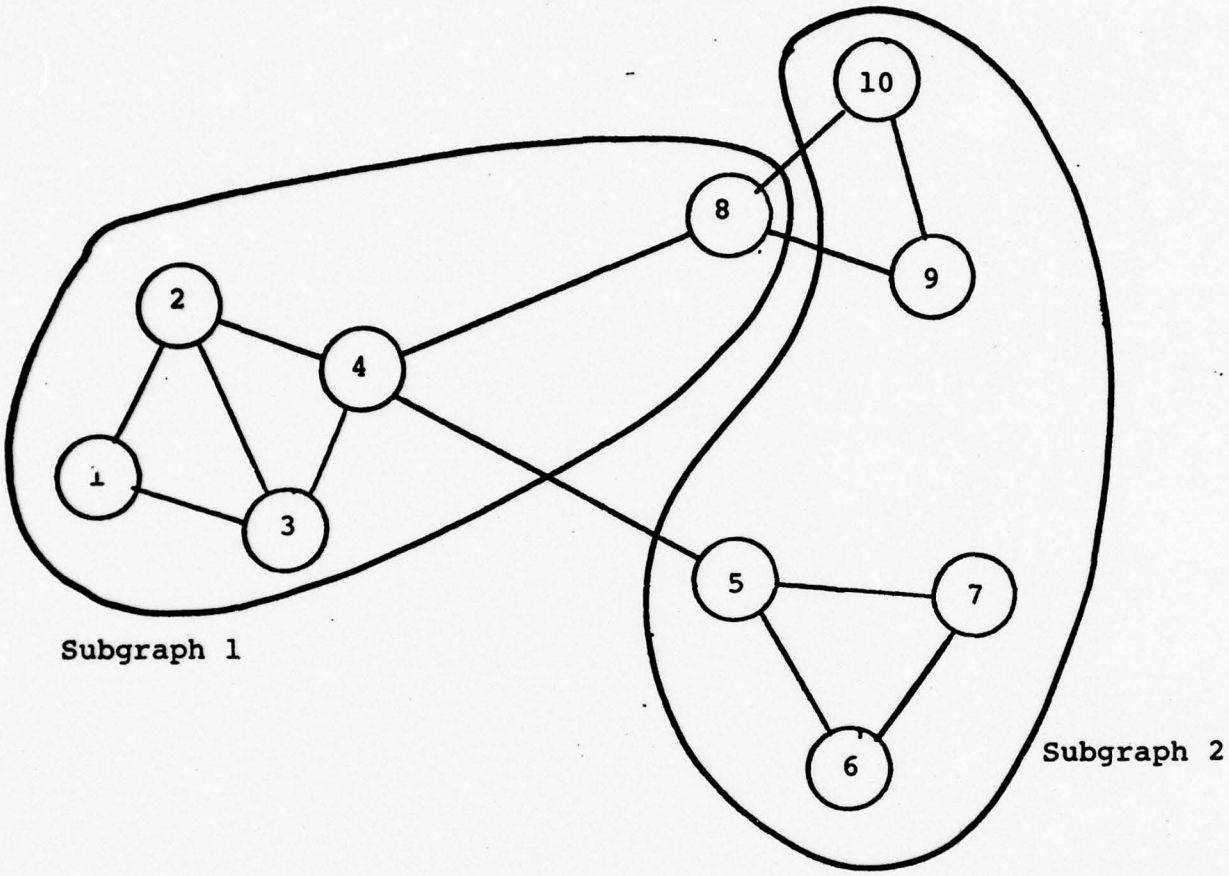
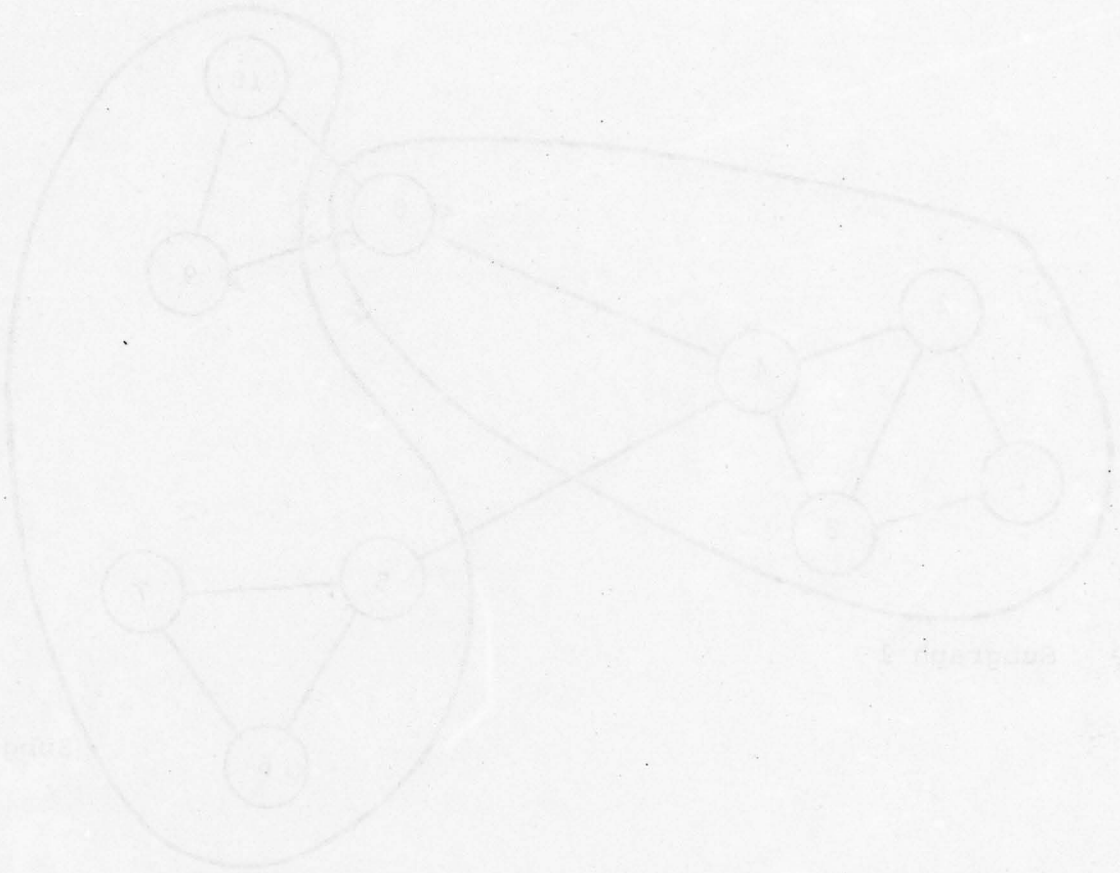


Figure 2.9

Best final partition for second example.

the utility of the basic interchange algorithm.



3 Algorithm Extensions - Partition Size Bounding and Unequal-sized Subgraphs.

Up to this point a target graph containing 2n nodes has been assumed, with the objective being to subdivide such a graph into two subgraphs containing n nodes each. These are quite constraining assumptions; there is no reason in general that it would necessarily be appropriate to divide a given graph exactly in half, nor that this even be possible (i.e., it might contain an odd number of nodes). Therefore these assumptions will now be relaxed. At the same time a very useful feature will be added to the basic algorithm, through the introduction of a simple extension.

3.1 Dummy Nodes.

Consider now the task of partitioning a given graph of n nodes (not "2n" as before) into two subgraphs, each of which contains at least n_1 nodes, and at most n_2 nodes, with $n_1 + n_2 = n$. It is clear that $n_1 \leq n_2$. Note that n need not be an even number.

To accomplish this task using the basic interchange algorithm, the original graph G is augmented through the introduction of a certain number of dummy nodes. The original graph together with the dummy nodes will be termed the augmented graph, G_a. The dummy nodes have no links to other nodes - that is, they are represented by a row and column of zeroes in the graph adjacency matrix.

In order to make the dummy node technique effective, a small adjustment must be made to the calculation of the decomposition goodness, M. Specifically, M must be defined so as to be unaffected by the introduction and distribution of the dummy nodes. This is quite easy to accomplish. The individual S_i and C_{ij} terms making up M all have the following general form:

$$f_1(\text{link weight sum})*(\text{average link weight})/f_2(\text{node count}),$$

where f_1 and f_2 are slightly different functional forms for the strength and coupling terms (see Section 4.1). Now, since the dummy nodes are disconnected from the rest of the network, their inclusion in the graph only impacts the denominator terms, i.e., node counts. By adding the minor adjustment to the M calculation that disconnected nodes not be included in the node count term, the presence of dummy nodes can be made transparent to the value of M. This modification to the definition of M has no other effect, as it is assumed that there are no "real" disconnected nodes in the original graph.

The interchange algorithm is then applied to the graph G_a in the usual way. Dummy nodes and regular nodes both participate in the interchange. The dummy nodes are simply dropped from the final partition, to yield to yield a decomposition of the original graph into subgraphs that are not necessarily of equal size.

3.2 Choice of Dummy Nodes.

The question arises as to how many dummy nodes should be added to the original graph. A simple empirical argument will demonstrate the reasoning behind the answer.

Consider a starting graph G of 5 nodes, and assume one dummy node is added to give an augmented graph G_a of six nodes. The basic interchange algorithm is then applied, and the resulting final partition will have three nodes in each subgraph. One of the two subgraphs will contain the single dummy node and two "real" nodes, while the other subgraph will contain three "real" nodes. In this case, then, the original graph is of size n = 5, and the size bounds on the resulting subgraphs are $n_1 = 2$ and $n_2 = 3$.

Now suppose three dummy nodes are added to the same starting graph G. Then the final subgraphs will each contain four nodes. Since the dummy nodes may end up split in any combination between the two subgraphs, it is clear that the size bounds on the subgraphs are now $n_1 = 1$ and $n_2 = 4$.

It would make no sense in this case to consider adding more than 3 dummy nodes (i.e., 5 or more), since a "trivial" partition - a partition in which one of the subgraphs contained only dummy nodes - might otherwise result.

Repetition of the above argument for other sizes of G leads to Table 3.1. This table illustrates the number of dummy nodes n_d that must be added to a graph of n nodes in order to insure that each of the two subgraphs will have at least n_1, and at most n_2, "real" nodes.

From Table 3.1 it is clear that in general n_d, the number of dummy nodes, must be chosen such that

$$n_d = n - 2*n_1 .$$

With this choice of n_d, the number of nodes in the augmented graph G_a is always even (as required by the basic interchange algorithm), since

$$n_t = n + n_d = n + (n - 2*n_1) = 2*(n - n_1).$$

For example, if it was desired to partition a graph of 43 nodes into two subgraphs such that each contained at least 10 nodes, and therefore at most 33 nodes, this would imply that $n_1 = 10$ and $n = 43$. Therefore it would be necessary to add

$$n_d = n - 2*n_1 = 43 - 2*(10) = 23$$

dummy nodes before applying the interchange algorithm to the augmented graph.

3.3 An Example Using the Dummy Node Technique.

To illustrate both the operation of the interchange algorithm while dummy nodes are present, as well as the effectiveness of the dummy node technique itself, consider the graph of Figure 2.8 from the previous section. This is the same graph used in the earlier example of Section 2.3. Recall that the basic interchange algorithm without dummy nodes produced the decomposition of this graph shown in Figure 2.9 - a clearly suboptimal decomposition that resulted primarily because of the exact halving property of the basic algorithm.

n = number of "real" nodes in G

n_1	5	6	7	8 . . . .
1	3	4	5	6
2	1	2	3	4
3	--	0	1	2
4	--	--	--	0
5	--	--	--	--
.				
.				
.				
.				

lower bound on partition size

Table entries are n_d values.

Table 3.1

Number of dummy nodes to be added to a graph of n nodes to insure a minimum subgraph size of n_1 nodes.

The basic algorithm is now applied to the current graph, with the minimum subgraph size taken to be $n_1 = 3$. An augmented graph G_a must be defined, where G_a consists of the original graph G plus

$$n_d = n - 2*n_1 = 10 - 2*3 = 4$$

dummy nodes. Schematically, then, the graph to be partitioned is that shown in Figure 3.1.

An initial partition is selected, say,

subgraph 1: 1 2 3 9 10 11 12

subgraph 2: 4 5 6 7 8 13 14

The decomposition goodness measure for this partition turns out to be M' = -0.29. The final partition is determined to be:

subgraph 1: 1 2 3 4 8 9 10

subgraph 2: 5 6 7 11 12 13 14

After the dummy nodes are discarded, the decomposition illustrated in Figure 3.2 results. Table 3.2 gives the execution trace for this partition.

With the addition of the four dummy nodes, the interchange algorithm produced a partition with the smallest subgraph containing at least (in this case, exactly) three nodes, the selected value of n_1. Also, the final partitioning (Figure 3.2) is clearly better than that obtained earlier without dummy nodes (Figure 2.9). It is better in the sense that it represents a "natural"

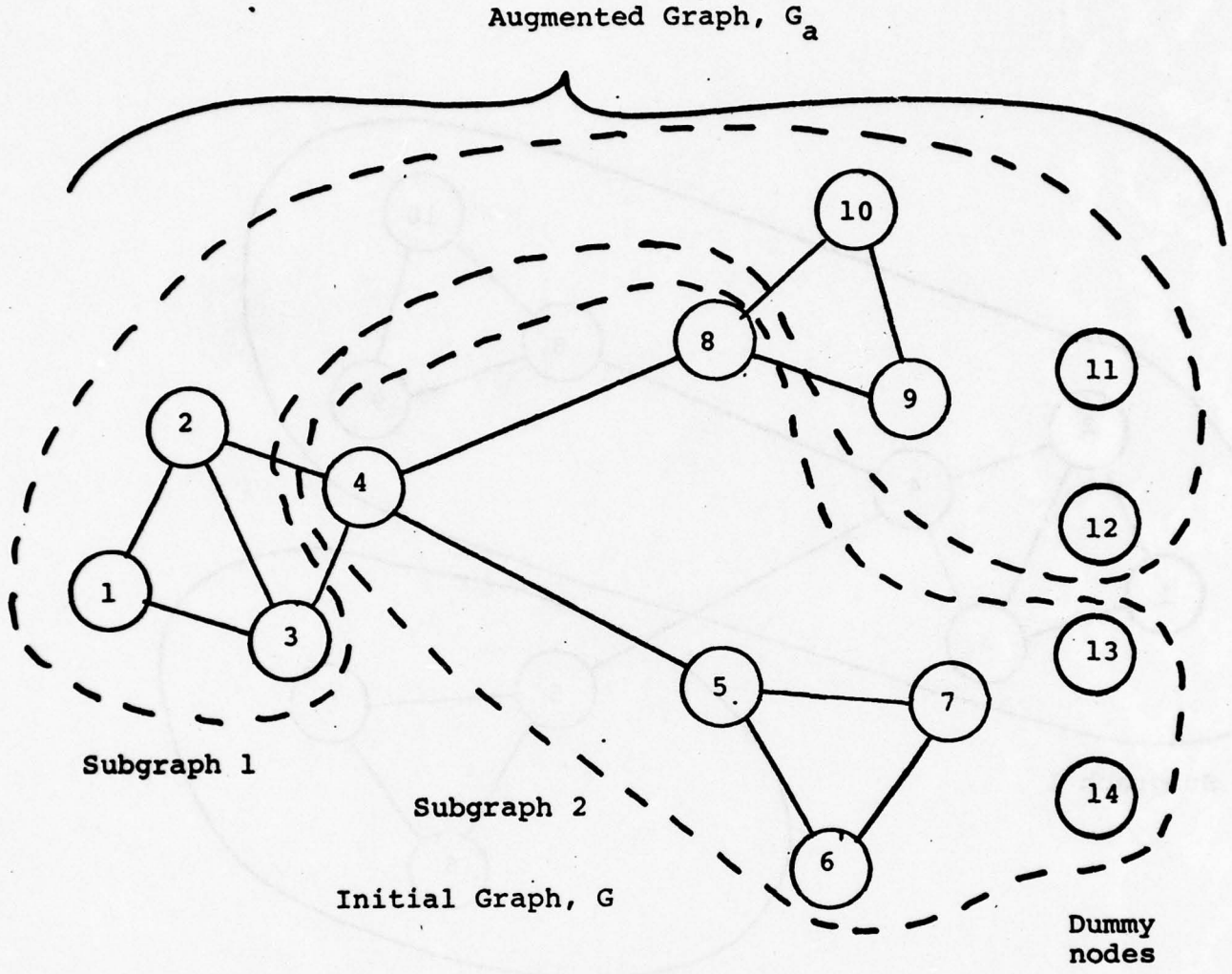


Figure 3.1

Graph for example of Section 3.3 showing dummy nodes and initial partition.

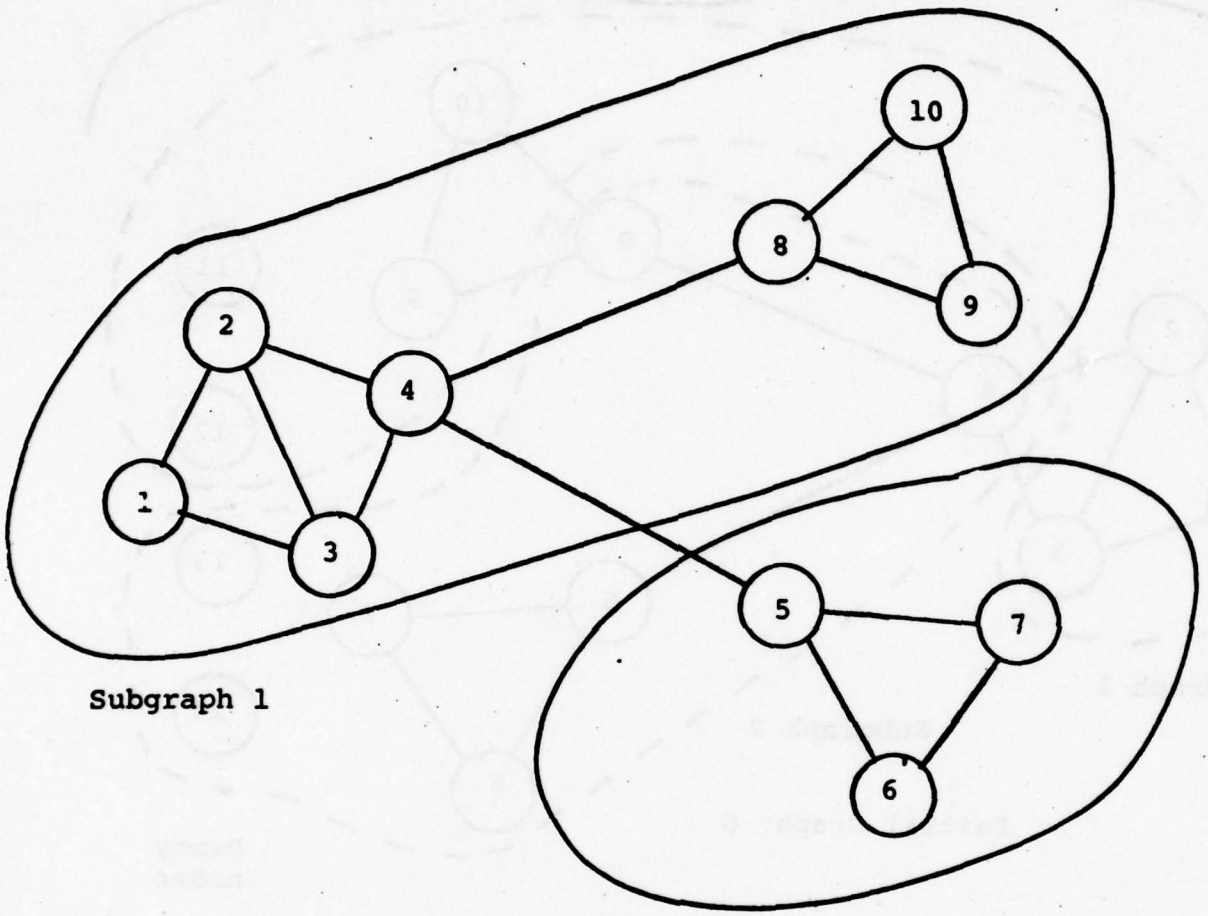


Figure 3.2

Final decomposition of example graph of Section 3.3 after one pass of the interchange algorithm (dummy nodes not shown).

Cycle	Node pair	Maximum gain	Partial sum
0	---	---	0.000
1	(12,8)	0.0876	0.0876
2	(11,4)	0.2284	0.3158
3	(10,13)	-0.2639	0.0519
4	(9,14)	0.0286	0.0804
5	(1,5)	-0.2686	-0.1881
6	(2,6)	-0.0359	-0.2239
7	(3,7)	0.2239	0.0000

Table 3.2

Interchange trace for example of Section 3.3

subdivision of the original graph. If the interchange algorithm was to be applied again, to subgraph 2 in Figure 3.2, it is reasonable to expect that the resulting partition would produce the optimal global partition of the original graph. (This is in fact what happens, as shown in Section 5.) This result should be contrasted with that shown in Figure 2.9, wherein it is clear that since the first partitioning is a poor one, further partitionings of the resulting subgraphs can never lead to the global optimum decomposition. As illustrated by this example, the dummy node technique contributes a very important and useful extension to the interchange algorithm.

3.4 Choosing the Number of Dummy Nodes.

Two alternative considerations arise in choosing n_d, the number of dummy nodes: whether or not it is desired to set the minimum subgraph size at some level greater than one node.

If the user of the interchange algorithm has in mind a specific minimum subgraph size n_1, then n_d is determined from the relationship $n_d = n - 2*n_1$, as discussed in Section 3.1. On the other hand, it may often happen that the investigator does not want to disallow any small subgraphs. Since the smallest possible subgraph consists of exactly one node, in such a case it would be necessary to set $n_1 = 1$. From this, n_d would be determined to be n - 2, so the total number of nodes in the augmented graph would be

$$n_t = n + (n-2) = 2*n - 2.$$

In the previous example, Section 3.3, if n_1 had been taken to be 1, it would have been necessary to add 8 dummy nodes, to produce an augmented graph of 18 nodes. In this case, the interchange algorithm would have produced the same partitioning obtained earlier, as is seen from the trace given in Table 3.3.

Initial Partition is:

Partition #1:	1	2	3	9	10	11	12	15	16
Partition #2:	4	5	6	7	8	13	14	17	18

Interchange Trace Table:

Cycle	Node pair	Maximum gain	Partial sum
0	---	---	0.0000
1	(16,8)	0.0876	0.0875
2	(15,4)	0.2284	0.3158
3	(11,13)	0.00	0.3158
4	(12,14)	0.00	0.3158
5	(10,17)	-0.2639	0.0519
6	(9,18)	0.0286	0.0804
7	(1,5)	-0.2686	-0.1881
8	(2,6)	-0.0359	-0.2239
9	(3,7)	0.2283	0.0000

Final partition is:*

Partition #1:	1	2	3	4	8	9	10	13	14
Partition #2:	5	6	7	11	12	15	16	17	18

* A second round of interchange calculations using this as the starting partition results in a maximum partial sum = 0.00.

Table 3.3

Initial and final partitions, and interchange trace, for example graph of Section 3.3 with $n_1 = 1$.

4 A Simplified Interchange Algorithm.

One of the reasons originally put forth for the development of this algorithm was the need for an efficient hierarchical partitioning technique. It was pointed out earlier, for instance, that the iterative partitioning technique studied by Andreu was interesting but impractical, due to its inefficient operational properties. In its present form, the new algorithm is much more efficient than the iterative approach. However, certain minor simplifications in the calculations of the interchange technique may be made in order to make it considerably faster still.

The main bottleneck in the present form of the algorithm centers on the gain calculation. If there are 2n eligible nodes, n^2 calculations are required to locate the largest gain value during the first cycle. In the next cycle, $(n-1)^2$ calculations will be required, etc. The total number of gain calculations will be, for a network originally containing 2n nodes,

$$T = \sum_{k=1}^{n} k^2 \quad .$$

This is a fairly large number of calculations steps for any non-trivial values of n. For example, if 2n = 200 nodes, then n = 100, and

$$T = 100*100 + 99*99 + \ldots + 2*2 + 1*1 = 338,350.$$

Some efficiencies are necessary if the interchange algorithm

is to be useful in problems of significant size.

Now, operation of the interchange technique could be made significantly faster by simplifying the interchange calculation itself - the elementary node pair interchange and corresponding measure calculation - since this step is executed so many times. In this section we develop such a simplification through incorporation of an approximate measure gain criterion, and give an example of its use. In practice the simplified algorithm, based on the approximate gain criterion has been found to behave essentially as well as the original algorithm, in terms of its capability for producing good partitions.

4.1 The Approximate Measure Gain Criterion.

The key simplification that can be made to improve the algorithm's efficiency involves an approximation to the measure gain calculation, $\Delta M(i,j)$.

Calculation of $\Delta M(i,j)$ basically consists of the calculation of the goodness measure for a particular bi-partition of a given graph, namely, the bi-partition obtained by interchanging the nodes x and y with respect to some initial bi-partition. Now, the definition of M for any graph that has been decomposed into k subgraphs is given as

$$M = \sum_{i=1}^{k} S_i - \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} C_{ij}$$

When the number of subgraphs k is equal to 2, this becomes

$$M = S_1 + S_2 - C_{12} .$$

The definitions of strength and coupling for a weighted graph (see Huff 79) are:

(1) strength S_i of subgraph i:

$$S_i = \frac{L_i - (n_i - 1)}{\frac{n_i(n_i - 1)}{2}} * \bar{w}_i$$

where L_i = the number of links within i;

n_i = the number of nodes within i;

$\bar{w}_i$ = the average weight on links within i.

(2) coupling C_{ij} between subgraphs i and j:

$$C_{ij} = \frac{L_{ij}}{n_i n_j} * \bar{w}_{ij}$$

where L_{ij} = the number of links between i and j;

n_i, n_j = the number of nodes in i,j;

$\bar{w}_{ij}$ = the average weight on links between i and j.

4.1.1 Simplifying S_i.

First consider the S_i term. This may be written

$$S_i = \frac{2}{n_i(n_i-1)} * L_i\bar{w}_i - \frac{2}{n_i} * \bar{w}_i$$

Now, a pairwise node interchange does not affect the value of n_i, so n_i is a constant as far as the calculation of $\Delta M(x,y)$ is concerned. Also, since $\bar{w}_i$ is an average, it may be assumed that its value is not substantially affected by interchanging nodes x and y. This is the case since most of the link weights that go into the calculation of $\bar{w}_i$ would remain unchanged. Therefore the entire second term in the S_i expression may be treated as a constant term. Since in the calculation of $\Delta M(x,y)$, this constant value will subtract out, it may be dropped henceforth.

Also, it may be noted that $\bar{w}_i$ is defined as W_i/L_i, where W_i is the sum of the weights on the links in subgraph i, and L_i is the link count in subgraph i. Hence the effective expression for S_i becomes

$$S_i = \frac{2}{n_i(n_i-1)} * W_i$$

4.1.2 Simplifying C_{ij} .

If W_{ij} is defined as the sum of the weights on links connecting subgraphs i and j, then $\overline{w}_{ij} = W_{ij}/L_{ij}$. Thus the expression for C_{ij} simplifies to

$$C_{ij} = \frac{1}{n_i n_j} * W_{ij}$$

4.1.3 Further Simplification of $\Delta M(x,y)$.

If we let S_1, S_2, and C_{12} be the values of strength and coupling for some starting bi-partitioned graph, and let S_1', S_2', and C_{12}' be the equivalent values after nodes x and y have been interchanged. Then

$$\Delta M(x,y) = (S_1' + S_2' - C_{12}') - (S_1 + S_2 - C_{12}).$$

Since equal-sized subgraphs are being considered, we may let $n = n_1 = n_2$, and the expression for $\Delta M(x,y)$ becomes

$$\Delta M(x,y) \cong \frac{2}{n(n-1)} \left[W_1' + W_2' - W_1 - W_2\right] - \frac{1}{n^2} \left[W_{12}' - W_{12}\right]$$

If we now approximate $n(n-1)$ by n^2, we may factor out a $1/n^2$ term and get

$$\Delta M(x,y) \cong \frac{1}{n^2} \left[2(W_1' + W_2' - W_1 - W_2) - (W_{12}' - W_{12})\right]$$

Now, the interchange algorithm is eventually going to make an ordinal comparison among all the $\Delta M(x,y)$ values for

all (x,y) combinations. Thus the constant $1/n^2$, which will be present as an external multiplier in each term, may be dropped from the $\Delta M(x,y)$ expression. This is so since it will be present in all such terms, and consequently serves only to scale all the values, having no net impact on their ordinal relationship.

Therefore the effective value for $\Delta M(x,y)$ reduces to

$$\Delta M(x,y) \propto 2(W_1' + W_2' - W_1 - W_2) - (W_{12}' - W_{12}) \quad \ldots\ldots \quad (1)$$

In order to go further with the reduction of the expression for $\Delta M(x,y)$, it is necessary to introduce some new terminology. We define:

(1) E_x = the sum of the weights on the external links connected to node x - that is, those links that connect node x to some node in the other subgraph;

(2) I_x = the sum of the weights on the internal links connected to node x - that is, those links that connect node x to other nodes within the same subgraph;

(3) O_x = the sum of the weights on the "other" links within the same subgraph as node x - i.e., those links not connected to node x;

(4) O_{xy} = the sum of the weights on the "other" links interconnecting the two subgraphs - i.e., those links not connected to either node x nor node y;

(5) w_{xy} = the weight on the link connecting node x to node y (if no such link exists, w is assumed to be zero).

Now, the W terms in the earlier expression for $\Delta M(x,y)$ may be translated into terms involving E,I,O, and w. In particular, it is fairly easy to see that

$$W_1 = I_x + O_x \quad \ldots\ldots (2)$$

$$W_2 = I_y + O_y \quad \ldots\ldots (3)$$

$$W_1' = E_y - w_{xy} + O_x \quad \ldots\ldots (4)$$

$$W_2' = E_x - w_{xy} + O_y \quad \ldots\ldots (5)$$

$$W_{12} = E_x + E_y - w_{xy} + O_{xy} \quad \ldots\ldots (6)$$

$$W_{12}' = I_x + I_y + w_{xy} + O_{xy} \quad \ldots\ldots (7)$$

The only slightly confusing part in the above equations involves the role of w_{xy}. When x and y are interchanged, all associated external links become internal, and vice versa, with the exception of the link connecting x and y (if any).

Now, if equations (2) through (7) are substituted into relation (1), the latter reduces to

$$\begin{aligned} \Delta M(x,y) \quad \propto \quad & 2\left[(E_x - I_x) + (E_y - I_y) - 2w_{xy}\right] \\ & - \left[(I_x - E_x) + (I_y - E_y) + 2w_{xy}\right] \\ & = 3\left[(E_x - I_x) + (E_y - I_y) - 2w_{xy}\right] \quad \ldots\ldots (8) \end{aligned}$$

It is no surprise that the "O" terms in (2) through (7) cancel out of (8), since by definition these terms represent a part of the graph structure that is unaffected by the interchange of nodes x and y.

Finally, we define

$$D_x = E_x - I_x \text{ , and}$$

$$D_y = E_y - I_y \text{ .}$$

Also, for the same reason as discussed earlier (immateriality of scale factors) we drop the factor 3 in relation (8). This yields the final result,

$$\Delta M(x,y) \propto D_x + D_y - 2w_{xy} \quad \ldots\ldots (9)$$

Expression (9) is the simplified measure gain expression. To be strictly correct, the right-hand side of expression (9) is <u>proportional</u> to an <u>approximation</u> of the exact measure gain $\Delta M(x,y)$. As was argued, proportionality constants are immaterial for purposes of the interchange algorithm's proper functioning; also, the approximations involved in the foregoing reduction are relatively minor, so that we would expect the simplified algorithm to still do a good job in partitioning graphs in a manner appropriate to our needs - i.e., in a manner that will usually locate a very good, or optimal, partition as measured by the true value of M.

4.2 A Verification Exercise.

To briefly illustrate the validity of relation (9) in the previous section, consider the graph shown in Figure 4.1, with the associated subgraphs 1 and 2. Two nodes x and y, one in each subgraph, are shown; also, the weights on certain links are specified, for illustrative purposes to follow.

In Figure 4.2, the same graph is shown, after nodes x and y have been interchanged between the two subgraphs. In terms of the foregoing definitions, the following may be seen to hold:

$$W_1 = w_1 + w_2 + o_1$$
$$W_2 = w_5 + w_6 + o_2$$

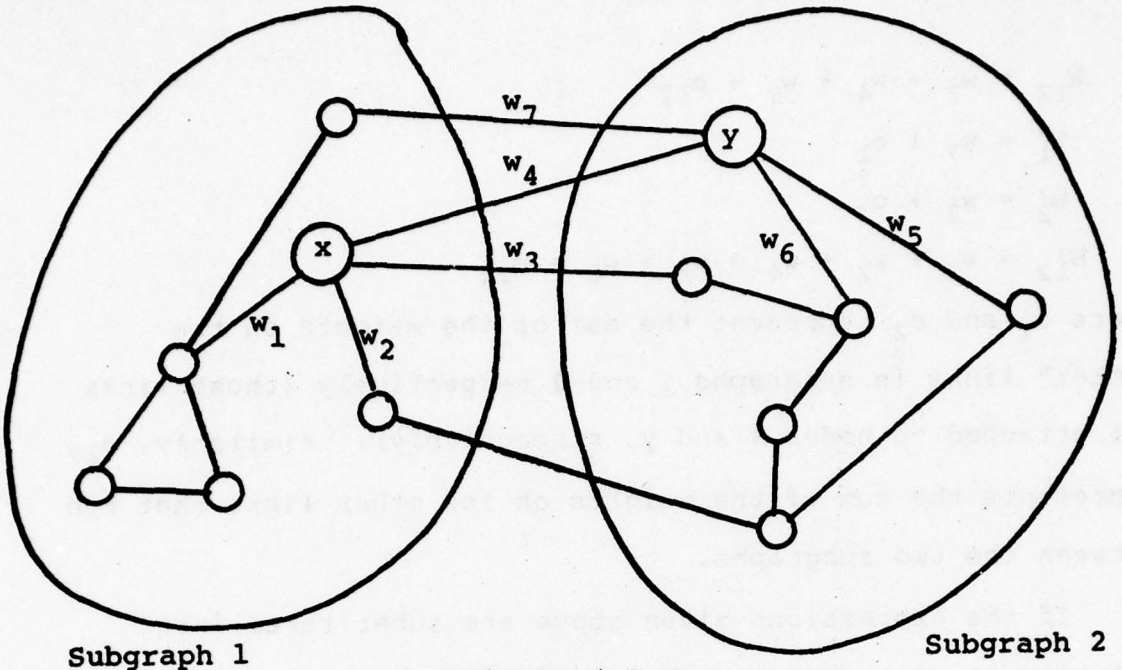


Figure 4.1

Initial graph for verification exercise

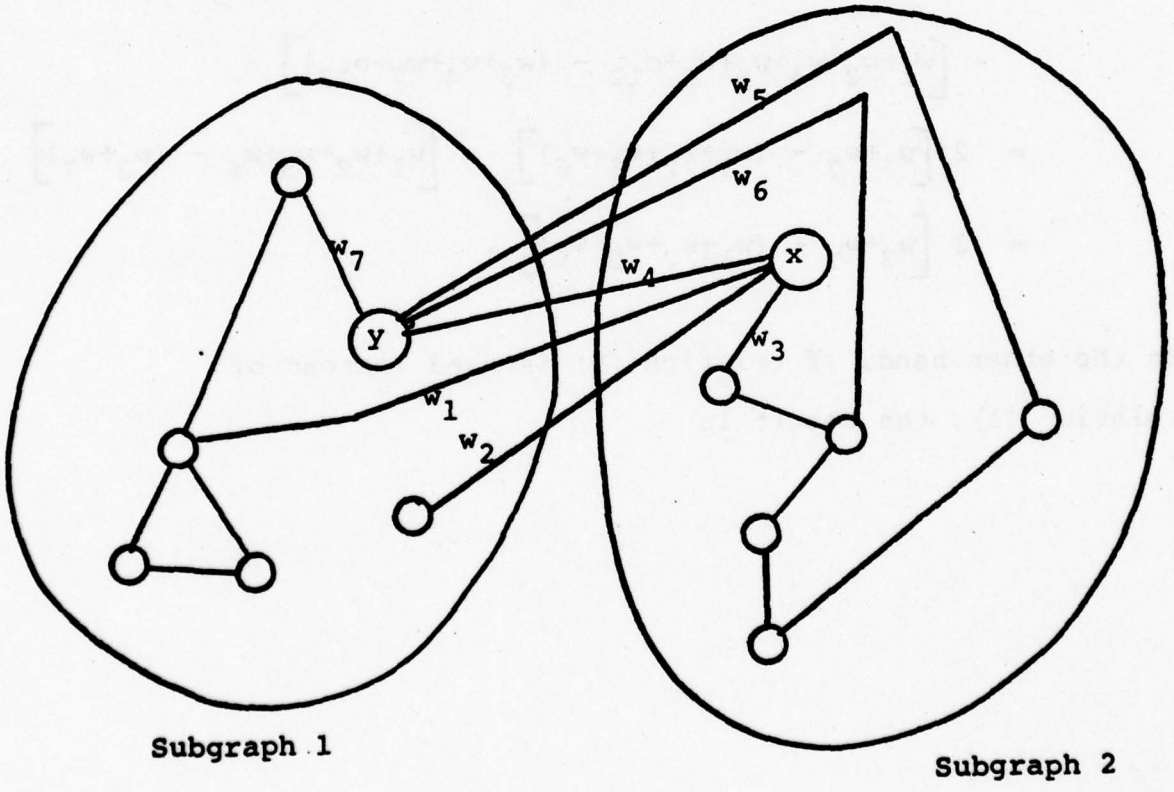


Figure 4.2

Graph of Figure 4.1 after nodes x and y have been interchanged.

$$W_{12} = w_3 + w_4 + w_7 + o_{12}$$

$$W_1' = w_7 + o_1$$

$$W_2' = w_3 + o_2$$

$$W_{12}' = w_1 + w_2 + w_4 + w_5 + w_6 + o_{12}$$

where o_1 and o_2 represent the sum of the weights on the "other" links in subgraphs 1 and 2 respectively (those links not attached to nodes x and y, respectively); similarly, o_{12} represents the sum of the weights on the other links that run between the two subgraphs.

If the expressions given above are substituted into relation (1) for $\Delta M(x,y)$, the result is

$$\begin{aligned}\Delta M(x,y) \propto\ & 2\left[w_7+o_1+w_3+o_2 - (w_1+w_2+o_1) - (w_5+w_6+o_2)\right] \\ & - \left[w_1+w_2+w_4+w_5+w_6+o_{12} - (w_3+w_4+w_7+o_{12})\right] \\ =\ & 2\left[w_3+w_7 - (w_1+w_2+w_5+w_6)\right] - \left[w_1+w_2+w_5+w_6 - (w_3+w_7)\right] \\ =\ & 3\left[w_3+w_7 - (w_1+w_2+w_5+w_6)\right] .\end{aligned}$$

On the other hand, if relation (9) is used instead of relation (1), the result is

$$\Delta M(x,y) \propto D_x + D_y - 2w_{xy}$$

$$= (E_x - I_x) + (E_y - I_y) - 2w_{xy}$$

$$= (w_3+w_4-w_1-w_2) + (w_4+w_7-w_5-w_6) - 2w_4$$

$$= w_3 + w_7 - (w_1+w_2+w_5+w_6) ,$$

which, except for the irrelevant factor 3, is the same as that obtained from relation (1). Thus the foregoing simplification analysis is verified in this example.

4.3 Summary of the Simplified Interchange Algorithm.

To review the approximate gain technique: in order to determine the relative gain that would result from interchanging the node pair (x,y), where node x currently resides in subgraph 1 and node y in subgraph 2, the quantities D_x and D_y are calculated. In general, D_k is calculated by summing the weights on the links that extend from node k to nodes in the other subset, then subtracting the sum of the weights on the links that extend from node k to other nodes within the same subset. Once D_x and D_y have been determined, the approximate gain term G(x,y) is calculated from the formula

$$G(x,y) = D_x + D_y - w_{xy} .$$

The quantity G(x,y) is the approximation to the quantity $\Delta M(x,y)$ used earlier in the basic interchange algorithm.

By inspection of the algorithms' code, the estimated

ratio of calculation times for G(x,y) to ΔM(x,y) is on the order of 1 to 100 - i.e., the calculation time for G(x,y) is approximately 1 percent of the calculation time for ΔM(x,y). This large difference is the source of the efficiency improvement that results from using the approximate gain criterion.

Once G(x,y) has been calculated for every eligible node pair, the procedure continues as before (see Section 2.3) to locate the largest gain value, logically interchange the corresponding nodes, mark those nodes as ineligible for further consideration, then repeat with one fewer node in each subset. The final calculation of the maximum partial sum of gain values, which determines the terminal partition, is also carried out exactly as before.

4.4 Further Efficiency Improvements.

The calculation of G(x,y) is almost fully uncoupled with respect to the two nodes x and y. That is, with the exception of the w_{xy} term, G(x,y) involves D_x and D_y, which can be calculated individually for nodes in subgraphs 1 and 2, respectively. This gives us a means of improving the speed of the interchange algorithm still further. The basic idea is as follows. First, all the D_x and D_y terms are sorted into descending order. Then G(x,y) is calculated for each pair (x,y) such that x and y are within some arbitrary count δ from the top of the D_x and D_y lists, respectively. The largest such value of G(x,y) is then selected, thereby

identifying the next pair of nodes to be interchanged. This technique avoids a global search over all (x,y) pairs at the cost of possibly missing the largest gain value. However, the liklihood of missing the overall largest value of G(x,y) may be made as small as desired by setting the value of high enough. The extra work involved in this approach consists of sorting two lists of n elements each, while the saved calculations involve a proportion $(n - \delta)/n$ of the n potential calculations. The extra improvement in algorithm efficiency mounts rapidly with n. Judging by a few cases examined in detail, δ need not be very large to insure locating the lragest G(x,y) value - e.g., δ = 10 percent seems to be more than sufficient in the cases examined. Additional studies may suggest ways in which to estimate δ on the basis of w_{xy}, D_x, and D_y values.

4.5 Comparison of the Basic and Simplified Interchange Algorithms.

Consider once again the 10-node graph introduced in Figure 2.8(a). This graph is augmented with the addition of four dummy nodes, and both the basic and simplified interchange algorithms are applied to it, using the starting partition given below:

subgraph 1: 1 3 5 7 9 11 13

subgraph 2: 2 4 6 8 10 12 14

The traces for the first pass of each algorithm are shown in Table 4.1(a) and Table 4.1(b) respectively.

Basically, both versions of the algorithm produced the same final partition at the end of the first pass, following essentially the same locus of interchanges. When a second pass was taken through each version of the algorithm, the results were again identical and the loci of interchanges were also nearly identical. Of course, in all cases the actual numerical values of gain were rather different between the two versions, since the gain calculation differences are the essence of the difference between the two algorithms. Also, the order in which dummy nodes were handled varied somewhat due to minor implementation differences. This has no impact on the interchanges of "real" nodes, which is the important issue. The main invariant factor is the ordinal relationship among the gain values in both cases. Inspection of Tables 4.1 (a) and (b) shows that this invariance of ordinality is indeed maintained for this example.

Cycle	Node pair	Maximum Gain	Partial Sum	
0	---	---	0.000	
1	(9,6)	0.338	0.338	
2	(13,2)	0.085	0.423	
3	(11,4)	0.139	0.562	(optimal)
4	(1,12)	-0.214	0.348	
5	(3,14)	-0.014	0.334	
6	(7,8)	-0.240	0.094	
7	(5,10)	-0.094	0.000	

Final Partition Is:

Partition #1: 1 2 3 4 5 6 7

Partition #2: 8 9 10 11 12 13 14

Table 4.1(a)

Interchange trace for example of Section 4.5, without simplifications.

Cycle	Node Pair	Maximum Gain	Partial Sum	
0	---	---	0.00	
1	(9,6)	2.60	2.60	
2	(11,2)	0.30	2.90	
3	(13,4)	0.80	3.70	(optimal)
4	(1,12)	-1.30	2.40	
5	(3,14)	-0.10	2.30	
6	(7,8)	-1.70	0.60	
7	(5,10)	-0.60	0.00	

Final Partition Is:

Partition #1: 1 2 3 4 5 6 7

Partition #2: 8 9 10 11 12 13 14

Table 4.1(b)

Interchange trace for example of Section 4.5, with simplifications.

5 Hierarchical Partitioning Using the Interchange Technique.

The original objective in developing this partitioning algorithm was to devise a means of decomposing a graph into subgraphs so as to locate the best overall decomposition as determined by the value of M. At this point we have demonstrated an efficient algorithm that is capable of dividing a given graph into <u>two</u> subgraphs, with the option of controlling the size of the subgraphs. All that is required to meet the original objective is some sort of "master" algorithm, within which the interchange algorithm may be imbedded, which will perform the following tasks:

- keep track of the original graph and the list of subgraphs
- decide which subgraph on the list should be partitioned next (using the interchange routine)
- generate initial partitions to accompany each call to the interchange routine;
- monitor the results of the interchange execution for the occurrence of the stopping condition (maximum partial sum of gains = 0) for any given starting subgraph and starting partition.

5.1 The Subgraph Selection Rule.

The only really non-trivial issue as far as the master algorithm is concerned involves the nature of the decision rule to be used to select the next subgraph for partitioning. Consider for illustrative purposes the initial 10-node graph given earlier (see Figure 2.8(a)). As was shown in the

previous section, the best general bi-partitioning of this graph is that illustrated in Figure 5.1 below.

Now we wish to determine which of the two subgraphs indicated in the figure to select for the next round of partitioning. In this simple case the answer is intuitively clear: subgraph 1 should be partitioned next, as it has the clearest division into two subgraphs.

The "next subgraph" decision rule may be made operational in the following way. First, the strength S of each subgraph i in the current decomposition must be calculated. There may be anywhere from 1 to (n-1) such subgraphs. Then the subgraph with the <u>smallest</u> strength measure is selected as the next subgraph to be partititoned.

The logic behind this choice is simple: the lower the subgraph strength, the higher its propensity for being subdivided. Also, this rule is a reverse image of the rule for deciding which subset pair to merge next when clustering techniques are used in handling the decomposition problem (see Huff 79).

In the case of Figure 5.1, the strengths of the two subsets are 0.078 and 0.233 respectively. Thus the lowest strength rule would select subgraph 1 as the next subgraph to partition, which is certainly the correct choice in this case, as discussed earlier.

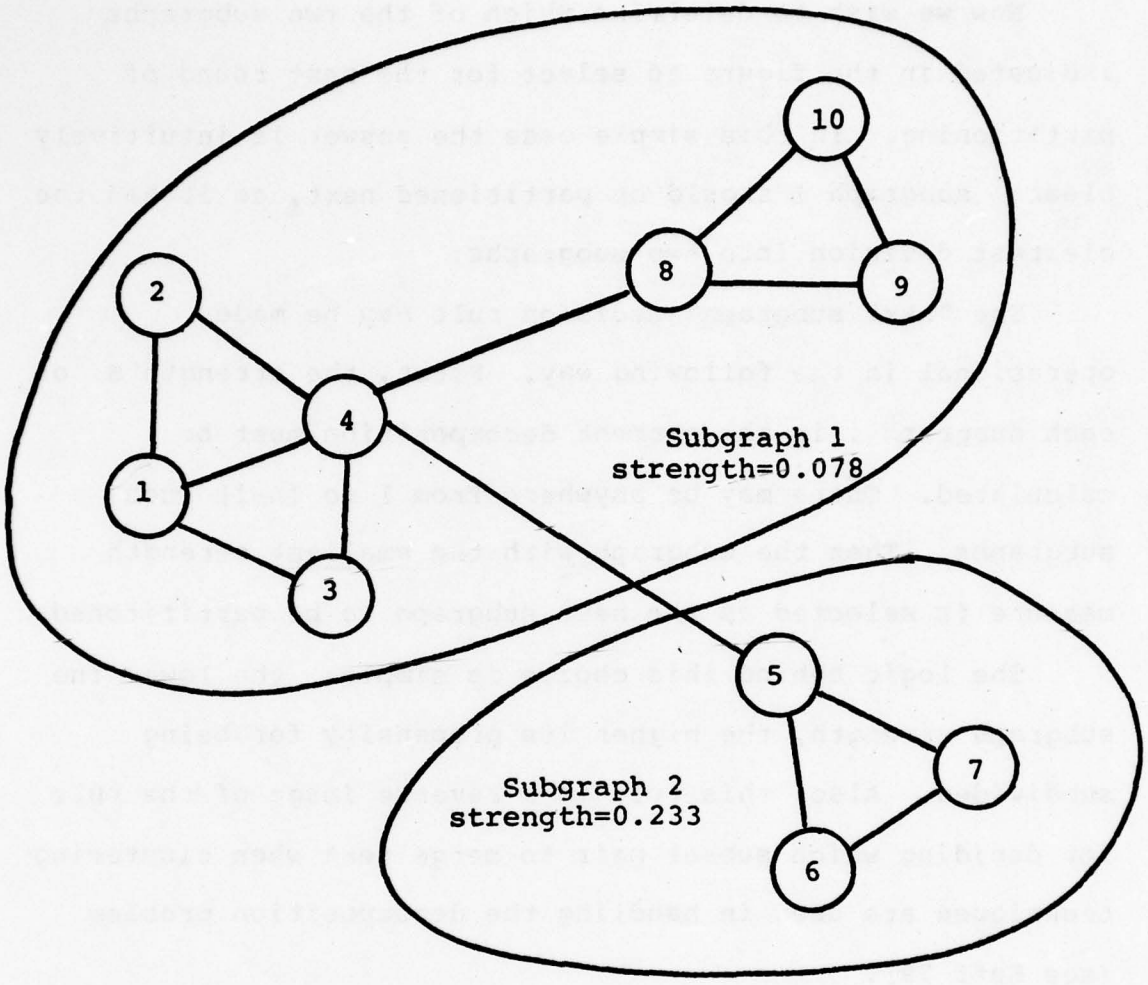


Figure 5.1

Partitions of the graph from Section 2.3 after one pass of the interchange algorithm, showing subgraph strengths.

5.2 A Stopping Rule for the Master Algorithm.

What stopping rule might be used to eventually halt the hierarchical partitioning procedure being implemented by the master algorithm? One obvious rule would be to continue partitioning until all current subgraphs are too small to be subdivided further - i.e., until $n_i < 2*n_1$, where n_i is the dimension of subgraph i and n_1 is the lower bound on subgraph size. Note that if $n_1 = 1$, the master algorithm would have to continue partitioning until all subgraphs consist of exactly one node.

It may be possible to devise a simple stopping rule that would halt the master algorithm well before this point is reached, however. Experience with partitioning a variety of graphs with the interchange method suggests that, generally speaking, the maximum value of M is attained after a fairly small number of "splits" have been made. In the above example, for example, only two splits were required to reach the optimum, whereas a total of nine splits were implemented before completely decomposing the original graph. Hence some simple rule such as stopping after k successive steps have yielded a lower value of M than the preceding step may prove useful and effective. A study of such stopping rules will be conducted in the future.

5.3 The Master Algorithm.

One possible implementation of a master algorithm for controlling the interchange partitioning method is given below.

(1) Get data on graph structure, link weights; also get value of n_1 (subgraph lower bound).

(2) Assign the entire graph as the current partition (CP);

CP <-- entire graph.

(3) Calculate the measure M for CP;

M_{opt} <-- M; CP_{opt} <-- CP.

(4) Calculate the strength S for each subgraph i in CP. Locate the subgraph with the lowest value; assume this is subgraph ℓ.

(i.e., $S_\ell \leq S_i \ \forall \ i$).

(5) Call the Simplified Interchange Algorithm to partition subgraph ℓ.

(6) Update CP: CP <-- (old CP with subgraph ℓ replaced by its sub-partitions as determined in step (5)).

(7) Calculate M for CP.

(8) If $M > M_{opt}$ then do:

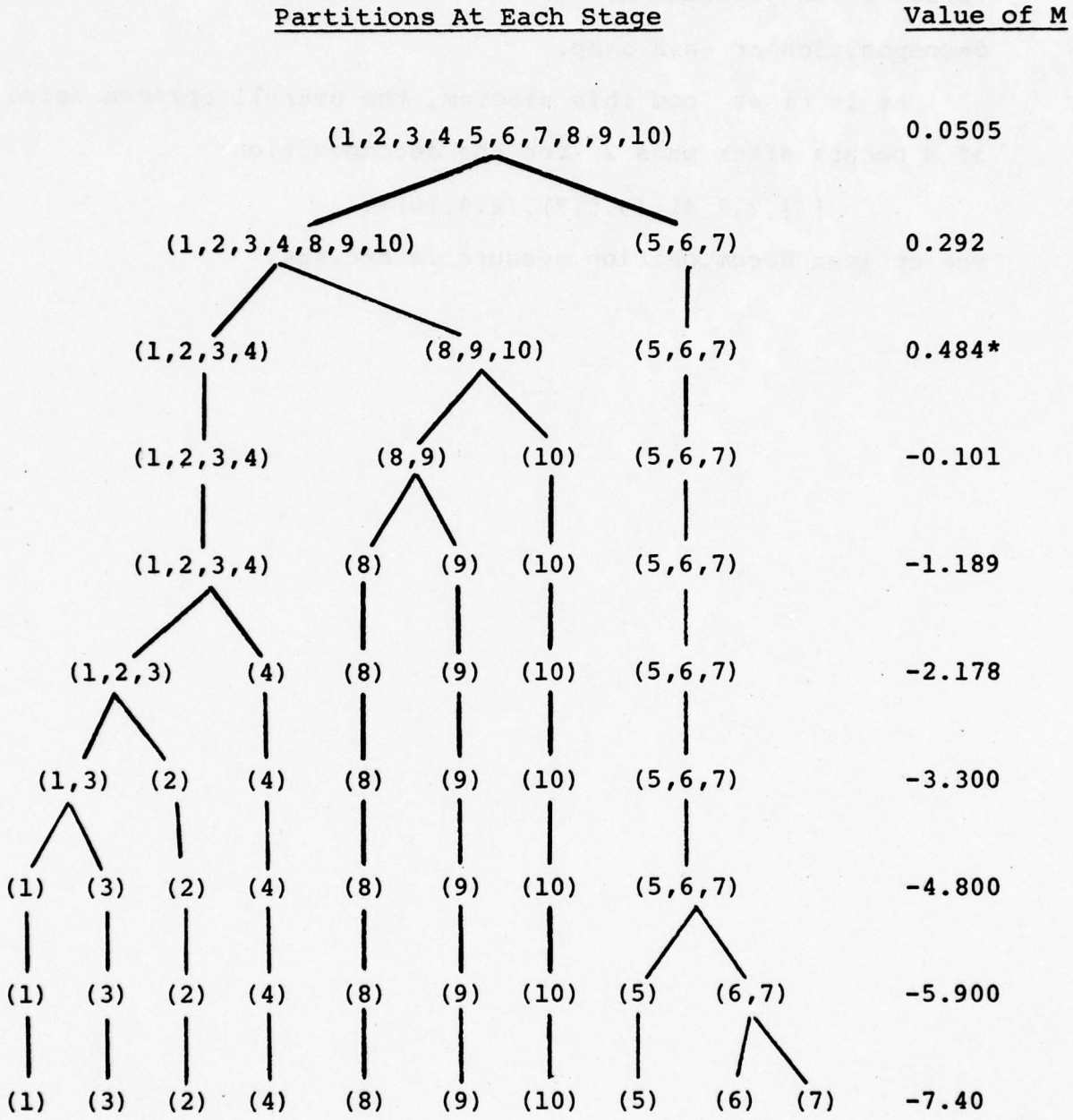
M_{opt} <-- M

CP_{opt} <-- CP.

(9) If there are further subgraphs that can be partitioned, then go to step (4); otherwise STOP.

5.4 An Example Using the Master Algorithm.

To complete this paper an example decomposition "from start to finish" is presented. Once again the ten-node example graph discussed in previous examples (see Figure 2.8(a)) will be used. Appendix A contains a complete trace of the execution path of the master algorithm in partitioning this graph. Figure 5.2 contains a schematic



*optimum measure

Figure 5.2

Decomposition "tree" showing complete decomposition of graph from Section 2.3.

representation of the partitioning process, and indicates the value of the goodness measure of the intermediate decomposition at each step.

As is clear from this diagram, the overall optimum value of M occurs after pass 2, for the decomposition

{(1,2,3,4),(5,6,7),(8,9,10)}.

The optimum decomposition measure is M=0.484.

6 Summary.

A new algorithm for performing top-down hierarchical partitioning upon a weighted graph has been defined and discussed. The algorithm functions by performing pairwise node interchanges and calculating the corresponding gain to the partition goodness measure M that results. The node pair exchanges that produce the maximum gain are retained, and those nodes are marked as ineligible for further interchanges, then the process repeated until no available node pairs remain. The sequence of node pair interchanges that leads to a maximum partial sum of the maximum gain values is then implemented. If the maximum partial sum is zero, then no interchanges are effected and no further improvement is possible.

The interchange algorithm is "driven" by the partition goodness measure M. In order to improve the efficiency of the algorithm certain approximations to the calculation of M were introduced. These approximations were verified by way of example, and were shown to lead to an equivalent set of interchanges in one example problem. Experience with the approximate gain technique has shown it to be essentially as effective, in terms of identifying good decompositions, as is the exact technique. The approximate technique is on the order of 100 times faster than the exact technique.

A "master" algorithm appropriate for controlling and driving the node interchange algorithm was also introduced.

A complete trace of the decomposition of a 10-node graph, using this master algorithm and the approximate interchange technique is included in the appendix.

It is believed that the interchange algorithm is a useful contribution to the body of general graph partitioning techniques. In the present context (SDM), it is especially powerful, as it may be used as the core of a top-down partitioning search for the optimal graph decomposition. Since our experience with SDM has consistently shown that the optimal decompositiion tends to reside fairly near the top of the decomposition tree (see Figure 5.2), a top-down search will generally reach the optimum fairly rapidly as compared with bottom-up clustering approaches used in this work previously. In later studies we intend to further investigate the relative strengths, in terms of both costs and effectiveness, of the top-down versus bottom-up decomposition techniques.

REFERENCES

1. Aho, A. V., J. E. Hopcroft, and J. D. Ullman: The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

2. Anderberg, M. R.: Cluster Analysis for Applications, Academic Press, 1973.

3. Andreu, R.: A Systematic Approach to the Design of Complex Systems: Application to Database Management Systems, unpublished PhD Thesis, M.I.T., Sloan School of Management, 1978.

4. Boehm, Barry: "Software and Its Impact: A Quantitative Assessment", Datamation, vol. 18, no. 5, May 1973.

5. Corneil, D. G., and M. S. Woodward: "A Comparison and Evaluation of Graph Theoretical Clustering Techniques", Infor, vol. 16, no. 1, February 1978.

6. Deo, N.: Graph Theory with Applications to Engineering and Computer Science, Prentice-Hall, 1974.

7. Ford, L. R., and D. R. Fulkerson: Flows in Networks, Princeton University Press, 1962.

8. Gottleib, J. C., and S. Kumar: "Semantic Clustering of Index Terms", Journal of the ACM, vol. 15, no. 4, October 1968.

9. Hartigan, J.: Clustering Algorithms, Wiley, 1975.

10. Huff, S. L., and S. E. Madnick: "An Approach to Constructing Functional Requirement Statements for System Architectural Design", M.I.T. Sloan School Technical Report No. 6, NTIS No. A057802, May 1978a.

11. Huff, S. L., and S. E. Madnick: "An Extended Model for a Systematic Approach to the Design of Complex Systems", M.I.T. Sloan School Technical Report No. 7, NTIS No. A058565, 1978b.

12. Huff, S. L.: "Analysis Techniques for Use With the Extended SDM Model," M.I.T. Sloan School Technical Report no. 9, February 1979.

13. Karp, R. M.: "On the Computational Complexity of Combinatorial Problems", Networks, vol. 5, no. 2, 1975.

14. Kernighan, B. W., and S. Lin: "An Efficient Heuristic Procedure for Partitioning Graphs", Bell System Technical Journal, vol. 49, no. 2, 1970.

15. McCormick, William, et. al.: "Problem Decomposition and Data Reordering by a Clustering Technique", Operations Research, vol. 20, no. 5, September 1972.

16. Salton, G., and A. Wong: "Generation and Search of Clustered Files", ACM TODS, vol. 3, no. 4, December 1978.

17. Sangiovanni-Vincentelli, A., et. al.: "An Efficient Heuristic Clustering Algorithm for Tearing Large-Scale Networks", I.E.E.E. Trans. on Circuits and Systems, vol. 24, no. 12, December 1977.

18. Silver, A.: "A Computer Analysis Tool for Structural Decomposition Using Entropy Metrics", Martin Marietta Corp., Denver, CO., 1978.

19. Ward, J. H.: "Hierarchical Grouping to Optimize An Objective Function", American Statistical Association Journal, March 1963.

APPENDIX

This appendix contains a computer-produced trace of the execution path calculations performed by the master control procedure and the imbedded interchange algorithm in the course of decomposing the 10-node graph first introduced in Section 2.3. The actual partitions effected by the algorithm as detailed here are also shown in Figure 5.2.

GRAPH ADJACENCY MATRIX

	1	2	3	4	5	6	7	8	9	10
1	0.00									
2	0.50	0.00								
3	0.80	0.40	0.00							
4	0.00	0.60	0.50	0.00						
5	0.00	0.00	0.00	0.40	0.00					
6	0.00	0.00	0.00	0.00	0.80	0.00				
7	0.00	0.00	0.00	0.00	0.40	0.90	0.00			
8	0.00	0.00	0.00	0.70	0.00	0.00	0.00	0.00		
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00	
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.30	0.00

(COMMENTS)

****

Start with entire graph

BEGIN PARTITICNING.
CLUSTERS ARE: (1 2 3 4 5 6 7 8 9 10)
MEASURE = 0.051

****

SET STARTING PARTITION 1
INITIAL PARTITION IS:
#1: 1 2 3 4 5 6 7 8 9
#2: 10 11 12 13 14 15 16 17 18

Add 8 dummy nodes ($n_1 = 1$)

Starting partitions are chosen arbitrarily

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	9	10	-0.100	-0.100
2	8	11	-0.600	-0.700
3	4	12	-0.800	-1.500
4	2	13	-0.300	-1.800
5	3	14	0.100	-1.700
6	1	15	1.300	-0.400
7	5	16	-0.800	-1.200
8	6	17	-0.100	-1.300
9	7	18	1.300	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 9

First starting partition leads to no interchanges.

FINAL PARTITICN:
#1: 1 2 3 4 5 6 7 8 9
#2: 10 11 12 13 14 15 16 17 18

SET STARTING PARTITION 2
INITIAL PARTITICN IS:
#1: 1 3 5 7 9 11 13 15 17
#2: 2 4 6 8 10 12 14 16 18

BEGIN ITERATION 1

INTERPCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	9	6	2.600	2.600

2	11	2	0.300	2.900
3	13	4	0.800	3.700
4	15	12	0.000	3.700
5	17	14	0.000	3.700
6	1	16	-1.300	2.400
7	3	18	-0.100	2.300
8	7	8	-1.700	0.600
9	5	10	-0.600	0.000

PARTIALMAX= 3.700 PARTIALMAX INDEX= 3

FINAL PARTITION:
#1: 1 2 3 4 5 6 7 15 17
#2: 8 9 10 11 12 13 14 16 18

BEGIN ITERATION 2

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	15	11	0.000	0.000
2	17	12	0.000	0.000
3	4	13	-0.800	-0.800
4	2	14	-0.300	-1.100
5	3	16	0.100	-1.000
6	1	18	1.300	0.300
7	5	10	-1.600	-1.300
8	6	9	-0.400	-1.700
9	7	8	1.700	0.000

PARTIALMAX= 0.300 PARTIALMAX INDEX= 6

FINAL PARTITION:
#1: 5 6 7 11 12 13 14 16 18
#2: 1 2 3 4 8 9 10 15 17

BEGIN ITERATION 3

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	11	15	0.000	0.000
2	12	17	0.000	0.000
3	13	10	-0.800	-0.800
4	14	9	-0.300	-1.100
5	16	8	0.400	-0.700
6	18	4	0.000	-0.700
7	7	2	-1.600	-2.300
8	6	3	0.200	-2.100
9	5	1	2.100	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 1

FINAL PARTITION:
#1: 5 6 7 11 12 13 14 16 18
#2: 1 2 3 4 8 9 10 15 17

Second starting partition locates good final partition

SET STARTING PARTITION 3
INITIAL PARTITION IS:
#1: 5 6 7 8 9 10 11 12 13
#2: 1 2 3 4 14 15 16 17 18

```
BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1        11         4        0.000       0.000
       2        12        14        0.000       0.000
       3        13        15        0.000       0.000
       4        10        16       -0.800      -0.800
       5         9        17       -0.300      -1.100
       6         8        18        0.400      -0.700
       7         7         2       -1.600      -2.300
       8         6         3        0.200      -2.100
       9         5         1        2.100       0.000

PARTIALMAX=  0.000  PARTIALMAX INDEX=  1

FINAL PARTITION:
#1:   5   6   7   8   9  10  11  12  13
#2:   1   2   3   4  14  15  16  17  18

RESULTING PARTITIONS NOT EQUIVALENT;
BEST PARTITION IS:
#1:   5   6   7
#2:   1   2   3   4   8   9  10

END OF PASS   1
CLUSTERS ARE: (   5   6   7) (   1   2   3   4   8   9  10)
MEASURE =   0.292
**********************************************************

LOCATE SUBGRAPH WITH LOWEST STRENGTH:
SUBGRAPH NO.      STRENGTH
    1              0.2332
    2              0.0777
FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH:  2

SET STARTING PARTITION   1
INITIAL PARTITION IS:
#1:   1   2   3   4   5   6
#2:   7   8   9  10  11  12

BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         6         7       -0.100      -0.100
       2         5         8       -0.600      -0.700
       3         4         9       -0.400      -1.100
       4         2        10       -0.300      -1.400
       5         3        11        0.100      -1.300
       6         1        12        1.300       0.000

PARTIALMAX=  0.000  PARTIALMAX INDEX=  6

FINAL PARTITION:
#1:   1   2   3   4   5   6
#2:   7   8   9  10  11  12

SET STARTING PARTITION   2
```

Same result as for second starting partition.

"Best" is determined using objective function M

Decomposition after 1 pass.

Subgraph 2 is best candidate for next pass.

Note that internal number identifiers are assigned to the nodes of the subgraph being partitioned.

```
INITIAL PARTITION IS:
#1:   1   3   5   7   9  11
#2:   2   4   6   8  10  12

BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         5         2       1.100       1.100
       2         7         4       1.200       2.300
       3         9         8       0.000       2.300
       4        11        10       0.000       2.300
       5         1        12      -1.300       1.000
       6         3         6      -1.000       0.000

PARTIALMAX=  2.300  PARTIALMAX INDEX=  2

FINAL PARTITION:
#1:   1   2   3   4   9  11
#2:   5   6   7   8  10  12

BEGIN ITERATION  2

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         9         8       0.000       0.000
       2        11        10       0.000       0.000
       3         4        12      -0.400      -0.400
       4         2         7      -1.100      -1.500
       5         3         6      -0.200      -1.700
       6         1         5       1.700       0.000

PARTIALMAX=  0.000  PARTIALMAX INDEX=  1

FINAL PARTITION:
#1:   1   2   3   4   9  11
#2:   5   6   7   8  10  12

SET STARTING PARTITION   3
INITIAL PARTITION IS:
#1:   4   5   6   7   8   9
#2:   1   2   3  10  11  12

BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         4        10       0.400       0.400
       2         8        11       0.000       0.400
       3         9        12       0.000       0.400
       4         5         1      -1.700      -1.300
       5         6         3       0.200      -1.100
       6         7         2       1.100       0.000

PARTIALMAX=  0.400  PARTIALMAX INDEX=  1

FINAL PARTITION:
#1:   5   6   7   8   9  10
#2:   1   2   3   4  11  12
```

```
BEGIN ITERATION  2

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         8         11        0.000       0.000
       2         9         12        0.000       0.000
       3        10          4       -0.400      -0.400
       4         7          2       -1.100      -1.500
       5         6          3       -0.200      -1.700
       6         5          1        1.700       0.000

PARTIALMAX=   0.000  PARTIALMAX INDEX=  1

FINAL PARTITION:
#1:   5   6   7   8   9  10
#2:   1   2   3   4  11  12

RESULTING PARTITIONS NOT EQUIVALENT;
BEST PARTITION IS:
#1:   1   2   3   4
#2:   5   6   7

END OF PASS   2
CLUSTERS ARE: (   5   6   7) (   1   2   3   4) (   8   9  10)
MEASURE =   0.484
**********************************************************

LOCATE SUBGRAPH WITH LOWEST STRENGTH:
SUBGRAPH NO.      STRENGTH
    1              0.2332
    2              0.1866
    3              0.1555
FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH:  3

SET STARTING PARTITION   1
INITIAL PARTITION IS:
#1:   1   2
#2:   3   4

BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         1          4       -0.100      -0.100
       2         2          3        0.100       0.000

PARTIALMAX=  0.000  PARTIALMAX INDEX=  2

FINAL PARTITION:
#1:   1   2
#2:   3   4

SET STARTING PARTITION   2
INITIAL PARTITION IS:
#1:   1   3
#2:   2   4

BEGIN ITERATION  1
```

Result after second pass (turns out to be optimal result - Fig 5.2)

These are the "real" node numbers (not internal identifiers).

```
     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         1         4        0.100       0.100
       2         3         2       -0.100       0.000

PARTIALMAX=  0.100  PARTIALMAX INDEX=  1

FINAL PARTITION:
#1:   3   4
#2:   1   2

BEGIN ITERATION  2

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         3         2       -0.100      -0.100
       2         4         1        0.100       0.000

PARTIALMAX=  0.000  PARTIALMAX INDEX=  2

FINAL PARTITION:
#1:   3   4
#2:   1   2

SET STARTING PARTITION   3
INITIAL PARTITION IS:
#1:   2   3
#2:   1   4

BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         2         4        0.300       0.300
       2         3         1       -0.300       0.000

PARTIALMAX=  0.300  PARTIALMAX INDEX=  1

FINAL PARTITION:
#1:   3   4
#2:   1   2

BEGIN ITERATION  2

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         3         2       -0.100      -0.100
       2         4         1        0.100       0.000

PARTIALMAX=  0.000  PARTIALMAX INDEX=  2

FINAL PARTITION:
#1:   3   4
#2:   1   2

ALL PARTITIONS EQUIVALENT, SO
BEST PARTITION IS:
#1:   1   2
#2:   3
```

```
END OF PASS   3
CLUSTERS ARE: (   5   6   7) (   1   2   3   4) (   8   9) (  10)
MEASURE =  -0.101
**********************************************************

LOCATE SUBGRAPH WITH LOWEST STRENGTH:
SUBGRAPH NO.      STRENGTH
    1              0.2332
    2              0.1866
    3              0.0000
FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH:  3
```

No calculations - subgraph has only 2 nodes.

```
END OF PASS   4
CLUSTERS ARE: (   5   6   7) (   1   2   3   4) (   8) (  10) (   9)
MEASURE =  -1.189
**********************************************************

LOCATE SUBGRAPH WITH LOWEST STRENGTH:
SUBGRAPH NO.      STRENGTH
    1              0.2332
    2              0.1866
FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH:  2

SET STARTING PARTITION   1
INITIAL PARTITION IS:
#1:   1   2   3
#2:   4   5   6

BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE   NODE 1   NODE 2   MAX GAIN   PAR SUM
       1        1        4      -0.200     -0.200
       2        3        5      -0.100     -0.300
       3        2        6       0.300      0.000

PARTIALMAX=   0.000  PARTIALMAX INDEX=  3

FINAL PARTITION:
#1:   1   2   3
#2:   4   5   6

SET STARTING PARTITION   2
INITIAL PARTITION IS:
#1:   1   3   5
#2:   2   4   6

BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE   NODE 1   NODE 2   MAX GAIN   PAR SUM
       1        5        2       0.300      0.300
       2        1        4      -0.200      0.100
       3        3        6      -0.100      0.000

PARTIALMAX=   0.300  PARTIALMAX INDEX=  1

FINAL PARTITION:
#1:   1   2   3
```

Strengths are only calculated for subgraphs with at least 2 nodes.

```
#2:   4   5   6

BEGIN ITERATION  2

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         1         4       -0.200      -0.200
       2         3         5       -0.100      -0.300
       3         2         6        0.300       0.000

PARTIALMAX=   0.000  PARTIALMAX INDEX=  3

FINAL PARTITION:
#1:   1   2   3
#2:   4   5   6

SET STARTING PARTITION   3
INITIAL PARTITION IS:
#1:   2   3   4
#2:   1   5   6

BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         4         1        0.200       0.200
       2         2         5       -0.300      -0.100
       3         3         6        0.100       0.000

PARTIALMAX=   0.200  PARTIALMAX INDEX=  1

FINAL PARTITION:
#1:   1   2   3
#2:   4   5   6

BEGIN ITERATION  2

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1         1         4       -0.200      -0.200
       2         3         5       -0.100      -0.300
       3         2         6        0.300       0.000

PARTIALMAX=   0.000  PARTIALMAX INDEX=  3

FINAL PARTITION:
#1:   1   2   3
#2:   4   5   6

ALL PARTITIONS EQUIVALENT, SO
BEST PARTITION IS:
#1:   1   2   3
#2:   4


END OF PASS   5
CLUSTERS ARE: (   5   6   7) (   1   2   3) (   8) (  10) (   9) (   4)
MEASURE =  -2.178
**************************************************
```

```
LOCATE SUBGRAPH WITH LOWEST STRENGTH:
SUBGRAPH NO.      STRENGTH
    1              0.2332
    2              0.1888
FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH:  2

SET STARTING PARTITION    1
INITIAL PARTITION IS:
#1:    1   2
#2:    3   4

BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1        1         4         0.300       0.300
       2        2         3        -0.300       0.000

PARTIALMAX=  0.300  PARTIALMAX INDEX=  1

FINAL PARTITION:
#1:    2   4
#2:    1   3

BEGIN ITERATION  2

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1        2         3        -0.300      -0.300
       2        4         1         0.300       0.000

PARTIALMAX=  0.000  PARTIALMAX INDEX=  2

FINAL PARTITION:
#1:    2   4
#2:    1   3

SET STARTING PARTITION    2
INITIAL PARTITION IS:
#1:    1   3
#2:    2   4

BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1        1         4        -0.300      -0.300
       2        3         2         0.300       0.000

PARTIALMAX=  0.000  PARTIALMAX INDEX=  2

FINAL PARTITION:
#1:    1   3
#2:    2   4

SET STARTING PARTITION    3
INITIAL PARTITION IS:
#1:    2   3
#2:    1   4
```

```
BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1        2         1         0.400       0.400
       2        3         4        -0.400       0.000

PARTIALMAX=  0.400  PARTIALMAX INDEX=  1

FINAL PARTITION:
#1:   1   3
#2:   2   4

BEGIN ITERATION  2

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1        1         4        -0.300      -0.300
       2        3         2         0.300       0.000

PARTIALMAX=  0.000  PARTIALMAX INDEX=  2

FINAL PARTITION:
#1:   1   3
#2:   2   4

ALL PARTITIONS EQUIVALENT, SO
BEST PARTITION IS:
#1:   2
#2:   1   3


END OF PASS   6
CLUSTERS ARE: (   5   6   7) (   2) (   8) (  10) (   9) (   4) (   1   3)
MEASURE =  -3.300
**********  **********************************************

LOCATE SUBGRAPH WITH LOWEST STRENGTH:
SUBGRAPH NO.      STRENGTH
    1              0.2332
    7              0.0000
FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH:  7


END OF PASS   7
CLUSTERS ARE: (   5   6   7) (   2) (   8) (  10) (   9) (   4) (   1) (   3)
MEASURE =  -4.800
**********  **********************************************

LOCATE SUBGRAPH WITH LOWEST STRENGTH:
SUBGRAPH NO.      STRENGTH
    1              0.2332
FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH:  1

SET STARTING PARTITION   1
INITIAL PARTITION IS:
#1:   1   2
#2:   3   4

BEGIN ITERATION  1
```

```
     INTERCHANGE TRACE:
     CYCLE   NODE 1   NODE 2   MAX GAIN   PAR SUM
       1       1        3        0.100      0.100
       2       2        4       -0.100      0.000

PARTIALMAX=  0.100  PARTIALMAX INDEX=  1

FINAL PARTITION:
#1:   2   3
#2:   1   4

BEGIN ITERATION  2

     INTERCHANGE TRACE:
     CYCLE   NODE 1   NODE 2   MAX GAIN   PAR SUM
       1       2        4       -0.100     -0.100
       2       3        1        0.100      0.000

PARTIALMAX=  0.000  PARTIALMAX INDEX=  2

FINAL PARTITION:
#1:   2   3
#2:   1   4

SET STARTING PARTITION   2
INITIAL PARTITION IS:
#1:   1   3
#2:   2   4

BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE   NODE 1   NODE 2   MAX GAIN   PAR SUM
       1       1        2        0.500      0.500
       2       3        4       -0.500      0.000

PARTIALMAX=  0.500  PARTIALMAX INDEX=  1

FINAL PARTITION:
#1:   2   3
#2:   1   4

BEGIN ITERATION  2

     INTERCHANGE TRACE:
     CYCLE   NODE 1   NODE 2   MAX GAIN   PAR SUM
       1       2        4       -0.100     -0.100
       2       3        1        0.100      0.000

PARTIALMAX=  0.000  PARTIALMAX INDEX=  2

FINAL PARTITION:
#1:   2   3
#2:   1   4

SET STARTING PARTITION   3
INITIAL PARTITION IS:
#1:   2   3
#2:   1   4
```

```
BEGIN ITERATION  1

     INTERCHANGE TRACE:
     CYCLE    NODE 1    NODE 2    MAX GAIN    PAR SUM
       1        2         4        -0.100      -0.100
       2        3         1         0.100       0.000

PARTIALMAX=  0.000  PARTIALMAX INDEX=  2

FINAL PARTITION:
#1:   2   3
#2:   1   4

ALL PARTITIONS EQUIVALENT, SO
BEST PARTITION IS:
#1:   2   3
#2:   1


END OF PASS   8
CLUSTERS ARE: (   6   7) (   2) (   8) ( 10) (   9) (   4) (   1) (   3) (5)
MEASURE =  -5.900
**************************************************************

LOCATE SUBGRAPH WITH LOWEST STRENGTH:
SUBGRAPH NO.      STRENGTH
    1              0.0000
FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH:  1


END OF PASS   9
CLUSTERS ARE: (   6) (   2) (   8) ( 10) (   9) (   4) (   1) (   3) ( 5) (7)
MEASURE =  -7.400
**************************************************************


OVERALL BEST PARTITION:

OPTIMUM MEASURE M* =     0.4837
OPTIMAL DECOMPOSITION IS:

  (   5   6   7 )
  (   1   2   3   4 )
  (   8   9  10 )
```

Overall optimal partition remembered and printed out.