An Efficient Heuristic Procedure for Partitioning Graphs

By B. W. KERNIGHAN and S. LIN

We consider the problem of partitioning the nodes of a graph with costs on its edges into subsets of given sizes so as to minimize the sum of the costs on all edges cut. This problem arises in several physical situations—for example, in assigning the components of electronic circuits to circuit boards to minimize the number of connections between boards.

This paper presents a heuristic method for partitioning arbitrary graphs which is both effective in finding optimal partitions, and fast enough to be practical in solving large problems.

I. INTRODUCTION

1.1 *Definition of the Problem*

This paper deals with the following combinatorial problem: given a graph G with costs on its edges, partition the nodes of G into subsets no larger than a *given maximum size*, so as to minimize the total cost of the edges cut.

One important practical example of this problem is placing the components of an electronic circuit onto printed circuit cards or substrates, so as to minimize the number of connections between cards. The components are the nodes of the graph, and the circuit connections are the edges. There is some maximum number of components which may be placed on any card. Since connections between cards have high cost compared to connections within a board, the object is to minimize the number of interconnections between cards.

This partitioning problem also arises naturally in an attempt to improve the paging properties of programs for use in computers with paged memory organization. A program (at least statically) can be thought of as a set of connected entities. The entities might be subroutines, or procedure blocks, or single instruction and data items, depending on viewpoint and the level of detail required. The connections between the entities might represent possible flow or transfer of control, or references from one entity to another. The problem is to assign the objects to "pages" of a given size so as to minimize the number of references between objects which lie on different pages.

To pose the partitioning problem mathematically, we shall need the following definitions. Let G be a graph of n nodes, of sizes (weights) $w_i > 0, i = 1, \cdots, n$. Let p be a positive number, such that $0 < w_i \leqq p$ for all i. Let $C = (c_{ij}), i, j = 1, \cdots, n$ be a weighted connectivity matrix describing the edges of G.

Let k be a positive integer. A *k-way partition* of G is a set of nonempty, pairwise disjoint subsets of G, $v_1, \cdots, v_k$ such that $\bigcup_{i=1}^{k} v_i = G$. A partition is *admissible* if

$$|v_i| \leqq p \quad \text{for all} \quad i,$$

where the symbol $|x|$ stands for the *size* of a set x, and equals the sum of the sizes of all the elements of x. The *cost* of a partition is the summation of c_{ij} over all i and j such that i and j are in different subsets. The cost is thus the sum of all external costs in the partition.

The partitioning problem we consider here is to find a minimal-cost admissible partition of G.

There are three other problems which are equivalent to this one. First, minimizing external cost is equivalent to maximizing internal cost because the total cost of all edges is constant. Further, by changing the signs of all c_{ij}'s, we can maximize external cost, or minimize internal cost.

1.2 *Exact Solutions*

A strictly exhaustive procedure for finding the minimal cost partition is often out of the question. To see this suppose that G has n nodes of size 1 to be partitioned into k subsets of size p, where $kp = n$. Then there are $\binom{n}{p}$ ways of choosing the first subset, $\binom{n-p}{p}$ ways for the second, and so on. Since the ordering of the subsets is immaterial, the number of cases is

$$\frac{1}{k!}\binom{n}{p}\binom{n-p}{p}\cdots\binom{2p}{p}\binom{p}{p}.$$

For most values of n, k, and p, this expression yields a very large number; for example, for $n = 40$ and $p = 10$ ($k = 4$), it is greater than 10^{20}.

Formally the problem could also be solved as an integer linear programming problem, with a large number of constraint equations necessary to express the uniformity of the partition.

Because it seems likely that any direct approach to finding an optimal solution will require an inordinate amount of computation, we turn to an examination of heuristics. Heuristic methods can produce good solutions (possibly even an optimal solution) quickly. Often in practical applications, several good solutions are of more value than one optimal one.

The first and foremost consideration in developing heuristics for combinatorial problems of this type is finding a procedure that is powerful and yet sufficiently fast to be practical. A process whose running time grows exponentially or factorially with the number of vertices of the graph is not likely to be practical. In most cases, a growth rate of more than the square of the number of vertices is still not too practical. (If the running time of a procedure grows as $f(n)$, where n is the number of vertices involved, we shall refer to it as an $f(n)$-procedure.)

1.3 *False Starts*

To point out a few pitfalls, we mention some unsuccessful attempts at heuristic solutions to the partitioning problem.

1.3.1 *Random Solutions*

One tactic is simply to generate random solutions, keeping the best seen to date, and terminating after some predetermined time or value is reached. This is quite fast, although actually an n^2-procedure. Unfortunately, this approach is unsatisfactory for problems of even moderate size, since there are generally few optimal or near-optimal solutions, which thus appear randomly with very low probabilities. Experience with 2-way partitions for a class of 0–1 matrices of size 32×32, for example, has indicated that there are typically 3 to 5 optimal partitions, out of a total of $\frac{1}{2}\binom{32}{16}$ partitions, giving a probability of success on any trial of less than 10^{-7}.

1.3.2 *Max Flow-Min Cut*

Another partitioning method is the Ford and Fulkerson max flow-min cut algorithm[1]. The graph is treated as a network in which edge costs correspond to maximum flow capacities between pairs of nodes. A cut is a separation of the nodes into two disjoint subsets. The max flow-min cut theorem states that the maximal flow values between any pair of nodes is equal to the minimal cut capacity of all cuts which separate the two nodes. In our terminology, a cut is a 2-way partition, and the cut capacity is the cost of the partition. The Ford and Fulkerson algorithm finds a cut with maximal flow, which is thus a minimal cost cut; this represents a minimum cost partition of the graph into two subsets of unspecified sizes.

There are several difficulties involved in using the Ford and Fulkerson algorithm for our partitioning problem. The most severe of these is the fact that the algorithm has no provision for constraining the sizes of the resultant subsets, and there seems to be no obvious way to extend it to include this. Thus if flow methods are used to perform a split, then further processing is necessary to make the resulting subsets the correct size. If the subsets are greatly different in size, then use of this algorithm will have produced essentially no benefit. Hence in spite of its theoretical elegance, the Ford and Fulkerson algorithm is not suitable for this application. (Note however, that since it does find the minimal cost unconstrained 2-way partition, the value it produces is a lower bound for solutions produced by any method.)

Reprinted with permission from *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, Feb. 1970.

1.3.3 *Clustering*

A class of much more intuitive methods is based on identifying "natural clusters" in the given cost matrix—that is, groups of nodes which are strongly connected in some sense. For example, one can use very simple heuristics for building up clusters, based on collecting together elements corresponding to large values in the cost matrix. But again these methods do not in general include much provision for satisfying constraints on the sizes of the subsets, nor do they provide for systematic assignment of "stragglers" (nodes which do not obviously belong to any particular subset).

1.3.4 *λ-Opting*

Lin, working on the Traveling Salesman Problem, [See Ref. 2] categorized a set of methods of improving given solutions by rearranging single links, double links, triplets, and in general, λ links. He referred to a change involving the movement of λ links as a λ-change. If a configuration of the system is reached in which no λ-change can be made which results in a decrease in cost, the configuration is said to be "λ-opt."

For the partitioning problem, an analogous operation is the interchange of groups of λ points between a pair of sets. Thus a 1-change is the exchange of a single point in one set with a single point in another set. A configuration is then said to be "1-opt" if there exists no interchange of two points which decreases the cost of the partition. Experiments to evaluate 1-opting for 2-way partitions of 0-1 matrices (32 × 32) within which about one-half of the elements were nonzero, show that apparently optimal values can be achieved in about 10 percent of the trials; values within 1 or 2 of the optimal can be achieved in about 75 percent of cases.

It appears fruitless to extend λ beyond 1 (1-opting is already an n^2-procedure), or to extend 1-opting experiments to partitions into more than two subsets, since more powerful methods have been developed. These methods are the topic of the next sections.

II. TWO-WAY UNIFORM PARTITIONS

2.1 *Introduction*

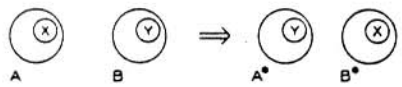
The simplest partitioning problem which still contains all the significant features of larger problems is that of finding a minimal-cost partition of a given graph of $2n$ vertices (of equal size) into two subsets of n vertices each. The solution of the 2-way partitioning problem is the subject of this section. The solution provides the basis for solving more general partitioning problems. In Section 2.6, we discuss 2-way partitions into sets of unequal size.

Let S be a set of $2n$ points, with an associated cost matrix $C = (c_{ij})$, $i, j = 1, \cdots, 2n$. We assume without loss of generality that C is a symmetric matrix, and that $c_{ii} = 0$ for all i. There is no assumption about nonnegativity of the c_{ij}'s. We wish to partition S into two sets A and B, each with n points, such that the "external cost" $T = \sum_{a \in A, b \in B} c_{ab}$ is minimized.

In essence, the method is this: starting with any arbitrary partition A, B of S, try to decrease the initial external cost T by a series of interchanges of subsets of A and B; the subsets are chosen by an algorithm to be described. When no further improvement is possible, the resulting partition A', B' is locally minimum with respect to the algorithm. We shall indicate that the resulting partition has a fairly high probability of being a globally minimum partition.

This process can then be repeated with the generation of another arbitrary starting partition A, B, and so on, to obtain as many locally minimum partitions as we desire.

Given S and (c_{ij}), suppose A^*, B^* is a minimum cost 2-way partition. Let A, B be any arbitrary 2-way partition. Then clearly there are subsets $X \subset A$, $Y \subset B$ with $|X| = |Y| \leq n/2$ such that interchanging X and Y produces A^* and B^* as shown below.



$A^* = A - X + Y$
$B^* = B - Y + X$

The problem is to identify X and Y from A and B, without considering all possible choices. The process we describe finds X and Y approximately, by sequentially identifying their elements.

Let us define for each $a \in A$, an *external cost* E_a by

$$E_a = \sum_{x \in B} c_{ax}$$

and an *internal cost* I_a by

$$I_a = \sum_{x \in A} c_{ax} .$$

Similarly, define E_b, I_b for each $b \in B$. Let $D_a = E_a - I_a$ for all $x \in S$; D_x is the difference between external and internal costs.

Lemma 1: Consider any $a \in A$, $b \in B$. If a and b are interchanged, the gain (that is, the reduction in cost) is precisely $D_a + D_b - 2c_{ab}$.

Proof: Let z be the total cost due to all connections between A and B that do not involve a or b. Then

$$T = z + E_a + E_b - c_{ab} .$$

Exchange a and b; let T' be the new cost. We obtain

$$T' = z + I_a + I_b + c_{ab}$$

and so

$$\text{gain} = \text{old cost} - \text{new cost} = T - T'$$
$$= D_a + D_b - 2c_{ab} .$$

2.2 *Phase 1 Optimization Algorithm*

In this subsection we present the algorithm for 2-way partitioning.

First, compute the D values for all elements of S. Second, choose $a_i \in A$, $b_i \in B$ such that

$$g_i = D_{a_i} + D_{b_i} - 2c_{a_i b_i},$$

is maximum; a_i and b_i correspond to the largest possible gain from a single interchange. (We will return shortly to a discussion of how to select a_i and b_i quickly.) Set a_i and b_i aside temporarily, and call them a_1' and b_1', respectively.

Third, recalculate the D values for the elements of $A - \{a_i\}$ and for $B - \{b_i\}$, by

$$D_x' = D_x + 2c_{xa_i} - 2c_{xb_i}, \qquad x \in A - \{a_i\},$$
$$D_y' = D_y + 2c_{yb_i} - 2c_{ya_i}, \qquad y \in B - \{b_i\}.$$

The correctness of these expressions is easily verified: the edge (x, a_i) is counted as internal in D_x, and it is to be external in D_x', so c_{xa_i} must be added twice to make this correct. Similarly, c_{xb_i} must be subtracted twice to convert (x, b_i) from external to internal.

Now repeat the second step, choosing a pair a_2', b_2' from $\underline{A - \{a_1'\}}$ and $\underline{B - \{b_1'\}}$ such that $g_2 = D_{a_2'} + D_{b_2'} - 2c_{a_2' b_2'}$ is maximum (a_1' and b_1' are *not* considered in this choice). Thus g_2 is the additional gain when the points a_2' and b_2' are exchanged as well as a_1' and b_1'; this additional gain is maximum, given the previous choices. Set a_2' and b_2' aside also.

Continue until all nodes have been exhausted, identifying $(a_3', b_3'), \cdots, (a_n', b_n')$, and the corresponding maximum gains $g_3, \cdots, g_n$. As each (a', b') pair is identified, it is removed from contention for further choices so the size of the sets being considered decreases by 1 each time an (a', b') is selected.

If $X = a_1', a_2', \cdots, a_k'$, $Y = b_1', b_2', \cdots, b_k'$, then the decrease in cost when the sets X and Y are interchanged is precisely $g_1 + g_2 + \cdots + g_k$. Of course $\sum_1^n g_i = 0$. Note that some of the g_i's are negative, unless all are zero.

Choose k to maximize the partial sum $\sum_{i=1}^{k} g_i = G$. Now if $G > 0$, a reduction in cost of value G can be made by interchanging X and Y. After this is done, the resulting partition is treated as the initial partition, and the procedure is repeated from the first step.

If $G = 0$, we have arrived at a locally optimum partition, which we shall call a *phase 1 optimal partition*. We now have the choice of repeating with another starting partition, or of trying to improve the phase 1 optimal partition. We shall discuss the latter option shortly. Figure 1 is a flowchart for the phase 1 optimization procedure.

2.3 *Effectiveness of the Procedure*

One general approach to solving problems such as this one is to find the *best* exchange involving say λ pairs of points, for some λ specified in advance[3]. The difficulty encountered is that use of a small value of λ is not sufficient to identify good exchanges, but the computational effort required grows rapidly as λ increases.

The procedure we have described *sequentially* finds an approximation to the best exchange of λ pairs. λ is not specified in advance, but rather is chosen to make the improvement as large as possible. This technique sacrifices a certain amount of power for a considerable gain in speed.

Since we construct a sequence of gains g_i, $i = 1, \cdots, n$, and find the

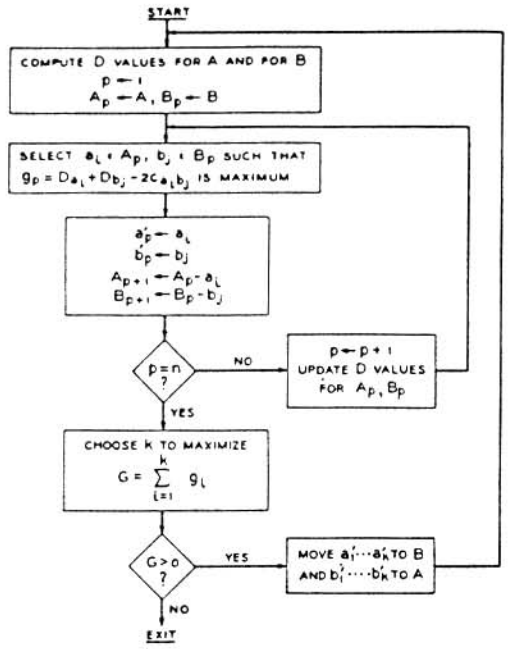


Fig. 1 — Flowchart of phase 1 optimization procedure.

maximum partial sum, *the process does not terminate immediately when some g_i is negative*. This means that the process can sequentially identify sets for which the exchange of only a few elements would actually increase the cost, while the interchange of the entire sets produces a net gain.

Numerous experiments have been performed to evaluate the procedure on different types of cost matrices. The matrices used have included (*i*) 0-1 matrices, with density of nonzero elements ranging from 5 percent to 50 percent, (*ii*) integer matrices with elements uniformly distributed on $[0, k]$, $k = 2, \cdots, 10$, (*iii*) matrices with clusters of known sizes and binding strength. Results on all of these matrices have been similar, so we shall only summarize them here. A more extended discussion may be found in Ref. 3.

A useful measure of the power of a heuristic procedure is the probability that it finds an optimal solution in a single trial. Suppose that p is the probability that a phase 1 optimal solution found using a random starting partition is globally optimal. We have examined the behavior of this probability as the size of the matrices involved is varied. Experiments show p is around 0.5 for matrices of size 30 × 30, 0.2 to 0.3 for 60 × 60, and 0.05 to 0.1 for 120 × 120. The functional behavior of p is approximately $p(n) = 2^{-n/30}$.

These values are derived primarily from 0-1 matrices having about 50 percent 1's (randomly placed). Experiments on matrices with lower densities of 1's yield larger variances, but substantially identical mean values for p.

2.4 *Running Time of the Procedure*

Let us define a *pass* to be the operations involved in making one cycle of identification of $(a'_1, b'_1), \cdots, (a'_n, b'_n)$, and selection of sets X and Y to be exchanged. The total time for a pass can be estimated this way. First, the computation of the D values initially is an n^2-procedure, since for each element of S, all the other elements of S must be considered. The time required for updating the D values is proportional to the number of values to be updated, so the total updating time in one pass grows as

$$(n - 1) + (n - 2) + \cdots + 2 + 1$$

which is proportional to n^2.

The dominant time factor is the selection of the next pair a'_i, b'_i to be exchanged. The method we have used to perform this searching is to sort the D values so that

$$D_{a_1} \geq D_{a_2} \geq \cdots \geq D_{a_n}$$

and

$$D_{b_1} \geq D_{b_2} \geq \cdots \geq D_{b_n}.$$

When sorting is used, only a few likely contenders for a maximum gain need be considered. This is because when scanning down the set of D_a's and D_b's, if a pair D_{a_i}, D_{b_j} is found whose sum does not exceed the maximum improvement seen so far in this pass, then there cannot be another pair a_k, b_l with $k \geq i$, $l \geq j$, with a greater gain, (assuming $c_{kl} \geq 0$) and so the scanning can be terminated. Thus the next pair for interchange is found rapidly. Sorting is an $n \log n$ operation, so in this method, the total time required to sort D values in a pass will be approximately

$$n \log n + (n - 1) \log (n - 1) + \cdots + 2 \log 2$$

which grows as $n^2 \log n$.

To reduce the time for selection of an (a, b) pair, it is possible to use techniques which are faster than sorting, but which do not necessarily always give the maximum gain at each stage. For example, one method is to scan for the largest D_a and the largest D_b, and use the corresponding a and b as the next interchange. This method is essentially linear-time and would probably be implemented as part of the recomputation of the D values. It is best suited for sparse matrices, where the probability that $c_{ab} > 0$ is small. A slight extension, involving negligible extra cost, is to save the largest two or three D_a's and D_b's, so that if the largest pair does not give the maximum gain (because c_{ab} is too large), then another can be tried. Experience indicates that three values are sufficient in virtually all cases, even for matrices with a relatively high percentage of nonzero entries. Use of this method reduces running time by about 30 percent in the present implementation, with very small degradation of power.

The number of passes required before a phase 1 optimal partition is achieved is small. On all matrix sizes tested at the time of writing (up to 360 points), it has been almost always from 2 to 4 passes. On the basis of this experimental evidence, the number of passes is not strongly dependent on the value of n.

From the foregoing observations, it is possible to estimate the total running time of the procedure. If we use a method which sorts the D values at each stage (time proportional to $n^2 \log n$), then the running time should grow as $n^2 \log n$. If a fast-scan method is used, and the number of passes is constant, the running time should have an n^2 growth rate; this is a lower bound.

For comparison, examination of all pairs of sets X and Y, and evaluation of the costs would require time proportional to

$$\begin{aligned} n^2 \sum_{k=1}^{n/2} \binom{n}{k}^2 &\sim \frac{n^2}{2} \sum_{k=0}^{n} \binom{n}{k}^2 \\ &= \frac{n^2}{2} \binom{2n}{n} \\ &\sim \frac{n^2}{2} 4^n \left(\frac{1}{\pi n}\right)^{\frac{1}{2}} \end{aligned}$$

for large n. This function grows as $n^{3/2}4^n$.

Running times have been plotted in Fig. 2. The observed times have an apparent growth rate of about $n^{2.4}$, which is reasonably close to n^2. Although on the logarithmic plot this curve is close to linear over the range $n = 20$ to $n = 130$, it may actually be $n^2 \log n$; insufficient data is available to check this. All times are based on an implementation in FORTRAN G on an IBM System 360 Model 65.

2.5 *Improving the Phase 1 Optimal Partition*

In this section, we discuss a method which might be used to improve the partition produced by the phase 1 procedure, which may not be globally optimum. The method suggested in this section is based heavily on experimental evidence, although there are quite plausible reasons for performing the particular set of operations. The basic idea is to perturb the locally optimal solution in what we hope is an enlightened manner, so that an iteration of the process on the perturbed solution will yield a further reduction in the total cost. If this tactic fails, nothing has been lost except some computation time, since the best solution seen so far is always saved.

Computer results for problems with up to 64 points suggest that whenever a phase 1 optimal solution is not globally optimal, $|X| = |Y| \approx n/2$. Roughly, this implies that if $|X|$ and $|Y|$ had been small compared to $n/2$, they would have been found by the process; it is only larger sets which are not identified all the time.

A successful heuristic to find the correct X and Y in this case is to find a phase 1 optimal partition for each of the sets A and B, say $A \rightarrow [A_1, A_2]$ and $B \rightarrow [B_1, B_2]$. (That is, find near-optimal partitions of A and of B separately.) Recombine the 4 sets into 2, say $A_0 = A_1 \cup B_1$ and $B_0 = A_2 \cup B_2$, and continue with phase 1 optimization. If our expectation is correct, the new X and Y will be small, and thus readily identified by the phase 1 process.

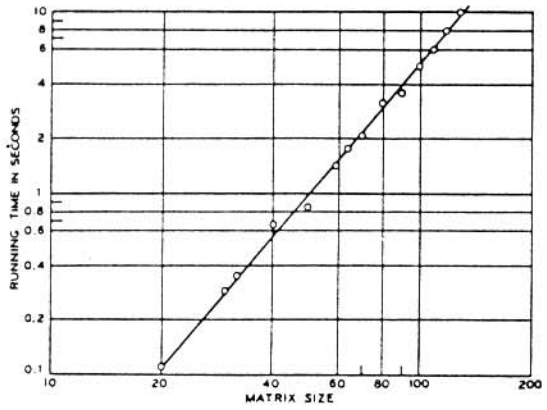


Fig. 2 — Running time.

When A is split into A_1, A_2 and B into B_1, B_2 there are two ways in which the smaller sets can be recombined. A series of tests was made on matrices of moderate size (up to 64 × 64), in which both possible recombinations were done, generating three phase 1 optimal values for each starting partition. For matrices of size 32 × 32, the apparent optimal value was observed at least once in each triple of values, for a large number of cases. With matrices of size 64 × 64, there were occasional failures.

It might be noted that the extra time involved for the recombination approach is three times that required to do a completely new partition from a random start, assuming an n^2-procedure.

It is possible to estimate whether a particular improvement tactic is profitable or not in the following way. Suppose that some method increases the probability of finding an optimal partition from p to p', while it increases the running time from t to t'. Then in a fixed amount of time, it is possible to do k trials of the basic procedure, and kt/t' trials of the improved method. The corresponding probabilities of achieving an optimal solution are $1 - (1 - p)^k$ and $1 - (1 - p')^{kt/t'}$ respectively. The improved method is then desirable if the second expression is greater than the first; by simple manipulation, this condition becomes

$$1 - p' < (1 - p)^{t'/t}.$$

On the basis of the numerical values in this section, it may be useful to try the recombination method.

2.6 *Partitioning into Unequal-Sized Sets*

It is simple to modify the procedure to partition a set S with n elements into two sets of specified sizes n_1 and $n_2 (n_1 + n_2 = n)$. Assume $n_1 < n_2$. Then restrict the maximum number of pairs that can be exchanged in one pass of the procedure to n_1. All other operations are performed on all elements of each set. (The starting partition is into two sets, of n_1 and n_2 elements respectively.)

Suppose we wish to partition S into two sets, such that there are at least n_1 elements and at most n_2 elements in each subset; $n_1 + n_2 = n$, but they are not specified further.

The procedure is easily modified to handle this sort of constraint by the addition of "dummy" elements. These are elements which have no connections whatsoever; that is, they have zero entries in the cost matrix wherever they appear. Add $2n_2 - n$ dummies so S has $2n_2$ elements, and perform the procedure on it. The resulting partition will assign the dummy elements to the two subsets so as to minimize the external cost; at this point the dummies are discarded, leaving a partition into two subsets that satisfy the size constraints given.

2.7 *Elements of Unequal Sizes*

We have made the assumption so far that the elements (vertices) of the graph are all of the same size. This requirement may be relaxed to a large extent by converting any node of size $k > 1$ to a cluster of k nodes of size 1, bound together by edges of appropriately high cost. The size of the problem will obviously increase proportionally to the value of k, so it may be necessary to sacrifice some accuracy to keep the number of generated nodes within reasonable bounds.

III. MULTIPLE-WAY PARTITIONS

3.1 *Reduction to 2-Way Partitioning Problem*

So far, the discussion has been concerned exclusively with the basic problem of performing a 2-way partition on a set of $2n$ objects. In this section we extend the technique to perform k-way partitions on a set of kn objects, using the 2-way procedure as a tool.

The essential idea is to start with some partition into k sets of size n and by repeated application of the 2-way partitioning procedure to pairs of subsets, make the partition as close as possible to being pairwise optimal. (Section 3.2 treats the question of what starting sets to use.) Of course pairwise optimality is only a necessary condition for global optimality. There may be situations where some complex interchange of three or more items from three or more subsets is required to reduce a pairwise optimal solution to globally optimum; at the moment, no reasonable method for identifying such sets is known.

There are $\binom{k}{2}$ pairs of subsets to consider, so the time for one pass through all pairs is (assuming an n^2-procedure) $\binom{k}{2}n^2 \approx (kn)^2/2 =$ (number of points)$^2/2$. In general, more passes than this will actually be required, since when two sets are made optimal, this may change their optimality with respect to other sets.

Experience indicates that the number of passes is small and the process converges quickly. For example, our algorithm selects (i, j) as the next pair of sets to be optimized, where either i or j has been changed since the last time the pair (i, j) was selected. Using this selection process, the average number of passes through each pair of sets is a slowly growing function of both k and n. For matrices of size 100 or less and $k < 6$, the number of passes has been less than 5. [The average number of passes is computed as the average number of pairs considered to reach pairwise optimality, normalized by $\binom{k}{2}$.]

In any particular trial, there is a correlation between the number of pairs selected and the quality of the final partition. To get a better solution requires more work.

Convergence is rapid: two passes account for more than 95 percent of the improvement in most cases; the remaining passes contribute only small further reductions. Let $p(n, k)$ be the proportion of minimum cost solutions found for a particular n and k. For k fixed and small compared to n, the functional behavior of $p(n, k)$ is similar to the case $k = 2$, but the actual values are lower. Roughly, we observe $p(n, k + 1) \approx \frac{1}{2}p(n, k)$ for k in the range 2–4, and n up to 100, with considerable variation depending on the matrix being tested. For instance, for matrices of size about 40, $p(40, 2) \approx 0.4$, $p(42, 3) \approx 0.2$, and $p(40, 4) \approx 0.1$.

Another interesting question is measurement of how close to optimum the partitions found are. The solutions obtained by pairwise optimization have values concentrated in a narrow range. In almost all cases, the largest value found by the procedure is within 4–5 percent of the smallest. As another measure, if c is the mean cost of random partitions and b is the cost of the best partition observed, then virtually all partitions found have values v such that

$$v - b \leqq 0.1(c - b).$$

For instance, one test case was a series of 4-way partitions of a 0–1 matrix of size 80. This matrix had 1278 nonzero entries (a density of 0.2), corresponding to 639 edges in the graph. The mean value of randomly chosen partitions was 480.6. Twenty-four partitions of this matrix were found using the method described above. The lowest value encountered was 352 (1 time), the highest 365 (1 time); the mean value was 359.5, the median 360.

3.2 *Starting Partition*

In this subsection we discuss various methods of generating good starting partitions, based on modifications of the basic procedure.

The primary reason for choosing good starting partitions is that this particular form of preprocessing reduces the amount of work required to make the system pairwise optimal. It may also make the probability of an optimal solution higher, although this tendency is very difficult to evaluate.

Several methods for finding good multi-way starting partitions which are based on repeated application of the procedure itself have been investigated. The essential idea is to generate a k-way starting partition by first forming an r-way partition, then an s-way partition on each of the resulting subsets, and so on, up to t-way. (Here $k = rs \cdots t$.) The partitions found this way will in general be better than those which are completely arbitrary. A pairwise optimization stage is applied to the final set of subsets.

For example, if k is a power of 2, then perform a 2-way split, then a

2-way split on each of these subsets, and so on until the desired size of subsets is found.

This general approach is prone to the following difficulty: the first split divides the original set into r subsets by trying to make the internal connections in each subset as large as possible. Obviously this may conflict directly with the next stage, which is to try to divide each subset further. Carried to several levels, it can lead to a relatively poor overall solution. In experiments with 4-way partitions of matrices of sizes up to 64 × 64, this method yields optimal solutions approximately as often as does starting with a 4-way partition in the first place. In addition, this method will be effective if the matrix happens to have natural clusters of approximately the correct size (that is, equal to the final subset size).

A second method which can be used is to partition the set of kn elements into a set of n and a set of $(k-1)n$, using the slightly modified version of the basic procedure discussed in the first part of Section 2.6. The set of n elements is set aside, and the next n elements from the remaining $(k-1)n$ are identified. This continues until k subsets have been formed; again the pairwise optimization technique is used to improve on this partition.

This method can make an error in the identification of the first set which will bias the choice of the second, and so on; the effect is most severe for the case where k is large, so each set is small.

The method of breaking off subsets sequentially has another potential flaw: regardless of the starting configuration, it will identify approximately the same set each time it is used on a particular problem, and hence little is gained by using it twice on one cost matrix. However variations in the order of performing pairwise optimizations can still produce different final partitions in general.

Limited computational experience with sequential break-off followed by pairwise optimization suggests that it yields solutions which are on the average at least as good as (and sometimes slightly better than) those provided by pairwise optimization applied to an arbitrary k-way starting partition. Pairwise optimization yields the optimum with a higher probability, however, because it is less susceptible to error caused by a bad choice made early. For instance, in tests on the 80 point matrix mentioned previously, sequential break off yielded 4-way solutions with a mean value of 358.6, but the lowest value found was 355. (The highest was 363.) These may be compared to 359.5, 352 and 365 for the standard partitioning method.

Running time for the sequential break-off method is lower than for straight pairwise optimization.

Insufficient data is available for a direct comparison between sequential break off and the method of repeated subdivision.

In all cases, the original process, be it a completely random generation of some initial configuration, or the production of a good starting partition, is followed by a pairwise optimizing phase. It is unlikely that using better starting partitions will lead to worse results than random starts, on the average. Whether the possible improvement in results and running times will justify the extra computational effort required to generate the starting partition depends on the characteristics of the particular class of matrices being studied.

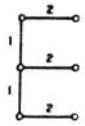


Fig. 3 — Cost reduction by expansion.

Some limited experiments were performed to compare the present procedure with a multi-dimensional scaling technique[4], on a Boolean matrix of 316 points, with about 1400 nonzero entries. The results indicated that the procedure identifies clusters well, even when no attempt is made to provide a good starting partition.

3.3 *Expansion Factor*

The introduction of dummy elements was mentioned in Section 2.6 as a method of handling partitioning into subsets of unequal sizes. This can be viewed equally well as a means of introducing "slack" into a solution, in an attempt to get a lower overall cost by allowing "expansion." That is, so far we have treated the problem of finding a partition with a constraint on the sizes of the subsets, *and* on the number of subsets, since given kn points, we have tried to find the best partitions into exactly k subsets of n points each. Suppose we now relax this second constraint by permitting the addition of dummy elements to increase the size of the problem, and attempt to find the best solution involving *any number* (greater than or equal to k) of subsets, with *at most* n points in each. This solution with k or greater subsets will in general have a lower cost than the constrained solution.

Figure 3 shows an example in which introducing slack permits a lower overall cost. Assume n is 3 and all nodes are size 1. The vertical edges have cost 1 and the horizontal ones cost 2. Any partition into 2 equal subsets has a cost of at least 3, but there is an obvious partition into 3 subsets with cost 2. Any nontrivial partition into 4 or more subsets has a cost greater than 2, so 3 subsets represents the optimal expansion. It is possible to find the minimal cost solution and the corresponding optimal amount of expansion as follows. Suppose the problem has kn points to be partitioned into k sets of n points each. Starting with no slack (kn points), the optimal assignment is found. Then n dummies, enough to create one extra subset, are added, making a $(k+1)n$ problem, and so on. Eventually, one subset is produced which consists entirely of dummies. When this occurs, we take the partition with this set of dummies removed as our optimum solution.

REFERENCES

1. Ford, L. R., and Fulkerson, D. R., *Flows in Networks*, Princeton, New Jersey: Princeton University Press, 1962, p. 11.
2. Lin, S., "Computer Solutions of the Traveling Salesman Problem," B.S.T.J., *44*, No. 10 (December 1965), pp. 2245–2269.
3. Kernighan, B. W., "Some Graph Partitioning Problems Related to Program Segmentation," Ph.D. Thesis, Princeton University, January 1969, pp. 74–126.
4. Kruskal, J. B., Multi-Dimensional Scaling by Optimizing Goodness of Fit to a Non-Metric Hypothesis," *Psychometrika*, *29*, No. 1 (March 1964), pp. 1–27, and No. 2 (June 1964), pp. 115–129.