ON PARTITIONING A GRAPH: A THEORETICAL AND EMPIRICAL STUDY

by

R. M. MacGregor

Memorandum No. UCB/ERL M78/14

22 March 1978

ON PARTITIONING A GRAPH: A THEORETICAL AND EMPIRICAL STUDY

by

R. M. MacGregor

Memorandum No. UCB/ERL M78/14

22 March 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ON PARTITIONING A GRAPH: A THEORETICAL AND EMPIRICAL STUDY*

Robert M. MacGregor

Division of Computer Science
University of California, Berkeley

Ph.D. Dissertation

June 1978

Abstract

This thesis studies a combinatorial problem which we call the k-Partition Problem: "Given an undirected graph G and an integer k, divide the nodes of G into k equal-sized sets in a way which minimizes the number of arcs which connect nodes in different sets". This problem finds application in the field of Design Automation and in problems of allocating program or data segments to pages of secondary storage. We have obtained a variety of results.

Our problem is shown to be NP-complete even when we require that the maximum node degree of G is at most three. A variant of the problem, which we call the General Partition Problem, is also proved to be NP-complete.

Iterative-improvement algorithms are often employed to find k-partitions. A probabilistic analysis proves that on a very simple set of graphs the simplest of the iterative-improvement algorithms almost surely produces a 2-partition which is far from optimal. Empirical tests showed that more complex iterative-improvement strategies also fail to find good solutions on these graphs.

*Research supported by National Science Foundation grants MCS74-17680 and MCS77-09906.

Suppose we randomly select a graph G from the space of all graphs with n nodes and N arcs. A probabilistic analysis produced two functions $L(n,N)$ and $U(n,N)$ such that if m^* is the number of cross arcs (arcs which cross from one set to the other) in an optimal 2-partition of G, then with probability going to one as $n \to \infty$, $L(n,N) \leq m^* \leq U(n,N)$.

We have derived some new heuristics which augment the standard iterative-improvement techniques. Also, we propose new approaches for extending 2-partition algorithms to find k-partitions.

Finally, we investigate properties of the 2-Partition Problem for special types of graphs. For instance, when we restrict the maximum node degree of a graph G we can improve the upper bound on $m^*(G)$, the number of cross arcs in an optimal 2-partition of G. When G has fewer than half as many arcs as nodes we have proved that $m^*(G) = 0$. If G is a tree with bounded node degree we can show that $m^*(G) = O(\log n)$ (assuming n nodes), and if G is planar with bounded node degree we prove that $m^*(G) = O(\sqrt{n})$.

R.M. Karp

Acknowledgment

I wish to express my thanks to my advisor, Professor Richard M. Karp, for our numerous discussions, his valuable ideas and suggestions, and his careful reading of this manuscript.

Table of Contents

CHAPTER 1
INTRODUCTION

1.1 The k-Partition Problem and Its Applications

This thesis is a study of a combinatorial problem which we call the k-Partition Problem: "Given an undirected graph G and an integer k, divide the nodes of G into k equal-sized sets in a way which minimizes the number of arcs which connect nodes in different sets." The problem can be generalized by attaching cost to each arc, and/or assigning a weight to each node. Another variation of the k-Partition Problem involves partitioning the nodes of a hypergraph. We commence this study by mentioning a number of areas where versions of our problem arise, and then outline the contents of the remaining chapters.

Consider the problem of placing a collection of intercommunicating subroutines onto a paged secondary storage device (see [3,17,21,27]). With the aim of minimizing the number of page faults during execution, it is desirable to place routines which reference each other on the same page. We can abstract the problem by creating a graph which contains a node corresponding to each subroutine, and an associated weight proportional to the size of the subroutine. For each pair of subroutines which access each other we create an arc between the corresponding pair of nodes, and place on it an arc cost proportional to the average number of accesses which occur between the two routines during execution.

A pagination of the subroutines defines a partition of those routines, and hence of the corresponding nodes. An arc connecting nodes which are separated by the partition is called a cross arc.

We seek a partition for which the sum of costs of the cross arcs is minimal, subject to the constraint that for each set in the partition the sum of its node weights doesn't exceed a certain constant (the page size). The techniques which have been developed for the k-Partition Problem can also be applied to this problem.

Partitioning algorithms can also be applied to a similar problem, that of placing data onto a paged memory device. The data in a structured database is usually accessed in a systematic manner. Hence, certain collections of records tend to be referenced as a group, and it is desirable that these records be contained within a small number of pages.

Large sparse square matrices are used in the computer solutions to a wide variety of problems (see [8]). The associated (directed) graph has a node for each row of the matrix, and an arc from the i^{th} node to the j^{th} node wherever entry i,j of the matrix is non-zero. Gaussian elimination methods can be made more efficient when blocks of nodes can be identified such that relatively few arcs connect nodes within each block to nodes outside of that block. Such blocks can be found by a k-partition algorithm.

In the field of Design Automation computers are being used to aid in grouping electrical modules into packages (e.g. assigning chips to circuit boards) so as to minimize the number of electrical connections between packages (see [11,19,24,33]). The problem is equivalent to partitioning a hypergraph, where a node represents each module, and an arc (which is a set of two or more nodes) exists for each signal net connecting a set of modules.

Graph partition problems are encountered in the everyday world, though computers aren't generally used to solve them. For instance, workers in a multi-story office building who must frequently interact with each other ought to be assigned to the same floor when possible. Problems of this type fall under the heading "Subdivision of a Business or Social Community".

1.2 Outline of Remaining Chapters

The 2-Partition Problem has been shown to be as hard as any NP-complete problem, which indicates that the existence of a polynomial-time algorithm to solve it is unlikely. In Chapter 2 we show that with the restriction that the maximum node degree of a graph is three, the 2-Partition Problem on such graphs is still NP-hard. A different version of the problem, the General Partition Problem, is also shown to be NP-hard.

Chapter 3 begins by proving that a 2-partition algorithm we call one-opting can always find a solution for which there are at most $\frac{N}{2} + o(N)$ cross arcs, assuming that the graph contains N arcs. Next we prove that on a certain special class of graphs the one-opting algorithm almost always finds a solution for the 2-Partition Problem which is very far from optimal. We have also observed empirically that on a slightly more complex class of graphs none of our heuristic 2-partition algorithms has a good chance of finding a near-optimal solution.

Our next approach to the k-Partition Problem, in Chapter 4, is that of a probabilistic analysis on the space of random graphs. Our graphs here have kn nodes and N randomly-chosen edges. We define a function $L(k,n,N)$, and prove that as n increases the probability goes to one that an optimal k-partition of a graph has at least $L(k,n,N)$ cross arcs. We also find a function $U(n,N)$, and an algorithm which with probability going to one as $n \to \infty$ will find a 2-partition with at most $U(n,N)$ cross arcs. These results give us some insight into the behavior of our k-partition algorithms -- in particular they tell us that the optimal k-partition of a relatively dense graph (one where the ratio of arcs to nodes is large) is likely to be only a little better, percentagewise, than a randomly-selected k-partition.

In Chapter 5 we introduce some new heuristics which suggest new algorithms to solve the 2-Partition Problem. Then we present the results of empirical tests where we executed all of our 2-partition algorithms on some pseudo-randomly generated graphs. Lastly we indicate how most of the algorithms can be implemented so that they run in linear time.

Up until Chapter 6 the only algorithms discussed are those for solving the 2-Partition Problem. Here we show how to extend the algorithms to do k-partitioning and general partitioning. Also, we show how arc costs and node weights can be dealt with, and how hypergraphs are treated.

In the final chapter we take a look at the properties of the 2-Partition Problem for special types of graphs, such as very sparse graphs, trees, and planar graphs. One of our observations is that if the maximum degree of a graph is $d \geq 3$, and there are N arcs, then a 2-partition with at most $\frac{d-1}{2d}N + o(N)$ cross arcs exists. Further, if that graph is a tree then a 2-partition with at most $O(d \log N)$ cross arcs exists. This indicates that graphs with special structure may often be more tractable (i.e. yield partitions with fewer cross arcs) than randomly-chosen graphs like those encountered in Chapter 4.

CHAPTER 2

NP-COMPLETENESS OF TWO PARTITION PROBLEMS

2.1 Definitions

We first introduce some notation which is used throughout this paper: $G = (N,A)$ is a finite undirected graph. N is the set of nodes (vertices) and A is the set of arcs (edges). There will be n (or an integer multiple of n) nodes in our graphs, and N arcs. If $S \subseteq N$ is a subset of the nodes then $\bar{S} \triangleq N - S$. A cut $(S;\bar{S})$ is the set of arcs which connect a node in S with a node in $\bar{S}$. "$\#(S;\bar{S})$" denotes the size of the cut $(S;\bar{S})$, i.e. the number of edges which cross between S and $\bar{S}$. "$|S|$" denotes the number of nodes in S.

A <u>k-partition</u> of G will mean a partition $\{P_1,P_2,\ldots,P_k\}$ of N into k <u>equal-sized</u> subsets. Any (undirected) arc (u,v) which crosses between two different sets of the partition ($u \in P_i$, $v \in P_j$, $i \neq j$) is termed a <u>cross-arc</u>.

The <u>k-Partition Problem</u> is to select from the space of all k-partitions of N one which minimizes the number of cross arcs. The remainder of this paper focuses primarily on aspects of this problem. The special case when $k = 2$ is the 2-Partition Problem. It embodies most of the features of the more general problem, and we will often find it convenient to deal with this simpler version.

The 2-Partition Problem, which is an optimization problem, is converted to a problem of existence by defining it slightly differently. Let "2-P" denote the problem:

> Given an integer m and a graph G, does there exist a 2-Partition of G with at most m cross arcs?

We note that any algorithm which solves the 2-Partition Problem also solves 2-P, so the 2-Partition Problem is at least as hard as 2-P. Suppose we add a restriction to the problem 2-P by requiring that no node of G may have degree greater than three. Then G is constrained to be a relatively sparse graph, and we might hope that in this case 2-P would be easier to solve. We call this restricted version "S2-P".

A problem similar to but distinct from the k-Partition Problem is the <u>General Partition Problem</u>. Here an integer W is fixed. A partition $\{P_1,P_2,\ldots,P_\ell\}$ is "feasible" if $|P_i| \leq W$ for $i = 1$ to ℓ. For this problem ℓ is not fixed. The problem is to find a partition which minimizes the number of cross arcs, from the space of all feasible partitions of G. If we are in addition given an integer m, and wish to know if there exists a feasible partition with at most m cross arcs, then we call the problem <u>GP</u>.

The General Partition Problem has been treated by Lukes [27,28,29].

2.2 <u>Graph Partitioning -- A Hard Problem</u>

Suppose our problem was to find a cut $(S;\bar{S})$ of G of minimum size. A polynomial-time algorithm exists [4] to solve this problem. However, requiring that both pieces of the cut be of equal size, as in the 2-Partition Problem, appears to make the problem much more difficult. We suspect that no polynomial-time algorithm exists for this problem.

A paper by Karp [14] showed that a number of important combinatorial problems have something in common: 1) they are polynomially reducible to each other, implying that if any one of them can be solved in polynomial-time they all can, and 2) no sub-exponential-time algorithm is known for any of them. These problems have since been joined by a long list of other problems, and have come to be called NP-complete (see [1,14] for a more rigorous discussion).

Two different polynomial reductions [6,7] have shown that the Simple Max Cut Problem ("Is there a cut of G containing at least m cross arcs?") is NP-complete, and [7] reduces Simple Max Cut to our problem 2-P, indicating that it is also NP-complete. Since the 2-Partition Problem answers the question raised in 2-P, it is at least as hard. The same is true a fortiori for the k-Partition Problem.

2.3 2-Partitioning with Maximum Node-Degree Three is NP-Complete

As stated above, the demonstration in [7] that 2-P is NP-complete starts with a maximum cut problem on a graph G. Sufficiently many new singleton nodes are added to G to form a modified graph G' so that if we limit our consideration only to those cuts which split G' exactly in half, we are still solving the original max cut problem. Solving the equal-sized max cut problem on G' is equivalent to solving a 2-Partition Problem on the graph complement of G'. The point here is that this construction produces a complementary graph which is dense, i.e. the number of arcs is of order n^2. The possibility is left open that if we are restricted to considering only sparse graphs -- we fix an upper bound d on the degree of any node -- then a nifty algorithm exists for solving 2-P.

In fact, this is not the case, as we show in our first theorem, where d has the value three.

<u>Theorem 2.1</u>. 2-P is polynomially reducible to S2-P.

(Theorem 2.1 implies that S2-P is NP-complete.)

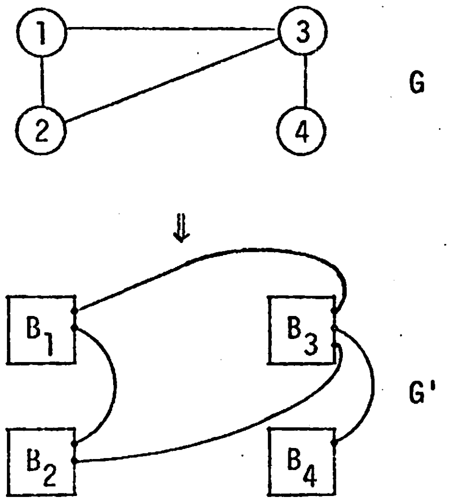
If the value of d is reduced to 2 then the 2-Partition Problem is no longer NP-complete. Every component of a graph with maximum node-degree two consists of a simple cycle or a simple path, and a 2-partition $\{L,R\}$ exists such that at most one component is not wholly contained in L or in R. Hence there exists a 2-partition with two or fewer cross arcs when $d \leq 2$.

We can test in polynomial time whether or not a 2-partition with two or fewer cross arcs exists: To test for a 2-partition with no cross arcs is equivalent to solving a unary-coded SUBSET-SUM problem (see [13]) -- dynamic programming techniques provide a polynomial-time algorithm. To test for a 2-partition with at most one cross arc, remove each arc in turn and test for a 2-partition with no cross arcs. To test for the existence of 2-partitions with two or fewer cross arcs, remove each arc in turn and test for the existence of 2-partitions with at most one cross arc. Induction can be applied to show that for any fixed m there is a polynomial-time algorithm to test for the existence of a 2-partition with m or fewer cross arcs.

Proof of Theorem 2.1. Given an n-node graph G for which a solution to the 2-Partition Problem is desired, we will produce (in time polynomial in the size of G) a graph G' having maximum node degree three such that, given any optimal 2-Partition of G', there exists a straightforward transformation to an optimal solution for G.

Our overall strategy is very simple. There is available to us a certain graph having nodes of degrees two and three which is relatively costly to split into pieces, i.e. there exists no cut of small cardinality which splits off a large set of nodes. We will refer to this special graph as a block. Our construction of G' replaces each node i in G by a block B_i. Some of the degree two nodes in B_i are designated as outlet nodes. Each arc incident with node i in G will be connected to a different outlet node of B_i, so that all node degrees in the final construction are at most three. A block will contain $2(4n+1)^2 - 2$ nodes.

Example.



All of the blocks B_i have the same number of nodes. We will demonstrate that in any optimal 2-partition of G', all of the nodes of a block B_i will lie in the same set. Hence, if G' can be 2-partitioned with m or fewer cross arcs, then G can also. The converse is trivially true, and Theorem 2.1 follows.

Claim 2.1. Suppose the nodes of a block B_i are contained in two disjoint sets S and $\bar{S}$, with $|S| \le |\bar{S}|$, and suppose S contains r outlet nodes. Then

$$\#(S;\bar{S}) - r \ge \max\{\tfrac{1}{2}\sqrt{|S|}, r\} .$$

The construction of the blocks B_i and a proof of Claim 2.1 appear directly after the proof of Theorem 2.1.

Suppose we have a 2-partition of G' in which some of the B_i are divided between the two sets. Using Claim 2.1 we show how to find another 2-partition with fewer cross arcs:

Call the original two sets of the 2-partition L and R. For each block B_i define $L_i = B_i \cap L$ and $R_i = B_i \cap R$.

<u>Algorithm I</u>.

```
1.  $L' \leftarrow \emptyset$,  $R' \leftarrow \emptyset$.
2.  For each block $B_i$:
        if $|L_i| \leq |R_i|$ then $R' \leftarrow R' \cup B_i$
        else $L' \leftarrow L' \cup B_i$.
3.  If $|L'| = |R'|$ stop.
4.  $M_X \leftarrow \max\{L',R'\}$,  $M_N \leftarrow \min\{L',R'\}$.
5.  $t \leftarrow$ (#blocks in $M_X$) - (#blocks in $M_N$).
6.  $H \leftarrow$ ($\frac{t}{2}$ blocks randomly chosen from $M_X$).
7.  $L' \leftarrow M_X - H$,  $R' \leftarrow M_N \cup H$.
```

Starting with a 2-partition $\{L,R\}$ which splits some of the blocks B_i we claim that Algorithm I finds an improved 2-partition $\{L',R'\}$.

Consider a block B_i which was split by the 2-partition $\{L,R\}$, and suppose for definiteness that $|L_i| \leq |R_i|$, with L_i containing r outlet nodes. During step 2 L_i is moved over to R_i's set. This causes a decrease in cross arcs of size at least

$$\#(L_i;R_i) - (\text{\# arcs connecting } L_i \text{ to } L-L_i) \geq \#(L_i;R_i) - r . \qquad (2.1)$$

By Claim 2.1, (2.1) is at least one, so at step 3 $\#(L';R') < \#(L;R)$.

If we are not lucky, and $|L'| \neq |R'|$ at step 3, then we need a further argument. Let ℓ be the number of blocks which we split by the partition $\{L,R\}$, and let b_j be the size of the smaller of the two pieces of the j^{th} split block. Applying Claim 2.1 to (2.1) tells us that step 2 created a decrease in cross arcs of size at least

$$\sum_{j=1}^{\ell} \frac{1}{2}\sqrt{b_j} \quad . \tag{2.2}$$

If our blocks contain m nodes each, then to create the imbalance at step 3 at least $\frac{t}{2}m$ nodes must have moved in step 2, implying that

$$\sum_{j=1}^{\ell} b_j \geq t\,\frac{m}{2} \; . \tag{2.3}$$

By definition we have

$$b_j \leq \frac{m}{2} \; . \tag{2.4}$$

Minimizing (2.2) subject to (2.3) and (2.4) tells us

$$(2.2) \geq \frac{t}{2}\sqrt{\frac{m}{2}} \quad . \tag{2.5}$$

Performing step 7 causes an increase of at most $\frac{t}{2}n$ cross arcs, where n is the maximum degree of any node in G. Recalling that when constructing the B_i's, m was chosen to be $m = 2(4n+1)^2 - 2 > 8n^2$ implies with (2.5) that

$$(2.2) \geq tn \; .$$

Hence after step 7 there is still a net decrease in the number of cross arcs. $\square$

We now reveal what a block looks like. It is convenient to start by drawing what we will call a D-graph. The size of a D-graph (and of a block) is a function of n. The left graph of Fig. 2.1 is a D-graph for $n = \frac{1}{2}$, and Fig. 2.2b shows a D-graph for $n = 1$. In general each of the four "longer sides" of a D-graph consists of $4n$ edges.

To form a block, place a dot (a node) in each of the triangular faces of a D-graph. Next, draw a line (an arc) between every pair of dots which is separated by a single edge of the D-graph (see Fig. 2.1). Now erase the D-graph. The resulting graph is a block, and looks like a stack of hexagons if drawn correctly. A block has $2n$ output nodes which are selected by choosing every other node from among the degree-three nodes along one "side" of the block (see Fig. 2.2a).

It remains to show that Claim 2.1 holds for our block construction. The D-graph will aid us in this endeavor. If we regard the edges along the perimeter (dotted in Fig. 2.2b) as fictitious, then we have a one-to-one correspondence between edges in Fig. 2.1a and edges in Fig. 2.1b.

For Lemma 2.1 and Lemma 2.2 which follow we define S and $\bar{S}$ to be disjoint sets whose union is the nodes of a block. Furthermore $|S| \leq |\bar{S}|$. We define F and $\bar{F}$ to be the corresponding sets of faces in the D-graph. We will assume that each edge of the D-graph has unit length. Define $p(F)$ as the length of the perimeter around the region F (dotted edges don't count). For example, if $F = \{a,b,c,d\}$ in Fig. 2.1b, then $p(F) = 3$. A cut $(S;\bar{S})$ in a block has the same value as $p(F)$ in its D-graph.

Forming a Block

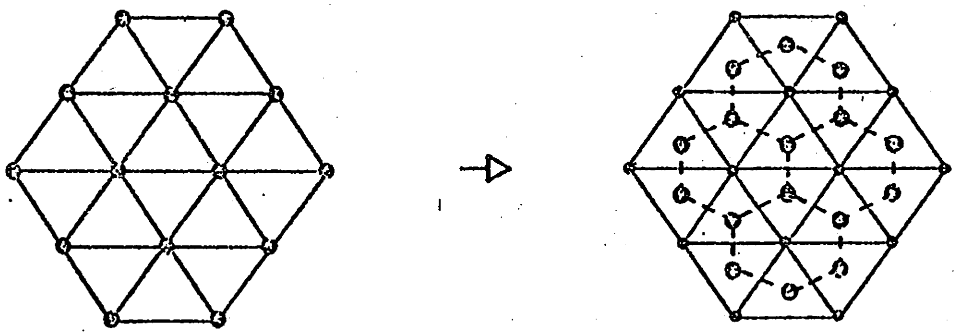


Fig. 2.1

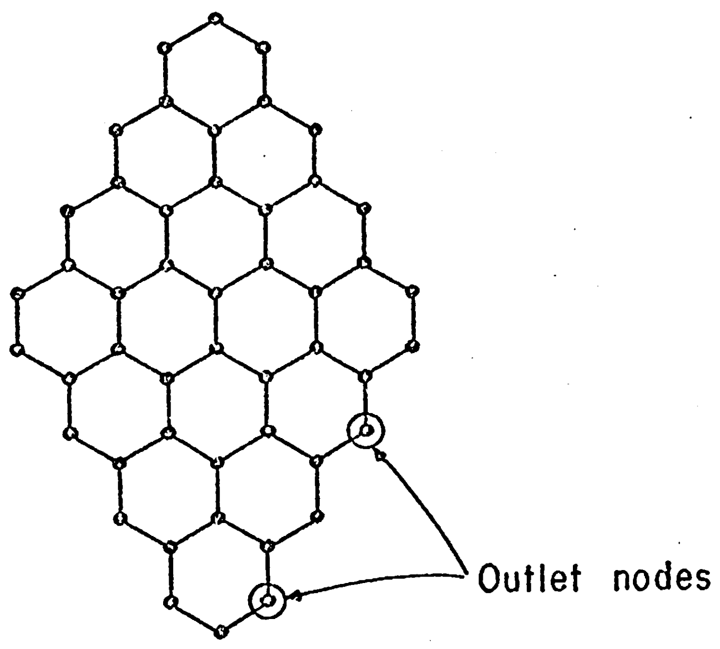


Fig. 2.2a

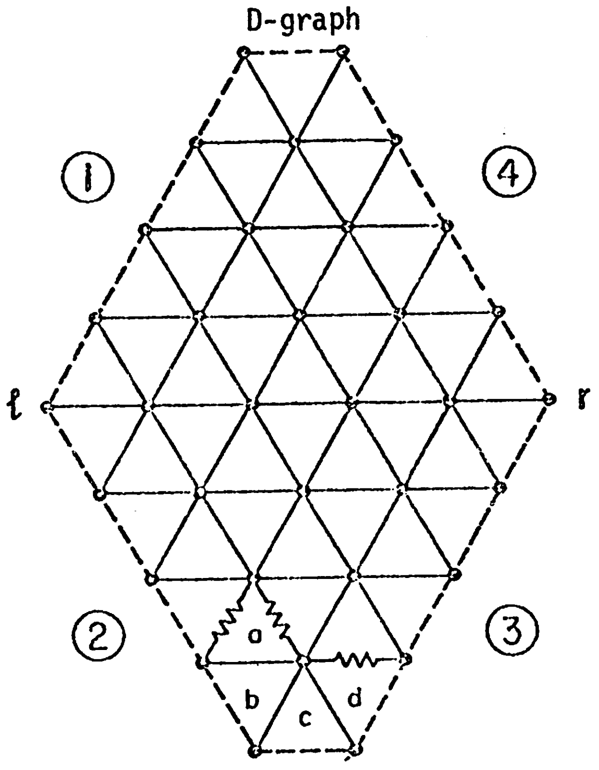


Fig. 2.2b

Lemma 2.1. Given sets of faces F and $\bar{F}$, as defined above, for all F

$$p(F) \geq \sqrt{\lceil F \rceil} .$$

An immediate consequence of Lemma 2.1 is:

Corollary 2.1. Given a block, and sets of nodes S and $\bar{S}$, as defined above, then $\#(S;\bar{S}) \leq \sqrt{\lceil S \rceil}$.

Lemma 2.2. Given a block, and sets of nodes S and $\bar{S}$, as defined above, let S contain r outlet nodes. Then $\#(S;\bar{S}) \geq 2r$.

Claim 2.1 follows directly from Corollary 2.1 and Lemma 2.2.

Proof of Lemma 2.1. For ease of reference we have numbered the sides of the D-graph, and distinguished two nodes with the labels "ℓ" and "r" (see Fig. 2.2b). We will refer to faces as being above or below the line segment $\overline{\ell r}$. Each side has length $s = 4n$.

We observe that if F is a set of faces, then the faces can be "moved" so that they form a connected region whose perimeter has not increased. Hence we may always assume that F is a connected region, and the region is adjacent to at least two of the sides of the D-graph.

Case 1. F touchs only sides 2 and 3, and lies entirely below the line $\overline{\ell r}$ (see Fig. 2.3a). Draw a line parallel to $\overline{\ell r}$ through the highest point of the region F. Let F' be the region containing all faces below this line (Fig. 2.3b). Then

$$p(F) \geq p(F') = \sqrt{\lceil F' \rceil + 1} \geq \sqrt{\lceil F \rceil} .$$

Case 1

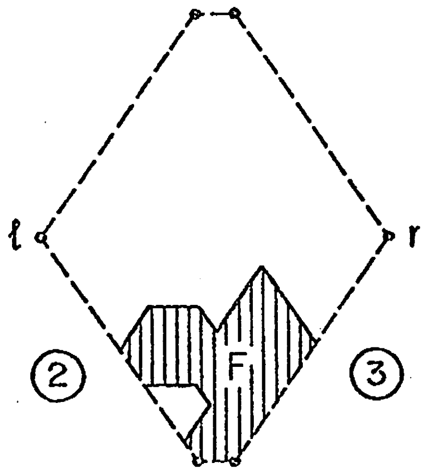


Fig. 2.3a

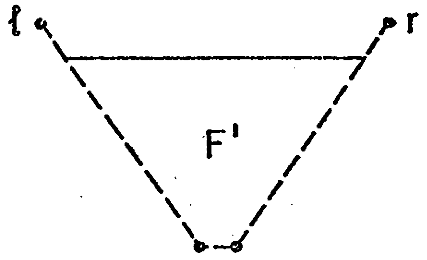


Fig. 2.3b

Case 2. Region F touchs sides 2 and 3 only, but rises above the line $\overline{\ell r}$. We first can add in all faces not in F which lie below $\overline{\ell r}$ without increasing the perimeter. Now our region looks like the shaded part in Fig. 2.3c. Circumscribe the faces above $\overline{\ell r}$ with a trapezoid, and add any faces necessary so that the space inside the trapezoid is filled. Call this new, bigger region F'. Let the trapezoid have sides of length h, and a base of length w, as in Fig. 2.3d. We see that

$$p(F') = s+h+1$$

(the distance from ℓ to r is s+1). Also the number of faces in F' is

$$|F'| = (s+1)^2 - 1 + 2wh - h^2 .$$

Because $w < s$, $p(F')^2 > |F'|$. Hence $p(F)^2 > |F|$.

Case 3. Region F touchs only sides 3 and 4. We can circumscribe F by a rhomboid F' which has a perimeter no larger than p(F). We let the sides have length w and h (see Fig. 2.3e). Then the number of faces in F' is 2wh. The perimeter $p(F') = w+h$, and $(w+h)^2 < 2wh$ for $w, h \geq 1$. The desired inequality follows.

Case 4. Our region F touchs exactly three sides of the D-graph. We first observe that it must then have a perimeter of length $p(F) \geq s+1$. Secondly, there are $2(s+1)^2 - 2$ faces altogether, and F contains at most half of these by definition. Hence $p(F) > \sqrt{|F|}$.

Case 5. F touchs all four sides of the D-graph. We turn our attention to the complementary region $\bar{F}$. Suppose it is made up of t subregions $\bar{F}_1, \bar{F}_2, \ldots, \bar{F}_t$, each of which is connected. Then each subregion must touch at most two of the sides of the D-graph. This allows us to apply the results of Cases 1, 2, and 3 to say that $p(\bar{F}_i) \geq \sqrt{|\bar{F}_i|}$

Case 2

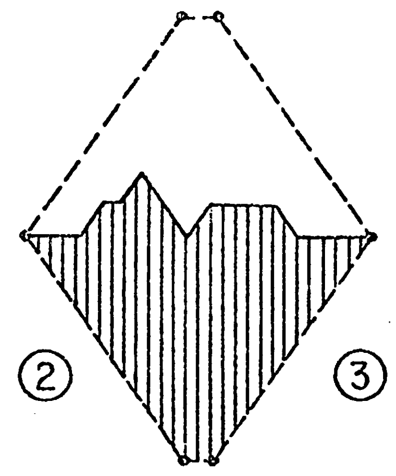


Fig. 2.3c

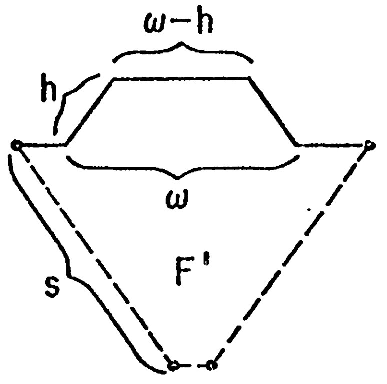


Fig. 2.3d

Case 3

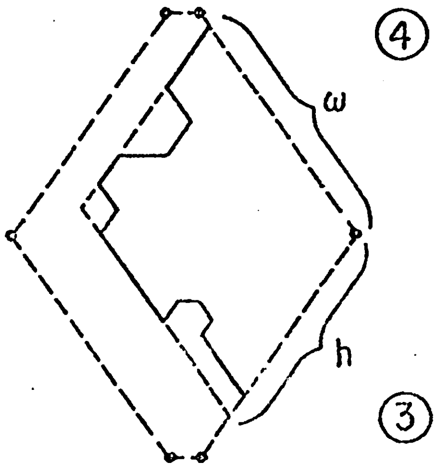


Fig. 2.3e

least $\frac{n}{5} \cdot \frac{4n}{5} = \frac{4n^2}{25} > \frac{5n}{2}$ for $n \geq 16$. Contradiction. Hence, for definiteness and without loss of generality we may assume that P_1 contains at least $\frac{4}{5}$ of the nodes in N_1, and P_2 contains at least $\frac{4}{5}$ of the nodes in N_2. We can now show that in fact $N_1 \subseteq P_1$ and $N_2 \subseteq P_2$.

Suppose y_1 of N_1's nodes lie somewhere else than in P_1, and y_2 of the nodes in N_2 are not in P_2. If we move the y_1 nodes of N_1 to P_1, and the y_2 to P_2, the number of cross arcs decreases by at least $(\frac{4}{5}n - 1)(y_1+y_2)$. This move may cause one of P_1 or P_2 to exceed their maximum allowance of $\frac{3n}{2}$ nodes. Suppose P_1 now has t too many nodes ($t \leq y_1$). Choose t nodes from $P_1 - N_1$ and put them in P_2. This causes an increase of at most $3t$ cross arcs. The new partition is feasible and shows a net decrease of

$$\begin{aligned}(\tfrac{4n}{5} - 1)(y_1+y_2) - 3t &\geq (\tfrac{4n}{5} - 4)(y_1+y_2) \\ &\geq (y_1+y_2) \quad \text{for } n > 5 .\end{aligned}$$

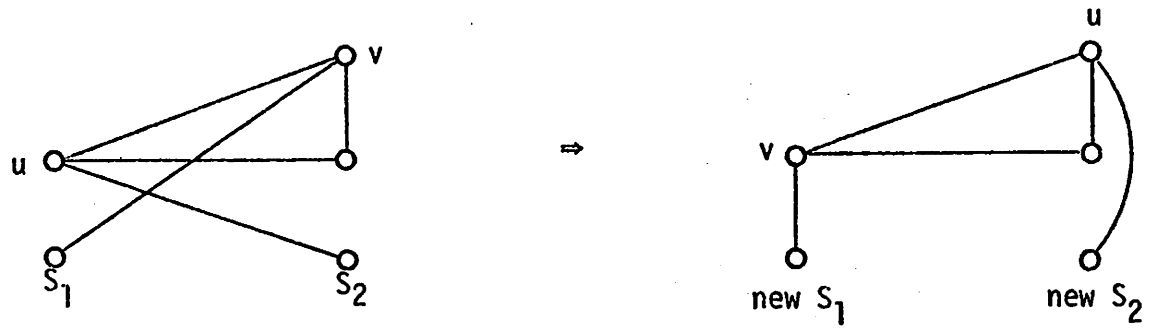
This is a contradiction to the optimality of the starting partition unless $y_1 = y_2 = 0$. This proves Claim 2.2.

Now let us start with a partition $\{P_1, P_2, \ldots, P_\ell\}$ of G', supposing that $N_1 \subseteq P_1$, $N_2 \subseteq P_2$. Let $H = P_3 \cup P_4 \cup \cdots \cup P_\ell$. Since $|H| < \frac{3n}{2}$, the partition $\{P_1, P_2, H\}$ is feasible, and has no more cross arcs than the former partition.

Let $s_1 = \frac{3n}{2} - |P_1|$, $s_2 = \frac{3n}{2} - |P_2|$, $h = |H|$, so that $s_1 + s_2 = h$. In the subgraph induced by H no node degree exceeds three. We claim that H can always be split into disjoint sets S_1 and S_2 such that $|S_1| = s_1$, $|S_2| = s_2$, and $\#(S_1;S_2) \leq h$:

Suppose S_1 and S_2 are disjoint sets whose union is H, and assume that no pairwise exchange of nodes between S_1 and S_2 reduces $\#(S_1;S_2)$. Define $x(u)$ = "number of cross arcs incident to node u." If $x(u) \leq 2$ for all $u \in S_1 \cup S_2$ then $\#(S_1;S_2) \leq 2 \min\{s_1,s_2\} \leq h$, so we're done. Otherwise there exists a u such that $x(u) = 3$. Suppose $u \in S_1$. If there is a $v \in S_2$ with $x(v) \geq 2$, an exchange of u and v reduces the number of cross arcs by at least two.

<u>Example</u>.



Contradiction. Hence $x(v) \leq 1$ for all $v \in S_2$. Then $\#(S_1;S_2) \leq s_2 \leq h$.

Back to the partition $\{P_1,P_2,H\}$. Each node in H is adjacent to a node in P_1 and a node in P_2. Hence if we consider the (feasible) partition $\{P_1 \cup S_1, P_2 \cup S_2\}$, we note that at least h cross arcs have disappeared, and at most h new ones formed when H was split into S_1 and S_2.

We have shown that given any feasible partition of G' with x cross arcs, we can find a 2-partition $\{P_1',P_2'\}$ of G' with not more than x cross arcs and such that $N_1 \subseteq P_1'$, $N_2 \subseteq P_2'$. By construction, the partition $\{P_1'-N_1, P_2'-N_2\}$ is a 2-partition of G with at most $x-n$ cross arcs.

Conversely, if $\{S,\bar{S}\}$ is a 2-partition of G with x cross arcs, then $\{S \cup N_1, \bar{S} \cup N_2\}$ is a partition of G' with $x+n$ cross arcs subject to capacity $\frac{3n}{2}$. □

CHAPTER 3

WORST-CASE RESULTS FOR k-PARTITIONING

3.1 Introduction

This chapter investigates worst-case properties of the k-Partition Problem. Let us define $m^* = m^*(G)$ to be the minimum number of cross arcs among all k-partitions of a graph G with kn nodes and N arcs. We prove that for every such graph G

$$\frac{m^*}{N} \leq \frac{k-1}{k} + \frac{k-1}{k^2 n-k} .$$

This bound is shown to be best possible.

Suppose we have an algorithm in mind to solve the 2-Partition Problem; call it algorithm "A". Then we can define $m_A = m_A(G)$ to be the number of cross arcs in the 2-partition of G found by algorithm A. Assuming that m^*, the value of an optimal 2-partition, is greater than zero, it would be desirable to find a constant upper bound on the ratio $\frac{m_A}{m^*}$, for then we are guaranteeing that algorithm A always gets us to within a fixed multiple of the optimum solution. Unfortunately, all of our results are negative. We prove that on a series of very simple graphs the algorithm we call "one-opting" will with very high probability find a solution which is far from optimum.

For the more complex algorithms we investigated the ratio $\frac{m_A}{m^*}$ empirically. We found a series of simple graphs upon which all of the usual heuristic algorithms fell far short of the optimum. It appears that for all of these algorithms examples of graphs exist for which m_A is arbitrarily large, while $m^* = 1$. Thus none of the available k-partition algorithms can provide the assurance that their solutions are near-optimal. Later chapters address the question of how well they do on the average.

3.2 An Upper Bound on the Number of Cross Arcs

Let G have kn nodes, and suppose one node connects to all of the others, so that it has degree $kn-1$, and all other nodes have degree one. Every k-partition of G has $(k-1)n$ cross arcs, and there are $kn-1$ arcs altogether. Hence,

$$\frac{m^*}{N} = \frac{(k-1)n}{kn-1} = \frac{k-1}{k} + \frac{k-1}{k^2n-k} . \tag{3.1}$$

Again, let G be the complete graph on kn nodes. Then $m^* = \binom{k}{2}n^2$ and $N = \binom{kn}{2}$. The ratio $\frac{m^*}{N}$ equals that in (3.1). We will show that for every graph the ratio $\frac{m^*}{N}$ never exceeds this value.

Fix $\lambda > 0$ and let $\{P_1,P_2,\ldots,P_k\}$ be a k-partition of a graph. Suppose for no pair of sets of nodes U and V is it true that

i) $U \subseteq P_i$, $V \subseteq P_j$ for some i, j, $i \neq j$,

ii) $|U| = |V| \leq \lambda$, and

iii) exchanging U and V reduces the number of cross arcs.

Then we say that the partition $\{P_1,P_2,\ldots,P_k\}$ is "λ-optimal" (in [25] Lin refers to an equivalent notion as "λ-opt").

For the case of 2-partitioning of graphs having $2n$ nodes, the notions "optimal" and "n-optimal" are equivalent, and in fact a solution which is $\frac{n}{2}$-optimal is optimal. Most heuristic 2-partition algorithms only guarantee that their solutions are 1-optimal.

<u>Theorem 3.1</u>. Let G be a graph with $2n$ nodes and N arcs, and suppose $\{P_1,P_2\}$ is a 2-partition of G which is 1-optimal. Then

$$\frac{\#(P_1;P_2)}{N} \le \frac{1}{2} + \frac{1}{4n-2} .$$

<u>Corollary 3.1</u>. Let G be a graph with kn nodes and N arcs, and let m be the number of cross arcs in a 1-optimal k-partition of G. Then

$$\frac{m}{N} \le \frac{k-1}{k} + \frac{k-1}{k^2n-k} .$$

<u>Proof of Corollary 3.1</u>. Let $\{P_1,P_2,\ldots,P_k\}$ be a k-partition of G which is 1-optimal. Define

m_{ij} = "the number of arcs connecting nodes in P_i with nodes in P_j, for $i \neq j$"

ℓ_{ij} = "the number of arcs connecting nodes within P_i"

If G_{ij} is the subgraph of G induced by $P_i \cup P_j$, for $i \neq j$, then $\{P_i,P_j\}$ is a 2-partition of G_{ij}, and it is 1-optimal with respect to G_{ij}. Hence Theorem 3.1 applies to tell us

$$m_{ij} \div (m_{ij}+\ell_i+\ell_j) \le \frac{1}{2} + \frac{1}{4n-2}$$

implying

$$m_{ij} \le \frac{n}{n-1}(\ell_i + \ell_j) . \tag{3.2}$$

Let $m = \sum_{i=1}^{k} \sum_{j=i+1}^{k} m_{ij}$. By (3.2)

$$m \le \frac{n}{n-1} \sum_{i=1}^{k} \sum_{j=i+1}^{k} (\ell_i+\ell_j) = \frac{n}{n-1}(k-1) \sum_{i=1}^{i} \ell_i . \tag{3.3}$$

Hence

$$\begin{aligned}\frac{m}{N} &= m \div \left(m + \sum_{i=1}^{k} \ell_i\right) \\ &\le \frac{(k-1)\frac{n}{n-1}}{(k-1)\frac{n}{n-1}+1} \qquad \text{by (3.3)} \\ &= \frac{k-1}{k} + \frac{k-1}{k^2 n - k}\ . \end{aligned} \qquad \square$$

Comparing Corollary 3.1 with the example resulting in equation (3.1), we observe that our upper bound is tight. We now turn to the

Proof of Theorem 3.1. We are given a graph $G = (N,A)$ with $2n$ nodes and N arcs. Let $\{L,R\}$ be a 1-optimal 2-partition of G. Define

$$x(u) = \text{"number of cross arcs incident with node } u\text{"}$$
$$\Delta(u) = 2x(u) - \text{degree}(u)$$

For $u \in L$, $v \in R$ define

$$a(u,v) = \begin{cases} 1 & \text{if } (u,v) \in A \\ 0 & \text{otherwise} \end{cases}$$

$$\delta(u,v) = \Delta(u) + \Delta(v) - 2a(u,v)$$

If nodes u and v are exchanged, the number of cross arcs decreases by $\delta(u,v)$. Hence, if $\{L,R\}$ is 1-optimal then for all $u \in L$, $v \in R$,

$$\delta(u,v) \le 0\ . \tag{3.4}$$

Let

$$S_L = \sum_{u \in L} \Delta(u)\ , \qquad S_R = \sum_{v \in R} \Delta(v)\ .$$

Assume that the partition $\{L,R\}$ has m cross arcs. Then

$$m = \#(L;R) = \frac{1}{2}N + \frac{1}{4}(S_L+S_R)$$

so that

$$\frac{m}{N} = \frac{1}{2} + \frac{S_L+S_R}{4N} . \qquad (3.5)$$

Our objective is an upper bound on (3.5), so we will seek to maximize $(S_L+S_R)/4N$ subject to the restriction (3.4). Using the fact that

$$N = 2m - \frac{1}{2}(S_L+S_R)$$

we can write

$$\frac{S_L+S_R}{4N} = \frac{1}{2}\,\frac{S_L + S_R}{4m-(S_L+S_R)} . \qquad (3.6)$$

If we regard S_L and S_R as fixed, then (3.6) is maximized by making the m in the denominator as small as possible. We can use (3.4) to determine a lower bound for m.

We introduce new variables to make explicit m's dependence on S_L and S_R. Define for $|i| \leq n$

$$C_i = \{u \in L | \Delta(u) = i\}$$
$$D_i = \{v \in R | \Delta(v) = i\}$$

and let $c_i = |C_i|$, $d_i = |D_i|$. Assume for definiteness that

$$\max\{i | c_i > 0\} \geq \max\{i | d_i > 0\}$$

and define

$$r = \max\{i | c_i > 0\} - 1 .$$

If $r < 0$ then $\Delta(w) \le 0$ for all nodes $w \in L \cup R$, implying $\frac{m}{N} \le \frac{1}{2}$. Otherwise $r \ge 0$. Then (3.4) implies that for all $v \in R$, $\Delta(v) \le 1 - r$. (3.4) further implies that

$$(u \in C_{r+1}) \wedge (v \in D_{1-r} \cup D_{-r}) \Rightarrow (u,v) \in A$$
$$(v \in C_{1-r}) \wedge (u \in C_{r+1} \cup C_{r}) \Rightarrow (u,v) \in A$$

These implications give us a lower bound on m:

$$m \ge c_{r+1}d_{1-r} + c_r d_{1-r} + c_{r+1}d_{-r} \quad . \tag{3.7}$$

Next,

$$\begin{aligned} S_L &= (r+1)c_{r+1} + rc_r + (r-1)c_{r-1} + \cdots + (-n)c_{-n} \\ &\le rn + c_{r+1} - c^- \end{aligned}$$

where $c^- \triangleq c_{r-1} + c_{r-2} + \cdots + c_{-n}$. Similarly,

$$S_R \le -rn + d_{1-r} - d^-$$

where $d^- \triangleq d_{-1-r} + d_{-2-r} + \cdots + d_{-n}$. Hence

$$S_L + S_R \le c_{r+1} + d_{1-r} - c^- - d^- \quad . \tag{3.8}$$

We now wish to plug (3.8) and (3.9) into (3.6). The particular value of the variable r has no bearing on the resulting equation, so we will assume for notational convenience that $r = 0$. Note that with this assumption

$$c_1 + c_0 + c^- = d_1 + d_0 + d^- = n \tag{3.9}$$

(all variables are non-negative). We have

$$(3.6) \leq \frac{c_1 + d_1 - c^- - d^-}{4(c_1d_1+c_0d_1+c_1d_0) - (c_1+d_1-c^--d^-)}$$

$$= [8(\frac{c_1d_1 + c_0d_1 + c_1d_0}{c_1 + d_1 - c^- - d^-}) - 2]^{-1}$$

$$= (8E - 2)^{-1} \qquad (3.10)$$

where we can eliminate c^- and d^- by using (3.9) to get

$$E = E(c_0,c_1,d_0,d_1) \triangleq \frac{c_1d_1 + c_0d_1 + c_1d_0}{2c_1 + 2d_1 + c_0 + d_0 - 2n} .$$

The constraint (3.9) implies that a lower bound on the function E exists, and hence we can bound (3.6) above.

$$\frac{\partial E}{\partial c_1} = \frac{c_0(d_0-d_1) + (d_1+d_0)(2d_1+d_0-2n)}{(2c_1+c_0+2d_1+d_0-2n)^2} . \qquad (3.11)$$

Equation (3.11) is non-positive when $d_1 \geq d_0$. If all variables except c_1 are fixed, and $d_1 \geq d_0$, then E is at a minimum when c_1 is maximal.

$$\frac{\partial E}{\partial c_0} = \frac{c_1(d_1-d_0) + d_1(2d_1+d_0-2n)}{(2c_1+c_0+2d_1+d_0-2n)^2} . \qquad (3.12)$$

Here (3.12) is negative when $d_1 \leq d_0$. Hence if c_0 is the free variable, and $d_1 \leq d_0$, then E is at a minimum when c_0 is maximal.

We conclude that E is at a minimum when $c_1 + c_0$ is maximized, i.e. if $c_1 + c_0 = n$ (so that $c^- = 0$). By symmetry $d_1 + d_0 = n$ is also necessary if E is to be minimal.

We plug $c_0 = c_1 - n$, $d_0 = d_1 - n$ into E to get

$$E' = E(n-c_1, c_1, n-d_1, d_1) = \frac{(c_1+d_1)n - c_1 d_1}{c_1 + d_1}$$

$$\frac{\partial E'}{\partial c_1} = \frac{-d_1^2}{(c_1+d_1)^2} < 0$$

E' decreases as c_1 increases. By symmetry we should also maximize d_1. Hence, setting $c_1 = d_1 = n$ we have

$$E \geq E(0,n,0,n) = \frac{n^2}{2n} = \frac{n}{2} .$$

Next, (3.10) implies (3.6) $\geq (4n-2)^{-1}$ and hence by (3.5)

$$\frac{m}{N} \geq \frac{1}{2} + \frac{1}{4n-2} .$$

This proves Theorem 3.1. $\square$

3.3 The One-Opting and λ-Opting Algorithms

All of the heuristic algorithms under our consideration for solving the 2-Partition Problem are called "iterative-improvement" algorithms (see [19]). Such an algorithm is handed an initial 2-partition which it seeks to improve. Each time a better 2-partition is found, it is regarded as a new initial partition, and the algorithm is re-applied. Eventually a 2-partition is found which the algorithm cannot improve upon. This solution is called "locally optimal" with respect to that algorithm. It may or may not be an optimum solution.

Commonly the initial 2-partition is regarded as "randomly chosen". By a "randomly-chosen 2-partition" we mean that the 2-partition $\{S,\bar{S}\}$ is constructed by selecting half of the nodes at random and placing them in S, and placing the remaining nodes in $\bar{S}$.

The most elementary of the iterative-improvement algorithms scans the nodes to find a pair of nodes lying in opposite sets whose exchange improves the graph, exchanges them, and repeats until no such pairs exist. Let us label this algorithm R. In section 3.2 $\delta(u,v)$ was defined as "the decrease in cross arcs if nodes u and v are exchanged".

<u>Algorithm R.</u>

Assume we start with a 2-partition $\{L,R\}$.

```
1.  If there exists no pair of nodes $u, v$ such that $u \in L$,
    $v \in R$, and $\delta(u,v) > 0$, then stop.
2.  Select any $u \in L$, $v \in R$ such that $\delta(u,v) > 0$.
3.  $L \leftarrow L \cup \{v\} - \{u\}$
    $R \leftarrow R \cup \{u\} - \{v\}$
4.  Go to step 1.
```

Algorithm S is a variation of algorithm R. It is the same except for step 2, which reads

```
2.  Select $u \in L$, $v \in R$ to maximize $\delta(u,v)$.
```

This version finds the exchange which maximizes the decrease in cross arcs for that iteration. Algorithms R and S are the fastest and simplest of the iterative-improvement algorithms. Their performance forms a standard against which other algorithms are usually compared.

Algorithms R and S are specific instances of a more general technique which Shen Lin has called "λ-opting" (see [25] or [16, p.83]). Here exchanges of groups of nodes of any size from one up to λ are performed. The resulting solution is λ-optimal (defined in section 3.2). Algorithms R and S represent one-opting, and their final 2-partitions are one-optimal. λ-opting for $\lambda \geq 2$ is seldom used for the 2-partition problem because it is too time-consuming.

3.4 An Example Where One-Opting Does Poorly

It is our objective to discover examples where the λ-opting algorithms are expected to fail, i.e. we look for graphs whose λ-opting solutions are expected to be far from optimal.

We build a "chain graph" by linking the n nodes of a graph together to form one long chain or path. Specifically, $G = (N,A)$ where

$$N = \{v_1,v_2,\ldots,v_n\} \text{ and}$$
$$A = \{(v_i,v_{i+1}) \mid v_i \in N,\ i < n\} .$$

Assume n is even. A 2-partition of G exists which has only one cross arc: set $L = \{v_1,v_2,\ldots,v_{n/2}\}$ and $R = \{v_{\frac{n}{2}+1},\ldots,v_n\}$.

We will prove that with high probability the usual one-opting algorithms, operating on chain graphs, find 2-partitions with a large number of cross arcs. The indications are that a λ-opting algorithm, with $\lambda > 1$, will also do badly on a chain graph.

We will assume that our one-opting algorithms are patterned after algorithm R, and are started on a randomly-chosen 2-partition $\{L,R\}$ of an n-node chain graph. We place one requirement on the method of implementing step 2. Let us call a node u "good" if $\Delta(u) > 0$. Then step 2 must obey

> Rule γ: If upon entering step 2 of algorithm R there exist good nodes $u \in L$ and $v \in R$ such that $\delta(u,v) > 0$, then step 2 must select a pair of good nodes for the next exchange.

Any version of algorithm R which looks first among the sets of good nodes for the next candidates for exchange satisfies Rule γ. Also, algorithm S satisfies Rule γ. The purpose of the rule is to limit the number of times that nodes u with $\Delta(u) = 0$ participate in an exchange.

Theorem 3.2. Fix $\varepsilon > 0$ and let G be an n-node chain graph. Let R' be a one-opting algorithm obeying Rule γ which is started on a randomly-chosen 2-partition $\{L,R\}$ of G. Then the probability that $m_{R'} \geq \frac{n}{24}(1-\varepsilon)$ goes to one as $n \to \infty$, where $m_{R'}$ is the number of cross arcs in R''s final solution.

Proof. We define a "run of length ℓ" to be a sequence of nodes $v_i, v_{i+1}, \ldots, v_{i+\ell-1}$ such that

i) $v_{i-1} \in \bar{S}$ (unless $i = 1$),

ii) $v_{i+\ell} \in \bar{S}$ (unless $i+\ell-1 = n$), and

iii) $\{v_i, v_{i+1}, \ldots, v_{i+\ell-1}\} \subseteq S$

where $S = L$ or $S = R$.

For short we will refer to a run of length ℓ as an "ℓ-run". Referring to Fig. 3.1a, we see five 1-runs, four 2-runs, and one 3-run. A one-opting algorithm will always find exchanges which eliminate the one-runs. For example, if in Fig. 3.1a v_1 is exchanged with v_5, and v_{11} is exchanged with v_{16} then the resulting partition is as

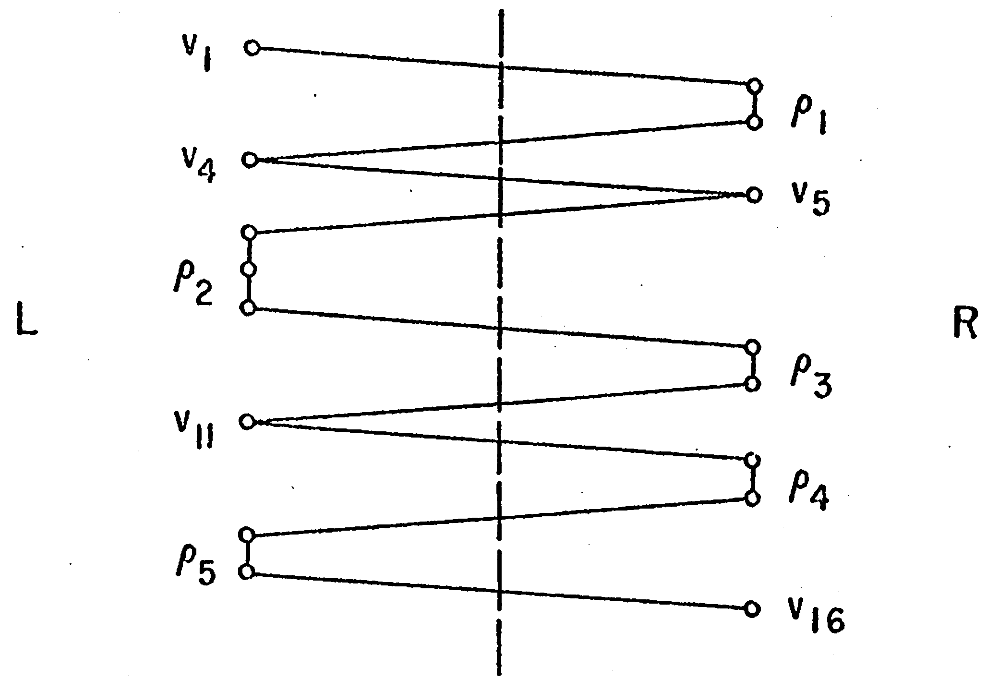


Fig. 3.1a

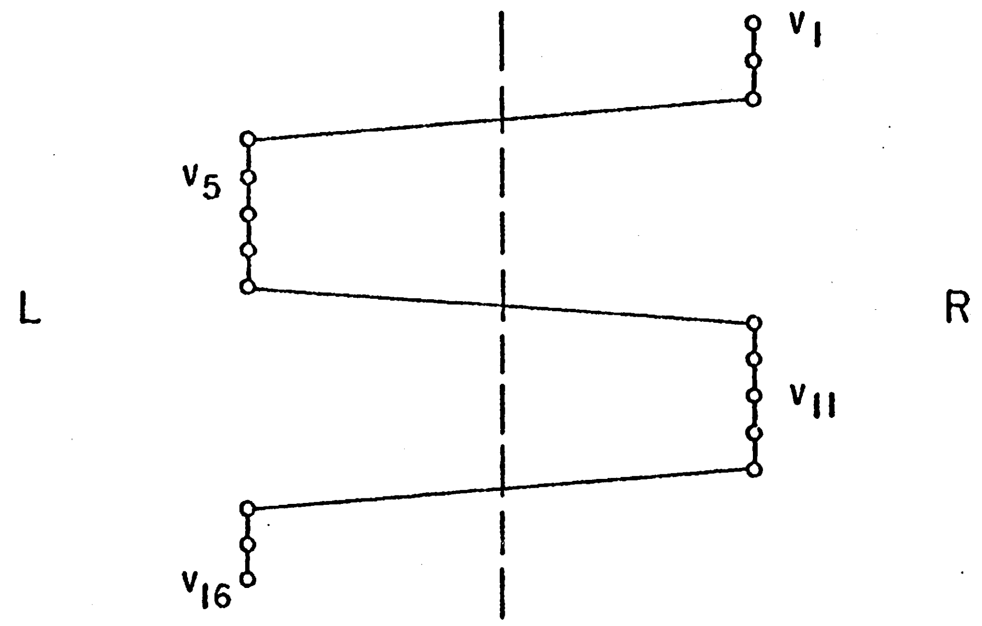


Fig. 3.1b

shown in Fig. 3.1b. There are no one-runs left, and the partition is 1-optimal (it is also 2-optimal, but not 3-optimal). There are two more cross arcs than in an optimal partition.

Our next definition gives us an indication of which cross arcs might be left after a one-opting algorithm has been applied to our starting partition. First we order the runs of length greater than one. Let us say that a run $v_i, v_{i+1}, \ldots, v_{i+j}$ is "below" a run $v_{i'}, v_{i'+1}, \ldots, v_{i'+j'}$ if $i < j$. Next, label the lowest run ρ_1, the next lowest ρ_2, and so on. The runs of length greater than one in Fig. 3.1a have been labelled in this fashion. Now, if runs ρ_i and ρ_{i+1} are on opposite sides of the partition, we say an "alternation" has occurred. In Fig. 3.1a alternations occur between the pairs ρ_1 and ρ_2, ρ_2 and ρ_3, and ρ_4 and ρ_5. If neither of the runs defining an alternation is moved during a one-opting procedure, then a cross arc will remain between them in the final 2-partition. This happened in Fig. 3.1b. We will find it useful to know how many ℓ-runs and alternations are expected to occur in a randomly-chosen 2-partition of a chain graph.

We introduce a notation which is used extensively in Chapter 4: If $a = a(G)$ and $b = b(G)$ are real-valued random variables associated with an n-node graph G, and if for any $\varepsilon > 0$ the probability that $\{|a-b| > \varepsilon\}$ goes to zero as $n \to \infty$ then we write

$$a \equiv_{\varepsilon} b$$

(read "a is epsilon equivalent to b").

Lemma 3.1. Fix an integer $\ell > 0$ and let $\{L,R\}$ be a randomly-chosen 2-partition of an n-node chain graph. Let μ_ℓ = "the number of ℓ-runs defined by $\{L,R\}$". Then

$$\mu_\ell \equiv_{\varepsilon n} \frac{n}{2^{\ell+1}} \cdot$$

Lemma 3.2. Let $\{L,R\}$ be a randomly-chosen 2-partition of an n-node chain graph. Let α = "the number of alternations which occur in $\{L,R\}$". Then for any $\varepsilon > 0$,

$$\alpha \geq \frac{n}{8}(1-\varepsilon)$$

with probability going to one as $n \to \infty$.

The behavior of our one-opting algorithm can be conveniently split into three stages. Stage one exists as long as both L and R contain good nodes (on a chain graph a node is good if and only if it is a 1-run). Define an "iteration" to be the selection of a pair of nodes, and their exchange. Rule γ requires that two good nodes are selected and exchanged at each iteration of stage one. An alternation present before such an exchange still exists after the exchange, since runs of length greater than one haven't been affected.

Observe in Fig. 3.1a that when v_5 is moved to L, v_4 no longer is a good node. This is a general phenomenon -- when a good node is moved, any adjacent runs are absorbed into one longer run. For our purposes it is sufficient to observe that at most three good nodes on each side disappear at each iteration of stage one. Stage one ends when one side has run out of good nodes. From the proof of Lemma 3.1 it can be deduced that initially $\equiv_{\varepsilon n} \frac{n}{8}$ 1-runs occur on each side of the partition. To exhaust these good nodes takes at least $\frac{n}{24}(1-\varepsilon)$ iterations, with a probability going to one as $n \to \infty$.

Assume for definiteness that side L runs out of good nodes first. Stage two is now in effect, and continues as long as R still contains good nodes. Subtracting the number of iterations in stage one from the number of good nodes in R initially, we find that if t is "the number of iterations during stage two", then for $\varepsilon > 0$

$$t < \frac{n}{12}(1+\varepsilon) \tag{3.13}$$

with probability going to one as $n \to \infty$.

If the ℓ nodes of an ℓ-run in L all are moved to R, there is an accompanying decrease of two cross arcs. At the beginning of stage two no 1-runs exist in L, so moving x nodes from L to R results in a decrease of at most x cross arcs (that happens when $\frac{x}{2}$ 2-runs are moved). Hence by (3.13) at most $\frac{n}{12}(1-\varepsilon)$ cross arcs are eliminated as a result of moving nodes from L to R during stage two.

The alternations account for at least $\frac{n}{8}(1-\varepsilon)$ cross arcs whose existence is unaffected by the moving of any of the original one-runs. Thus, at the end of stage two at least $\frac{n}{24}(1-\varepsilon)$ cross arcs still remain, with probability going to one as $n \to \infty$ (we made the subtraction $\frac{n}{8}-\frac{n}{12}$).

Stage three is in effect from the point when the last of R's original 1-runs disappears until the algorithm halts. Its behavior is described in the proof of Claim 3.1 at the end of this section.

Claim 3.1. Stage three contains fewer than εn iterations with probability going to one as $n \to \infty$.

This concludes the proof of Theorem 3.2. □

It should be possible to show that fewer than ϵn iterations of stage two occur, with probability going to one as $n \to \infty$. This would imply that a one-opting algorithm's solution is expected to have at least $\frac{n}{8}$ cross arcs. Empirical evidence indicates that stage two seldom lasts more than 3 or 4 iterations.

Fig. 3.2 shows the results of empirical testing. A version of algorithm S was applied to four chain graphs of varying sizes. On an n-node graph the algorithm was launched from $6 \log_2 n$ different initial 2-partitions. The final solutions contained just under $\frac{n}{5}$ cross arcs on the average, and it was observed that solutions with fewer than $\frac{n}{6}$ cross arcs never appeared on the larger graphs.

Intuitively, the one-opting algorithms do badly on chain graphs because they always remove 1-runs, but eliminate longer runs only accidently. We would expect that a λ-opting algorithm (with $\lambda > 1$) would similarly ignore runs of length greater than λ. Since Lemma 3.1 guarantees that a lot of these long runs exist, we expect that when λ is greater than one the λ-opting algorithms will still do poorly on large chain graphs.

We remark that for the SIMPLE MAX CUT problem (described in section 2.2) a one-opting type of procedure always finds a solution which is within a factor of two of optimal. The algorithm described in [32] finds a solution in which at least one half of all arcs are cross arcs.

Performance of Algorithm S on a Chain Graph

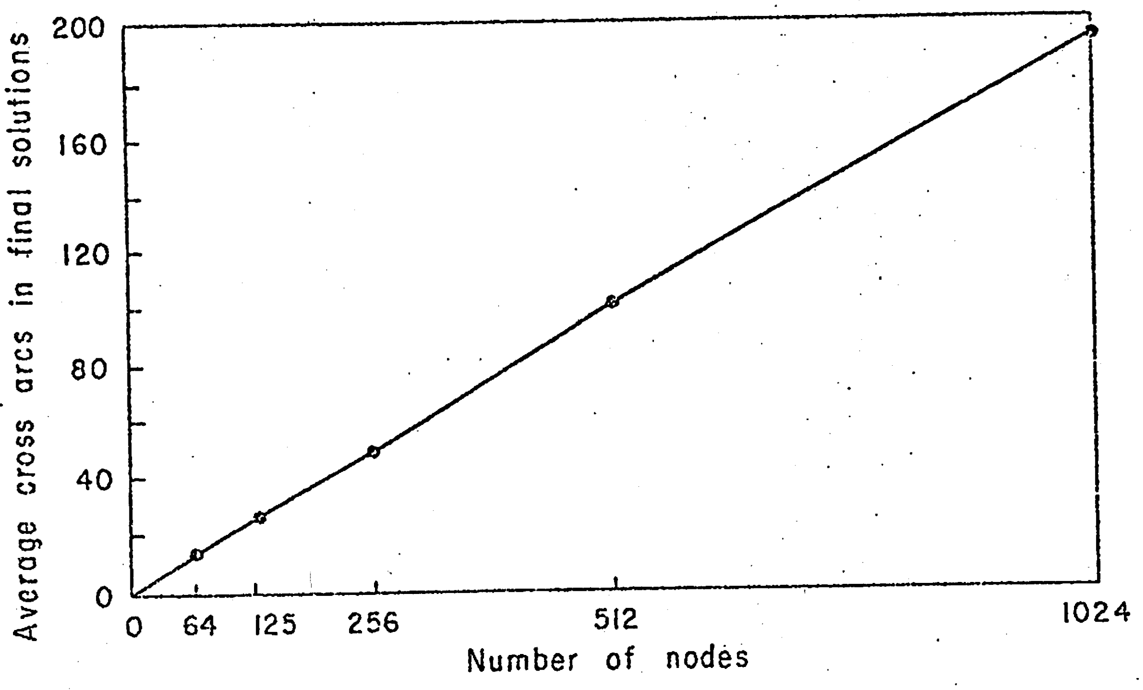


Fig. 3.2

Proof of Lemma 3.1. Given a randomly-chosen 2-partition of an n-node chain graph, we wish to show that the number of ℓ-runs $\mu_\ell \equiv_{\varepsilon n} \frac{n}{2^{\ell+1}}$.

It is convenient for us to adopt a slightly different probabilistic model.

> Assume that each node is placed in the set L with probability $\frac{1}{2}$, and otherwise it is placed in R. (3.14)

Lemma 4.2 of Chapter 4 tells us that with this model $|L| \equiv_{\varepsilon n} |R| \equiv_{\varepsilon n} \frac{n}{2}$ It is to be understood that after performing all of our calculations using the model (3.14), we can patch things up by moving $p = \frac{1}{2}\left||L| - \frac{n}{2}\right|$ nodes from the larger set to the smaller. This patching will affect the existence of at most $3p$ runs. For any $\varepsilon > 0$ we have $3p \leq \varepsilon n$ with probability going to one as $n \to \infty$. Since our results are accurate only to within εn anyway, they will remain unaltered by the patching.

The probability that a particular 2-partition is selected is the same if we chose a 2-partition at random, or if we use the method (3.14) and then patch things up. Hence this new model is interchangeable with the old.

Define the random variables

$$X_i = \begin{cases} 1 & \text{if } v_i, v_{i+1}, \ldots, v_{i+\ell-1} \text{ is a run in R} \\ 0 & \text{otherwise} \end{cases}$$

for $1 \leq i \leq n-\ell+1$. Assume $n > \ell$. For $1 < i < n-\ell+1$, under assumption (3.14) $v_i, v_{i+1}, \ldots, v_{i+\ell-1}$ is a left run with probability $2^{-(\ell+2)}$, and a right run with equal probability. Hence

$$E\,X_i = \frac{1}{2^{\ell+2}}\,.$$

At the ends of our graph we have $E\,X_1 = E\,X_{n-\ell+1} = \frac{2}{2^{\ell+2}}$. So

$$E(\mu_\ell) = 2\sum_{i=1}^{n-\ell+1} EX_i = \frac{n-\ell+3}{2^{\ell+1}} \equiv_{\varepsilon n} \frac{n}{2^{\ell+1}}\,. \tag{3.15}$$

Next we will show that the variance $D(\mu_\ell) = O(n)$, and then Lemma 4.1 in Chapter 4 assures us that $\mu_\ell \equiv_{\varepsilon n} E(\mu_\ell)$. We will bound the variance on the random variable S, "the number of ℓ runs occurring in the set R".

$$S = \sum_{i=1}^{n-\ell+1} X_i\ , \quad \text{so} \quad ES^2 = E\sum_{i=1}^{n-\ell+1}\ \sum_{j=1}^{n-\ell+1} X_iX_j\ .$$

Let us examine the cross products X_iX_j, ignoring the special cases when $i \in \{1,n-\ell+1\}$ or $j \in \{1,n-\ell+1\}$. For $i+\ell+1 < j$, $EX_iX_j = EX_iEX_j = (EX_i)^2$, because assumption (3.14) guarantees independence. For $i \le j \le i+\ell+1$, X_i and X_j are dependent.

Suppose $X_i = 1$. Then $X_{i+1} = X_{i+2} = \cdots = X_{i+\ell} = 0$, because runs can't overlap. This tells us that at least $\ell(n-2\ell-1)$ of the terms EX_iX_j are zero. However,

$$\text{Prob}\{X_{i+\ell+1} = 1 \mid X_i = 1\} = 2 * \text{Prob}\{X_i = 1\}\ ,$$

so that $(n-2\ell-4)$ of the terms EX_iX_j are twice the value of $(EX_i)^2$. The zero terms cancel out the oversized terms to give us

$$E(S^2) \le (n-\ell+1)EX_i^2 + (n-\ell-1)(n-\ell-2)(EX_i)^2\ .$$

Then

$$\begin{aligned} D(S) &= ES^2 - (ES)^2 = ES^2 - (n-\ell-1)(EX_i)^2 \\ &\le (n-\ell+1)(EX_i^2 - (EX_i)^2) \\ &\le nD(X_i) = O(n) \ . \end{aligned} \qquad \square$$

Proof of Lemma 3.2. We wish to show that the number of alternations α is at least $\frac{n}{8}(1-\varepsilon)$ for any $\varepsilon > 0$ with probability going to one as $n \to \infty$.

Assume there are μ runs of length two or more: $\rho_1, \rho_2, \ldots, \rho_\mu$. By Lemma 3.1

$$\mu \equiv_{\varepsilon n} \sum_{\ell=2}^{n} \frac{n}{2^{\ell+1}} \equiv_{\varepsilon n} \frac{n}{4} \ . \tag{3.16}$$

As in the proof of Lemma 3.1 we will simplify things by using the assumption (3.14). Define random variables Y_i, for $i = 1$ to $\mu-1$:

$$Y_i = \begin{cases} 1 & \text{if there is an alternation between } \rho_i \text{ and } \rho_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

Choose an i from $\{1,2,\ldots,\mu-1\}$ and suppose for definiteness that the nodes in the run ρ_i are in L. Let v_j be the first node following the run ρ_i. Then $v_j \in R$, and starts the next run. Assumption (3.14) tells us that $\text{Prob}\{v_{j+1} \in R\} = \frac{1}{2}$, which implies $\text{Prob}\{\rho_{i+1} \in R\} \ge \frac{1}{2}$. Thus

$$\text{Prob}\{\rho_i \text{ and } \rho_{i+1} \text{ define an alternation}\} \ge \frac{1}{2} \ .$$

Hence

$$E(\alpha) = \sum_{i=1}^{\mu-1} EY_i \ge \frac{\mu-1}{2} \ .$$

The assumption (3.14) guarantees that the Y_i's are independent, so $EY_iY_j = EY_iEY_j$ for $i \neq j$. We can now apply Lemma 4.2 of section 4.3 and conclude

$$E(\alpha) \geq \frac{\mu-1}{2}(1-\varepsilon)$$

with probability going to one as $n \to \infty$. Plugging in (3.16) we get

$$E(\alpha) \geq \frac{n}{8}(1-\varepsilon)$$

with probability going to one. □

Proof of Claim 3.1. We will describe the third stage of our one-opting algorithm, and show that its lifetime is brief. We are assuming that at the end of stage one side L ceased to have any good nodes, and hence during stage two L never contained more than one good node at a time, while the number of good nodes in R was steadily decreasing.

The last iteration of stage two may or may not create a 1-run in L. If it doesn't, then all good nodes have vanished, and the algorithm terminates without entering stage three.

The other possibility is that during the last iteration of stage two, as the last good node in R is used up, a node in L belonging to a 2-run is moved to R -- creating a 1-run in L, but none in R. This signals the beginning of stage three. The next iteration exchanges the good node in L with a node w in R for which $\Delta(w) = 0$. If w belongs to a run of length three or longer then all good nodes have been used up, and we stop. However, if w was one half of a 2-run, then a 1-run has just been created in R. The situation here is a mirror image of what it was upon entering the previous iteration. Hence as long as the only good node is exchanged with a member of a 2-run stage three will continue to improve the 2-partition, but the first time that our node w is selected from a run longer than two the algorithm terminates.

We will demonstrate that the chance that the algorithm terminates quickly is very high. Upon entering stage three there are at least $\frac{n}{24}(1-\varepsilon)$ cross arcs remaining (with probability going to one). This implies that there are $z \geq \frac{n}{48}(1-\varepsilon)$ runs of length two or more whose nodes are in R. Label these runs $\rho_1', \rho_2', \ldots, \rho_z'$. We will prove that at least half of these runs are of length three or more, with very high probability. This implies that the chance of terminating whenever node w is selected from R is 50% or better. Thus the chance of stage three continuing for εn iterations goes to zero as $n \to \infty$.

At the end of stage two none of the original runs in R of length 2 or more has been disturbed. Consider a run ρ_i', for $i < z$. Either ρ_i' coincides with one of the original runs ρ_j, or it was formed later by the merging of two or more runs. If the latter case is true then ρ_i' has length at least three. In the former case we observe that as far as the algorithm is concerned, the nodes in ρ_i' $(= \rho_j)$ have never been looked at, because a one-opting algorithm doesn't distinguish 2-runs from longer runs. Since there is no conditioning, we can apply assumption (3.14) to obtain $\text{Prob}\{\rho_j \text{ is a 2-run}\} = \frac{1}{2}$. Hence in either case

$$\text{Prob}\{\rho_i' \text{ is longer than } 2\} \geq \frac{1}{2} .$$

Applying Lemma 4.2 as in the proof of Lemma 3.2 finishes this proof. □

3.5 The Kernighan-Lin Algorithm

Brian Kernighan and Shen Lin have developed a heuristic algorithm to solve the 2-Partition Problem which is more powerful than one-opting (see [16,17]). We will refer to their method as algorithm K. It happens that algorithm K usually finds the optimal partition of a chain graph. However, it fails markedly on a graph which is only slightly more complex.

Algorithm K consists of a series of passes, each pass trying to improve the result of the previous one. A single pass commences by exchanging the pair of nodes u, v which effects the largest decrease in the number of cross arcs. Nodes u and v are then fixed so that they cannot be moved again during this pass. Algorithm K now selects a new pair u', v' to maximize the decrease in cross arcs, but u and v are not considered as candidates for exchange. Nodes u' and v' are exchanged and fixed to their new sides. The algorithm continues in this fashion until all nodes have exchanged sides, resulting in a partition equivalent to the original. The best of the intermediate 2-partitions has been saved, and if it is better than the original partition it becomes a new starting partition for the next pass. Otherwise the algorithm terminates.

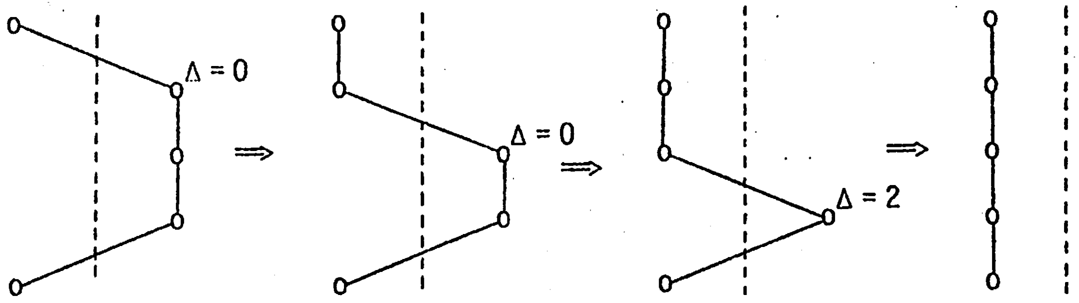
Here is a more formal version of the algorithm:

The input is a 2-partition $\{L,R\}$ of a graph on $2n$ nodes.

Algorithm K

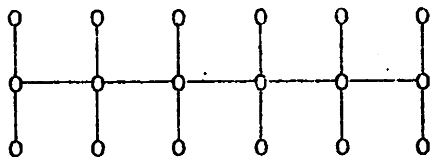
```
i, k, and m are integer variables; U[ ] and V[ ] are arrays of
nodes; L, $L_0$, R, and $R_0$ are sets of nodes; u and v are nodes.
1. $m \leftarrow \#(L,R)$.
2. $L_0 \leftarrow \emptyset$, $R_0 \leftarrow \emptyset$, $k \leftarrow 0$.
3. For i = 1 to n do Steps 4, 5, and 6.
    4. Select $u \in L-L_0$ and $v \in R-R_0$ to maximize $\delta(u,v)$.
    5. (exchange u and v)
        $L \leftarrow L \cup \{v\}-\{u\}$,
        $L_0 \leftarrow L_0 \cup \{v\}$,
        $V[i] \leftarrow v$,
        $R \leftarrow R \cup \{u\}-\{v\}$,
        $R_0 \leftarrow R_0 \cup \{u\}$,
        $U[i] \leftarrow u$.
    6. (save the index of the best intermediate partition)
        If $\#(L;R) < m$ set
            $m \leftarrow \#(L;R)$,
            $k \leftarrow i$.
7. If k = 0 stop (no intermediate partition was an improvement).
8. For i = 1 to k do
        $L \leftarrow L \cup \{V[i]\}-\{U[i]\}$,
        $R \leftarrow R \cup \{U[i]\}-\{V[i]\}$.
9. Goto Step 2.
```

When algorithm K is applied to a chain graph it usually finds a partition with only one cross arc -- an optimal partition. It appears to eliminate the runs of length two or longer by moving nodes (whose delta values are zero) in such a way that the runs get shorter and shorter until they disappear. An illustration should make the process clear:



In order to fool the algorithm it is necessary to add something to the chain graph which makes such a judicious choice of moves less probable. This can be done by adding "leaves" to the graph.

From an n-node chain graph we will build what we call an r-leaf chain graph. For each node v_i in the original graph we create r new nodes $v_{i1}, v_{i2}, \ldots, v_{ir}$ (the leaves) and add them to the graph. The leaves are connected to v_i by adding the arcs $(v_i, v_{i1}), (v_i, v_{i2}), \ldots, (v_i, v_{ir})$. For example, here is a two-leaf chain graph:



Our r-leaf chain graph has $(r+1)n$ nodes. For the case of n even, all r-leaf chain graphs can be partitioned so as to have only one cross arc.

Algorithm K usually finds the optimal 2-partitions of 0-leaf and 1-leaf chain graphs. However, it finds poor solutions on a 2-leaf chain graph -- its solutions here average about (.06)n cross arcs.

Chapter 5 introduces new heuristic 2-partition algorithms, some of which do better than algorithm K on 2-leaf chain graphs. Still, the number of cross arcs in an average solution appears to grow linearly in n for all of the algorithms (see Fig. 5.2). We conclude that none of the available heuristic 2-partition algorithms can assure us that their solutions are always close to optimal.

CHAPTER 4

A PROBABILISTIC ANALYSIS OF THE k-PARTITION PROBLEM

4.1 <u>The Random Graph Model</u>

We have indicated that the existence of a polynomial-time algorithm to solve the k-Partition Problem is very unlikely. In fact, we have shown that the commonly-used polynomial-time algorithms for this problem may find solutions which are arbitrarily far from optimal.

There remains the possibility that one or more of these algorithms might do very well almost all of the time, i.e. the probability that it finds a near-optimal solution is high. We will begin an exploration of this avenue via the theoretical tool of "random graph analysis".

A classic introduction to random graph analysis is a study by Erdös and Renyi [5] in which probability distributions for numerous graph properties are discovered. The expected behavior of algorithms on randomly-chosen graphs has been explored in [2, 9, 15], where NP-complete problems such as the clique, coloring, and Hamiltonian circuit problems have been dealt with.

The graph which we wish to 2-partition is the "random graph" $\Gamma_{n,N}$, which has n nodes and N undirected arcs. We suppose that $\Gamma_{n,N}$ was randomly selected from the $\binom{\binom{n}{2}}{N} = C_{n,N}$ labelled graphs on n nodes and N arcs, i.e. the N arcs are chosen at random among the $\binom{n}{2}$ possible arcs. Thus if $G_{n,N}$ is any one of the $C_{n,N}$ graphs from our sample space of graphs, the probability that $\Gamma_{n,N} = G_{n,N}$ is $C_{n,N}^{-1}$.

Assume that $N = N(n)$ is a fixed function of n. Define the random variable $X = X(\Gamma_{n,N})$ by

X = "the number of cross arcs in an optimal 2-partition of $\Gamma_{n,N}$".

Possible values of X range from 0 to N. We don't know how to find the probability distribution for X, but asymptotic estimates are possible. We can find a function $x_L = x_L(n)$ such that $\mathrm{Prob}\{X < x_L\}$ goes to zero as $n \to \infty$. Also we find a function $x_U = x_U(n)$ for which $\mathrm{Prob}\{X \leq x_U\}$ goes to one as $n \to \infty$. We have in essence bounded X above and below, but not with absolute certainty. We call such bounds "probabilistic" to distinguish them from bounds which are absolute.

4.2 A Probabilistic Lower Bound

For convenience we will assume in this chapter that we are given a graph with kn nodes and $N = ksn$ arcs (for notational convenience we will often write N instead of $N(k,s,n)$). k and s are regarded as fixed constants. Our results will be asymptotic -- increasing in accuracy as $n \to \infty$. Our problem is to divide the graph into k n-node sets, and examine the number of arcs which cross between sets. Let

$$\alpha(f) \triangleq (1-f)^{-(1-f)}\left(\frac{k-1}{f}\right)^f$$

($\log_2\alpha(f)$ equals the entropy function plus the term $f \log_2(k-1)$). If we set $h(f) = \log_e\alpha(f)$ we find that

$$h'(f) = \log_e\left((k-1)\left(\frac{1-f}{f}\right)\right) > 0$$

for f on the interval $[0,\frac{k-1}{k}]$. We conclude that $\alpha(f)$ is an increasing function of f on $[0,\frac{k-1}{k}]$.

Theorem 4.1. Fix s and k, and choose f on $[0,\frac{k-1}{k}]$ such that $\alpha(f) < k^{(\frac{s-1}{s})}$. Then for a randomly chosen graph $\Gamma_{kn,N(k,s,n)}$ the probability that it can be k-partitioned with not more than fN cross arcs goes to zero as $n \to \infty$.

We will have a better idea of what this theorem is saying if we prove a few corollaries.

We know that there always exists a k-partition with $\leq \frac{k-1}{k}N(1+o(1))$ cross arcs for every graph $\Gamma_{kn,N}$.

Corollary 4.1.1. Given any $\varepsilon > 0$, for sufficiently large s the existence of a k-partition of $\Gamma_{kn,N(k,s,n)}$ with $\leq (\frac{k-1}{k} - \varepsilon)N$ arcs has probability going to zero as $n \to \infty$.

Proof. Let $f = \frac{k-1}{k}$. Then

$$\alpha(f) = k^{k^{-1}}\left(\frac{k(k-1)}{k-1}\right)^{\frac{k-1}{k}} = k\ .$$

Since $\alpha(f)$ is strictly increasing on $[0,\frac{k-1}{k}]$ we know that $\alpha(f-\varepsilon) < k$. Hence if s is large enough, then $\alpha(f-\varepsilon) < k^{\frac{s-1}{s}}$. By Theorem 4.1 we know that the chance of a k-partition with $\leq (f-\varepsilon)N$ cross arcs existing goes to zero as $n \to \infty$. □

Corollary 4.1.1 tells us that if the number of arcs in our graphs grows faster than linearly with the number of nodes, then with probability tending to one as $n \to \infty$ an optimal k-partition will be within a factor of $(1-\varepsilon)$ of the upper bound $\frac{k-1}{k}N$. Hence only for the case when the number of arcs is linear (or less) in the number of nodes can we expect to be able to find a k-partition with significantly fewer than $\frac{k-1}{k}N$ cross arcs.

For the special case $k = 2$ we can derive a convenient asymptotic expression for the "f" in Theorem 4.1.

Corollary 4.1.2. Fix an $s \geq 9$. For a randomly-chosen graph $\Gamma_{kn,N}$ the probability of a 2-partition existing with less than $(\frac{1}{2} - \sqrt{\frac{\ln 2}{2s}})N$ arcs goes to zero as $n \to \infty$.

Proof. We will require $(1-f)^{-(1-f)}f^{-f} < 2^{\frac{s-1}{s}}$ to apply Theorem 4.1. Set $f = \frac{1}{2} - \frac{c}{\sqrt{s}}$. We will determine c:

$$(\tfrac{1}{2}+\tfrac{c}{\sqrt{s}})^{-(\frac{1}{2}+\frac{c}{\sqrt{s}})}(\tfrac{1}{2}-\tfrac{c}{\sqrt{s}})^{-(\frac{1}{2}-\frac{c}{\sqrt{s}})} = (\tfrac{1}{4}-\tfrac{c^2}{s})^{-\frac{1}{2}}\left(\frac{\frac{1}{2}-\frac{c}{\sqrt{s}}}{\frac{1}{2}+\frac{c}{\sqrt{s}}}\right)^{\frac{c}{\sqrt{s}}}$$

$$= 2(1-\tfrac{4c^2}{s})^{-\frac{1}{2}}(1-\tfrac{4c}{\sqrt{s}+2c})^{\frac{c}{\sqrt{s}}}$$

$$\sim 2\exp(\tfrac{2c^2}{s} - \tfrac{4c^2}{s+2c\sqrt{s}}) \quad \text{as} \quad s \to \infty .$$

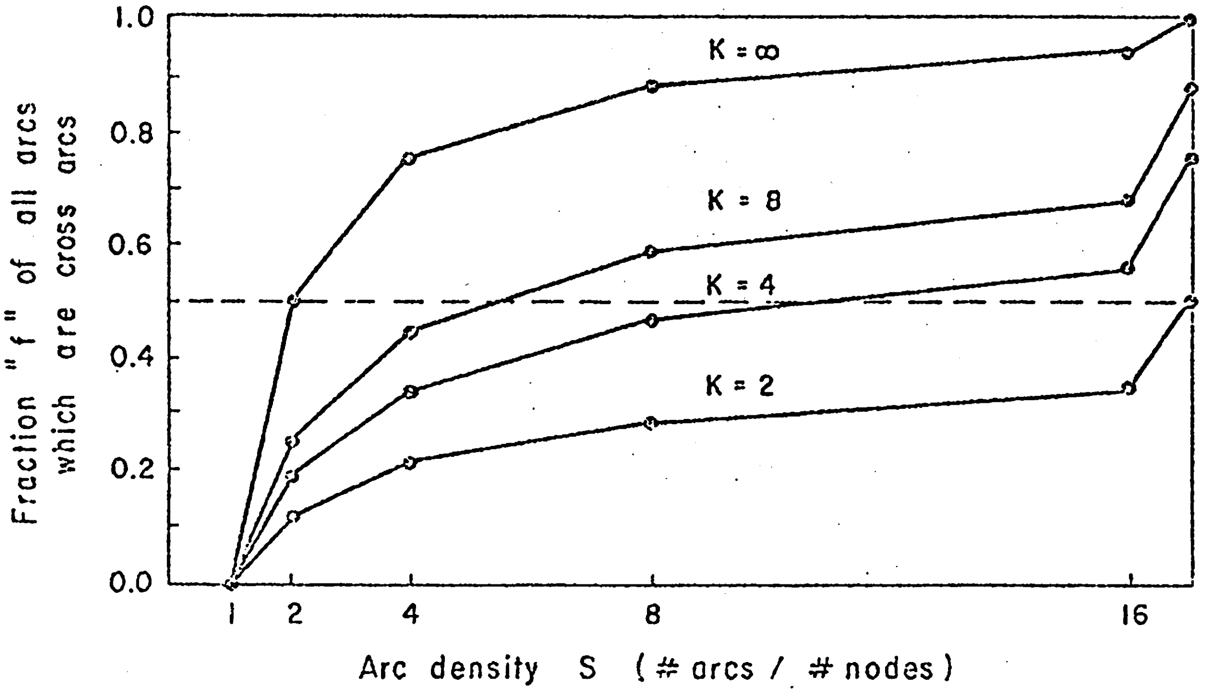
(using (4.2) and (4.5) which appear farther on)

$$\sim 2e^{-\frac{2c^2}{s}}$$

We set $2e^{-\frac{2c^2}{s}} = 2^{\frac{s-1}{s}}$ and solve for c, obtaining $c = \sqrt{\frac{\ln 2}{2}}$. Let $f = f(x) = \frac{1}{2} - \sqrt{\frac{\ln 2}{2s}}$. A more detailed analysis than the one just shown proves that $\alpha(f) < 2^{\frac{s-1}{s}}$ for $s \geq 9$. Numeric computation of $\alpha(f(s))$ and $2^{\frac{s-1}{s}}$ for $s = 1, 2, 4, 6, 8$ showed $\alpha(f)$ to be less than $2^{\frac{s-1}{s}}$ for those values of s as well. □

Corollary 4.1.2 gives us an easily-computed probabilistic lower bound for the 2-Partition Problem. The graph in Fig. 4.1 shows what

K-Partition: Probabilistic Lower Bound



Values for "f" are determined by the equation $\alpha(f) = k^{\frac{s-1}{s}}$ in Theorem 4.1.

Fig. 4.1

our probabilistic lower bound looks like for various values of k and x.

It is interesting to note a simplification which occurs as $k \to \infty$. $\alpha(f)$ converges to $(k-1)^f$, and hence $(k-1)^f < k^{\frac{s-1}{s}}$ implies that f cannot exceed $(\frac{s-1}{s})$. For example, if $N = 2kn$, our probabilistic lower bound approaches the limit $\frac{1}{2}N$ as $k \to \infty$.

Before we begin the proof of Theorem 4.1 we will list a number of inequalities which will be useful further on.

$$n! = (\tfrac{n}{e})\sqrt{2\pi n}(1 + o(1)) \tag{4.1}$$

For $m > 0$,

$$\frac{1}{e} > (1 - \frac{1}{m})^m \ . \tag{4.2}$$

For $m \geq 6$,

$$\frac{1}{e} - \frac{1}{5m} < (1 - \frac{1}{m})^m \ . \tag{4.3}$$

For all natural numbers s and r, with $s < r$,

$$(1 - \frac{s}{2r})^s > \prod_{i=1}^{s} (1 - \frac{i}{r}) \geq (1 - \frac{s}{r})^{\frac{s+1}{2}} \ . \tag{4.4}$$

For $y \geq 6$,

$$(1 - \frac{1}{y}) > e^{-\frac{1}{y}(1 + \frac{e/5}{y - e/5})} \ . \tag{4.5}$$

Inequality (4.3) is derived by expanding $(1 - \frac{1}{m})^m$:

$$\begin{aligned}(1 - \frac{1}{m})^m &= 1 - \binom{m}{1}m^{-1} + \binom{m}{2}m^{-2} - \cdots \pm \binom{m}{m}m^{-m} \\ &= 1 - 1 + \frac{(1 - \frac{1}{m})}{2} - \frac{(1 - \frac{1}{m})(1 - \frac{2}{m})}{6} + \cdots \\ &= (\frac{1}{2} - \frac{1}{6} + \frac{1}{24} - \cdots \pm \frac{1}{m!}) - \frac{1}{m}(\frac{1}{2} - \frac{3}{6} + \frac{6}{24} - \frac{10}{120} + \cdots) + O(m^{-2}) \ .\end{aligned}$$

Now $(\frac{6}{24} - \frac{10}{120} + \frac{15}{720} - \cdots) < \frac{1}{5}$ so

$$(1 - \frac{1}{m})^m > \frac{1}{e} - \frac{1}{5m} + O(m^{-2})$$
$$> \frac{1}{e} - \frac{1}{5m} \quad \text{when} \quad m \geq 6$$

Inequality (4.4) is found in [20]. We can derive (4.5) from (4.3):

$$(1 - \frac{1}{y}) > (\frac{1}{e} - \frac{1}{5y})^{y^{-1}} \qquad \text{by (4.3)}$$
$$= \exp[-y^{-1}](1 - \frac{e}{5y})^{y^{-1}}$$
$$> \exp[-y^{-1}(1 + \frac{e}{5}y^{-1} + (\frac{e}{5})^3 y^{-2} + \cdots + (\frac{e}{5})^{\binom{\ell}{2}} y^{-\ell})]$$
$$\cdot (1 - (\frac{e}{5})^{\ell} y^{-1})^{(\frac{e}{5})^{\binom{\ell}{2}} y^{-\ell}}$$
$$\text{(after applying (4.3) } \ell \text{ times)}$$
$$> \exp[-y^{-1} \prod_{i=0}^{\ell} (\frac{e}{5y})^i](1 + o(\ell^{-1})) .$$

Finally the limit as $\ell \to \infty$ is

$$\exp[-y^{-1}(1 - \frac{e}{5y})^{-1}] = \exp[-y^{-1}(1 + \frac{e/5}{y - e/5})] \quad .$$

A proof of a result similar to Theorem 4.1 for the case $k = 2$ appears in [20]. Our version sharpens their result, and generalizes to an arbitrary k-partition.

Note. In the proofs of Theorems 4.1 and 4.2 the asymptotic notations $O(\ )$, $o(\ )$ and $\sim$ are understood to hold for increasing n unless dependence on another variable is specified.

<u>Proof of Theorem 4.1</u>. The probability that an optimal k-partition of $\Gamma_{n,N}$ has no more than m cross arcs is bounded above by the expected number of k-partitions of $\Gamma_{n,N}$ having $\leq m$ cross arcs. The value of this expression is

$$\frac{(kn)!}{(n!)^k} \sum_{\ell=0}^{m} \binom{\binom{k}{2}n^2}{\ell} \binom{k\binom{n}{2}}{N-\ell} \binom{\binom{kn}{2}}{N}^{-1} \tag{4.6}$$

which is the number of such partitions in the space of all graphs with kn nodes and N arcs, divided by the number of those graphs. We will now bound (4.6) above for increasing n.

$$(4.6) \leq k^{kn} \sum_{\ell=0}^{m} \binom{N}{\ell} \frac{\prod_{i=0}^{\ell-1} \left(\frac{k(k-1)n^2}{2} - i\right) \prod_{i=0}^{N-\ell-1} \left(\frac{kn(n-1)}{2} - i\right)}{\ell_{i=0}^{N-1} \left(\frac{kn(kn-1)}{2} - i\right)}$$

$$= k^{kn} \sum_{\ell=0}^{m} \binom{N}{\ell} \frac{(k-1)^{\ell}\left(1 - \frac{1}{n}\right)^{N-\ell} \prod_{i=0}^{\ell-1} \left(1 - \frac{i}{\binom{k}{2}n^2}\right) \prod_{i=0}^{N-\ell-1} \left(1 - \frac{i}{k\binom{n}{2}}\right)}{k^N\left(1 - \frac{1}{kn}\right)^N \prod_{i=0}^{N-1} \left(1 - \frac{i}{\binom{kn}{2}}\right)}$$

We note that for $c > 0$,

$$\prod_{i=0}^{N-1} \left(1 - \frac{i}{cn^2}\right) \geq e^{-\frac{k^2s^2}{2c}}(1 + o(1)) \, . \tag{4.7}$$

This is because

$$\prod_{i=0}^{N-1} \left(1 - \frac{i}{cn^2}\right) \geq \left(1 - \frac{N-1}{cn^2}\right)^{\frac{N}{2}} \qquad \text{by (4.4)}$$

$$= \left(1 - \frac{ks}{2cn}\right)^{ksn}(1 + o(1))$$

$$\geq e^{-\frac{k^2s^2}{2c}}(1 + o(1)) \qquad \text{by (4.3)} \, .$$

From (4.5) we have

$$(1 - \frac{1}{kn})^N \geq e^{-s(1+O(n^{-1}))} \quad . \tag{4.8}$$

Now we use (4.7) and (4.8) to get

$$(4.6) \leq k^{kn} \sum_{\ell=0}^{m} \binom{N}{\ell}(k-1)^{\ell} k^{-N} \cdot O(1) \ .$$

For any f such that $0 \leq f \leq 1$,

$$\binom{N}{fN} = O(N^{-\frac{1}{2}}(1-f)^{-(1-f)N} f^{-fN}) \ . \tag{4.9}$$

This is discovered by application of the Stirling approximation (4.1). Now (4.9) gives us

$$\begin{aligned}(4.6) &\leq k^{kn-N} N^{-1/2} \sum_{\ell=0}^{m} [(1-f_\ell)^{-(1-f_\ell)} (\frac{k-1}{f_\ell})^{f_\ell}]^N \cdot O(1) \\ &\quad \text{(where we set } f_\ell \triangleq \frac{\ell}{N}) \\ &= k^{kn-N} N^{-1/2} \sum_{\ell=0}^{m} \alpha(f_\ell)^N \cdot O(1) \\ &\leq k^{kn-N} N^{-1/2} (m+1) \alpha(f_m)^N \cdot O(1) \end{aligned} \tag{4.10}$$

because α is increasing on $[0, \frac{k-1}{k}]$ and we can assume that $f_m \leq \frac{k-1}{k}$. Let $r = k^{(\frac{1}{s}-1)} \alpha(f_m)$. Then $(4.10) = O(r^N N^{1/2})$, which converges to zero when $r < 1$. This is true if $\alpha(f_m) < k^{(1-\frac{1}{s})}$, which is assumed in the statement of Theorem 4.1. $\square$

4.3 A Probabilistic Upper Bound

We now turn our attention to the problem of finding a probabilistic upper bound for the k-partition problem, using our random graph model. We present an algorithm which can be applied to the 2-partition problem, and which can be analyzed with a good deal of precision. Chapter 6 discusses how several algorithms for the 2-partition problem, including this one, can be extended to find a k-partition. Extending our analysis to cover the case of arbitrary k proved to be overly complex. For fixed values of s and k one could grind out some numeric approximations, but no satisfactory general formulas or asymptotic approximations were found.

Thus, the problem we deal with here is, given a randomly chosen undirected graph $\Gamma_{2n,N}$ with $2n$ nodes and $N = 2sn$ arcs, divide the nodes into two equal sized sets L and R, so as to minimize the number of arcs which cross from L to R.

To simplify the statement of the next theorem, let us define a few functions:

$$I_j(2s) \triangleq \sum_{i=0}^{\infty} \frac{s^{2i+j}}{i!(i+j)!} \qquad (j^{\text{th}} \text{ order modified Bessel function})$$

$$b(s) = \lceil 1.71\sqrt{s} \rceil \qquad (\text{smallest integer} \geq 1.71\sqrt{s})$$

$$H(s) = \left[\sum_{j=b(s)}^{\infty} e^{-2s} j I_j(2s)\right] \cdot \left[1 - 2\sum_{j=b(s)}^{\infty} e^{-2s} I_{j+1}(2s) - \sum_{j=b(s)}^{\infty} e^{-2s} \frac{j}{s} I_j(2s)\right]$$

(humongous equation)

<u>Theorem 4.2</u>. Fix $s > 0$. For a randomly chosen graph $\Gamma_{2n,N(2,s,n)}$ the probability that it has a 2-partition with $\leq N(\frac{1}{2} - H(s))$ cross arcs goes to one as $n \to \infty$.

We expect that upon seeing function $H(s)$ one will not be infused with a feeling of comfortable familiarity. Fortunately we can provide an alternative. We will show in the proof of Theorem 4.2 that as s grows to infinity

$$H(s) \sim .238s^{-1/2} \quad \text{(ignoring decimal roundoff)}$$

This approximation is fairly good even when s is as small as unity.

Let us compare our lower and upper bounds. Corollary 4.1.2 says that with probability tending to one the optimal 2-partition will have at least $N(\frac{1}{2} - (.589)s^{-1/2})$ cross arcs. If we view ourselves as starting from an initial solution of $\frac{N}{2}$ cross arcs (the worst case upper bound), we have beaten our solution down about $\frac{4}{10}$ of the distance from $\frac{N}{2}$ to the lower bound. The values of the upper and lower bounds for various values of s are shown in Fig. 4.2.

In [20] a much weaker, non-constructive upper bound is derived. That result states that the probability that a graph can be 2-partitioned with not more than $N(\frac{1}{2} - (.085)s^{-1})$ cross arcs is at least ninety-eight percent.

Our algorithm commences by dividing the $2n$ nodes arbitrarily into two equal-sized sets. This constitutes a randomly-chosen 2-partition. We then seek to improve the initial solution by exchanging appropriately chosen nodes among the two sets. This format is common to most 2-partition algorithms. Their differences arise in the manner in which these changes are effected.

Let $\Gamma_{2n,N} = (\mathcal{N},\mathcal{A})$ where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of arcs. Let us refer to the two sets in our initial 2-partition as L and R, so $L \cap R = \mathcal{N}$. For any node $u \in \mathcal{N}$ we define

Comparison of the Probabilistic Lower and
Upper Bounds for the Two-Partition Problem

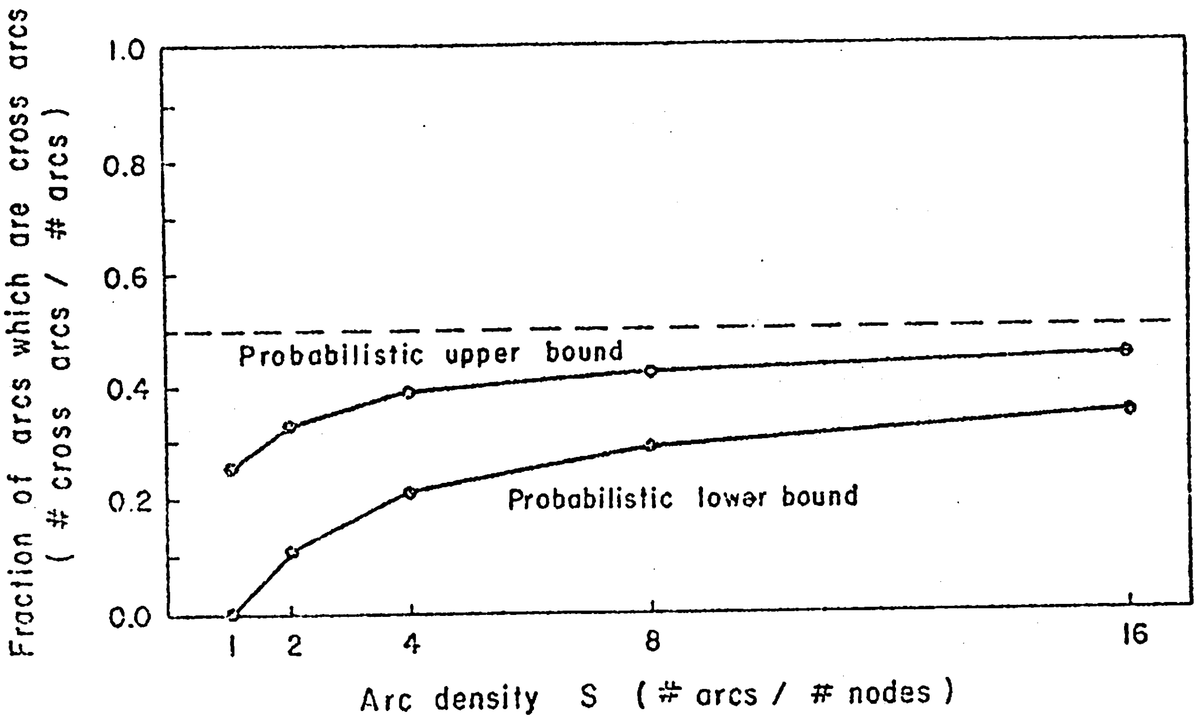


Fig. 4.2

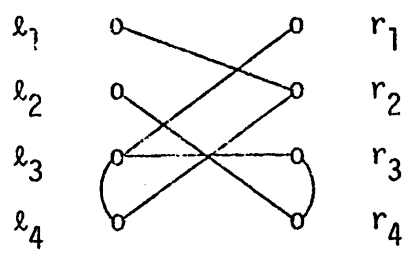
$$\deg_L(u) \triangleq |\{\{u,j\} \in A \mid j \in L\}| \ ,$$
$$\deg_R(u) \triangleq |\{\{u,j\} \in A \mid j \in R\}| \ \text{ and}$$
$$\deg(u) \triangleq \deg_L(u) + \deg_R(u) \ .$$

Next we define, for any node $u \in L$,

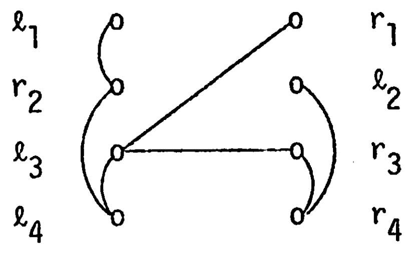
$$\Delta(u) \triangleq \deg_R(u) - \deg_L(u) \ .$$

If $u \in R$, then we define $\Delta(u)$ as $\deg_L(u) - \deg_R(u)$. In each case, $\Delta(u)$ is the number of arcs adjacent to u which cross to the other set, minus these arcs adjacent to u which don't leave u's set. If $\Delta(u) > 0$ then moving u to the other set will decrease the number of cross arcs by an amount $\Delta(u)$. We refer to the nodes u such that $\Delta(u) > 0$ as "good" nodes.

Our algorithm will identify the good nodes in L and R, and exchange some of them. An example will indicate some of the consequences of such an exchange.



Before



After exchanging ℓ_2 and r_2

Initially, assuming $L = \{\ell_i\}$ and $R = \{r_i\}$, nodes ℓ_1, ℓ_2, ℓ_3, r_1 and r_2 are "good". The decrease in cross arcs is $\Delta(\ell_2) + \Delta(r_2) = 1 + 2 = 3$. If we had exchanged ℓ_1 and r_2 the decrease would have been $\Delta(\ell_1) + \Delta(r_2) - 2$ because the arc $\{\ell_1, r_2\}$ remains a cross arc after the exchange. Although ℓ_3 and r_1 are still good nodes after the first exchange, exchanging them won't improve the graph -- the decrease is $\Delta(\ell_3)+\Delta(r_1)-2=0$. $\Delta(\ell_1)$ decreased, although ℓ_1 didn't move, and it is no longer a good node.

In general, many of the initially good nodes are no longer good after some exchanges have taken place. Also, some nodes which were bad initially may become good somewhere along the way. The usual one-opting algorithms (see section 3.3) alternately swap a pair of good nodes (or a good node and a $\Delta = 0$ node), and then update the Δ-values of the other nodes, until no pairwise exchanges exist which improve the 2-partition. The partition is now "1-optimal" (see section 3.1). Chapter 5 explores empirically how well one-opting algorithms and others perform on randomly-generated graphs.

For purposes of analysis we looked for a simpler algorithm. One possibility is an algorithm which considers as candidates for exchange only those nodes which were good initially, and have not been previously moved. This version can be viewed as selecting a block of good nodes from each side and swapping the two blocks. To select the nodes for each block we chose to set a parameter "b", and put into the blocks those nodes u such that $\Delta(u) \geq b$ (a small patch is necessary to insure that sets L and R are of equal size after the exchange). The forthcoming analysis finds the optimal value for b, and determines the expected improvement in the number of cross arcs. Section 4.4 verifies these quantities empirically.

Our analysis succeeds because certain quantities relative to any graph $\Gamma_{n,N}$ can be shown to be nearly equal for almost all graphs with n nodes and N arcs. For example, given any $\varepsilon > 0$, we will show that the probability that {the number of cross arcs for a randomly-chosen 2-partition is within εn of $\frac{N}{2}\}$ goes to one as $n \to \infty$. We introduce a special notation for this type of occurrence: If $a = a(\Gamma)$ and $b = b(\Gamma)$ are numerical quantities associated with a graph Γ_1 and if for any $\varepsilon > 0$ the probability that $\{|a-b| > \varepsilon\}$ is $O(P(n,\varepsilon))$, with $P(n,\varepsilon) = o(1)$, then we write

$$a \equiv_\varepsilon b \quad \text{with r.c.} \quad P(n,\varepsilon)$$

(real "a is epsilon-equivalent to b with rate of convergence $P(n,\varepsilon)$").

We now prove a couple of lemmas which give sufficient conditions for ε-equivalence.

Lemma 4.1. Let $\theta_n = \theta_n(\Gamma_{n,N})$ be a random variable with $E(\theta_n) = n\mu$ and $D(\theta_n) = O(n)$. ($D(\ )$ is the variance). Then

$$\frac{\theta_n}{n} \equiv_\varepsilon \mu \quad \text{with r.c.} \quad \frac{1}{n\varepsilon^2} .$$

Proof. Chebyshev's inequality [12] tells us that

$$\text{Prob}\{|\theta_n - n\mu| \geq \varepsilon\} \leq \frac{D(\theta_n)}{\varepsilon^2}$$

so that

$$\text{Prob}\{|\theta_n - n\mu| \geq \varepsilon n\} \leq \frac{D(\theta_n)}{n^2\varepsilon^2} = O(\frac{1}{n\varepsilon^2})$$

which goes to zero as $n \to \infty$. □

Corollary 4.1.1. Let $\{\xi_i\}_{i=1,n}$ be a set of 0-1 random variables (determined by $\Gamma_{n,N}$) and let $S_n = \sum_{i=1}^{n} X_i$ have a hypergeometric (or binomial) distribution with mean $n\mu$. Then $\frac{S_n}{n} \equiv_\varepsilon \mu$ with r.c. $O(\frac{\mu}{n\varepsilon^2})$.

Proof. In [12] $E(S_n)$ is shown to equal $n\mu$, and $D(S_n) = O(n\mu)$. Apply Lemma 4.1. □

Lemma 4.2. Let $X_1, X_2, \ldots, X_n$ be a sequence of random variables (with values determined by $\Gamma_{n,N}$) having identical means μ, bounded variance, and such that

(i) $EX_i^2 = EX_j^2$ for all i, j and

(ii) $EX_iX_j = EX_{i'}X_{j'}$ for all $i \neq j$, $i' \neq j'$.

Suppose furthermore that

(iii) $EX_iX_j \leq EX_iEX_j$ for all $i \neq j$.

Define $S_n = \sum_{i=1}^{n} X_i$. Then $\frac{S_n}{n} \equiv_\varepsilon \mu$ with r.c. $\frac{1}{n\varepsilon^2}$.

Proof. We need only to show that the variance $D(S_n) = O(n)$ and then Lemma 4.1 gives us our result.

$$\begin{aligned} E(S_n^2) &= E(\sum_{i=1}^{n} \sum_{j=1}^{n} X_iX_j) \\ &= E(\sum_{i=1}^{n} X_i^2) + E(\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n} (X_iX_j)) \\ &= nEX_1^2 + (n^2-n)E(X_1X_2) \\ &\leq nEX_1^2 + (n^2-n)EX_1EX_2 \,. \end{aligned}$$

Then

$$\begin{aligned} D(S_n) &= ES_n^2 - (ES_n)^2 \\ &\leq nEX_1^2 + (n^2-n)\mu^2 - (n\mu)^2 \\ &= n(EX_1^2-\mu^2) \\ &= nD(X_1) \\ &= O(n) \,. \end{aligned}$$

□

Remark 4.1. There is a trade-off between the value of ε and the rate of convergence for any ε-equivalence expression. We will sometimes find it necessary to let $\varepsilon = \varepsilon(n) = o(1)$. In this case convergence to zero is less rapid, but in the case of Lemma 4.1 it still occurs if $\varepsilon = \frac{1}{o(n^{1/2})}$. For example, if $\varepsilon = (\log n)^{-1}$ the rate of convergence is $O(\frac{\log^2 n}{n^2}) = o(1)$ for $D(\theta_n) = O(n)$.

Note. For the remainder of the chapter "log( )" will denote the natural logarithm. Also, we will write "$\lfloor x \rfloor$" to mean "the largest integer not greater than x".

Using the elementary inequality, for all real x,

$$1 + x \leq e^x$$

and inequality (4.5), we can derive a useful asymptotic formula:

For $\varepsilon > 0$, m increasing, with $\varepsilon = \varepsilon(m) = o(1)$ we have

$$(1 \pm \tfrac{\varepsilon}{m})^m \sim 1 . \tag{4.11}$$

We now have the main event:

Proof of Theorem 4.2. We suppose that we are handed a randomly-chosen graph $\Gamma_{2n,N}$ with $N = 2sn$. The nodes are divided into two n-node sets L and R. We set

qn = # of cross arcs from L to R

$p_L n$ = # of arcs connecting nodes in L

$p_R n$ = # of arcs connecting nodes in R

Between each pair of nodes i and j is a slot where an arc may appear. Let

$$X_{ij} = \begin{cases} 1 & \text{if arc } \{i,j\} \in A \\ 0 & \text{otherwise .} \end{cases}$$

Then

$$\begin{aligned} EX_{ij} &= \frac{\#\text{ arcs in } \Gamma_{2n,N}}{\#\text{ slots}} \\ &= \frac{2sn}{\frac{1}{2}(4n^2-2n)} \\ &\sim \frac{s}{n} \quad \text{as} \quad n \quad \text{grows.} \end{aligned}$$

There are n^2 slots connecting nodes in L with nodes in R. The corresponding X_{ij}'s are related such that $\sum_{i\in L} \sum_{j\in R} X_{ij} = qn$ has a hypergeometric distribution (with variance $O(n^2 \frac{s}{n})$). Corollary 4.1.1 applies to tell us that

$$q \equiv_\varepsilon \frac{n^2}{n} EX_{12} \sim s \quad \text{with r.c.} \quad n^{-1} .$$

Reapplying the same type of argument obtains

$$q \equiv_\varepsilon s \ , \quad p_L \equiv_\varepsilon \frac{s}{2} \ , \quad p_R \equiv_\varepsilon \frac{s}{2} \tag{4.12}$$

all with r.c. n^{-1}.

We now have our first item of information about our graph $\Gamma_{2n,N}$. The next thing we would like to know is the probability that a randomly-chosen node u has $\Delta(u) = j$, for $j \in \{0,1,\ldots,n\}$. Let us assume for definiteness that $u \in L$. As a preliminary we compute

$$\text{Prob}\{\deg_L(u) = j\} \quad \text{and}$$
$$\text{Prob}\{\deg_R(u) = j\} \quad \text{for} \quad 0 \leq j \leq n .$$

Fig. 4.3 diagrams our situation.

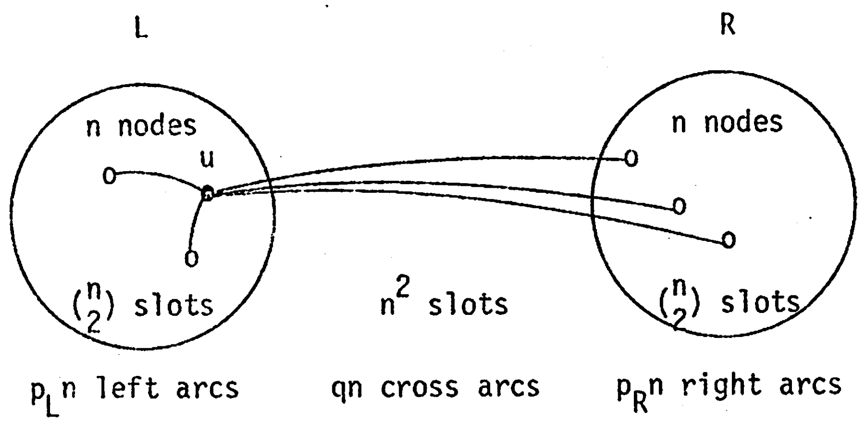


Fig. 4.3

Node u sees n slots crossing to the other side. The number of these actually filled by arcs is determined by a hypergeometric distribution, so

$$\text{Prob}\{\deg_R(u)=j\} = \frac{\binom{qn}{j}\binom{n^2-qn}{n-j}}{\binom{n^2}{n}} . \tag{4.13}$$

(We are choosing from a population of n^2 objects, of which qn are distinguished. We choose n of them.). See [12].

Also node u sees $(n-1)$ slots connected to other nodes in L, out of a total of $\binom{n}{2}$ on that side. $p_L n$ of them are filled with arcs, so

$$\text{Prob}\{\deg_L(u)=j\} = \frac{\binom{p_L N}{j}\binom{\binom{n}{2}-p_L n}{n-1-j}}{\binom{\binom{n}{2}}{n-1}}$$

$$= \frac{\binom{p_L n}{j}\binom{\frac{n^2}{2}-(p_L+\frac{1}{2})n}{n-1-j}}{\binom{\frac{n^2}{2}-\frac{n}{2}}{n-1}} . \tag{4.14}$$

For $j = o(n^{1/2})$ we can use the elementary asymptotic formula

$$\binom{n}{j} \sim \frac{n^j}{j!} \ . \tag{4.15}$$

We also need:

<u>Lemma 4.3</u>. For any fixed c and d, and $j = o(m^{1/2})$

$$\binom{cm^2-dm}{m-j} \sim (cm)^{(m-j)} e^{-(m+\frac{2d+1}{2c})} \sqrt{2\pi m} \ .$$

Lemmas 4.3-4.7 are proved at the end of the chapter. For $j = o(n^{1/2})$ we can plug (4.15) and Lemma 4.3 into (4.13) to get

$$(4.13) \sim \frac{(qn)^j}{j!} n^{(n-j)} e^{-(n+q+\frac{1}{2})} n^{-n} e^{(n+\frac{1}{2})} \frac{\sqrt{2\pi n}}{\sqrt{2\pi n}} \ .$$

Hence for $j = o(n^{1/2})$

$$\text{Prob}\{\deg_R(u) = j\} \sim \frac{q^j}{j!} e^{-q} \ . \tag{4.16}$$

Note that this is just a Poisson distribution with mean q. We similarly plug (4.15) and Lemma 4.3 into (4.14) to get

$$\text{Prob}\{\deg_L(u) = j\} \sim \frac{(2p_L)^j}{j!} e^{-2p_L} \ . \tag{4.17}$$

We state without proof that as $n \to \infty$

$$\text{Prob}\{\deg(u) = j\} \sim \frac{(2s)^j}{j!} e^{-2s} \ . \tag{4.18}$$

The proof technique is the same. Knowing (4.18) makes it easy to prove:

Lemma 4.4.

$$\text{Prob}\{\text{there exists } u \in N \text{ such that } \deg(u) \geq \log n\} = o(1) .$$

Hence for the remainder of this chapter we need never consider cases where the degree of a node exceeds $\log n$.

Knowing the values q and p_L serves to isolate probabilistic dependencies between cross arcs and arcs joining nodes in L (left arcs). Hence the probabilities expressed in (4.16) and (4.17) are independent. This fact allows us to easily determine the probability distribution for $\Delta(u)$:

$$\begin{aligned}\text{Prob}\{\Delta(u)=j\} &= \sum_{i=0}^{n-j} \text{Prob}\{\deg_R(u)=i+j\}\cdot\text{Prob}\{\deg_L(u)=i\} \\ &\sim e^{-(q+2p_L)} q^j \sum_{i=0}^{\lfloor\log n\rfloor} \frac{(q\cdot 2p_L)^i}{i!(i+j)!} + o(1) .\end{aligned}$$

Here we have used Lemma 4.4 to truncate our summation. It assures us that with probability one the remainder is $o(1)$. We have another lemma, which Lemma 4.4 also makes use of:

Lemma 4.5. For any positive constants ℓ and t,

$$n^{\ell} \sum_{i=\lfloor\log n\rfloor}^{\infty} \frac{t^i}{i!} = o(1) .$$

Lemma 4.5 allows us to say

$$\text{Prob}\{\Delta(u)=j\} \sim e^{-(q+2p_L)} q^j \sum_{i=0}^{\infty} \frac{(q\cdot 2p_L)^i}{i!(i+j)!} . \qquad (4.19)$$

From (4.12) we remember that $q \equiv_\varepsilon s$ and $p_L \equiv_\varepsilon \frac{s}{2}$, so we can set

$$q = s + \varepsilon_1$$
$$2p_L = s + \varepsilon_2$$

where ε_1 and ε_2 can be made arbitrarily close to zero. (4.19) becomes

$$e^{-(2s+\varepsilon_1+\varepsilon_2)}(s+\varepsilon_1)^j \sum_{i=0}^{\infty} \frac{(s+\varepsilon_1)^i(s+\varepsilon_2)^i}{i!(i+j)!} .$$

Let $\varepsilon_3 = \sqrt{(s+\varepsilon_1)(s+\varepsilon_2)} - s$. ε_3 can also be made arbitrarily close to zero. Now (4.19) equals

$$e^{-(2s+\varepsilon_1+\varepsilon_2)}\left(\frac{s+\varepsilon_1}{s+\varepsilon_3}\right)^j \sum_{i=0}^{\infty} \frac{(s+\varepsilon_3)^{(2i+j)}}{i!(i+j)!} . \tag{4.20}$$

We can assume that $|\varepsilon_1|$, $|\varepsilon_2|$ and $|\varepsilon_3|$ are $O(n^{-1/3})$ (using Remark 4.1). Then for $j = O(\log n)$ we use (4.11) to find

$$(4.20) = e^{-2s} \sum_{i=0}^{\infty} \frac{(s+\varepsilon_3)^{(2i+j)}}{i!(i+j)!}(1 + o(1)) .$$

Next we notice that

$$(4.20) = e^{-2s} I_j(2s+2\varepsilon_3)(1 + o(1)) \tag{4.21}$$

where

$$I_j(z) \triangleq \sum_{i=0}^{\infty} \frac{(z/2)^{2i+j}}{i!(i+j)!}$$

is a "jth order modified Bessel function of the second kind". There is no convenient expression for this Bessel function, but it turns out that we can find an elementary function which closely approximates $I_j(2s)$ for values of s near one, and is asymptotically equal as s goes to infinity.

Lemma 4.6. Given $\varepsilon > 0$, choose any k_0 large enough so that

$$\left(\frac{e}{4}\right)^{k_0}\sqrt{2\pi k_0} < \frac{\varepsilon}{4} .$$

Then for s sufficiently large, and any $j \in [0,\sqrt{k_0 s}]$ we have

$$\left| I_j(2s)\cdot 2\sqrt{\pi s}e^{-2s} - e^{-\frac{j^2}{4s}}\right| < \varepsilon .$$

This lemma is proved at the end of the chapter.

Lemma 4.6 tells us that

$$I_j(2s) = \frac{e^{2s}}{2\sqrt{\pi s}} e^{-\frac{j^2}{4s}}(1 + o(1)) \tag{4.22}$$

for large values of s. This is a very handy formula for our purposes. Because $I_j(2s)$ is analytic we know that

$$I_j(2s+\varepsilon_3) = I_j(2s)(1 + o(1))$$

so (4.21) becomes, for $j = O(\log n)$

$$\mathrm{Prob}\{\Delta(u) = j\} = e^{-2s} I_j(2s)(1 + o(1)) . \tag{4.23}$$

Expression (4.23) gives us a precise formula for the distribution of $\Delta(u)$, and (4.22) gives us an asymptotic approximation which we will need later on in our analysis.

We will now take the time to find out how many nodes u in L (or R) there are with $\Delta(u) = j$. Define a random variable

$$X_{uj} = \begin{cases} 1 & \text{if } \Delta(u) = j \\ 0 & \text{otherwise} \end{cases}$$

(4.23) tells us the value of EX_{uj}, and hence we know the expected number of u's in L such that $\Delta(u) = j$ is nEX_{uj}. The precise dependency between the variables X_{uj} is hard to compute, but we can easily show that $EX_{uj}X_{vj} \le EX_{uj}EX_{vj}$ for any nodes u and v in L. Then we can apply Lemma 4.2, and conclude that the number of nodes in L with $\Delta = j$ is within εn of

$$ne^{-2s}I_j(2s) \tag{4.24}$$

with probability tending to one as $n \to \infty$ (εn-equivalence).

The probability that $\Delta(u) = j$ for a node u in L depends on the ratio

$$(\text{\# cross arcs}) \div (\text{\# places where left arcs intersect left nodes})\ . \tag{4.25}$$

For the case $j = 0$, the $\text{Prob}\{X_{u0} = 1\}$ is greatest when ratio (4.25) is closest to one. Initially the ratio is $\frac{qn}{2p_L n}$. Knowing that $\Delta(u) = 0$ effectively decreases the value of the numerator and denominator by the amount $\deg(u)$, as seen by another node v. This causes the effective ratio for the node v to diverge from 1 (unless $\frac{qn}{2p_L n} = 1$, which has probability zero). Hence $EX_{u0}X_{v0} \le EX_{u0}EX_{v0}$.

For the case $j > 0$, EX_{uj} is greatest when ratio (4.25) is largest. We use:

<u>Lemma 4.7</u>. Let $d = O(\log n)$ and $q = p_L \pm \varepsilon$ where $\varepsilon \le n^{-1/3}$. Then for $j \ge 1$ and n large

$$\frac{qn - \frac{d+j}{2}}{p_L n - \frac{d-j}{2}} < \frac{qn}{p_L n} .$$

We see that when $d = \deg(u)$ is within reasonable bounds, the ratio that v sees decreases when we know that $\Delta(u) = j$. Hence $EX_{uj}X_{vj} \lesssim EX_{uj}EX_{vj}$.

Now we know the number of nodes u in L with $\Delta(u) = j$. We also need to know the total number of arcs which intercept these nodes. We first calculate

$$\begin{aligned} P(d,j) &\triangleq \mathrm{Prob}\{\deg(u) = d \text{ and } \Delta(u) = j\} \\ &= \mathrm{Prob}\{\Delta(u) = j \mid \deg(u) = d\}\mathrm{Prob}\{\deg(u) = d\} . \qquad (4.26) \end{aligned}$$

We already know the second factor, from (4.18). The first is determined by a hypergeometric distribution:

$$\begin{aligned} \mathrm{Prob}\{\deg_L(u) = \ell \mid \deg(u) = d\} &= \frac{\binom{qn}{\ell}\binom{2p_L n}{d-\ell}}{\binom{(q+2p_L)n}{d}} \\ &\sim \frac{(qn)^j(2p_L n)^{d-j} d!}{\ell!(d-\ell)!((q+2p_L)n)^d} \\ &\quad \text{using (4.15) and requiring } d = O(\log n) \\ &= \frac{(sn)^d}{(2sn)^d}\binom{d}{\ell}\frac{(1+\varepsilon')^j(1+\varepsilon'')^{d-j}}{(1+\frac{1}{2}(\varepsilon'+\varepsilon''))^d} \\ &\quad \text{setting } q = s(1+\varepsilon') \text{ and } 2p_L = s(1+\varepsilon'') \\ &= \binom{d}{\ell}2^{-d}(1+o(1)) \\ &\quad \text{using } \varepsilon' \text{ and } \varepsilon'' = O(n^{-1/3}) \text{ and (4.11).} \end{aligned}$$

We set $\ell = \frac{d-j}{2}$ and get that when $j \le d$ and Parity(j) = Parity(d)

$$(4.26) = \frac{d!2^{-d}}{(\frac{d-j}{2})!(\frac{d+j}{2})!} e^{-2s} \frac{(2s)^d}{d!}(1+o(1))$$

so

$$P(d,j) = e^{-2s}s^d[(\tfrac{d-j}{2})!(\tfrac{d+j}{2})!]^{-1}(1+o(1)) \ . \qquad (4.27)$$

This expression is correct with probability tending to one. Define random variables

$$Y_{ud} = \begin{cases} 1 & \text{if } \Delta(u) = j \text{ and } \deg(u) = d \\ 0 & \text{otherwise} \end{cases}$$

$$X_u = \begin{cases} d & \text{if } Y_{ud} > 0 \text{ for some } d \\ 0 & \text{otherwise} \end{cases}$$

We wish to find $\sum_{u=1}^{n} X_u$. So far we know the distribution for the Y_{ud}'s. Using the same type of argument that we used to derive (4.24) we know for any two nodes u and v, and all d,

$$\begin{aligned}
\text{Prob}\{Y_{ud}Y_{vd} = 1\} &= \text{Prob}\{\Delta(u) = \Delta(v) = j|\deg(u) = \deg(v) = d\} \\
&\quad \cdot\text{Prob}\{\deg(u) = \deg(v) = d\} \\
&\le \text{Prob}\{\Delta(u) = \Delta(v) = j|\deg(u) = \deg(v) = d\} \\
&\quad \cdot\text{Prob}\{\deg(u) = d\}\cdot\text{Prob}\{\deg(v) = d\} \\
&\le \text{Prob}\{\Delta(u) = j|\deg(u) = d\}\text{Prob}\{\Delta(v) = j|\deg(v) = d\} \\
&\quad \cdot\text{Prob}\{\deg(u) = d\}\text{Prob}\{\deg(v) = d\} \\
&= \text{Prob}\{Y_{ud} = 1\}\text{Prob}\{Y_{vd} = 1\} \ .
\end{aligned}$$

Hence $EY_{ud}Y_{vd} \le EY_{ud}EY_{vd}$ and Lemma 4.2 is applicable! Remembering (4.27) we know that

$$\frac{1}{n}\sum_{u=1}^{n} Y_{ud} \equiv_\varepsilon P(d,j) \quad \text{with r.c.} \quad \frac{1}{n\varepsilon^2} \ . \qquad (4.28)$$

Next,

$$\sum_{u=1}^{n} X_n = \sum_{u=1}^{n} \sum_{d=j}^{2n} dY_{ud}$$

$$= \sum_{d=j}^{\lfloor \log n \rfloor} d \sum_{u=1}^{n} Y_{ud} + o(n^{-1}) \qquad \text{(using Lemma 4.4)}$$

$$= \sum_{d=j}^{\lfloor \log n \rfloor} dnP(d,j) + n \sum_{d=j}^{\lfloor \log n \rfloor} d\varepsilon_{\alpha j} + o(n^{-1}) \qquad (4.29)$$

where we have set

$$\varepsilon_{dj} \triangleq \frac{1}{n} \sum_{u=1}^{n} Y_{ud} - P(d,j) .$$

We will require, for $d = j, j+1, \ldots, \lfloor \log n \rfloor$, that $|\varepsilon_{\alpha j}| \leq \log^{-4} n$. This implies that

$$\left| \sum_{d=j}^{\lfloor \log n \rfloor} d\varepsilon_{\alpha j} \right| \leq n \log^{-2} n$$

so (4.29) becomes

$$n \sum_{d=j}^{\lfloor \log n \rfloor} dP(d,j) + O(n \log^{-2} n) . \qquad (4.30)$$

Remark 4.2. The probability that any one of the ε_{dj}'s is greater than $\log^{-4} n$ in absolute value is less than $n^{-1}\log^{9} n = o(1)$, from (4.28).

We set d in (4.27) equal to $2m + j$, plug it into (4.30) and get

$$\frac{1}{n}\sum_{u=1}^{n} X_u \equiv_\varepsilon e^{-2s}\sum_{m=0}(2m+j)s^{(2m+j)}\frac{1}{(m+j)!m!} \qquad (\text{with } \varepsilon = \log^{-2} n)$$

$$= e^{-2s}\Big(2\sum_{m=0}^{\infty} s^{(2m+j)}\frac{m}{(m+j)!m!} + jI_j(2s)\Big)\big(1+o(1)\big)$$

$$= e^{-2s}\left[2s\sum_{\ell=0}^{\infty}\frac{s^{(2\ell+j+1)}}{(\ell+j+1)!\ell!} + jI_j(2s)\right]\big(1+o(1)\big)$$

$$= e^{-2s}\big(2sI_{j+1}(2s) + jI_j(2s)\big)\big(1+o(1)\big). \tag{4.31}$$

We have learned that $\sum_{\substack{u\in L\\ \Delta(u)=j}} \deg(u)$ is within εn of {n times (4.31)}. Let us denote this quantity by t_j. Let w_j denote the number of nodes u with $\Delta(u) = j$, calculated in (4.24). Then

$$\tfrac{1}{2}(t_j + jw_j) = \{\text{the number of cross arcs emanating from nodes } u \text{ such that } \Delta(u) = j\}$$

$$\equiv_{\varepsilon n} ne^{-2s}\big(sI_{j+1}(2s) + jI_j(2s)\big)\ . \tag{4.32}$$

Also,

$$\tfrac{1}{2}(t_j - jw_j) = \{\text{the number of points where left arcs emanate from nodes } u \text{ such that } \Delta(u) = j\}$$

$$\equiv_{\varepsilon n} ne^{-2s}sI_{j+1}(2s)\ . \tag{4.33}$$

We are now getting to the algorithmic part of this proof. Fix a constant b (to be determined precisely later on).

Let

$$B_L \triangleq \{u\in L \mid \Delta(u) \geq b\}$$
$$B_R \triangleq \{u\in R \mid \Delta(u) \geq b\}\ .$$

We are going to swap these two blocks of nodes, so that $L \leftarrow L + B_R - B_L$ and $R \leftarrow R + B_L - B_R$, and observe a decrease in the number of cross arcs for this new 2-partition.

Using Lemma 4.4 and (4.32) we find that the number of arcs spanning B_L and R is

$$cn \equiv_{\varepsilon n} \sum_{j=b}^{\lfloor \log n \rfloor} ne^{-2s}(sI_{j+1}(2s) + jI_j(2s)) \quad .$$

We note that $\varepsilon = O(\sum_{j=b}^{\lfloor \log n \rfloor} \log^{-2} n) = O(\log^{-1} n)$, from (4.30). Also, our rate of convergence may have slowed by a factor of $\log n$. Looking at Remark 4.2 this gives us r.c. $= n^{-1}\log^{10} n$.

We let $\ell \cdot n$ be the number of points where left arcs leave nodes in B_L, so

$$\ell \cdot n \equiv_{\varepsilon n} \sum_{j=b}^{\lfloor \log n \rfloor} ne^{-2s} sI_{j+1}(2s) \ .$$

Note that $\ell \cdot n$ is not the number of left arcs which leave B_L, because many arcs connect two points inside of B_L. In fact, we need to know how many arcs cross from B_L to $L - B_L$. Let us denote this quantity by $\ell_0 n$. Let $c_0 n$ be the number of arcs crossing from B_L to $R - B_R$. Then the decrease in cross arcs when we swap B_L and B_R will be $\equiv_{\varepsilon n} (2c_0 - 2\ell_0)n$. Fortunately we know how to find c_0, knowing c, and how to find ℓ_0, knowing ℓ:

Number the cross arcs leaving B_L $1,2,\ldots,cn$. Let the random variable

$$Z_i = \begin{cases} 1 & \text{if the } i^{th} \text{ cross arc spans } B_L \text{ and } B_R \\ 0 & \text{if the } i^{th} \text{ cross arc spans } B_L \text{ and } R\text{-}B_R \end{cases}$$

We know that $EZ_i \equiv_\varepsilon \frac{cn}{sn}$ because the number of cross arcs leaving B_R is $\equiv_{\varepsilon n} cn$. Also it is clear that for any two cross arcs i and j, $EZ_iZ_j \leq EZ_iEZ_j$. Hence Lemma 4.2 is applicable, and tells us that the number of arcs spanning B_L and B_R is

$$(c-c_0)n \equiv_{\varepsilon n} \sum_{i=1}^{cn} EZ_i \equiv_{\varepsilon n} \frac{c^2n^2}{sn} = \frac{c^2}{s}n$$

so

$$c_0 \equiv_\varepsilon c - \frac{c^2}{s} .$$

Using similar reasoning we find that

$$\ell_0 \equiv_\varepsilon \ell - \frac{\ell^2}{s} .$$

The situation is diagrammed in Fig. 4.4.

The decrease in cross arcs when we swap B_L and B_R is now $\equiv_{\varepsilon n} 2n(c - \ell - \frac{c^2}{s} + \frac{\ell^2}{s})$. Plugging in values for c and ℓ we have

$$c - \ell \equiv_\varepsilon \sum_{j=b}^{\infty} e^{-2s} jI_j(2s) \tag{4.34}$$

and

$$c^2 - \ell^2 \equiv_\varepsilon 2s\Big(\sum_{j=b}^{\infty} e^{-2s} I_{j+1}(2s)\Big)\Big(\sum_{j=b}^{\infty} e^{-2s} jI_j(2s)\Big) + \Big(\sum_{j=b}^{\infty} e^{-2s} jI_j(2s)\Big)^2 . \tag{4.35}$$

As it stands, expressions (4.34) and (4.35) are fairly intractable. We can compute values numerically, but we don't know what the best choice for the value of b would be. For the case when s is large, (4.22) comes to our rescue:

$$\begin{aligned}\sum_{j=b}^{\infty} e^{-2s} jI_j(2s) &= \frac{1}{2\sqrt{\pi s}} \sum_{j=b}^{\infty} je^{-j^2/4s}(1 + o(1)) \\ &= \frac{1}{2\sqrt{\pi s}} \int_b^\infty je^{-j^2/4s} dj (1 + o(1)) \\ &= \sqrt{\frac{s}{\pi}}\, e^{-b^2/4s}(1 + o(1))\end{aligned} \tag{4.36}$$

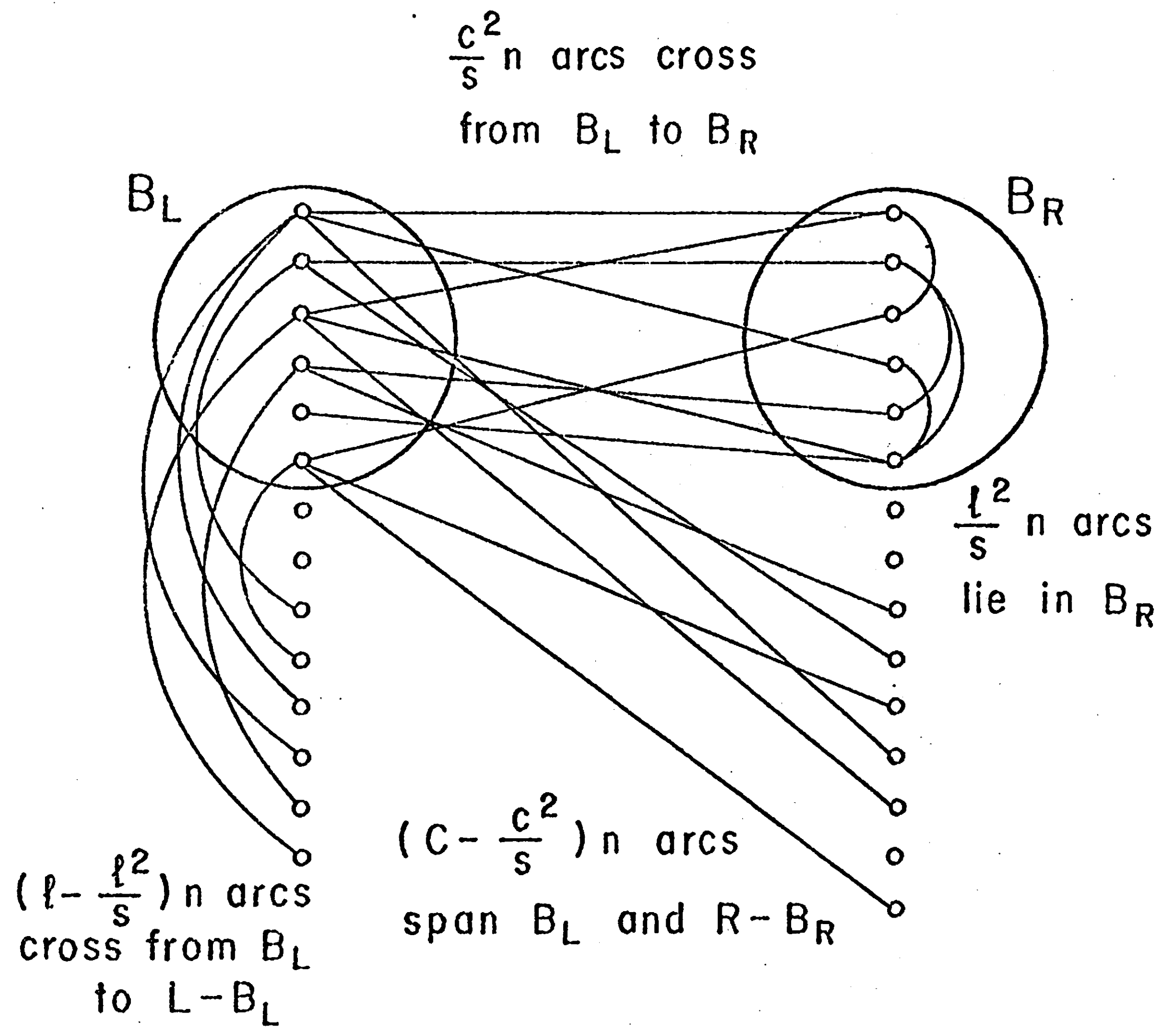

$\frac{c^2}{s}n$ arcs cross from B_L to B_R
B_L
B_R
$\frac{\ell^2}{s}n$ arcs lie in B_R
$(C-\frac{c^2}{s})n$ arcs span B_L and $R-B_R$
$(\ell-\frac{\ell^2}{s})n$ arcs cross from B_L to $L-B_L$

Fig. 4.4

Next we observe, for increasing s, that

$$2s \sum_{j=b}^{\infty} e^{-2s} I_{j+1}(2s) + \sqrt{\frac{s}{\pi}}\, e^{-b^2/4s}$$

$$= s\left(-\frac{1}{\sqrt{\pi s}} \sum_{j=b}^{\infty} e^{-(j+1)^2/4s} dj + \frac{1}{\sqrt{\pi s}}\, e^{-b^2/4s}\right)(1 + o(1))$$

$$= s\, \frac{1}{\sqrt{\pi s}} \int_0^{\infty} e^{-j^2/4s} dj\, (1 + o(1))$$

$$= s\left(1 - \frac{1}{\sqrt{\pi s}} \int_0^{b} e^{-j^2/4s} dj\right)(1 + o(1)) \quad . \qquad (4.37)$$

Insert (4.36) and (4.37) into (4.35) to get

$$\frac{1}{s}(c^2 - \ell^2) \equiv_{\varepsilon} \sqrt{\frac{s}{\pi}}\, e^{-b^2/4s}\left(1 - \frac{1}{\sqrt{\pi s}} \int_0^{b} e^{-j^2/4s} dj\right) \quad . \qquad (4.38)$$

We subtract (4.38) from (4.34) (using (4.36)) to get

$$c - \ell - \frac{1}{s}(c^2 - \ell^2) \equiv_{\varepsilon} \sqrt{\frac{s}{\pi}}\, e^{-b^2/4s}\, \frac{1}{\sqrt{\pi s}} \int_0^{b} e^{-j^2/4s} dj$$

$$= 2\sqrt{\frac{s}{\pi}}\, e^{-b^2/4s}\left(\operatorname{erf}\left(\frac{b}{\sqrt{2s}}\right) - \frac{1}{2}\right) \quad . \qquad (4.39)$$

We can simplify (4.39) by setting $b = t\sqrt{2s}$. Ignoring the leading constant, the function we have left, which we would like to maximize, is

$$e^{-t^2/2}\left(\operatorname{erf}(t) - \frac{1}{2}\right) \quad . \qquad (4.40)$$

We finally break down at this point and use numerical techniques. (4.40) is maximized around the point $t = 1.21$, so the optimal value for b is

$$b = 1.71\sqrt{s} \ .$$

At this point (4.40) has value .211 and (4.39) has value $.238\sqrt{s}$.

We conclude that the decrease in the number of cross arcs after exchanging B_L and B_R is (ignoring roundoff error)

$$\equiv_{\varepsilon n} 2n(.238)\sqrt{s} \ . \tag{4.41}$$

Hence we expect a final result of

$$\equiv_{\varepsilon n} 2ns\left(\frac{1}{2} - \frac{.238}{\sqrt{s}}\right) \text{ cross arcs .}$$

It is unlikely that $|B_L| = |B_R|$, so we have to insert a small patch which insures that L and R are of equal size after the exchange.

First we calculate the expected size of the set B_L:

$$\begin{aligned}
|B_L| &= \sum_{j=b}^{\infty} |\{u \in L \mid \Delta(u) = j\}| \\
&= \sum_{j=b}^{\lfloor \log n \rfloor} |\{u \in L \mid \Delta(u) = j\}| + o(1) \qquad \text{(using Lemma 4.4)} \\
&\equiv_{\varepsilon n} \sum_{j=b}^{\lfloor \log n \rfloor} ne^{-2s} I_j(2s) \ . \qquad (4.42)
\end{aligned}$$

To get (4.42) we used (4.24) and the fact that the sum of errors introduced for each term is $o(n)$ (see the derivation of (4.30)).

We can find $|B_R|$ similarly, and conclude that $|B_L| \equiv_{\varepsilon n} |B_R|$. Suppose $|B_L| - |B|_R = m > 0$. Then to increase R's size to be equal to that of L we move $\frac{m}{2}$ nodes from L to R. We can always choose $\frac{m}{2}$ nodes from L all of which have degree not more than $4s$ (when

$\frac{m}{2} \leq \frac{n}{2}$). Hence we can insure that our patch increases the number of cross arcs by an amount

$$\leq \frac{m}{2}4s < \varepsilon 2sn \quad \text{for any given} \quad \varepsilon > 0 .$$

This leaves our result (4.41) unaffected.

We will note that for the case of large s we can evaluate (4.42) by plugging in (4.22). We find that

$$|B_L| \equiv_{\varepsilon n} \frac{n}{2}(.3174) .$$

When s is large L contains about $\frac{n}{2}$ good nodes initially, so we are moving the top .32 of those nodes.

This concludes our proof of Theorem 4.2. □

Before proving Lemmas 4.3-4.7 we have a comment on our choice of the algorithm for analysis. No reason was given for constructing B_L and B_R from among those good nodes with the largest Δ-values. In fact, the case when B_L and B_R represent randomly chosen subsets of the good nodes has been analyzed. It can be shown that the optimal strategy is to place one half of all of the good left nodes into B_L, and an equal number into B_R. After swapping B_L and B_R a decrease of $\approx 2n(.141)\sqrt{s}$ cross arcs is observed. This is significantly less than the value (4.41), so we conclude that this alternative algorithm is inferior.

Now we set to work on the lemmas.

Lemma 4.3. Fix non-negative real numbers c and d, and assume that $j = o(m^{1/2})$. Then for increasing m

$$\binom{cm^2 - dm}{m - j} \sim (cm)^{(m-j)} e^{-(m+\frac{2d+1}{2c})} \sqrt{2\pi m} .$$

Proof. First consider the product

$$\prod_{i=0}^{m-j-1} \left(1 - \frac{q}{cm} - \frac{i}{cm^2}\right) \tag{4.43}$$

$$\begin{aligned}
(4.43) &\le \left(1 - \frac{q}{cm} - \frac{m-j+1}{2cm^2}\right)^{(m-j-1)} && \text{similar to (4.4)} \\
&= \left(1 - \frac{2q + 1 + O(\frac{1}{m})}{2cm}\right)^{(m-j-1)} && \text{using (4.2)} \\
&\le e^{-\frac{2q+1+O(\frac{1}{m})}{2cm}(m-j-1)} \\
&\sim e^{-\frac{2q+1}{2c}} .
\end{aligned}$$

On the other hand

$$\begin{aligned}
(4.43) &\ge \left(1 - \frac{2q}{cm} - \frac{m-j+1}{cm^2}\right)^{(\frac{m-j}{2})} && \text{similar to (4.4)} \\
&= \left(1 - \frac{2q + 1 + O(\frac{1}{m})}{cm}\right)^{(\frac{m-j}{2})} \\
&\ge e^{-(\frac{2q+1+O(\frac{1}{m})}{cm})(\frac{m-j}{2})(1+O(\frac{1}{m}))} && \text{(using (4.5))} \\
&\sim e^{-(\frac{2q+1}{2c})} .
\end{aligned}$$

Hence

$$(4.43) \sim e^{-(\frac{2q+1}{2c})} . \tag{4.44}$$

Next we observe

$$\begin{aligned}(m-j)! &\sim \left(\frac{m-j}{e}\right)^{(m-j)}\sqrt{2\pi(m-j)} \qquad \text{(Stirling)}\\ &= \left(\frac{m}{e}\right)^{(m-j)}\left(1-\frac{j}{m}\right)^{(m-j)}\sqrt{2\pi(m-1)}\\ &= m^{(m-j)}e^{-m}\sqrt{2\pi m}(1+o(1)) \qquad (4.45)\end{aligned}$$

using (4.2), (4.5), and $j = o(m^{1/2})$. Finally (4.44) and (4.45) give us

$$\begin{aligned}\binom{cm^2 - qm}{m-j} &= \frac{\prod_{i=0}^{m-j-1}(cm^2-qm-i)}{(m-j)!}\\ &\sim (cm)^{(m-j)}e^{-(m+\frac{2d+1}{2c})}\sqrt{2\pi m} \ . \qquad \square\end{aligned}$$

Lemma 4.4. Prob{there exists a node u such that $\deg(u) \geq \log n$} $= o(1)$.

Proof. For all $j \leq 2n-1$, (4.18) is an upper bound on the probability that $\deg(u) = j$. This can be verified by straight calculation, starting from the hypergeometric distribution for $\deg(u)$. Hence

$$\begin{aligned}\text{Prob}\{\deg(u) \geq \log(n)\} &\leq \sum_{\lfloor \log n\rfloor}^{2n-1} e^{-2s}\frac{(2s)^j}{j!}\\ &= o(n^{-2}) \qquad \text{by Lemma 4.5}\end{aligned}$$

Then

$$\begin{aligned}\text{Prob}\{(\exists u)(\deg(u) \geq \log(n))\} &\leq 2n\cdot o(n^{-2})\\ &= o(n^{-1}) \ . \qquad \square\end{aligned}$$

Lemma 4.5. For any positive constants ℓ and t,

$$n^{\ell}\sum_{i=\lfloor \log n\rfloor}^{\infty}\frac{t^i}{i!} = o(1) \ .$$

Proof.

$$n^{\ell} \sum_{i=\lfloor \log n \rfloor}^{\infty} \frac{t^i}{i!} \le n^{\ell} 2 \frac{t^{\lfloor \log n \rfloor}}{\lfloor \log n \rfloor !} \quad \text{when} \quad \lfloor \log n \rfloor > 2t$$

$$\sim n^{\ell} 2 \frac{(et)^{\log n}}{(\log n)^{\log n} \sqrt{\log n}} \qquad \text{(using (4.1))}$$

$$\le n^{(\ell + 1 + \log t - \log \log n)}$$

$$= o(1) \ . \qquad \square$$

Define

$$I_j(z) \triangleq \sum_{i=0}^{\infty} \frac{(\frac{z}{2})^{2i+j}}{i!(i+j)!} \ .$$

$I_j(z)$ is a j^{th} order modified Bessel function.

Lemma 4.6. Given $\varepsilon > 0$, choose any k_0 large enough so that

$$(\frac{e}{4})^{k_0} \sqrt{2\pi k_0} < \frac{\varepsilon}{4} \ . \qquad (4.46)$$

Then for s sufficiently large, and any integer $j \in [0, \sqrt{k_0 s}]$ we have

$$|I_j(2s) 2\sqrt{\pi s}\ e^{-2s} - e^{-j^2/4s}| < \varepsilon \ .$$

We make use of a Hankel expansion, found in [31], which is an asymptotic approximation to $I_j(2s)$ for the case of large s:

$$I_j(2s) \sim \frac{e^{2s}}{2\sqrt{\pi s}} \left[\sum_{i=0}^{j-1} (-)^i \frac{A(i,j)}{(2s)^i} + \gamma_j \right]$$

where $A(0,j) \triangleq 1$, and for $i > 0$

$$A(i,j) \triangleq \prod_{\ell=1}^{i} \frac{4j^2-(2\ell-1)^2}{8\ell} = \frac{(4j^2-1)(4j^2-9)\cdots(4j^2-(2i-1)^2)}{8^i i!} \ .$$

Also,

$$|\gamma_j| \leq 2\chi(j)\exp\left(\frac{\pi(j^2-\frac{1}{4})}{4s}\right)\frac{|A(j,j)|}{(2s)^j} \tag{4.47}$$

where

$$\begin{aligned}\chi(j) &\triangleq \frac{\Gamma(\frac{1}{2}j+1)}{\Gamma(\frac{1}{2}j+\frac{1}{2})} \\ &\sim \sqrt{\tfrac{1}{2}\pi j} \quad .\end{aligned} \tag{4.48}$$

We fix a constant b:

$$b = \begin{cases} 1 & \text{if } \frac{j^2}{s} < \frac{\varepsilon}{4} \\ k_0 & \text{otherwise} \end{cases}$$

and note that in all cases

$$\frac{j^2}{s} \leq b \ . \tag{4.49}$$

We will show that, for sufficiently large s,

$$\text{(i)} \quad \left|\sum_{i=b}^{j-1}(-)^i \frac{A(i,j)}{(2s)^i} + \gamma_j\right| < \frac{\varepsilon}{2}$$

$$\text{(ii)} \quad \left|e^{-j^2/4s} - \sum_{i=0}^{b-1}\frac{(-)^i}{i!}\left(\frac{j^2}{4s}\right)^i\right| < \frac{\varepsilon}{4}$$

$$\text{(iii)} \quad \left|\sum_{i=0}^{b-1}(-)^i \frac{A(i,j)}{(2s)^i} - \sum_{i=0}^{b-1}\frac{(-)^j}{i!}\left(\frac{j^2}{4s}\right)^i\right| < \frac{\varepsilon}{4}$$

Then from (i), (ii) and (iii) we can conclude:

$$\left|\sum_{i=0}^{j-1}(-)^i \frac{A(i,j)}{(2s)^i} + \gamma_j - e^{-j^2/4s}\right| < \varepsilon$$

which proves the lemma.

First we set

$$a_i \triangleq \frac{A(i,j)}{(2s)^i} .$$

Now from (4.47) and (4.49) we have

$$\begin{aligned}
|\gamma_j| &\le 2\chi(j)\exp(\frac{\pi k_0}{4})a_{j-1}(\frac{4j^2-(2j-1)^2}{j-16s}) \\
&\sim \sqrt{2\pi j}\ \exp(\frac{\pi k_0}{4})(\frac{4j-1}{16js})a_{j-1} \qquad \text{(using (4.48))} \\
&= O(s^{-3/4}a_{j-1}) \qquad \text{(becuase } j \le \sqrt{k_0 s}) \\
&= o(a_{j-1}) . \qquad\qquad (4.50)
\end{aligned}$$

We next show that, for $i \in [b,j-1]$, the terms a_i are positive and furthermore that

$$a_i > a_{i+1} . \qquad (4.51)$$

By definition

$$a_i - a_{i+1} = a_i(1 - \frac{4j^2-(2i-1)^2}{16si}) \qquad (4.52)$$

but

$$\begin{aligned}
0 &< \frac{4j^2-(2i-1)^2}{16si} \qquad \text{(because } i \le j-1) \\
&\le \frac{4b}{16b} - \frac{(2i-1)^2}{16si} \qquad \text{(using (4.49) and } b \le i) \\
&= \frac{1}{4} + O(s^{-1/2}) . \qquad\qquad (4.53)
\end{aligned}$$

We see that (4.52) and (4.53) imply (4.51). From (4.50) and (4.51) we know

$$\begin{aligned}\sum_{i=b}^{j-1}(-)^i a_i + \gamma_j &\le a_b + a_{j-1} \\ &\le 2a_b && \text{(from (4.51)} \\ &\le 2\left(\frac{4j^2}{16s}\right)^b (b!)^{-1} \\ &\sim 2\left(\frac{j^2 e}{4bs}\right)^b \sqrt{2\pi b} && \text{(using (4.1))} \\ &\le 2\left(\frac{e}{4}\right)^b \sqrt{2\pi b} && \text{(using (4.49))} \\ &< \frac{\varepsilon}{4}\,. && \text{(from (4.46))}\end{aligned}$$

This proves claim (i). A similar proof shows that

$$\left|\sum_{i=b}^{\infty}\frac{(-)^i}{i!}\left(\frac{j^2}{4s}\right)^i\right| \le \left(\frac{j^2}{4s}\right)^b (b!)^{-1} < \frac{\varepsilon}{4}\,. \tag{4.54}$$

Combining (4.54) with the Taylor expansion

$$e^{-j^2/4s} = \sum_{i=0}^{\infty}\frac{(-)^i}{i!}\left(\frac{j^2}{4s}\right)^i$$

yields claim (ii). Finally, consider the sum

$$\sum_{i=0}^{b-1}\frac{(-)^i}{i!}\left[a_i - \left(\frac{4j^2}{16s}\right)^i\right]\,. \tag{4.55}$$

Each term in (4.55) can be expressed as

$$\frac{(-)^i}{i!}\left[(4j^2)^i + c_{i\,i-1}(4j^2)^{i-1} + \cdots + c_{i0} - (4j^2)^i\right](16s)^{-i}$$

where the $c_{i\ell}$'s are all constants. There are $\le b^2 \le k_0^2$ $c_{i\ell}$'s. so all of them can be bounded above by some constant independent of s. Hence each term in (4.55) is

$$O\left(\frac{j^{(2i-2)}}{s^i}\right) = O(s^{-2}) = o(1) .$$

We conclude that (4.55) $< \frac{\varepsilon}{4}$ for large s. This proves (iii). $\square$

Lemma 4.7. Let $d = O(\log n)$ and $q = p \pm \varepsilon$ where $\varepsilon \le n^{-1/3}$. Then for $j \ge 1$ and large n

$$\frac{qn - \frac{d+j}{2}}{pn - \frac{d-j}{2}} < \frac{q}{p} .$$

Proof. It is clear that if $q = p - \varepsilon$ then $\frac{q}{p} < 1$ and the lemma is true. Hence we assume that $q = p + \varepsilon$.

For large n we have $\frac{1}{d} > \frac{\varepsilon}{p}$ implying

$$\frac{q}{p} = 1 + \frac{\varepsilon}{p} < 1 + \frac{1}{d} \le 1 + \frac{j}{d} = \frac{d+j}{d} < \frac{d+j}{d-j}$$

$$\Rightarrow \frac{d-j}{2pn} < \frac{d+j}{2qn} .$$

Then

$$\frac{qn - \frac{d+j}{2}}{pn - \frac{d-j}{2}} = \frac{q}{p}\left(\frac{1 - \frac{d+j}{2qn}}{1 - \frac{d-j}{2pn}}\right) < \frac{q}{p} . \qquad \square$$

4.4 Empirical Testing of the Block Exchange Algorithm

The proof of Theorem 4.2 suggests an algorithm for improving the 2-partition of a graph which we call the Block Exchange algorithm. It has a very simple program:

Block Exchange Algorithm

Assume a starting 2-partition $\{L,R\}$ of a graph $\Gamma_{2n,N}$.

```
1.  $s \leftarrow \frac{N}{2n}$.  $b \leftarrow 1.71\sqrt{s}$.
2.  $L_0 \leftarrow \{u \in L | \Delta(u) \geq b\}$,  $R_0 \leftarrow \{v \in R | \Delta(v) \geq b\}$.
3.  $L \leftarrow L \cup R_0 - L_0$,  $R \leftarrow R \cup L_0 - R_0$.
4.  $x \leftarrow ||L|-|R||$.  Move $\frac{x}{2}$ randomly-chosen nodes from the
    larger of $L$ and $R$ to the smaller, so that $|L| = |R|$.
```

This algorithm, which we call algorithm B for short, was coded and executed on randomly-generated graphs to see how well its behavior would follow analytical predictions. Runs were made on graphs of 128 nodes and 128s arcs, for $s = 1,2,4,8,16$. The arcs were selected pseudo-randomly. For each value of s 6 graphs were generated, and on each graph algorithm B was tried from 7 different starting 2-partitions. Hence each data point in Figs. 4.5 and 4.6 represents the average of values from 42 different runs of the algorithm (Chapter 5 describes in more detail how the trials were carried out).

First we wished to find out if the predicted value of $b = 1.71\sqrt{s}$ was really the best choice for a threshold. The values $b = 1,2,3,\ldots,10$ were tried (bypassing Step 1), with the result shown in Fig. 4.5. The "X" on each line indicates the location of the point $1.71\sqrt{s}$. We observe that each "X" is fairly close to the lowest point on its curve.

If we let $b \to 0$ in equation (4.39) we obtain the somewhat counter-intuitive result that for large s exchanging all good nodes on both sides ($b = 1$) is expected to produce a partition little better than the original randomly-chosen 2-partition. We see this verified here: for $s = 16$ and $b = 1$ the final partitions had an average of

Performance of Block Exchange Algorithm
(128 nodes, 128s arcs)

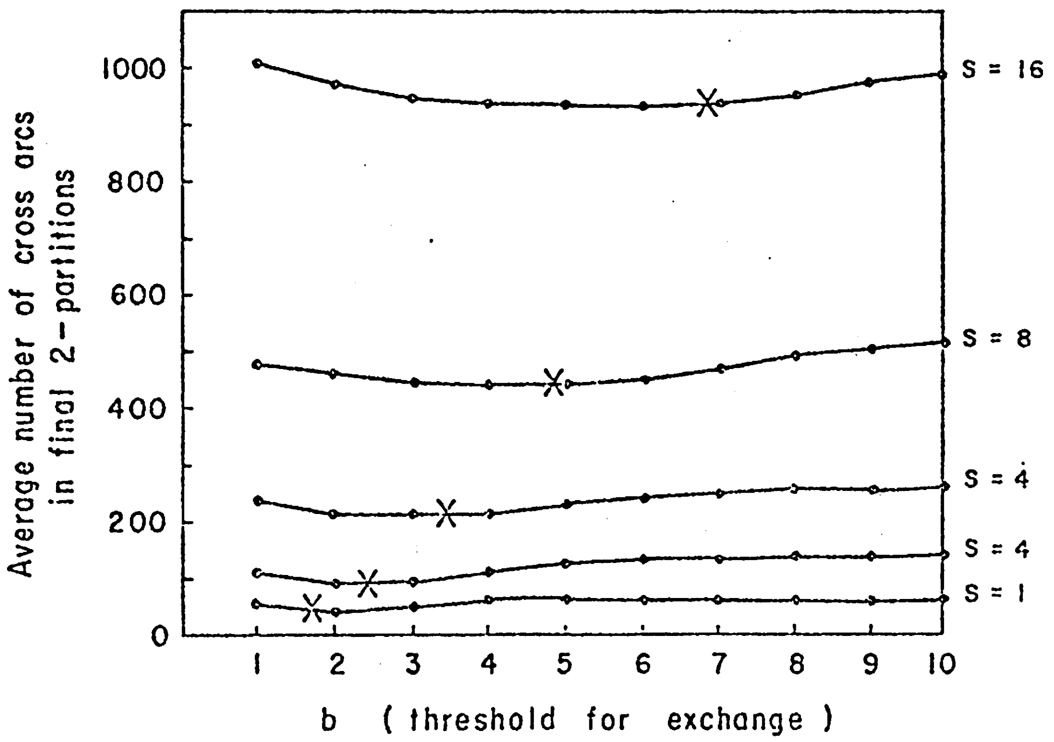


Fig. 4.5

Performance of Block Exchange Algorithm
(2n = 128 nodes, 2ns arcs)

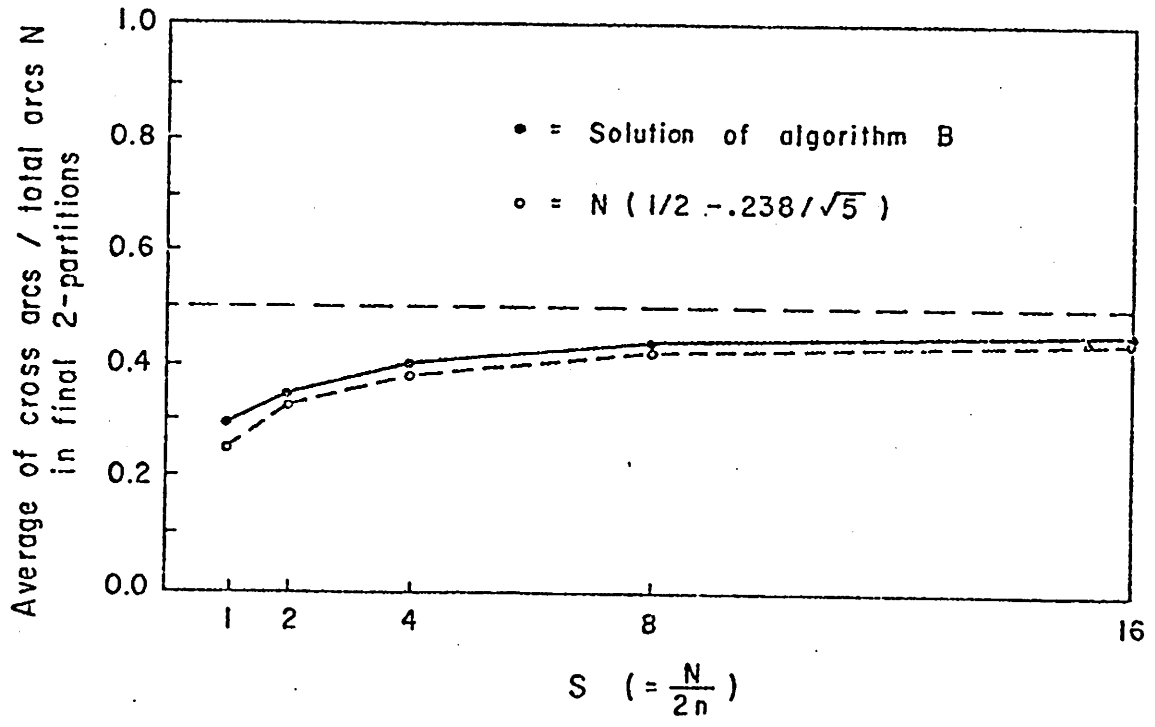


Fig. 4.6

1007 cross arcs, which is 49% of all of the arcs. A random 2-partition is expected to have about 50% of all arcs be cross arcs.

We also checked to see if the average number of cross arcs in a 2-partition produced by algorithm B converged on the value $N(\frac{1}{2} - \frac{.238}{\sqrt{s}})$ predicted analytically. Fig. 4.6 shows that the two quantities are fairly close, and become closer as s increases. This is to be expected, since the quantity $\frac{.238}{\sqrt{s}}$ is only an asymptotic approximation to the function $H(s)$ (defined when Theorem 4.2 was stated), which is the true expected decrease in cross arcs.

CHAPTER 5

A LOOK AT THE 2-PARTITION ALGORITHMS

5.1 Overview

This chapter begins by introducing two new iterative-improvement algorithms for the 2-Partition Problem. The "Lookahead" algorithm is somewhat like algorithm S, but it uses a more sophisticated objective function for determining which nodes are to be exchanged. The "Maxcut Analogue" is an algorithm which is patterned after a one-opting algorithm to solve the Maxcut Problem (see section 2.2) -- it allows the two sets in the 2-partition to become unequal in size. A penalty function which encourages balance between the sets eventually restores equality of the sets.

The heuristics embodied in the Kernighan-Lin, Lookahead, and Maxcut Analogue algorithms can be combined in various ways to create hybrid algorithms. We discuss some of these, and then proceed to the testing grounds, which is the empirical evaluation of all of our algorithms.

The programs for the algorithms were run on randomly-generated graphs. An ordering of the algorithms according to the quality of the 2-partitions produced proved to be invariant over a set of graphs having a large variation in the ratio $\frac{N}{n} = \frac{\text{\# arcs}}{\text{\# nodes}}$. However, when the programs were tried on chain graphs a different ordering was produced. Hence, no one algorithm can claim to be the best for all applications.

We show how to implement the algorithms so that most of them run in linear time, and then compare the relative speeds of the algorithms.

5.2 The Lookahead Algorithm

When a node w is moved from one side to the other of a 2-partition there is a decrease $\Delta(w)$ in the number of cross arcs. Given a 2-partition $\{L,R\}$, define

$$P = P(L,R) \triangleq \sum_{\substack{w \in L \cup R \\ \Delta(w)>0}} \Delta(w) .$$

A large value for P indicates the existence of a relatively large number of good nodes and/or the existence of good nodes with large Δ-values. We expect that the bigger P is the greater will be the decrease in cross arcs effected by a one-opting algorithm. Hence P may be considered an indicator of a 2-partition's potential for improvement. When choosing a pair of nodes for exchange it would seem desirable to select nodes whose exchange would minimize the decrease in potential P. This is the basic idea behind the algorithm we call the Lookahead algorithm.

Given a 2-partition $\{L,R\}$ let $u \in L$, $v \in R$, and define

$$\pi(u,v) \triangleq P(L \cup \{v\} - \{u\},\ R \cup \{u\} - \{v\}) - P(L,R) .$$

$\pi(u,v)$ is the change in potential when nodes u and v are exchanged. Next, fix constants c_1 and c_2 and define

$$\lambda(u,v) = \lambda(u,v,c_1,c_2) \triangleq c_1\delta(u,v) + c_2\pi(u,v) .$$

This will be our evaluation function in the Lookahead algorithm. ($\delta(\ )$ was defined in section 3.2).

If we set $c_1 = 3$, $c_2 = 1$, and substitute $\lambda(u,v)$ for $\delta(u,v)$ in algorithm S, the resulting algorithm has been shown by simulations to produce 2-partitions which are significantly better than those produced by algorithm S. However, the Lookahead algorithm incorporates another modification which improves its performance even further. We present algorithm L, dubbed the Lookahead algorithm because it takes into account side effects of an exchange before selecting the nodes for that exchange:

Algorithm L

Assume a starting partition $\{L,R\}$. Select non-negative integers c_1, c_2, and θ.

```
1.  If there exists no pair of nodes $u, v$ with $u \in L$, $v \in R$,
    and $\lambda(u,v) > 0$ go to Step 5.
2.  Select $u \in L$, $v \in R$ to maximize $\lambda(u,v)$.
3.  $L \leftarrow L \cup \{v\} - \{u\}$,  $R \leftarrow R \cup \{u\} - \{v\}$.
4.  Go to Step 1.
5.  $c_1 \leftarrow c_1 + 1$.
6.  If $c_1 \leq \theta \cdot c_2$ go to Step 1.
7.  Stop.
```

A few comments are in order regarding the choice of the parameters c_1, c_2, and θ. When c_1 and c_2 are approximately equal the algorithm will often choose exchanges which increase both the potential P and the number of cross arcs. As c_1 increases the evaluation function $\lambda(u,v)$ behaves more and more like $\delta(u,v)$, so exchanges which increase the number of cross arcs are seldom chosen. Hence the idea of the algorithm is to increase the 2-partition's potential in the early stages, and then take advantage of the abundance of large Δ-values later on. Experimentation indicates that choosing $c_1 = 2$, $c_2 = 1$

produces good results. It also indicates that no improvement occurs in the later passes beyond the point where c_1 is four times bigger than c_2. Hence $\theta = 4$ is a good choice.

It is not immediately obvious that algorithm L terminates for all choices of c_1 and c_2. Let us define a "pass" to be one of the intervals during which c_1's value is unchanged.

<u>Claim 5.1</u>. During one pass of algorithm L, with c_1 and c_2 chosen to be non-negative integers, the number of exchanges never exceeds $(c_1+2c_2)N$, where N is the number of arcs in the graph on which the algorithm is operating.

<u>Proof</u>. Suppose x exchanges occur during a pass. Let $u^{(j)}$ and $v^{(j)}$ be the nodes exchanged during the jth exchange, for $1 \le j \le x$. Let

$$S_\delta = \sum_{j=1}^{x} \delta(u^{(j)},v^{(j)}) \quad \text{and}$$
$$S_\pi = \sum_{j=1}^{x} \pi(u^{(j)},v^{(j)}) \; .$$

Because c_1 and c_2 are integers we know that for all j

$$\lambda(u^{(j)},v^{(j)}) = c_1\delta(u^{(j)},v^{(j)}) + c_2\pi(u^{(j)},v^{(j)}) \ge 1 \; .$$

Hence we know that

$$c_1S_\delta + c_2S_\pi \ge x \; .$$

The total decrease in cross arcs is at most N during this pass, and the change in potential is at most $2N$, so

$$S_\delta \le N \quad \text{and} \quad S_\pi \le 2N$$

implying

$$x \le (c_1+2c_2)N . \qquad \square$$

5.3 The Maxcut Algorithm

Our next algorithm was inspired by examining a one-opting algorithm for the Simple Maxcut Problem. Suppose we wish to find a minimal 2-partition of a graph $G = (N,A)$. Let $G^c = (N,A^c)$ be the complementary graph: $A^c = \{(i,j)|(i,j)\notin A\}$. If $\{L,R\}$ is a partition of G^c such that $\#(L;R)$ is maximal, and if $|L| = |R|$ as well, then $\{L,R\}$ is a minimal 2-partition of G. If $|L|$ and $|R|$ are close, then we can easily patch things up to get a near-optimal 2-partition of G.

With respect to a partition $\{L,R\}$ (possibly $|L| \neq |R|$) and the graph G^c, define for each node $w \in N$

$\mu^c(w)$ = "the increase in cross arcs if w is moved to the 'other' set" .

The one-opting algorithm we have in mind for the Maxcut Problem finds a node w to maximize $\mu^c(w)$, moves it to the other set if $\mu^c(w) > 0$, and repeats this sequence until $\mu^c(w) \le 0$ for all w.

Now we turn to the graph G. With respect to G and the same partition $\{L,R\}$, fix coefficients d_1 and d_2 and define for each node w

$$\mu(w) = \mu(w,d_1,d_2) \triangleq \begin{cases} d_1(|R|-|L|-1)+d_2\Delta(w) & \text{if } w \in L \\ d_1(|L|-|R|-1)+d_2\Delta(w) & \text{if } w \in R . \end{cases}$$

If $d_1 = d_2 = 1$ then $\mu(w) = \mu^c(w)$. Our next algorithm, without Steps 5, 6, 7 and with $d_1 = d_2 = 1$, is the analogue of the Maxcut one-opting algorithm.

<u>Algorithm M</u>

Assume a starting partition $\{L,R\}$ and fix positive numbers d_1, d_2, and τ.

```
1.  If there exists no $w \in N$ such that $\mu(w) > 0$ go to Step 5.
2.  Choose $w \in N$ to maximize $\mu(w)$.
3.  If $w \in L$ then $L \leftarrow L - \{w\}$, $R \leftarrow R \cup \{w\}$
    else $L \leftarrow L \cup \{w\}$, $R \leftarrow R - \{w\}$.
4.  Go to Step 1.
5.  If $|L| = |R|$ stop.
6.  $c_1 \leftarrow c_1 + \tau$.
7.  Go to Step 1.
```

As with algorithm L we need to prove that algorithm M always terminates. When $d_1 = d_2 = 1$ we know that with respect to G^c the partition $\{L,R\}$ gains at least one cross arc for every iteration of Steps 1, 2, 3, 4. Hence we are guaranteed in this case to reach Step 5 sooner or later. A proof similar to that for Claim 5.1 shows that the cycle of Steps 1-4 terminates for every choice of positive d_1 and d_2. We have proved that algorithm M terminates if we can show that Step 6 is always executed finitely many times.

The term $d_1(|L|-|R|-1)$ or $d_1(|R|-|L|-1)$ in $\mu(\ )$ acts as a penalty function. As d_1 grows the nodes have an increased tendency to migrate towards the smaller of the two sets L and R. If

$d_1 > 2nd_2$ the nodes are guaranteed to move so as to balance the two sets in the next cycle of Steps 1-4. Hence for $\tau > 0$ Step 6 is executed finitely many times.

The best choice for the parameters d_1, d_2, and τ was determined empirically. We first tried $d_1 = d_2 = 1$, with disappointing results. The 2-partitions produced were not as good as those produced by algorithm S. But when d_2 was increased the algorithm's performance improved significantly. $d_1 = 1$, $d_2 = 6$ turned out to be a good choice. Surprisingly, even for this choice of d_1 and d_2 the intermediate partitions produced by algorithm M as Step 5 was first reached were often perfectly balanced, and $||L|-|R||$ was seldom over 6. Hence the choice of τ is usually not critical -- $\tau = d_2$ is a reasonable choice.

When the ratio $\frac{N}{2n} = \frac{\#\text{ arcs}}{\#\text{ nodes}}$ is high a phenomenon we call an <u>avalanche</u> sometimes occurs. In an avalanche the high average degree of the nodes has caused the term $d_2\Delta(\ )$ to dominate the function $\mu(\ )$ to the extent that all of the nodes migrate to one set or the other. Balance is eventually restored, since τ keeps increasing the value of d_1, but the mass migration in an avalanche represents a lot of wasted moves. This situation can be avoided either by decreasing the value of d_2 for large values of $\frac{N}{2n}$ (say when $\frac{N}{2n} \geq 12$), or inserting a procedure before Step 3 which would cause a jump to Step 6 if the imbalance is about to exceed a specified threshold.

5.4 Hybrid Algorithms

Here we discuss ways in which algorithms K, L, and M can be modified or combined to produce more powerful 2-partition algorithms.

A very simple change to algorithm K has been shown to improve its performance: Algorithm K (described in section 3.5) stops if no intermediate partition generated in Steps 3-6 was better than the one it already had at Step 2. Instead of stopping (at Step 7) the algorithm can be modified to try Steps 2-6 several more times, hoping to find a different sequence of exchanges which will result in a still better 2-partition.

It is essential that a mechanism is put in so that different sequences of exchanges are possible. Suppose algorithm K has been implemented with a queue structure such that the entries for $\Delta = 0$ nodes are linked together, the entries for $\Delta = -1$ nodes are together, and so on. If each time that the queue is constructed the entries for nodes with the same Δ-values are strung together in a different order, then they will be accessed in a different order at Step 4. This provides the random element which allows the extra passes to sometimes find a better 2-partition.

This modification was incorporated in the algorithm we refer to as algorithm Kx. Our version makes up to 5 extra passes through Steps 2-6 whenever the algorithm is blocked at Step 7. The results are discussed in the next section.

Now recall the objective function $\lambda(\ )$ which is used by algorithm L (section 5.2). It can be substituted directly for $\delta(\ )$ in algorithm K. Let us suppose that Step 4 of algorithm K is modified to read

4. Select $u \in L-L_0$ and $v \in R-R_0$ to maximize $\lambda(u,v)$.

The parameters c_1 and c_2 which define $\lambda(\ )$ have to be selected. A version which works fairly well sets $c_1 = 3$, $c_2 = 1$.

Algorithm L was improved when the parameters c_1 and c_2 were not fixed, but instead c_1 ran through the values 2, 3, 4 while c_2 equalled one. We can do a similar thing here. Change Step 7 in algorithm K to read

```
7.  _If_ k = 0 _then_
      _begin_ $c_1 \leftarrow c_1+1$;
         _if_ $c_1 \le \theta \cdot c_2$ _then_ go to Step 2
         _else_ stop
      _end_
```

Experimentation has shown that the choice $c_1 = 2$ (initially), $c_2 = 1$ and $\theta = 3$ works well. This version will be referred to as algorithm K/L (algorithm K with lookahead).

Algorithm M can also be increased in power by adding the lookahead function. However, algorithm L exchanges pairs of nodes, while algorithm M moves nodes one at a time, so we need a function $\Lambda(\ )$ similar to $\lambda(\ )$ which is defined for individual nodes. For each node $w \in S$ ($S = L$ or $S = R$) define

$$\pi'(w) \triangleq P(S - \{w\},\ \bar{S} \cup \{w\}) - P(S,\bar{S})$$

(P was defined in section 5.3). Now define

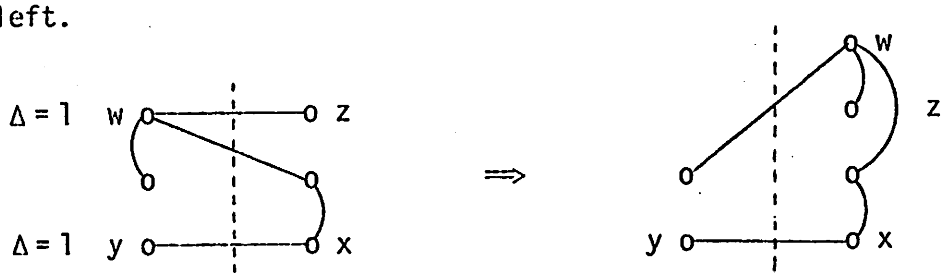
$$\Lambda(w) = \Lambda(w,c_1,c_2) \triangleq c_1\Delta(w) + c_2\pi'(w)\ .$$

Algorithm M uses the objective function $\mu(w)$, which has embedded in its definition the term "$d_2\Delta(w)$". Let us substitute $d_2\Lambda(w)$ for this term. We call this modified algorithm <u>algorithm M/L</u>. The choice $d_1 = 1$, $d_2 = 2$, $c_1 = 3$, $c_2 = 1$, $\tau = 5$ works well, and is used for our simulations in the next section.

5.5 Empirical Evaluation of the 2-Partition Algorithms

Algorithms R, S, K, L, M, M/L, Kx, and K/L have all been programmed and executed on a large number of graphs. This section presents the findings -- and discusses what they imply.

We must first make a disclaimer. Most of the algorithms mentioned progress by repeatedly exchanging pairs of nodes. None of our programs did this. Instead, all algorithms except M and M/L alternately select a node from the set L and move it to R, and then select a node from R and move it to L. Usually the result will be the same no matter which method of exchange is used, but it can happen that our method of exchange will end up exchanging a pair of nodes u, v such that $\delta(u,v)$ is not maximal (or such that $\lambda(u,v)$ is not maximal if look-ahead is being used). In the left graph below, a decrease of two cross arcs occurs if nodes y and z are exchanged. If however, our algorithm chooses to move node w to the right instead of y (as shown in the right graph), its best subsequent move is to move x to the left.



The net decrease is then only one cross arc.

For the algorithms where $\delta(u,v)$ is defined to be the objective function, our implementations choose $u \in L$ such that $\Delta(u) = \max_{w \in L} \{\Delta(w)\}$, move u to R, and then choose $v \in R$, with $\Delta(v) = \max_{w \in R} \{\Delta(w)\}$, and move v. Where the lookahead function is used, the function $\Lambda(u)$ (defined for algorithm M/L in the last section) is used in place of $\lambda(u,v)$.

There are several reasons why our method of selecting nodes for exchange is to be preferred. First of all, it is simpler. $\Delta(u)$ is easier to compute than $\delta(u,v)$, and $\Lambda(u)$ is much easier to compute than $\lambda(u,v)$. The most reasonable way to determine the pair u, v which maximizes $\delta(u,v)$ is to first calculate $\Delta(w)$ for all $w \in L \cup R$ and then compute $\delta(u,v)$ for those $u \in L$ and $v \in R$ which have the largest Δ-values. Our method does away with the second step of this procedure. The same argument applies for algorithms using the lookahead function. Hence, our method is not only simpler, but faster as well. Finally, the results of the trials indicate that algorithms which sometimes choose to exchange nodes whose objective function values are not quite maximal exhibit little if any decrease in the quality of their final 2-partitions.

Our first set of runs was designed to imitate the random graph model described in Chapter 4. Each graph had 128 nodes, labelled $1,2,\ldots,128$. The ratio s of arcs to nodes was selected, and then pseudo-random pairs of labels were generated and inserted as arcs until the total reached $128 \cdot s$. On each graph an algorithm was tried from 7 different starting 2-partitions. Because the labels have no meaning attached to them, the partition $\{\{1,2,\ldots,64\}\{65,66,\ldots,128\}\}$ can be considered "randomly-chosen", and was the first starting partition. In general the i^{th} starting partition was formed by placing the first 2^{7-i} nodes in L, the second 2^{7-i} in R, the third 2^{7-i} in L, and so on. For example, the seventh partition was $\{\{1,3,5,\ldots,127\}, \{2,4,6,\ldots,128\}\}$. For any two of our starting 2-partitions $\{L,R\}$ and $\{L',R'\}$, $|L \cap L'| = \frac{1}{2}|L|$, so they may be regarded as "far apart".

For each value of s, six different graphs were generated, and all of the algorithms were tried on those graphs from the 7 different starting positions. The number of cross arcs in each final 2-partition was divided by $128 \cdot s$ to obtain the fraction of total arcs which were cross arcs. Each data point in Fig. 5.1 is the average of the 42 runs made by that algorithm for that value of s. The variance was observed to be quite small. Several runs were made on graphs of 256 or 512 nodes, with s equal to 1 or 2. The ratios of cross arcs to total arcs were essentially the same as for the 128-node graphs.

We now turn our attention to Fig. 5.1, which contains the results for our algorithms on randomly-generated graphs. Some of the curves have been merged wherever they were extremely close together. For example, algorithms Kx and K/L were nearly identical in performance, while bettering algorithm K by only around .005 on the denser graphs.

The probabilistic lower and upper bounds, predicted in Chapter 4 to bracket the solutions obtained by our algorithms (with probability going to one as $n \to \infty$), are observed to do just that. In general, the slower, more complex algorithms found better 2-partitions. On the graphs with higher arc densities ($s \geq 8$) however, the percentage difference between the best and worst partitions is so slight (less than 8%) that one of the simple, fast algorithms like S or M would

Performance of 2-Partition Algorithms on Random Graphs

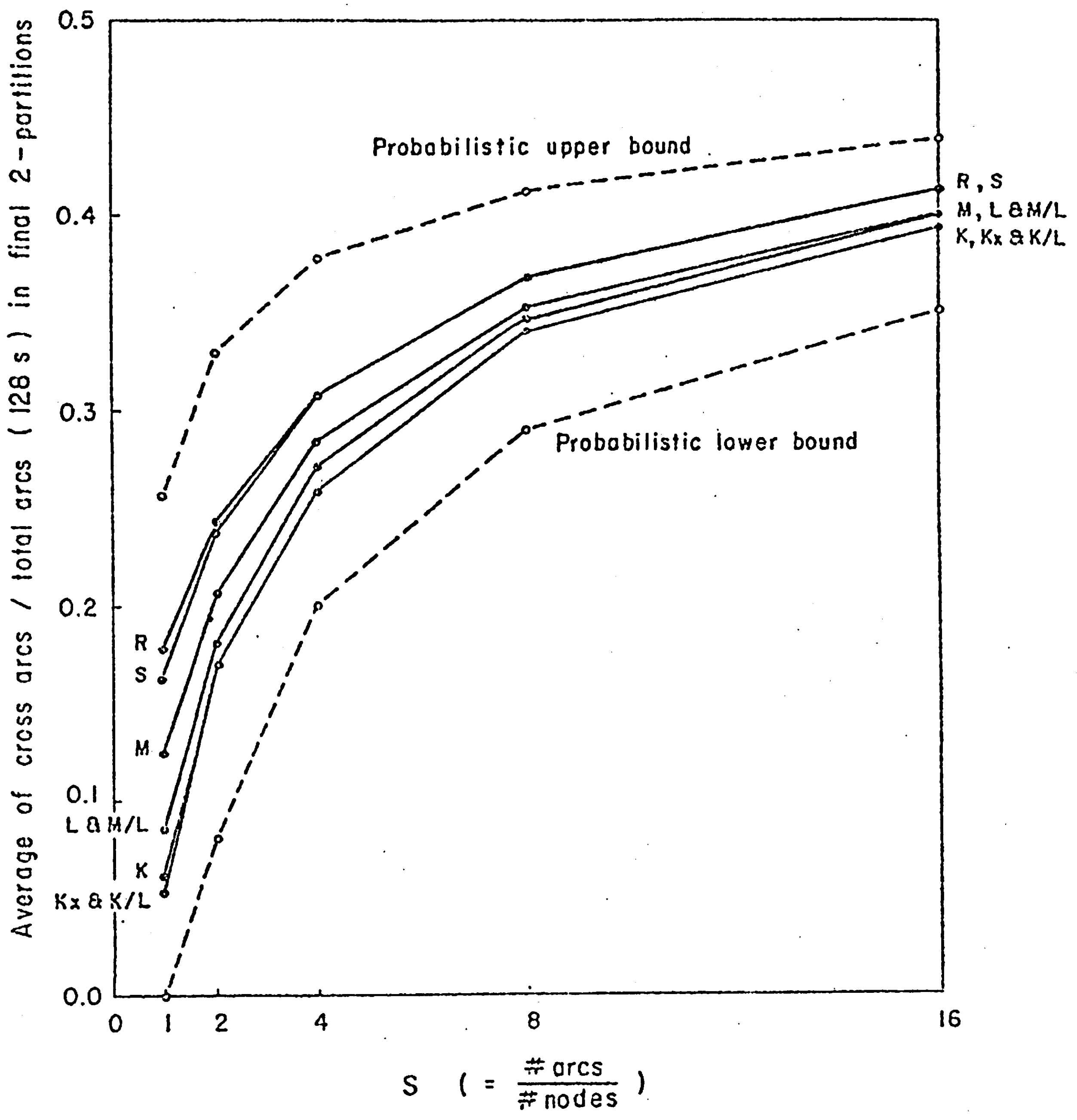


Fig. 5.1

generally be the most practical choice. On the sparsest graphs (s = 1) the differences are significant, and when faced with a real-life partitioning problem one might wish to invest in the additional computer time necessary to run a more complex algorithm. The next section sheds some light on how fast the various algorithms run.

Before making any conclusions it is instructive to look at Fig. 5.2. The relationship (the ordering by quality of solutions found) of the various algorithms changes markedly on the chain graphs. In particular, we see that using the lookahead function has a much more positive effect here than for the random graph simulations. This disparity in the relative performance of the algorithms on different types of graphs cautions us that the determination of the most appropriate algorithm for a specific graph partition problem cannot usually be made a priori.

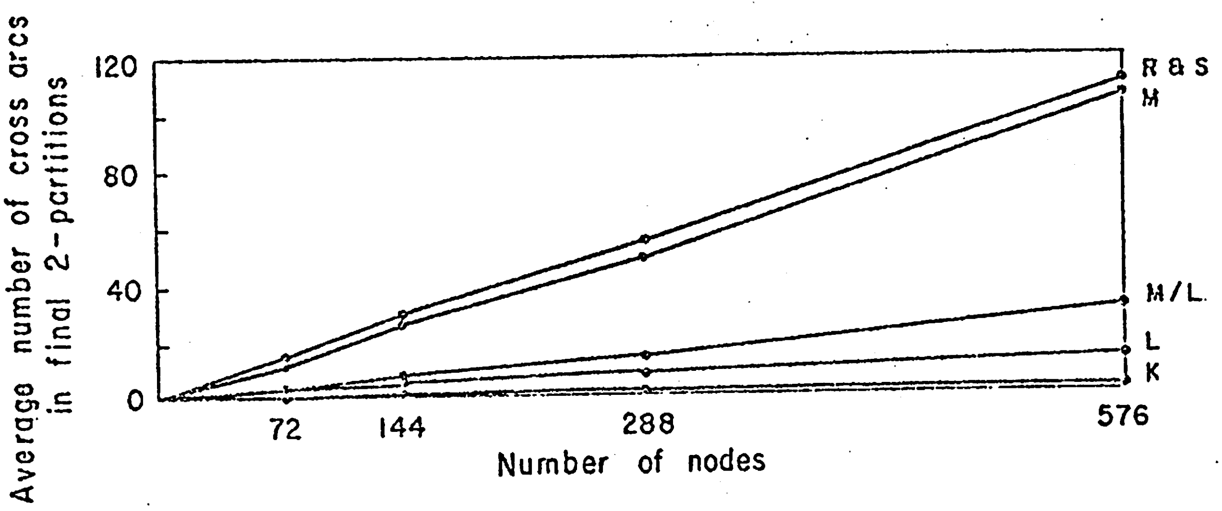
5.6 Implementation and Running Times

We indicated in section 5.5 that our implementation of the 2-partition algorithms uses a non-standard procedure to perform the pairwise exchange of nodes. It also employs a novel type of data structure from which it selects the nodes to be exchanged. These two decisions have resulted in an implementation which is both flexible and fast -- all of our algorithms make use of the same data structures, and most of them have an average running time which is linear in the size of the graph.

We will now describe our implementation, and at the same time derive an estimate of the expected running times for most of the

Results for 2-Partition Algorithms

Performance on 0-Leaf Chain Graphs



Performance on 2-Leaf Chain Graphs

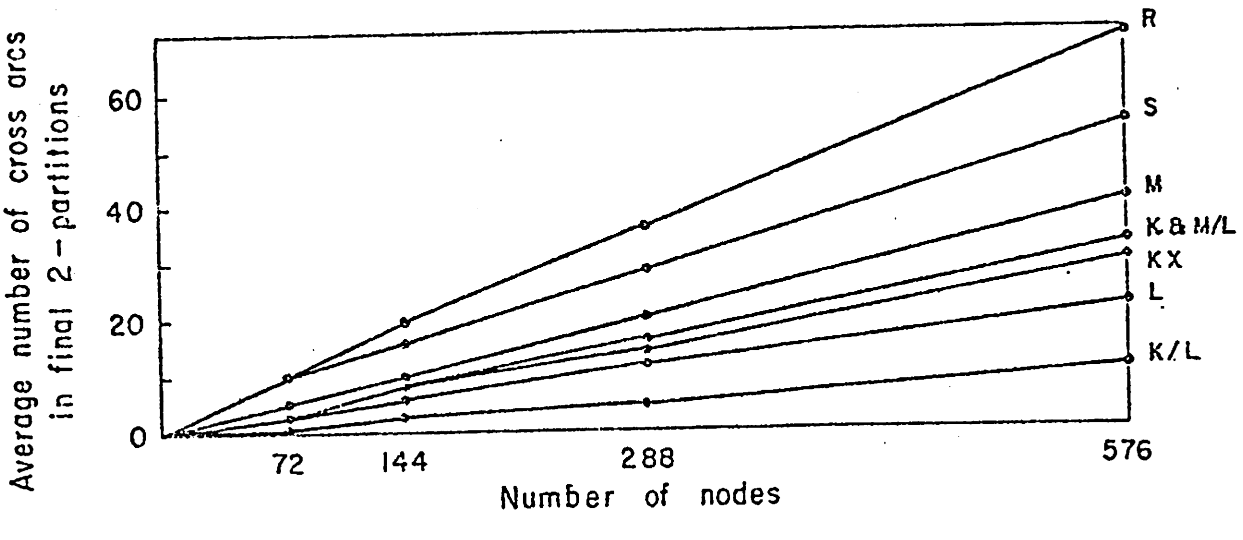


Fig. 5.2

2-partition algorithms. Let us call the operation of moving a node from one set to the other a move (a pairwise exchange involves two moves). It is essential that we have an idea of how many moves an algorithm is likely to make as it operates on a 2n-node graph.

It has been observed empirically that the number of moves made by each of the algorithms (except for K, K/L, and Kx) never exceeds $2n$, and is generally a fraction like $\frac{2n}{3}$. Algorithm K makes about $4\cdot(2n)$ moves on random graphs. Hence, we feel safe in making

Assumption 5.1. On the average the number of moves made by any of our algorithms except for Kx and K/L is $O(n)$.

(Algorithms Kx and K/L probably also use $O(n)$ moves). Proving that an algorithm like algorithm S uses an average of $O(n)$ moves is an interesting open problem. We conjecture that in fact algorithm S uses $O(n)$ moves in the worst case.

Remembering that A is the set of arcs in our graphs, define

$$A(S) \triangleq \{j \mid (i,j) \in A;\ i \in S\}$$

$A(S)$ is the set of nodes adjacent to nodes in the set S. When a node u is moved by a 2-partition algorithm, its Δ-value and the Δ-values of all nodes in $A(\{u\})$ are changed. If we are using the lookahead function when we move node u then the value of $\Lambda(v)$ must be updated for all nodes v in $AA = A(\{u\}) \cup A(A(\{u\}))$. It is straightforward to update the Δ-values for all nodes in $A(\{u\})$ in time $O(|A(\{u\})|)$. In our implementations using lookahead, the record for a node w contains both the value of $\Delta(w)$ and the value of

$\Lambda(w)$. This information can be used to update the Λ-values of the nodes in AA in time $O(|AA|)$.

We wish to count the total number of nodes whose objective functions must be updated during the course of an algorithm's execution. If we knew that "no single node is moved more than a constant number of times, say c times", then we could prove that for algorithms using the objective function $\Delta(w)$ not more than $c \cdot 2 \cdot n \cdot s$ updates will occur. However, that knowledge would also prove Assumption 5.1, which we don't know how to prove. Hence, based on the knowledge that the average degree of a node is $2s$, we will make

Assumption 5.2. Suppose one of the 2-partition algorithms is executed. Let d_i = "the degree of the i^{th} node moved", and $d_i^{(2)}$ = "the number of nodes a distance 2 from the i^{th} node moved". Let x be the actual number of nodes moved. Then

$$\sum_{i=1}^{n} d_i = O(ns) \quad \text{and} \quad \sum_{i=1}^{x} d_i^{(2)} = O(ns^2) .$$

Now we will start counting the number of operations made by the algorithms. Since we repeatedly wish to select the node from L (or R) with the highest objective function value, we need two priority queues -- one containing entries for nodes in L, and one with entries for nodes in R. Our algorithm first calculates the Δ-values for all nodes, taking time $O(ns)$. Algorithms S, M, and K fill the priority queues at the same time (taking time $O(n)$ if heaps are used, $O(n \log n)$ if the nodes are sorted). If lookahead is used we next calculate $\Lambda(u)$ for each node u, and place u's entry in the appropriate queue. Calculating the Λ-values also takes time $O(ns)$. (If we are using algorithm M or M/L, the node with the highest μ-value will always be either at the top of the left queue or at the top of the right queue -- depending on the value of $|L| - |R|$.)

Most of the remaining time is spent repeating the basic sequence which consists of selecting a node, moving it, and updating the queue entries of nodes affected by the move.

If we are using a heap or sorted list to maintain each priority queue the time to select a node is $O(\log n)$. Moving a node takes constant time. If the queue entries are ordered by their Δ-values then under Assumption 5.2 updating takes time $O(ns \log n)$. If lookahead is used updating is more expensive, taking time $O(ns^2 \log n)$.

Assumption 5.1 implies that the basic sequence is repeated $O(n)$ times, so that the time to update dominates the other terms in an expression of total tunning time. (Algorithm K does some extra bookkeeping in addition to the basic sequence, but under Assumption 5.1 it can be done in time $O(n)$. We are choosing to neglect algorithms Kx and K/L in the rest of our discussion.)

If we could eliminate the factor $(\log n)$ which appears in the running time we would have algorithms which run in time $O(ns)$ (or $O(ns^2)$ with lookahead). We can do this by using a version of address calculation (bucket sorting, see [18]) to build faster priority queues for which most basic queue operations take constant time.

Let HIGH and LOW stand for integers, with HIGH > 0 and LOW $\leq$ 0. Our priority queue consists of (HIGH - LOW + 1) doubly-linked lists, one for each of the integers in the interval [LOW,HIGH]. Suppose node u's objective function value is "t" if $t \geq$ HIGH, node u is put on the front of HIGH's list. If $t \leq$ LOW, u is put on LOW's list, and otherwise u is put on the intermediate list corresponding to the integer t. A pointer, TOP, always points to the highest-valued non-empty list.

During a basic sequence the node selected is the first one on the list pointed to by TOP. This takes constant time. After the node is moved, some nodes have had their objective values changed. Each of these nodes can be relocated to the front of the appropriate new list in constant time. Hence the only operation on our queue which takes more than constant time is the adjustment of the pointer TOP, which can take on the order of (HIGH-LOW) steps in the worst case. We shall decree that the quantity $\text{HIGH} - \text{LOW} = O(s)$, and can then conclude that the total running time for our algorithms under Assumptions 5.1 and 5.2, and using our priority queue, is $O(ns)$ (or $O(ns^2)$ if we are using lookahead).

The big question which remains is, "What have we sacrificed by using a priority queue which does not discriminate between nodes whose objective functions equal HIGH or better, or between those with values less than or equal to LOW?" We would like to point out first that the average Δ-value of a node, given that it is positive, is $O(\sqrt{s})$. This was proved in Theorem 4.2. Hence we would expect that if HIGH is some suitable constant times s, then few nodes' Δ-values will exceed HIGH. The expected value of $\Lambda(w)$ in the calculation of lookahead functions has not been calculated, but is clearly $O(s)$.

If algorithms R, S, or L are being implemented then LOW is set equal to zero, because they ignore all nodes with $\Delta < 1$ (a more efficient implementation would not even put such nodes in the queue). Algorithms M, M/L, and K move nodes with non-positive Δ-values, so that LOW should be negative, but since the algorithms rarely move nodes with large negative Δ-values, LOW doesn't have to be very negative.

Hence we have a plausible argument that the algorithms should be able to work well with suitably chosen values for HIGH and LOW such that $(\text{HIGH} - \text{LOW}) = O(s)$. Empirical tests were made to test this hypothesis. Let us say that when a node is moved whose objective function exceeds HIGH that an <u>overflow</u> has been observed, and when one is moved whose value is less than LOW an <u>underflow</u> has occurred. On our graphs on 128 nodes, with $s = 1,2,4,8,16$ overflow occurred very infrequently when algorithms S, L, M, M/L, and K had HIGH set as low as 10, and the partitions produced were just as good as those produced when HIGH was large enough that there were no overflows at all. With HIGH set equal to 5 algorithms S, M, and K performed as well as before, and algorithms L and M/L were a little poorer only when s was 8 or bigger. Note that if algorithm S is implemented with HIGH set equal to one, it is equivalent to algorithm R. On random graphs algorithms R and S perform almost identically.

Algorithms K, M, and M/L were able to run with LOW equal to -10 without any occurrence of underflow, unless an avalanche occurred. As noted before, algorithms M and M/L won't cause avalanches if they are programmed correctly. Hence, for $s \leq 16$ with $\text{HIGH} = -\text{LOW} = 10$, we have implementations whose running times are linear in n for algorithms S, L, M, M/L, and K, and which would work as well as the more elaborate versions which contain full scale sorting routines or heaps.

If we are partitioning a graph so large that only a small portion of the records for its nodes will fit into main memory, then the running time will depend mainly on the number of times records outside of core are accessed. The majority of record accesses occur i) when nodes are moved, ii) when objective functions are updated, and iii) when entries in the priority queue are changed.

The choice of algorithm determines the number of moves and updates. Algorithms S and L make the fewest moves, around $\frac{n}{4}$ on most graphs in our tests. Algorithm M averages half again as many, and algorithms R and M/L average about three times as many. Algorithm K makes about 16 times as many moves as algorithm S. For algorithms without look-ahead the number of updates is about $2s$ times the number of moves; for those with lookahead the factor is about $4s^2$.

With our implementation, moving an entry in the priority queue to a different linked list requires accessing several pointers. Hence, updates which leave a node's entry on the same linked list are much cheaper than those which require relinking. Algorithms S and L save a lot of time because the majority of their entries always stay on the LOW list (because their objective function values are non-positive). Algorithms M, M/L and K can move nodes whose Δ-values are zero or less, but if LOW is set high enough the majority of entries should still occur on the LOW list most of the time.

We have shown that there are significant differences in the execution times of our algorithms, even though they all run in linear time for a fixed s. Furthermore, the speed of execution is affected by such things as the values of HIGH and LOW or, if the maxcut analogue is being implemented, the choice of the coefficients d_1, d_2, and τ. Hence the selection of a 2-partition algorithm to solve a real problem involves a large number of decisions.

CHAPTER 6

EXTENDING THE 2-PARTITION ALGORITHMS

6.1 k-Partitioning

The algorithms considered in Chapter 5 are designed to solve only the 2-Partition Problem. The applications described in Chapter 1 generally require that a graph be partitioned into a number of pieces. Hence we have need of an algorithm which can find a good k-partition, for any specified k. It is also desirable that a partitioning algorithm adapts to solve more complex partition problems, such as ones where a cost is assigned to each arc, or a weight assigned to each node. We will show how the algorithms of Chapter 5 can be modified to solve these more complex problems. We begin by looking at k-partition algorithms.

Two methods suggested in the literature for k-partitioning a graph might be called the chipping method and the splitting method. Given a kn-node graph, the chipping method begins by dividing the nodes into two sets of sizes n and $(k-1)n$ so as to minimize the cross arcs between the two sets. The n-node set is set aside, and the remaining nodes are divided into sets of size n and $(k-2)n$. The process continues until k n-node sets have been produced. All of our 2-partition algorithms can be adapted to the chipping method. We will not pursue the details because k-partition algorithms which are apparently more powerful exist.

The splitting method is best described by an algorithm:

Input to SPLIT - A set R of nodes, and an integer $n > 0$ such that n divides $|R|$. SPLIT produces a $\frac{|R|}{n}$-partition of the subgraph induced by R.

```
Procedure SPLIT(R,n)
    1. Set $k \leftarrow \frac{|R|}{n}$. _If_ $k = 1$ _then_ record the set $R$ and _return_.
    2. Find a partition $\{S,\bar{S}\}$ of $R$ such that $|S| = \lfloor \frac{k}{2} \rfloor n$.
    3. SPLIT(S,n); SPLIT($\bar{S}$,n).
    4. _Return_.
```

The sets which have been "recorded" during the recursive calls to SPLIT make up the desired k-partition. All of our 2-partition algorithms can be easily adapted to find the type of partition defined in Step 2 of procedure SPLIT. References [16, pp. 117-123] and [3] suggest partitioning algorithms similar to the chipping and splitting methods.

Some of the weaknesses inherent in the splitting and chipping methods can be illustrated by an example:

Define S_1 and S_2 to be two simple paths of n nodes each (in Chapter 3 these were called n-node 0-leaf chains). Let C_1, C_2, and C_3 be cliques each having $\frac{2n}{3}$ nodes. Let G be a graph consisting of these five components. It is our object to find a good 4-partition $\{P_1,P_2,P_3,P_4\}$ of G.

Assign C_i to the set P_i, for $i = 1$ to 3, and set $P_4 = S_1$. Place the first third of S_2's nodes in P_1, the second third in P_2, and the remaining third in P_3. This defines a 4-partition which has only two cross arcs.

Suppose we have chipping and splitting algorithms which find an optimal partition at each step. In this case the chipping method will put one of the S_i's in P_1, the other in P_2, and then will have to split one of the cliques in half to form P_3 and P_4 from C_1, C_2, and C_3. The splitting method will begin by placing S_1 and S_2 in one set of a 2-partition, and C_1, C_2, and C_3 in the other. Then it will split each set in half, again requiring that one of the C_i's is split in half by the final 4-partition. Hence both methods find a 4-partition having $\frac{n^2}{9}$ cross arcs.

The chipping and splitting algorithms represent a blend of the constructive and the iterative-improvement approaches. They are expected to run faster than the pure iterative-improvement algorithms we are about to present. Kodres [19] describes some constructive algorithms which should also run very fast. It is possible that initial use of one of these fast algorithms to generate a starting k-partition, rather than using a randomly-chosen starting k-partition, would reduce the overall running time when an iterative-improvement algorithm is subsequently employed.

The one-opting algorithms R and S can be easily generalized to produce k-partitions. Assume a graph (N,A) with a k-partition $\{P_1,P_2,\ldots,P_k\}$. For each node $u \in P_i$ $(i = 1 \text{ to } k)$ define

$$\Delta_{ij}(u) \triangleq |\{v \in P_j | (u,v) \in A\}| - |\{v \in P_i | (u,v) \in A\}| \quad .$$

$\Delta_{ij}(u)$ is the decrease in cross arcs when node ω is moved from P_i to P_j. Now define for $u \in P_i$, $v \in P_j$

$$a(u,v) \triangleq \begin{cases} 1 & \text{if } (u,v) \in A \\ 0 & \text{otherwise} \end{cases}$$

and

$$\delta_{ij}(u,v) \triangleq \Delta_{ij}(u) + \Delta_{ji}(v) - 2a(u,v) .$$

Algorithm S now becomes "repeatedly select a maximal $\delta_{ij}(u,v)$ and exchange nodes u and v, until no δ_{ij}'s exceed zero". Unlike the previous k-partition algorithms, this one guarantees that the partitions it finds are 1-optimal (defined in section 3.2).

In the manner of algorithm K we can "lock" the nodes u and v to their new sets after each exchange, and continue exchanging pairs of nodes until all nodes have been moved and locked in place. Then the best intermediate k-partition is found, and the sequence is repeated as long as improvements occur. However, as in the case for the 2-partition algorithms in Chapter 5, we can design more efficient algorithms if nodes are moved one at a time, instead of being exchanged pairwise. This results in what we call the <u>cyclic</u> method.

Here is the cyclic version of Algorithm S:

<u>Algorithm S-C</u>.

Assume kn nodes and a starting partition $\{P_1, P_2, \ldots, P_k\}$. M[] is an array of size k whose entries take the values OPEN or CLOSED.

```
1. For i = 1 to k do M[i] ← OPEN.
2. If $\Delta_{ij}(u) = 0$ for all $u \in N$ stop.
3. Choose u such that
       $\Delta_{i_0 j_0}(u) = \max_{\substack{1 \le i \le k \\ 1 \le j \le k}} \{\Delta_{ij}(v) \mid v \in P_i,\ M[i] = \text{OPEN}\}$
4. If M[$i_0$] = CLOSED then M[$i_0$] ← OPEN.
5. Move u from $P_{i_0}$ to $P_{j_0}$.
6. If $|P_{j_0}|$ = n then go to Step 2.
```

```
7. Choose $u \in P_{j_0}$ such that
       $\Delta_{j_0\ell_0}(u) = \max_{1\le\ell\le k} \{\Delta_{j_0\ell}(v) | v \in P_j, M[\ell] = \text{OPEN}\}$
8. If $\Delta_{j_0\ell_0}(u) \le 0$ then $M[j_0] \leftarrow$ CLOSED.
9. $i_0 \leftarrow j_0$; $j_0 \leftarrow \ell_0$; go to Step 5.
```

When Algorithm S-C selects a node in Step 2 and then moves it in Step 5 an imbalance is created -- one of the sets has one too few nodes, and one has one too many. This condition may continue for several moves before a move happens to be directed into the deficient set -- in which case balance is restored. Thus a cycle of improving moves occurs, rather than a simple pairwise exchange.

To ensure termination a set P_j is "closed" when it has no good nodes ($\delta_{j\ell}(v) \le 0$ for all $v \in P_j$, and for all ℓ), meaning that the algorithm will no longer move nodes into it. The OPEN-CLOSE mechanism could be implemented differently. For example, Δ-values of some nodes must be updated after the move in Step 5. If a node belonging to a closed set becomes good at this point, the set could be opened immediately, rather than waiting for Step 4 to open it. Also, instead of immediately closing a set P_j when it has no good nodes, a counter could be started which would allow some fixed number of moves into and out of P_j before closing it. This would allow more $\Delta = 0$ nodes to be moved, and might uncover some additional improving moves.

Algorithm K can be adapted to a cyclic format more elegantly than Algorithm S, because no problems with termination occur. Here is an outline of Algorithm K-C's principal subroutine:

Subroutine for Algorithm K-C.

Assume kn nodes and a k-partition $\{P_1, P_2, \ldots, P_k\}$. M[] is an array of size kn which takes the values FREE or USED.

```
1. For u = 1 to kn M[u] ← FREE.
2. Choose a free node u such that
       $\Delta_{i_0 j_0}(u) = \max_{\substack{1 \le i \le k \\ 1 \le j \le k}} \{\Delta_{ij}(v) \mid v \in P_i,\ M[v] = \text{FREE}\}$
3. Move u from $P_{i_0}$ to $P_{j_0}$ and set M[u] ← USED.
4. If $|P_{j_0}| = n$ then go to Step 2.
5. Choose a free node $u \in P_j$ such that
       $\Delta_{j_0 \ell_0}(u) = \max_{1 \le \ell \le k} \{\Delta_{j_0 \ell}(v) \mid v \in P_\ell,\ M[v] = \text{FREE}\}$
6. $i_0 \leftarrow j_0$; $j_0 \leftarrow \ell_0$; go to Step 3.
```

A mechanism to keep track of intermediate partitions has to be inserted into this subroutine. The sets making up the partitions are all equal in size only occasionally. The lowest number of cross arcs will often belong to a partition which has one node "out of place". If $\{P'_1, P'_2, \ldots, P'_k\}$ is an intermediate partition, and $|P'_i| + 1 = |P'_j| - 1 = n$, then to form a k-partition a node has to be moved from P'_i to P'_j, possibly changing the number of cross arcs. It is simple to compute what the value (in cross arcs) of each intermediate partition will be after this adjustment is made, and the intermediate partition with the best adjusted value is the one which should be remembered by the subroutine.

Algorithm M adapts to solve the k-Partition Problem in a straightforward way. For a node $u \in P_i$ and some constants d_1 and d_2 we define

$$\mu_{ij}(u) = \mu_{ij}(u,d_1,d_2) \triangleq d_1(|P_i|-|P_j|) + d_2\Delta_{ij}(u) .$$

We repeatedly move the node with the highest μ_{ij}-value until all μ_{ij}-values are non-positive. Then d_1 is increased and the process repeated until all sets are of equal size. From now on we will refer to this k-partitioning version as Algorithm M.

We have a few remarks on the implementation of our k-partition algorithms. Algorithm S-C should have a priority queue Q_i for each set P_i. For a node u in P_i there are up to $k-1$ entries in queue Q_i -- an entry containing $\Delta_{ij}(u)$ for each set $j \neq i$ which hasn't been closed. Step 7 of Algorithm S-C selects the node u corresponding to the top entry in queue Q_{j_0}. We could facilitate the selection of u in Step 3 by maintaining another queue containing the top entry from each queue Q_i, but probably Step 3 is executed a small enough percentage of the time that simply finding the maximum among all of the queues whenever Step 3 occurs is faster than maintaining the extra queue.

If Algorithm K-C has a structure which can quickly produce the value $\max\{\Delta_{ij}(u)|u \in P_i\}$, given i and j, then the "adjusted values" of the intermediate partitions can be easily computed. This is realized if we maintain k^2 priority queues -- there are k^2-k queues Q_{ij} $(i=1 \text{ to } k, j=1 \text{ to } k, i \neq j)$ such that each queue Q_{ij} contains the values $\{\Delta_{ij}(u)|u \in P_i\}$, and we maintain k more priority queues Q_i $(i=1 \text{ to } k)$ such that queue Q_i contains the highest entry from each of the queues Q_{ij} $(j=1 \text{ to } k, j \neq i)$. Though this scheme requires k times as many queues as the scheme for Algorithm S-C, the total number of entries in all of the queues is not much greater than before.

Algorithm M changes a large number of μ_{ij}-values with each move, because the sizes of the sets are constantly changing. Our strategy is to have $k^2 - k$ queues Q_{ij} $(i \neq j)$ containing Δ_{ij}-values, as for Algorithm K-C. Only the μ_{ij}-value corresponding to the top Δ_{ij}-value of each queue Q_{ij} needs to be calculated. Thus, of all of the μ_{ij}-values which change, only a fraction of them have to be updated.

The cyclic algorithms and Algorithm M appear to be more powerful than the previous k-partition algorithms because they can perform cyclic exchanges of nodes as well as pairwise exchanges. Their implementations are relatively straightforward. Lookahead (Chapter 5) can be added to any of them.

6.2 General Partitioning

Suppose we are asked to partition a graph G into exactly k pieces of _approximately_ equal size so as to minimize the number of cross arcs. Specifically, assume we have a graph with kn nodes, and a positive integer ε is fixed: we seek a partition $\{P_1, P_2, \ldots, P_k\}$ such that $|P_i| \leq n + \varepsilon$ for $i = 1$ to k. To make this a k-Partition Problem all we have to do is add $k\varepsilon$ new singleton (dummy) nodes to G. Call the augmented graph G'. Let $\{P_1', P_2', \ldots, P_k'\}$ be a k-partition of G' which minimizes the number of cross arcs. Set

$$P_i = P_i' - \{\text{the dummy nodes in } P_i'\} \quad (i = 1 \text{ to } k) .$$

Then $\{P_1, P_2, \ldots, P_k\}$ is a partition which minimizes the number of cross arcs subject to the constraint $|P_i| \leq n + \varepsilon$.

If ε is relatively large then it will take significantly more space to store the graph G' than it takes to store G. We can eliminate this problem, and probably speed up the execution time somewhat, by eliminating the dummy nodes, and substituting in their place a counter for each set P_i which indicates how many nodes P_i contains. Moving a dummy node from P_i to P_j is performed by decrementing P_i's counter by one, and adding one to P_j's counter.

We now show how to convert the General Partition Problem (mentioned in Chapter 2) into a k-Partition Problem. We are given a graph G and a constant W. A partition $\{P_1, P_2, \ldots, P_\ell\}$ of G is _feasible_ if $|P_i| \leq W$ for $i = 1$ to ℓ. The number of sets allowed in a feasible partition is arbitrary, and they are not necessarily of equal size. We seek among all feasible partitions of G one which minimizes the number of cross arcs.

For convenience assume that G has n nodes and that W divides n evenly (we can always add some dummy nodes to G to insure that W evenly divides the total number of nodes). Set $k = \frac{n}{W}$. Let $\{P_1, P_2, \ldots, P_\ell\}$ be a feasible partition of G. If there exist sets P_i and P_j $(i \neq j)$ such that $|P_i| + |P_j| \leq W$ then we can combine these two sets into one to form a new feasible partition of smaller cardinality. Given any feasible partition of G, there is one with the same number of cross arcs and cardinality at most $2k-1$. Our problem has become "find a partition $\{P_1, P_2, \ldots, P_{2k-1}\}$ of G such that $|P_i| \leq W$ (for $i = 1$ to $2k-1$) which minimizes the number of cross arcs". At the beginning of this section we showed how to solve this version of the problem. Hence, our k-partition algorithms can be employed to solve the General Partitioning Problem.

6.3 More Complex Cost Functions

In this section we show how to modify the definition of Δ_{ij} in the implementations of our k-partition algorithms to solve a variety of other problems.

Let us assign a cost c_{uv} to each arc (u,v) of a graph G, and define $c_{uv} = 0$ if arc (u,v) isn't present in G. Instead of minimizing the number of cross arcs in a k-partition, some problems ask for a k-partition of G which minimizes the sum $\sum_{\substack{(u,v) \text{ is a} \\ \text{cross arc}}} c_{uv}$.

We can accommodate this change by redefining our Δ_{ij}'s: for a node $u \in P_i$ define

$$\Delta_{ij}(u) = \sum_{v \in P_j} c_{uv} - \sum_{v \in P_i} c_{uv} .$$

Our k-partition algorithms now are defined exactly as before, except that they utilize this new objective function in place of the old one.

Assume that the arc costs are non-negative integers, and let $t = \sum_{u \neq v} c_{uv}$. The number of exchanges made by our algorithms is bounded above by some polynomial $P(n,t)$. On any class of graphs for which t grows exponentially with the size of n we have no proof that our algorithms will run in polynomial time. It is an open problem to show that Algorithm S, say, runs in time polynomial in n, regardless of the value of t.

From the field of Design Automation comes the problem of partitioning elements of electrical circuits so as to minimize the number of electrical connections made between different sets of the partition (see Lawler [23] or Kodres [19]). We abstract this problem by using a node to represent an element of the circuit. The elements are interconnected by signal nets. A net connecting elements $u_1, u_2, \ldots, u_\ell$ is denoted in our abstracted problem by the set $\{u_1, u_2, \ldots, u_\ell\}$. This collection of nodes and sets defines a hypergraph (see Lawler [24]). Let $S_1, S_2, \ldots, S_r$ be the signal nets in our problem. With respect to a partition $\{P_1, P_2, \ldots, P_k\}$ of the nodes define

$$\theta_{ij} = \begin{cases} 1 & \text{if } S_i \cap P_j \text{ is non-null} \\ 0 & \text{if } S_i \cap P_j \text{ is empty} \end{cases}$$

The cost function we wish to minimize is

$$\sum_{i=1}^{t} \sum_{j=1}^{k} \theta_{ij} \tag{6.1}$$

All of our k-partition algorithms can be adapted to solve this problem. We set $\Delta_{ij}(u)$ = "the decrease in the cost function (6.1) when node u in P_i is moved to P_j". Schweikert and Kernighan discuss their experience with this model in [33].

An additional constraint is often found associated with this problem. Let $E(i)$ denote the number of signal nets which contain both nodes in P_i and nodes not in P_i. We require $E(i) \leq p$, for all i and some p -- where p corresponds to the maximum number of external connections, or pins, allowed each set of the partition. See [11,19,23] for discussions on this aspect of the problem. We suggest adding a penalty function to the objective function Δ_{ij} which encourages exchanges that lower the values of the larger $E(i)$'s, in the same spirit as the use of the penalty function $d_1(|P_1|-|P_j|-1)$ by Algorithm M.

6.4 Adding Weights to the Nodes

If we are handed a partitioning problem in which the elements to be partitioned are of different sizes (assuming a one-dimensional metric) then we must alter our abstract model by associating a weight ω_u with each node u. We also specify a capacity W and require that our partitions satisfy $\sum_{u \in P_i} \omega_u \leq W$, for $i = 1$ to k. If the difference $W - \frac{1}{k}\sum \omega_u$ is too small the problem becomes a packing problem as well as a partitioning problem. To avoid this we will assume that the difference is large enough so that given any partition $\{P_1, P_2, \ldots, P_k\}$ for which a set P_i is too heavy $(|P_i| > W)$, there exists a node $u \in P_i$ and a set P_j such that $|P_j| + \omega_u \leq W$.

If we wish to partition our graph into only two sets, the problem of node weights can be dealt with without much difficulty. Define $\omega(P_i) = \sum_{u \in P_i} \omega_u$. Suppose we have a starting partition $\{L,R\}$ with $\omega(L) \geq \omega(R)$. Algorithm S or K moves a node from L to R, and continues to move nodes from L to R until $\omega(R) \geq \omega(L)$. Next a node is moved from R to L, and more nodes follow it until $\omega(L) \geq \omega(R)$. This pattern is repeated until the algorithm arrives at a final partition. Algorithm M uses a different strategy, which we shall look at presently.

The chipping and splitting methods of finding partitions of cardinality k can be applied to graphs with node weights using the approach just outlined. However, the cyclic method used by Algorithms S-C and K-C on graphs with unit node weights doesn't seem to adapt to the problem of finding k-way partitions of graphs having different sized weights.

It is fortunate that Algorithm M comes to our rescue. If r unit-weight nodes are moved one-by-one from P_i to P_j, the sum of the penalties incurred (using M's original penalty function) is

$$\begin{aligned} &d_1[(|P_i|-|P_j|-1)+(|P_i|-|P_j|-3)+\cdots+(|P_i|-|P_j|-2r+3)] \\ &\quad = d_1[r(|P_i|-|P_j|)-r^2] \end{aligned}$$

Hence we suggest that the objective function $\mu_{ij}(u)$ for a node u with weight ω_u be

$$\mu_{ij} = d_1[\omega_u(|P_i|-|P_j|)-\omega_u^2]+d_2\Delta_{ij}(u) \tag{6.2}$$

On first glance it appeared to us that the function

$$d_1[|P_i|-|P_j|-\omega_u]+d_2\Delta_{ij}(u) \tag{6.3}$$

might be a better objective function than (6.2), because it doesn't exaggerate the importance attached to moving heavy nodes as much, and therefore would tend to let the heavy nodes move more freely. However, a simple example was found where an infinite sequence of moves could be made such that the objective function (6.3) was positive for each move, i.e. cycling can occur. Thus, (6.3) isn't usable.

The penalty incurred when a set P_i exceeds the maximum weight allowable should be greater than when a set if underweight. If some unit-weight dummy nodes are added, they will migrate to the sets having the lowest total weight, effectively reducing the penalty placed upon nodes which seek to exit from those sets.

We conclude that the k-partition algorithms can be easily adapted to solve a variety of more complex problems.

CHAPTER 7

PARTITIONING OF GRAPHS WITH SPECIAL STRUCTURE

7.1 Worst-Case 2-Partitioning of Graphs with Bounded Node Degree

Let $d = d(G)$ denote the maximum degree of any node in a graph G. In section 3.2 we proved that, if G has N arcs, then $m^* \leq \frac{N}{2} + o(N)$, where m^* is the number of cross arcs in an optimal 2-partition of G. If d is held constant we can derive a stronger result:

Theorem 7.1. Given a graph G with N arcs and maximum node degree $d \geq 3$,

$$m^* \leq \frac{d-1}{2d} N + O(d^2) .$$

For the case $d = 3$, Theorem 7.1 tells us that we can always find a 2-partition with not more than about $\frac{N}{3}$ cross arcs.

Proof. Assume we have a graph with $2n$ nodes, N arcs, and maximum node degree d. Let $\{L,R\}$ be an optimal 2-partition which contains m cross arcs.

A side arc is any arc which is not a cross arc. A side arc (u,v) is considered to consist of two half arcs: one half arc leaves node u, one leaves v, and they meet in the middle of the arc (u,v). We will find it convenient to count the number of half arcs leaving nodes in L (or R) and then to divide by two to find the number of side arcs. This approach is based upon the fact that a node u which is incident with c cross arcs must be incident with $c - \Delta(u)$ half arcs.

Let H_S denote a subset of S $(S = L \text{ or } R)$ which contains $2d^2$ nodes having the highest Δ-values among nodes in S (ties are broken arbitrarily). Define $L' \triangleq L - H_L$ and $R' \triangleq R - H_R$. Next define

$$MAX_{S'} \triangleq \max\{\Delta(u) | u \in S'\}$$

for $S' = L'$ or R'. For example, suppose $2d^2 = 18$, and L contains 10 nodes with $\Delta = 1$, 30 with $\Delta = 0$, and the rest have negative Δ-values. Then $MAX_{L'} = 0$.

For definiteness assume $MAX_{L'} \geq MAX_{R'}$. We start by showing that

$$MAX_{L'} + MAX_{R'} \leq 0 \tag{7.1}$$

Choose $u \in H_L$ such that $\Delta(u) \geq MAX_{L'}$. Choose any $v \in H_R$ such that u is not adjacent to v (since degree$(u) \leq d$ and $|H_R| = 2d^2$ this is always possible). Then $\Delta(v) \geq MAX_{R'}$, and exchanging nodes u and v produces a decrease of at least $MAX_{L'} + MAX_{R'}$ cross arcs. This is a contradiction unless (7.1) holds, since $\{L,R\}$ is defined to be an optimal 2-partition.

Given (7.1), we can prove our theorem by considering three different cases. Throughout, we define

ℓ = "the number of half arcs adjacent to nodes in L"

r = "the number of half arcs adjacent to nodes in R"

Assume that $n > 4d^2$.

<u>Case 1</u>. $MAX_{L'} + MAX_{R'} \leq -2$.

For all nodes $u \in L$, $\Delta(u) \leq -MAX_{R'}$, and for all nodes $v \in R$, $\Delta(v) \leq -MAX_{L'}$ (otherwise an exchange exists which would improve the partition $\{L,R\}$). At most $2d^2$ nodes in L (R) have Δ-values exceeding $MAX_{L'}$ $(MAX_{R'})$. Hence

$$\ell \geq m - (n-2d^2)\mathrm{MAX}_{L'} - 2d^2\mathrm{MAX}_{R'}$$

and

$$r \geq m - (n-2d^2)\mathrm{MAX}_{R'} - 2d^2\mathrm{MAX}_{L'}$$

implying

$$\ell + r \geq 2m - 2n + 8d^2 .$$

Then

$$N = m + \tfrac{1}{2}(\ell + r) \geq 2m + n - 4d^2 . \qquad (7.2)$$

A node with Δ-value ≤ -1 is incident with at most $\frac{d-1}{2}$ cross arcs. If some node in R has Δ-value greater than 2 then all Δ-values for nodes in L are negative, implying $m \leq \frac{d-1}{2}n$. Otherwise all Δ-values for nodes in R are 2 or less. Also $\mathrm{MAX}_{R'} \leq -1$. Hence

$$m \leq \frac{d-1}{2}n + \frac{3}{2}(2d^2) . \qquad (7.3)$$

Now we have

$$\begin{aligned} \frac{m}{N} &\leq \frac{m}{2m + n - 4d^2} && \text{by (7.2)} \\ &\leq \frac{\frac{d-1}{2}n + 3d^2}{dn + 2d^2} && \text{by (7.3)} \\ &\leq \frac{d-1}{2d}n + \frac{3d}{n} . \end{aligned}$$

Since $N \leq dn$, $m \leq \frac{d-1}{2d}n + O(d^2)$ as desired.

Case 2. $\mathrm{MAX}_{L'} + \mathrm{MAX}_{R'} = 0$.

For clarity we will prove Case 2 assuming that $\mathrm{MAX}_{L'} = \mathrm{MAX}_{R'} = 0$, but a proof analogous to ours goes through without using this assumption.

A node u such that $\Delta(u) > 0$ does not exist, or the 2-partition could be improved. Let

$$Z \triangleq \{u \in L \mid \Delta(u) = 0\} .$$

Define G_Z to be the subgraph induced by the subset $L \cup R - Z$. Consider the partition $\{L-Z, R\}$ of G_Z. Let $\Delta_Z(u)$ denote the Δ-value of a node u with respect to this new partition.

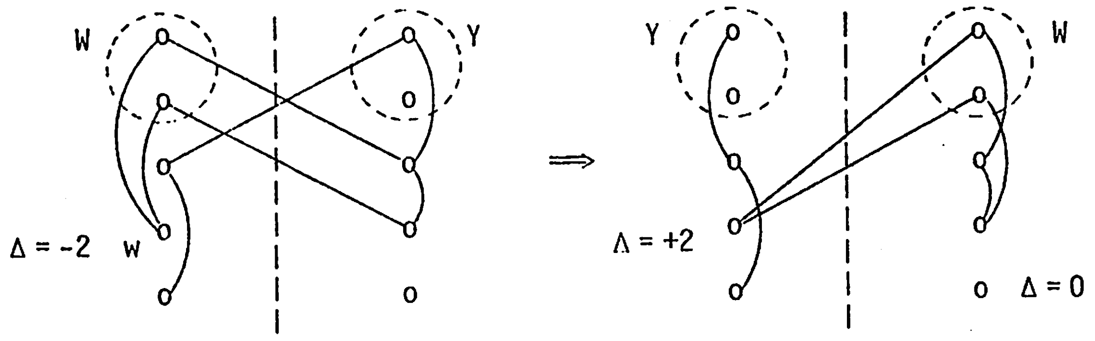
<u>Claim 7.1</u>. For all $u \in L-Z$, $\Delta_Z(u) \leq -1$.

<u>Proof of Claim 7.1</u>. Suppose that there exists $w \in L-Z$ such that $\Delta_Z(w) \geq 0$. Let $c = \Delta(w)$. By definition

$$\Delta_Z(w) - c > 0 \ . \tag{7.4}$$

Let $W = \{u \in H_L \text{ adjacent to } w\}$. Choose $|W|$ nodes in H_R such that none are adjacent to nodes in the set $\{w\} \cup W$; call this new set Y. (Note, $|W| \leq d$ and $|H_R| = 2d^2$, so such a set can always be found.) Exchange the nodes in W with those in Y. The number of cross arcs will not increase as a result of this exchange. Now consider the node w. With respect to the new partition formed by exchanging W and Y, $\Delta(w) = c + 2(\Delta_Z(w) - c) > 0$ by (7.4). But there exists a node $v \in H_L - Y$, not adjacent to w, such that $\Delta(v) = 0$. Exchanging w and v improves the 2-partition, a contradiction. Hence Claim 7.1 must be true.

We give an example of what happens when Claim 7.1 is not satisfied. Here $\Delta_Z(w) = 0$, $c = -2$, $t = 2$:



Suppose z cross arcs are incident with nodes in Z, so $m-z$ cross arcs are incident with nodes in $L-Z$. Let $|L-Z| = b$. Then the number of side arcs incident only with nodes in $L-Z$ is at least $\frac{1}{2}(m-z+b)$, by Claim 7.1.

Next, let A_Z be the set of side arcs incident with at least one node in Z. None of the nodes in Z are adjacent, or a 2-partition better than $\{L,R\}$ would exist (arguing as for Claim 7.1). Hence $|A_Z| = z$. Summing, we find that

$$\ell \geq 2z + m - z + b = m + z + b \ .$$

We can similarly define $Z' = \{v \in R | \Delta(v) = 0\}$ and find z' and b' for R such that $r \geq m + z' + b'$. Assume for definiteness that $z + b \leq z' + b'$. Then

$$N \geq 2m + z + b \ .$$

We also know

$$m \leq z + \frac{d-1}{2} b \ .$$

Hence

$$\frac{m}{N} \leq \frac{m}{2m+z+b} \leq \frac{z + \frac{d-1}{2}b}{3z + db} \leq \frac{d-1}{2d} \ ,$$

which proves the theorem for Case 2.

<u>Case 3</u>. $MAX_{L'} + MAX_{R'} = -1$.

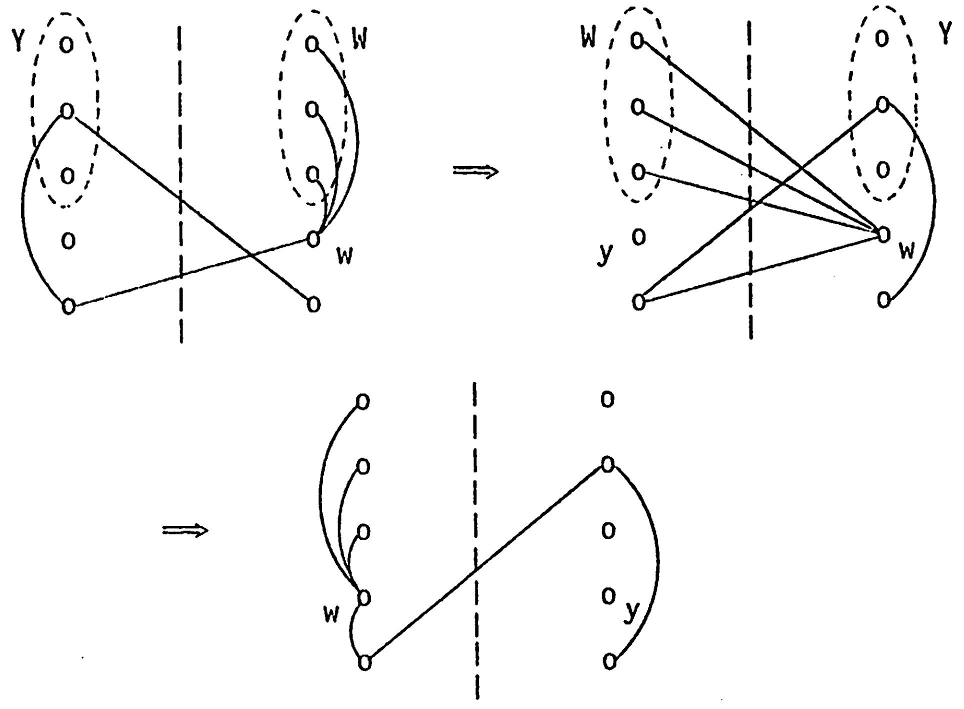
For clarity assume $MAX_{L'} = 0$, $MAX_{R'} = -1$. For the moment assume further that $\Delta(u) \leq 0$ for all $u \in L$, and $\Delta(v) \leq -1$ for all $v \in R$.

<u>Claim 7.2</u>. Let $u \in L$ and suppose $\Delta(u) = -i$. Then u is adjacent to at most $i+1$ nodes $v \in L$ such that $\Delta(v) = 0$.

<u>Claim 7.3</u>. Let $u \in R$ and suppose $\Delta(u) = -i$. Then u is adjacent to at most i nodes $v \in R$ such that $\Delta(v) = -1$.

We will prove Claim 7.3 here. The proof of Claim 7.2 is similar. Suppose $w \in R$, $\Delta(w) = -i$, and at least $i+1$ $\Delta = -1$ nodes in R are adjacent to w. Choose $i+1$ of these nodes and call this set W. Now choose $i+1$ $\Delta = 0$ nodes in L which are not adjacent to nodes in $\{w\} \cup W$, and call this set of nodes Y. Interchange the nodes in Y with those in W. The number of cross arcs increases by at most $i+1$. Now look at w. With respect to the new partition, $\Delta(w) = -i + 2(i+1) = i + 2$. Find a node y in L such that $\Delta(y) = 0$ and y is not adjacent to w. Exchanging y and w reduces the number of cross arcs by $i + 2$. Contradiction. Hence Claim 7.3 must be true.

<u>Example for Claim 7.3</u> $(i = 2)$:



Claim 7.4. $\ell \geq m + \frac{2(m-n)}{d+1}$

Claim 7.5. $r \geq m + n + \frac{2m}{d+2}$ if d is even.

$r \geq m + n + \frac{2m}{d+1}$ if d is odd.

We will prove Claim 7.5. Claim 7.4 is proved in a similar fashion, but makes use of Claim 7.2 rather than Claim 7.3 in its proof. Define

$$T = \{v \in R \mid \Delta(v) < -1\}$$

and

$$t_i = |\{v \in R \mid \Delta(v) = -i\}| \ .$$

Set $t = |T| = \sum_{i=2}^{d} t_i$. If d is even, then at most $\frac{t(d-2)}{2}$ cross arcs are incident with nodes in T, so at least $m - \frac{t(d-2)}{2}$ cross arcs are connected to nodes in $R - T$. Hence at least $m - \frac{t(d-2)}{2}$ side arcs cross from $R - T$ to T, because at least $m - \frac{t(d-2)}{2} + n - t$ half arcs leave nodes in $R - T$, and by Claim 7.3 $(i = 1)$ all but $n - t$ of these half arcs connect to nodes with $\Delta < -1$. Combining this result with Claim 7.3 where $i < 1$, we find

$$2t_2 + 3t_3 + \cdots + dt_d \geq m - \frac{t(d-2)}{2}$$

and hence

$$t_2 + 2t_3 + \cdots + dt_d \geq m - \frac{t(d-2)}{2} - t \ .$$

We also have $t_2 + t_3 + \cdots + t_d = t$ so

$$\begin{aligned} t_2 + 2t_3 + \cdots + (d-1)t_d &\geq \max\{m - \tfrac{td}{2},\ t\} \\ &\geq \frac{2m}{d+2} \ . \end{aligned}$$

Hence

$$\begin{aligned} r &\geq m + n + t_2 + 2t_3 + \cdots + (d-1)t_d \\ &\geq m + n + \frac{2m}{d+2} \end{aligned}$$

as desired. If d is odd the proof of Claim 7.3 starts by observing that at most $\frac{t(d-3)}{2}$ cross arcs are incident with nodes in T. This leads the slightly stronger lower bound on r. Thus, Claim 7.3 is proved.

We now split the proof into four cases.

Case 3a. $d = 3$.

If a node $u \in R$ is connected to a cross arc then $\Delta(u) \geq -1$. Hence all m cross arcs connect to nodes in $R-T$, implying

$$2t_2 + 3t_3 \geq m .$$

Also, $2m$ half arcs leave the m nodes in $R-T$ that are incident to cross arcs. Putting these results together we have $r \geq 3m$. Also, $\ell \geq m$, so

$$N \geq \frac{1}{2}(\ell+r) + m \geq 3m \Rightarrow \frac{m}{N} \leq \frac{d-1}{2d} .$$

Case 3b. $m \leq n$, $d \geq 4$.

By Claim 7.5, $r \geq m + n + \frac{2m}{d+2}$. Also, $\ell \geq m$, so

$$N \geq m + \frac{d+3}{d+2}m + \frac{n}{2} .$$

Hence

$$\frac{m}{N} \leq \frac{m}{\frac{2d+5}{d+2}m + \frac{n}{2}} \leq \frac{2(d+2)}{5d+12}$$
$$\leq \frac{d-1}{2d} \text{ when } d \geq 4 .$$

Case 3c. d is even, $d \geq 4$, $m > n$.

By Claims 7.4 and 7.5

$$\ell \geq m + \frac{2(m-n)}{d+1} \geq m + \frac{m-n}{d-1}$$

and

$$r \geq m + n + \frac{2m}{d+2} \geq m + n + \frac{m}{d-1} .$$

Hence

$$N \geq m(2 + \frac{1}{d-1}) + \frac{n}{2}(1 - \frac{1}{d-1}) .$$

Because at most $\frac{d-2}{2}$ cross arcs are incident to any node in R when d is even, we know $m \leq \frac{d-2}{2}n$. This implies

$$\frac{m}{N} \leq \frac{\frac{d-2}{2}}{\frac{d-2}{2}(\frac{2d-1}{d-1}) + \frac{1}{2}\cdot\frac{d-2}{d-1}} = \frac{d-1}{2d} .$$

<u>Case 3d</u>. d is odd, $d \geq 5$, $m > n$.

By Claims 7.4 and 7.5

$$\ell + r \geq 2m + n + \frac{4m-2n}{d+1}$$

so

$$N \geq m(2 + \frac{2}{d+1}) + \frac{n}{2}(1 - \frac{2}{d+1}) .$$

For d odd we know that $m \leq \frac{d-1}{2}n$, so

$$\frac{m}{N} \leq \frac{\frac{d-1}{2}}{\frac{d-1}{2}(\frac{2d+4}{d+1}) + \frac{1}{2}\cdot\frac{d-1}{d+1}} = \frac{d+1}{2d+5}$$

$$\leq \frac{d-1}{2d} \quad \text{for} \quad d \geq 5 .$$

Thus we have shown that Case 3 is true when all Δ-values in L are ≤ 0 and all Δ-values in R are ≤ -1. By definition of MAX_L, and MAX_R, at most $2d^2$ nodes in L could have $\Delta = +1$, and at most $2d^2$ nodes in R have $\Delta = 0$. This adds a term of size $O(d^2)$ to our upper bound on m. Hence Theorem 7.1 is proved. □

7.2 The Case of Very Sparse Graphs

Suppose we are given a graph G with n nodes and N arcs, with $N \leq \frac{n}{4}$. Then there are at least $\frac{n}{2}$ singleton nodes in G, and hence a 2-partition of G with zero cross arcs obviously exists. How big can we make N and still guarantee that for all graphs G, $m^*(G) = 0$?

Theorem 7.2. Given a graph G with n nodes and N arcs, if $N < \frac{n}{2}$ then there exists a 2-partition of G having zero cross arcs.

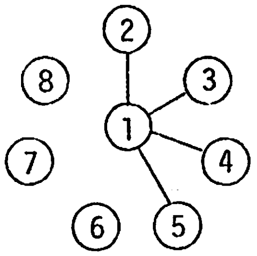
Fix an even positive integer n, choose a positive integer $N \leq n-1$, and let

$$\mathcal{N} = \{v_1, v_2, \ldots, v_n\}$$

and

$$A = \{(v_1, v_j) \mid 1 < j \leq N+1\}$$

define a star graph $G = (\mathcal{N}, A)$. Here is an example of G when $n = 8$ and $N = \frac{n}{2}$:



An optimal 2-partition of G contains one cross arc. This construction shows that the result in Theorem 7.2 is best possible. For $N \geq \frac{n}{2}$ the best 2-partition of a star graph has $N - \frac{n}{2} + 1$ cross arcs, so if we fix the ratio $s = \frac{N}{n} > \frac{1}{2}$ then the number of cross arcs grows linearly with n.

Suppose we require that no node degree exceeds three. For this case Theorem 7.1 finds an upper bound on $m^*(G)$ which grows linearly with n. The possibility remains that when s is small (but greater than $\frac{1}{2}$) $m^*(G)$ can be bounded above by a function which grows slower than linearly as n increases. Our next theorem indicates that this is not the case.

Theorem 7.3. Given any $s \in (\frac{1}{2},\frac{3}{2})$ there exists an $\varepsilon > 0$ such that given any n_0, there is an $n > n_0$ and a graph G with n nodes and at most sn arcs such that the maximum degree of any node is three, and $m^*(G) \geq \varepsilon n$.

We will now prove Theorems 7.2 and 7.3.

Proof of Theorem 7.2. We will prove a more general result for any graph G such that $N < \frac{n}{2}$:

Claim 7.6. Given the graph G, fix a $c \geq 2$ such that $\frac{n}{c}$ is an integer. Then there exists a partition $\{S,\bar{S}\}$ of G with $|S| = \frac{n}{c}$ which has zero cross arcs.

Claim 7.6 is proved by induction on the number of components of size at least two. Assume that G contains p components of size two or greater.

Basis: $p = 0$, i.e. G has no arcs. For this case Claim 7.6 is obviously true.

Induction step: Assume that Claim 7.6 is true whenever G contains exactly $p-1$ non-trivial components. We will show that the claim must also be true if G contains p non-trivial components, $p \geq 1$.

Suppose G has p non-trivial components. Choose any non-trivial component C. We know that

$$|C| \le N+1 < \frac{n}{2}+1 \ . \qquad (7.5)$$

We begin the construction of the sets S and $\bar{S}$ by assigning the nodes of C to $\bar{S}$. We must make sure that $|C| \le |\bar{S}|$: Suppose n is even. Then by (7.5), $|C| \le \frac{n}{2} \le n(1-\frac{1}{c})$, since $c \ge 2$. Otherwise n is odd. Then because $|\bar{S}|$ and $|C|$ are integers, $|\bar{S}| \ge \frac{n+1}{2}$, and by (7.5) $|C| \le \frac{n+1}{2}$.

Let $|C| = n_0$ and suppose that x cross arcs connect the nodes in the component C. Let G' be the subgraph of G induced by N-C, i.e. G' contains all components of G except C. Then G' has n-n_0 nodes, N-x arcs, and p-1 non-trivial components. Our object now is to find a partition $\{S,\bar{S}'\}$ of G' with zero cross arcs such that $|S| = \frac{n}{c}$ and $|\bar{S}'| = n - n_0 - \frac{n}{c}$. If we have such a partition, then setting $\bar{S} = \bar{S}' \cup C$ proves Claim 7.6 and the theorem. Such a partition $\{S,\bar{S}'\}$ must exist, according to the inductive hypothesis, if we can demonstrate that $N\text{-}x < \frac{n-n_0}{2}$.

Because C is non-trivial and connected we know

$$x \ge n_0 - 1 \qquad (7.6)$$

and

$$n_0 \ge 2 \ . \qquad (7.7)$$

Suppose n is even. Then by (7.6)

$$\begin{aligned} N - x &\le N - (n_0-1) \le \frac{n}{2} - 1 - n_0 - 1 \\ &< \frac{n}{2} - \frac{n_0}{2} \quad \text{because } n_0 > 0 \ . \end{aligned}$$

If n is odd then

$$N - x \le N - (n_0 - 1) \le \frac{n}{2} - \frac{1}{2} - (n_0 - 1)$$
$$= \frac{n+1}{2} - n_0 < \frac{n - n_0}{2} \text{ by (7.7).} \qquad \square$$

The proof of Theorem 7.2 suggests a very simple algorithm for finding a zero-cost cut when $N < \frac{n}{2}$. Assume $\frac{n}{c}$ is an integer and $c \ge 2$.

Algorithm to find a partition $\{S, \bar{S}\}$ such that $\#(S;\bar{S}) = 0$ and $|S| = \frac{n}{c}$:

```
1. Label the connected components of the graph $C_1, C_2, \ldots, C_p$
   in such a way that $|C_1| \ge |C_2| \ge \cdots \ge |C_p|$.
2. $S \leftarrow \emptyset$; $\bar{S} \leftarrow \emptyset$.
3. for $i = 1$ to $p$ do
       if $\frac{n}{c} - |S| \le (\frac{c-1}{c})n - |S'|$ then $\bar{S} \leftarrow \bar{S} \cup C_i$
       else $S \leftarrow S \cup C_i$.
```

Proof of Theorem 7.3. A construction by G.A. Margulis in [30] provides our starting point, which is

Lemma 7.1. There exists $\varepsilon_1 > 0$ such that for every integer $m > 0$ we can find a graph G_1 such that

i) G_1 has $2m^2$ nodes;

ii) the degree of each node is at most 10;

iii) if $\{S, \bar{S}\}$ is a partition of the nodes of G_1 with $|S| = f \cdot 2m^2 \le m^2$ then $\#(S;\bar{S}) \ge \varepsilon_1 f \cdot 2m^2$.

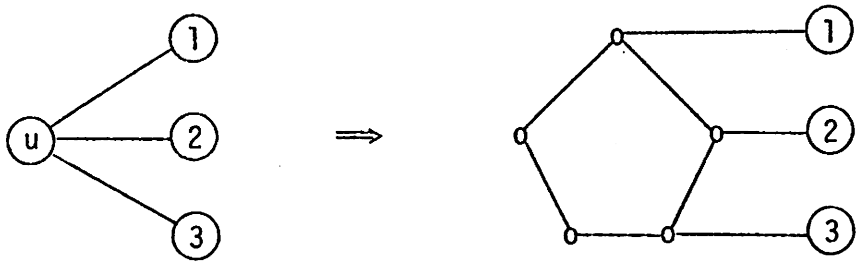
The degrees of the nodes in a graph G_1 provided by the lemma are too large for our purposes, so we will alter G_1 so that each node has degree three or less:

Define an <u>r-cycle</u> to be the graph (N_r, A_r) where

$$\begin{aligned} N_r &= \{u_1, u_2, \ldots, u_r\} \\ \text{and} \quad A_r &= \{(u_i, u_{i+1 \bmod r}) \mid 1 \le i \le r\} . \end{aligned}$$

Assume $r \geq 10$. Replace each of the original nodes u in G_1 by an r-cycle, assigning each of the arcs in G_1 originally incident with u to a different node of the r-cycle. Call this new larger graph G_2.

<u>Example</u> $(r=5)$:



Our graph G_2 has $n_2 = 2m^2r$ nodes, at most $2m^2r + 10m^2$ arcs, and contains $2m^2$ embedded r-cycles.

<u>Claim 7.7</u>. If $\{S, \bar{S}\}$ is a partition of G_2 with $|S| = fn_2 \leq m^2r$ then $\#(S;\bar{S}) \geq \frac{\varepsilon_1}{2r} fn_2$.

The proof of Claim 7.7 follows this proof of Theorem 7.3.

Suppose we set $r = 10$. Then for the graph G_2, $\frac{\#\text{ arcs}}{\#\text{ nodes}} = s \leq \frac{3}{2}$. If we set $\varepsilon = \frac{\varepsilon_1}{20}$ then for this particular value of s we have proved Theorem 7.3. To make the theorem valid for all s in the interval $(\frac{1}{2}, \frac{3}{2})$ we need a way to reduce the average degree of each node still further. We will do this by adding qn_2 singleton nodes to G_2, for a suitable $q < 1$. We will show that there is an $\varepsilon > 0$ such that any 2-partition of this new graph must have at least εn cross arcs (assuming n nodes).

Define $\delta = s - \frac{1}{2}$, let $q = \lceil\frac{(1-\delta)n_2}{1+2\delta}\rceil\frac{1}{n_2}$ and set $r = \lceil\frac{10}{\delta}\rceil$ ($\lceil x\rceil$ denotes the smallest integer not less than x). Our final graph G is constructed by finding a graph G_1, embedding r-cycles in it (for $r = \lceil\frac{10}{\delta}\rceil$) to form G_2, and then adding q singleton nodes. G is constructed so that

$$\frac{\#\ \text{arcs}}{\#\ \text{nodes}} = \frac{2m^2(r+5)}{2m^2r(1+q)} \le \frac{\frac{10}{\delta}+5}{\frac{10}{\delta}(1+\frac{1-\delta}{1+2\delta})} = \frac{1}{2}+\delta = s\ .$$

Every 2-partition of G must contain at least $n_2(\frac{1+q}{2} - q) = n_2\frac{1-q}{2}$ nodes of the subgraph G_2 in each half of the partition ($n_2 = 2m^2r$). Hence we can apply Claim 7.7 with $f = \frac{1-q}{2}$ to find that the number of cross arcs in any 2-partition of G is at least

$$\frac{n_2\varepsilon_1(1-q)}{4r} = \frac{n_2\varepsilon_1(1 - \lceil\frac{n_2(1-\delta)}{1+2\delta}\rceil\frac{1}{n_2})}{4\lceil\frac{10}{\delta}\rceil}$$

$$\ge \frac{n_2\varepsilon_1\delta(\frac{\delta}{1+2\delta}) - \frac{1}{n_2})}{40+4\delta}$$

$$\ge \frac{n_2\varepsilon_1\delta^2 - 1}{44(1+2\delta)}$$

$$\ge \frac{n\varepsilon_1\delta^2}{88\cdot 3}$$ for n large enough and because $\delta < 1$.

Setting $\varepsilon = \frac{\varepsilon_1\delta^2}{264}$ proves the theorem. □

In [30] Margulis gives a construction for a graph he calls a $(1+\frac{\varepsilon_0}{2}, \frac{1}{2})$-concentrator. For any $m > 0$ he finds a graph such that

i) there are $2m^2$ nodes, divided equally into sets A and B;

ii) each arc connects a node in A with a node in B, and the degree of each node is at most five;

iii) let X be any subset of A such that $|X| \leq \frac{m^2}{2}$, and let $\tau(X)$ be the set of nodes in B adjacent to nodes in X. Then there exists a constant $\varepsilon_0 > 0$ independent of m such that $|\tau(X)| \geq |X|(1+\frac{\varepsilon_0}{2})$.

To build the graph mentioned in Lemma 7.1 we start with a concentrator $G_0 = (N_0, A_0)$. Label the nodes in A and B:

$$A = \{u_1, u_2, \ldots, u_{m^2}\} ,$$
$$B = \{v_1, v_2, \ldots, v_{m^2}\} .$$

Then define

$$A_0' = \{(v_i, u_j) | (u_i, v_j) \in A_0\} .$$

Then the graph $G_1 = (N_0, A_0 \cup A_0')$ meets the requirements of Lemma 7.1, where $\varepsilon_1 = \frac{\varepsilon_0}{4}$.

Proof of Claim 7.7. Given a graph G_2 and a partition $\{S, \bar{S}\}$ of G such that $|S| \leq fn_2$, $f \leq \frac{1}{2}$ we show that $\#(S;\bar{S}) \geq \frac{\varepsilon_1}{2r} fn_2$.

Suppose there are p r-cycles which are split by the partition $\{S, \bar{S}\}$, i.e. for each of the p r-cycles, some of its nodes are in S, and some are in $\bar{S}$. If $p \geq \frac{n_2 f}{2(r-1)}$ then

$$\#(S;\bar{S}) \geq \frac{n_2 f}{r-1} \geq \frac{n_2 f \varepsilon_1}{2r} \qquad (\varepsilon_1 \leq 1) .$$

Otherwise $p \le \frac{n_2 f}{2(r-1)}$. Then at least $(n_2 f - p(r-1))r^{-1} \ge \frac{n_2 f}{2r}$ r-cycles are wholly contained in S. By Lemma 7.1 $\#(S;\bar{S}) \ge \frac{n_2 f \varepsilon_1}{2r}$. $\square$

7.3 The Case of Trees

We now restrict our attention to the partitioning of those graphs G which are trees. A star graph having n nodes and $N = n-1$ arcs (defined in section 7.2) provides an example of a tree whose optimal 2-partition has $\sim \frac{N}{2}$ nodes. Theorem 7.1 produced the upper bound $(\frac{d-1}{d} + o(1))N$ on $m^*(G)$ for graphs having node degrees at most d. For the case of trees with bounded node degree we can find an even lower upper bound on m^*.

Theorem 7.4. Given a tree with n nodes and maximum node degree d, the number of cross arcs for an optimal 2-partition of the tree is at most

$$\tau(d,n) \triangleq \begin{cases} \frac{1}{2} + 2\log_3 n & \text{if } d \in \{3,4\} \\ \frac{1}{2} + \frac{d+1}{2}\log_d n & \text{if } d \ge 5 \end{cases}.$$

This result provides a striking contrast to the result in Theorem 7.3, where we proved the existence of n-node graphs which are sparser than trees and have maximum node degree three, but whose best 2-partitions have at least εn cross arcs, for some $\varepsilon > 0$.

We will describe an algorithm -- the Tree Coloring Algorithm -- which can 2-partition a tree so that at most $\tau(d,n)$ cross arcs exist. We also have an algorithm for k-partitioning trees which finds partitions having at most $(k-1)(d-1)\log_{2d-3}\frac{2n}{k} + \frac{k-1}{2}$ cross arcs. For the case $d = 4$ we have an example of a tree whose optimal 2-partition has $\sim \log_3 n$ cross arcs, indicating that for this case Theorem 7.4's bound cannot be improved by more than a factor of two.

Dynamic programming techniques can be effectively applied to find 2-partitions of trees. We present an algorithm which finds an optimal 2-partition of a tree in time $O(n^3)$.

Our Tree Coloring Algorithm commences by "coloring" all of the nodes of a tree white. It then goes about changing the color of more and more nodes, coloring them red, until half of the nodes are red and half are white. This defines a 2-partition, where all red nodes are understood to make up one set, and all white nodes the other.

During the execution of our algorithm there will always be some node which is <u>distinguished</u>, and is labelled "u". With respect to the distinguished node u, we define $u_1, u_2, \ldots, u_\ell$ to be the ℓ nodes adjacent to u. <u>sub(u_i)</u> denotes the subtree containing u_i and all nodes v such that u_i is on the (unique) path from u to v. <u>bal(u_i)</u> is defined to be "(the number of white nodes) - (the number of red nodes) in sub(u_i)." For convenience we will assume that the u_i's are always labelled so that

$$\text{bal}(u_1) \geq \text{bal}(u_2) \geq \cdots \geq \text{bal}(u_\ell)$$

Changing the distinguished node u causes a redefinition of the nodes u_i, and hence alters the sub(u_i)'s and bal(u_i)'s.

Let b = (# white nodes) - (# red nodes) for the entire tree. The algorithm distinguishes a node u such that

$$\text{bal}(u_i) \leq \frac{b}{2} \tag{7.8}$$

for $i = 1$ to ℓ. If at any time the distinguished node fails to have property (7.8), then a new node is chosen to be distinguished which does have it. We will prove that such a node u always exists. Here is the algorithm:

<u>Tree Coloring Algorithm</u>

```
1. Color all nodes white; $b \leftarrow n$.
2. Choose any node $x$. Assign $u \leftarrow x$.
3. If there exists a $u_i$ such that $bal(u_i) > \frac{b}{2}$, assign
   $u \leftarrow u_i$ and repeat Step 3.
4. Let $j$ denote the largest index obeying $\sum_{i=1}^{j} bal(u_i) \leq \frac{b}{2}$.
   Set $B \leftarrow \sum_{i=1}^{j} bal(u_i)$.
5. If $(\frac{b}{2} - B) > (B + bal(u_{j+1}) - \frac{b}{2})$ set $B \leftarrow B + bal(u_{j+1})$ and
   $j \leftarrow j+1$.
6. Complement all nodes in $\bigcup_{1 \leq i \leq j} sub(u_i)$: White nodes are
   colored red, red become white.
7. $b \leftarrow b - 2B$. If $b < 0$ complement every node and set $b \leftarrow -b$.
8. If $b \geq 2$ go to Step 3. Otherwise stop (half of the nodes
   are red, half are white).
```

Let us verify that Step 3 will always terminate, and hence find a node obeying (7.8). Here is an abstracted tree (see Fig. 7.1). Suppose that, in Fig. 7.1, node u_5 was originally a distinguished node, but did not obey (7.8) because the subtree of the node now labelled u has a balance exceeding $\frac{b}{2}$. Step 3 will select u to be the new distinguished node. If (7.8) is true for this node u then Step 3 terminates here. If not, then $bal(u_i) > \frac{b}{2}$ for at least one of the nodes $\{u_1, u_2, u_3, u_4\}$. $bal(u_5)$ must be less than $\frac{b}{2}$, implying that it won't be chosen to be distinguished. Hence, the succession

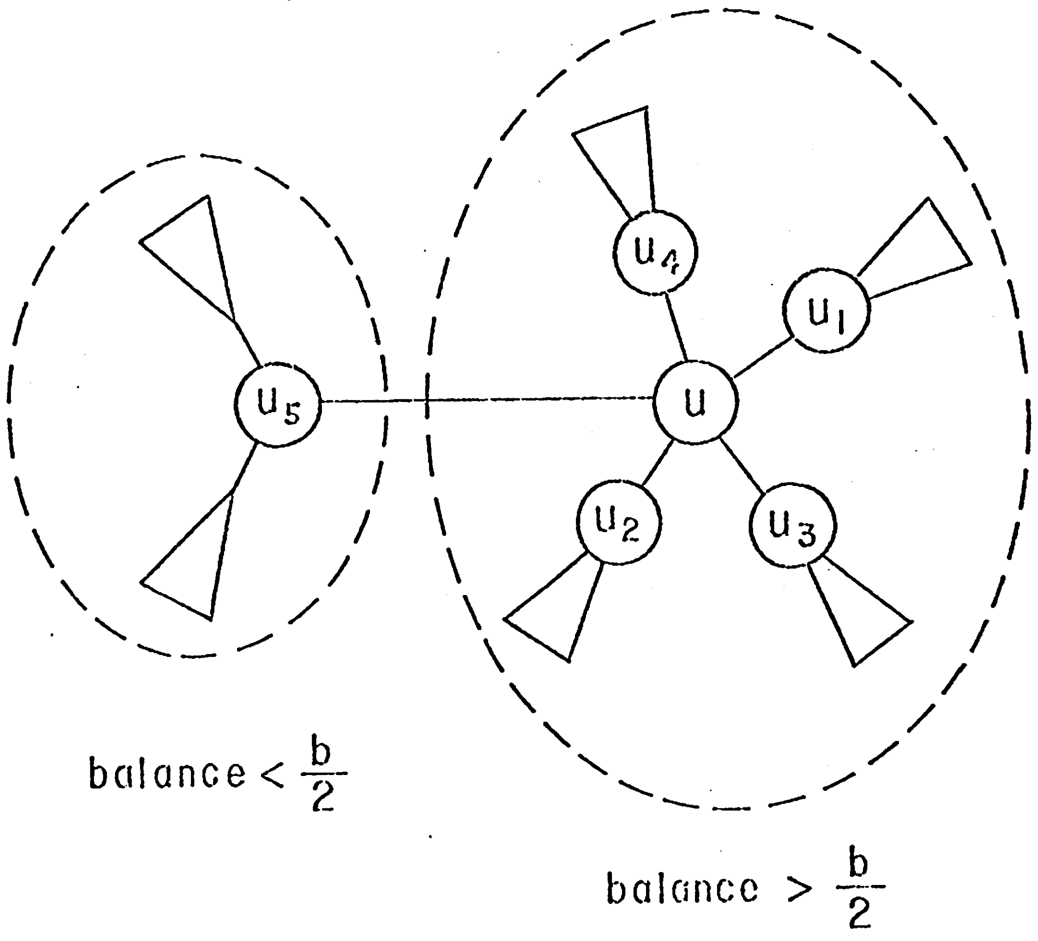


Fig. 7.1

of nodes chosen to be distinguished as Step 3 iterates forms a simple path. Since such a path is finite, the search for a node obeying (7.8) must terminate, and hence will succeed.

For any node u, $\sum_{i=1}^{\ell} \text{bal}(u_i) = b \pm 1$ (depending on whether u is white or red). If (7.8) holds for u then we conclude

$$\sum_{i=1}^{\lfloor\frac{\ell+1}{2}\rfloor} \text{bal}(u_i) \geq \frac{b-1}{2} \tag{7.9}$$

(7.9) implies that the number of subtrees whose nodes are complemented in Step 6 does not exceed $\lfloor\frac{\ell+1}{2}\rfloor$ during one iteration of Steps 2-8.

We have argued that the Tree Coloring Algorithm is correct and will always terminate. It remains to prove that its 2-partitions have at most $\tau(d,n)$ cross arcs.

Proof of Theorem 7.4. Proof is by induction on the balance b. We will show that when the algorithm is applied to a tree with overall balance b, it will create at most $\tau(d,b)$ new cross arcs as it proceeds to color various nodes.

Basis: $b \leq d$ (use $b \leq 3$ for the basis if $d \in \{3,4\}$). New cross arcs are only created at Step 6: if u and u_i are the same color initially, and $\text{sub}(u_i)$ is complemented, then (u,u_i) becomes a cross arc. Upon entering Step 6, $\text{bal}(u_i) \geq 1$ for $i \leq j$, implying that $j \leq \frac{b}{2}$, meaning that at most $\frac{b}{2}$ new cross arcs will be created. Here is an elementary fact about the logarithm function:

For $r \geq 2$ and $1 \leq x \leq r$,

$$\log_r x \geq \frac{x-1}{r-1} \tag{7.10}$$

For the case $d \geq 5$ we find

$$\begin{aligned}\frac{b}{2} &\leq \frac{d+1}{d-1}\,\frac{b-1}{2}+\frac{1}{2} \\ &\leq \frac{d+1}{2}\log_d b + \frac{1}{2} \qquad \text{by (7.10)} \\ &= \tau(d,b)\end{aligned}$$

For $d \in \{3,4\}$ we calculate

$$\frac{b}{2} = \frac{2(b-1)}{2}+\frac{1}{2} \leq 2\log_3 b+\frac{1}{2} = \tau(d,b) \ .$$

<u>Induction Step</u>: The balance is $b > d$ (for $d \in \{3,4\}$ assume $b > 3$).

Suppose that one iteration of the algorithm results in a new coloring with balance b' ($b' < b$). Our inductive hypothesis states that succeeding iterations will produce at most $\tau(d,b')$ new cross arcs.

<u>Claim 7.8</u>. Let p denote the value of j at Step 6. Then $b' \leq \frac{b}{2p-1}$.

<u>Proof</u>. Using the value for j upon entering Step 5 define

$$x = \sum_{i=1}^{j} \mathrm{bal}(u_i) \quad \text{and} \quad w = \mathrm{bal}(u_{j+1}) \ .$$

The algorithm is constructed so that at Step 7

$$b' \leftarrow |b-2B| \leq 2(\tfrac{w}{2}) = w \tag{7.11}$$

(because $x \leq \frac{b}{2}$ and $\sum_{i=1}^{\ell} \geq b-1$ we know $w > 0$). Returning to Step 5, if $x+\frac{w}{2} \geq \frac{b}{2}$ then $p = j$, so that $pw \leq x \leq \frac{b}{2}$, implying with (7.11) that $b' \leq \frac{b}{2p}$. Otherwise $x + \frac{w}{2} < \frac{b}{2}$. Then $p = j+1$, so $(p - \frac{1}{2})w \leq \frac{b}{2}$, implying $b' \leq \frac{b}{2p-1}$, proving Claim 7.8.

Claim 7.9. For $d \geq 5$ define

$$f(p,d) \triangleq p - \frac{d+1}{2} \log_d(2p-1) .$$

Then $f(\frac{d+1}{2},d) \geq f(p,d)$ for all $p \in [2,\frac{d+1}{2}]$.

Proof. $\frac{\partial^2 f}{\partial p^2}$ is positive for $p \in [2,\frac{d+1}{2}]$, so $f(p,d)$ must reach a maximum on the interval $[2,\frac{d+1}{2}]$ at one of the endpoints, 2 or $\frac{d+1}{2}$. $f(\frac{d+1}{2},d) = \frac{d+1}{2} - \frac{d+1}{2} = 0$. $f(2,d) = 2 - \frac{d+1}{2}\log_d 3$. We will show that $f(2,d) \leq f(2,5) < 0$:

Define $g(d) \triangleq \frac{d+1}{\ln d}$.

$$g'(d) = \frac{\ln d - \frac{1}{d} - 1}{\ln^2 d} \tag{7.12}$$

For $d > 1$, the quantity $\ln d - \frac{1}{d}$ increases as d increases. Hence, the numerator of (7.12) will always exceed $\ln 5 - \frac{b}{5} \approx 0.4 > 0$, implying that for $d \geq 5$, $g'(d)$ is positive and $g(d) \geq g(5)$. Therefore $f(2,d) \leq f(2,5) \approx -0.05 < 0$.

Now we apply the inductive hypothesis. Suppose p subtrees $\text{sub}(u_i)$ were complemented in Step 6. We claim that

$$p \in \{1,2\} \Rightarrow b' \leq \frac{b}{3} \tag{7.12a}$$

If $p = 1$ then $\text{bal}(u_1) + \frac{\text{bal}(u_2)}{2} \geq \frac{b}{2}$, implying $\text{bal}(u_1) \geq \frac{b}{3}$, implying $b' \leq \frac{b}{3}$. Claim 7.8 verifies (7.12) for $p = 2$.

If $d \in \{3,4\}$ then (7.12) and the inductive hypothesis imply that the number of new cross arcs is at most

$$2 + 2\log_3 b' + \frac{1}{2} \le 2 + 2\log_3 \frac{b}{3} + \frac{1}{2} = 2 + 2\log_3 b + \frac{1}{2} = \tau(d,b) .$$

Now suppose $d \ge 5$. The inductive hypothesis guarantees that the number of new cross arcs is at most

$$p + \frac{d+1}{2}\log_d b' + \frac{1}{2} \tag{7.13}$$

If $p \ge 2$ then by Claim 7.8

$$\begin{aligned}(7.13) &\le p + \frac{d+1}{2}\log_d \frac{b}{2p-1} + \frac{1}{2} && (7.14)\\ &= f(p,d) + \frac{d+1}{2}\log_d b + \frac{1}{2}\\ &\le f(\tfrac{d+1}{2},d) + \frac{d+1}{2}\log_d b + \frac{1}{2} && \text{by Claim 7.9}\\ &= \frac{d+1}{2}\log_d b + \frac{1}{2} = \tau(d,b) .\end{aligned}$$

If $p = 1$ then $(7.13) \le 1 + \frac{d+1}{2}\log_d \frac{b}{3} + \frac{1}{2} < 2 + \frac{d+1}{2}\log_d \frac{b}{3} + \frac{1}{2}$ which is the value of (7.14) when $p = 2$. □

By modifying some of the steps of the Tree Coloring Algorithm we can produce an algorithm which finds a partition $\{S,\bar{S}\}$ such that $|S| = \frac{n}{k}$ for any $k \ge 2$ which divides n. Steps 1, 2, 3 and 8 are changed to become:

1. Color all nodes white; $b \leftarrow \frac{2n}{k}$.
2. Choose any node x. Assign $u \leftarrow x$. If $\text{bal}(u_i) \le \frac{b}{2}$ for $i = 1$ to ℓ go to Step 4.
3. Put a check mark on node u. Find a node u_i such that $\text{bal}(u_i) > \frac{b}{2}$ and set $u \leftarrow u_i$.

3.1. Exactly one of the u_i's is checked; call it u_t. If $t > \ell$ then do:

```
TEMP ← $u_t$;
for i = t to $\ell$-1 do $u_i$ ← $u_{i+1}$;
$u_\ell$ ← TEMP;
```

Step 3.1 insures that a checked node will not be referenced by Steps 4, 5, or 6.

3.2. If all of the u_i's which are unchecked satisfy $\mathrm{bal}(u_i) \leq \frac{b}{2}$ then go to Step 4. Otherwise go to Step 3.

8. Erase all check marks. If $b \geq 2$ go to Step 2. Otherwise stop ($\frac{n}{k}$ of the nodes are red).

<u>Theorem 7.5</u>. Given a tree with n nodes and maximum node degree d, the Modified Tree Coloring Algorithm (above) can find a partition $\{S,\bar{S}\}$ with $|S| = \frac{n}{k}$ having at most

$$(d-1)\log_{2d-3}\frac{2n}{k}+\frac{1}{2} \text{ cross arcs .}$$

<u>Proof</u>. The correctness of the modified algorithm and the upper bound on the number of cross arcs can be verified in much the same manner as was done for the Tree Coloring Algorithm. Steps 2 and 3 (including 3.1 and 3.2) have been constructed to guarantee that

$$\sum_{i=1}^{\ell-1} \mathrm{bal}(u_i) \geq \frac{b}{2} \tag{7.15}$$

This inequality implies that the proof of Claim 7.8 is still valid, and hence that Claim 7.8 holds for the modified algorithm as well. Another implication of (7.15) is that $p \leq d-1$, where p is the value of j at Step 6. This inequality is weaker than the inequality $p \leq \lfloor\frac{d+1}{2}\rfloor$ implied by (7.8). Hence our upper bound in Theorem 7.5 is weaker than that derived for Theorem 7.4. □

Corollary 7.1. Given a tree with n nodes and maximum degree d, we can find a k-partition having at most

$$(k-1)\left[\frac{1}{2} + (d-1)\log_{2d-3}\frac{2n}{k}\right] \text{ cross arcs .}$$

Proof. We use the chipping method described in Chapter 6. The Modified Tree Coloring Algorithm is applied $k-1$ times. Each time that a set of $\frac{n}{k}$ nodes has been colored red, we set those nodes aside, set $n \leftarrow n-\frac{n}{k}$ and $k \leftarrow k-1$, and apply our algorithm to the remaining white nodes. If those white nodes form a forest which is not connected we add artificial arcs where needed to form a tree. □

The splitting method can be used in place of the chipping method. The resultant upper bound on the number of cross arcs is asymptotically the same as that in Corollary 7.1, but is not quite as good. It is not hard to show that the Tree Coloring Algorithm runs in time $O(n\log n)$ on an n-node tree. Hence we can use our Modified Tree Coloring Algorithm to find a k-partition of a tree in time $O(kn\log n)$.

A complete ternary tree on $n = \frac{3^\ell-1}{2}$ nodes (assume ℓ is a positive even integer) provides an example of a tree for which the number of cross arcs in an optimal 2-partition, m^*, is $\sim \log_3 n$. For this tree $d = 4$, indicating that for the case $d = 4$ the upper bound $\tau(4,n)$ on m^* cannot be improved by more than a factor of two. Fig. 7.2 is an example of our ternary tree for the instance $\ell = 4$.

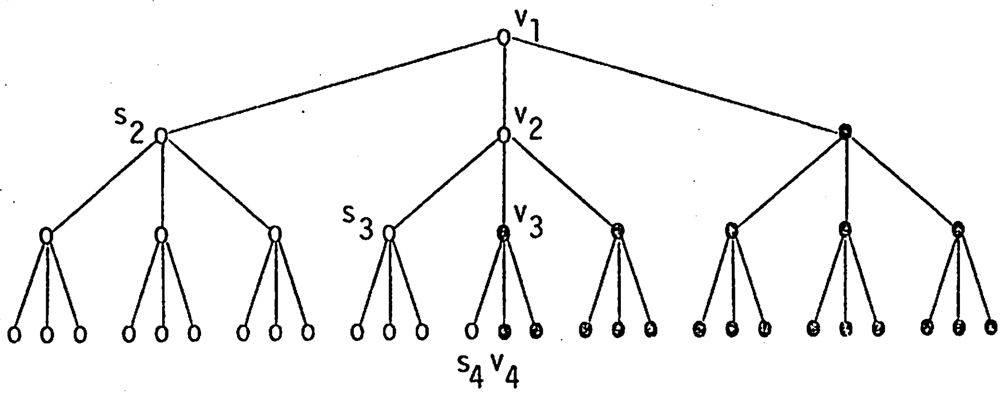


Fig. 7.2

For convenience we have labelled the root v_1, and defined v_{i+1} to be node v_i's middle son, for $i = 1$ to $\ell-1$. We define s_{i+1} to be the left son of v_i, for $i = 1$ to $\ell-1$. We claim

Theorem 7.6. An optimal 2-partition of a complete ternary tree on $n = \frac{3^\ell-1}{2}$ nodes has at least $\lceil \log_3 n \rceil - \lceil \log_3 \log_3 n \rceil$ cross arcs.

Proof. Suppose the nodes of our tree have been colored either black or white so as to define an optimal 2-partition. On each level, if a white node is to the right of a black node, complement the colors of both nodes, until on any level all black nodes will be to the right of all white nodes. This recoloring preserves the optimality of the induced 2-partition, since on a given level the new arrangement maximizes both the number of black nodes which have black fathers, and the number of black sons of black nodes. For definiteness we will assume that more than half of the leaves are black (there are an odd number of leaves).

Claim 7.10. No white node of our tree has three black sons, and no black node has three white sons.

If Claim 7.10 is false we can improve our 2-partition: Suppose a white node has three black sons. Color the white node black, and color some black leaf in the tree white. This reduces the total number of cross arcs by at least one. Claim 7.10 implies that all nodes to the right of the v_i's in our tree are black. Also, v_ℓ must be black. This leaves only $\frac{\ell}{2}-1$ black nodes unaccounted for (Fig. 7.2 illustrates what such a coloring looks like).

If all of the leaves of a subtree rooted at s_i are white, for $2 \leq i \leq \ell-1$, then by Claim 7.10, s_i must also be white. All of the leaves to the left of v_ℓ, except for up to $\frac{\ell}{2}-1$ of them, must be white. This means that the nodes $s_2, s_3, \ldots, s_{\ell-i}$ are white, for any i obeying $\frac{\ell}{2}-1 \leq \frac{3^i-1}{2}$. Hence

"at least $(\ell - \lceil \log_3(\ell-1) \rceil - 1)$ of the nodes labelled s_i are white" (7.16)

For every white s_j there is a corresponding cross arc, because the right son of node v_{j-1} is black. There is also at least one cross arc connecting a pair of nodes along the path $v_1, v_2, \ldots, v_\ell$. Hence (7.16) implies that

$$\begin{aligned} m^* &\geq \ell - \lceil \log_3(\ell-1) \rceil \\ &\geq \lceil \log_3 n \rceil - \lceil \log_3 \log_3 n \rceil \ . \end{aligned}$$

This proves Theorem 7.6. □

Note that if we color everything to the right of the v_i's black, color $v_{\frac{\ell}{2}+1}, v_{\frac{\ell}{2}+2}, \ldots,$ and v_ℓ black, and color all other nodes white then the induced 2-partition has ℓ cross arcs, implying

$$m^* \leq \log_3(2n+1) \ .$$

Our Tree Coloring Algorithm may produce a 2-partition which is far from optimal. Consider the tree represented in Fig. 7.3.

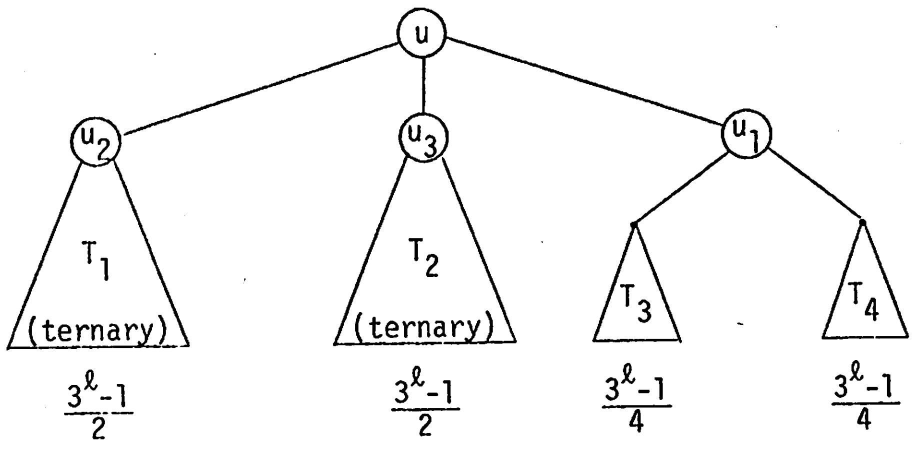


Fig. 7.3

Here we assume that T_1 and T_2 are complete ternary trees on $\frac{3^\ell-1}{2}$ nodes, and T_3 and T_4 are any trees having $\frac{3^\ell-1}{4}$ nodes each (ℓ is even). A solution which colors T_1 and T_3 white, and T_2 and T_4 red, will have three cross arcs.

When the Tree Coloring Algorithm commences, $b = n = (3^\ell-1)\frac{3}{2}+2$, and u is the distinguished node. Our algorithm will first complement sub(u_1), and then either u_2 or u_3 will become the distinguished node. Suppose for definiteness that u_3 is distinguished. The balance of all nodes not in the subtree T_2 is zero, so our algorithm from this point on will never attempt to alter the colors of nodes outside of T_2, except to complement them as a group. Hence it is forced to 2-partition the ternary tree T_2, requiring at least $\ell - \lceil \log_3 \ell-1 \rceil$ cross arcs.

A more detailed analysis reveals that the Tree Coloring Algorithm finds a 2-partition of an ℓ-level ternary tree which has $2\ell - 2$ cross arcs. Thus, while the optimal solution to the tree in Fig. 7.3 has three or fewer cross arcs, our algorithm's solution had about $2\log_3 n$ cross arcs.

We now outline a dynamic programming algorithm which will find an optimal 2-partition of a tree. Our abstracted tree (in Fig. 7.4) consists of a root r and d subtrees $S_1, S_2, \ldots, S_d$, where we let S_i denote both a subtree and the nodes of that subtree.

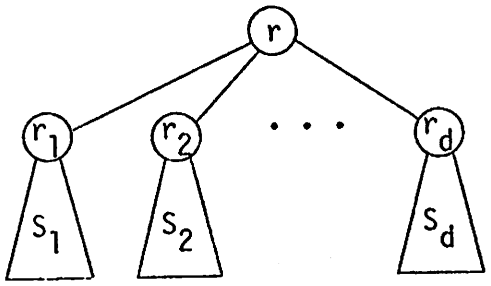


Fig. 7.4

We define r_i to be the root of the subtree S_i, and let $n_i = |S_i|$, so that $n = \sum_{i=1}^{d} n_i + 1$. Our algorithm will always assign the indices of our labels so that $n_1 \le n_2 \le \cdots \le n_d$.

Given an n-node tree T, for a node r in the tree and each j such that $0 \le j < n$ we define $V^*(r,j)$ to be the set of partitions $\{S,\bar{S}\}$ of T which minimize the number of cross arcs subject to $|S| = j$ and $r \in \bar{S}$. Next define a <u>complete set of partitions</u> of an n-node tree rooted at r to be any set (with cardinality n) containing one partition from each $V^*(r,j)$, for $j = 0$ to $n-1$.

The dynamic programming algorithm starts by choosing any node r and making it the root, as in Fig. 7.4. It then finds a complete set of partitions for each of the subtrees S_i rooted at r_i. Next it computes a complete set of partitions for the subtree $S_1 \cup \{r\}$ noted at r. One-by-one more subtrees are added: we find the complete set of partitions of the subtree $S_1 \cup S_2 \cup \cdots \cup S_i \cup \{r\}$ rooted at r, for $i = 2$ to d. The last complete set of partitions computed includes an optimal 2-partition of the tree.

Now let us define the algorithm more formally. Let an <u>optimal partition</u> denote a partition with a minimum number of cross arcs among all admissible partitions. For all pairs of integers i and j such that $1 \leq i \leq d$ and $0 \leq j \leq n_i-1$ define

$\{L_i^j,R_i^j\} \triangleq$ "an optimal partition of S_i, given that $|L_i^j| = j$ and $r_i \in R_i^j$"

$c_i^j \triangleq \#(L_j^i;R_j^i)$

$\{U_i^j,V_i^j\} \triangleq$ "an optimal partition of the nodes in $S_i \cup \{r\}$, given that $|U_i^j| = j$ and $r \in V_i^j$"

For integers i and j such that $1 \leq i \leq d$ and $0 \leq j \leq \sum_{\ell=1}^{i} n_\ell$ define

$\{X_i^j,Y_i^j\} \triangleq$ "an optimal partition of the nodes in $\bigcup_{\ell = 1 \text{ to } i} S_\ell \cup \{r\}$, given that $|X_i^j| = j$ and $r \in Y_i^j$"

Finally, define for $0 \leq j \leq n-1$,

$\{L^j,R^j\} \triangleq$ "an optimal partition of the tree, given that $|L^j| = j$ and $r \in R^j$"

Our algorithm is designed to produce the set of partitions $\{L^j,R^j\}$ and scalars c^j, for $j = 0$ to $n-1$. The definitions imply that $L^j = X_d^j$ for all j, and that $\{L^{n/2},R^{n/2}\}$ is an optimal 2-partition of the tree. Here is the algorithm:

Algorithm D

Assume $n > 1$, guaranteeing that r has at least one subtree.

```
1.  "label the nodes adjacent to r so that $n_1 \le n_2 \le \cdots \le n_d$."
    for $i = 1$ to $d$ do begin
        if $n_i = 1$ then begin
            $L_i \leftarrow \emptyset$; $R_i^0 \leftarrow \{r_i\}$; $c_i^0 \leftarrow 0$;
        end
        else "make a recursive call to algorithm D to generate $L_i^j$,
            $R_i^j$ and $c_i^j$ for $0 \le j \le n_i - 1$"
2.  for $i = 1$ to $d$ do begin
        $c_i^{n_i} \leftarrow \infty$;
        for $j = 0$ to $n_i$ do
            if $c_i^j \le c_i^{n_i - j} + 1$ then begin
                $U_i^j \leftarrow L_i^j$; $V_i^j \leftarrow R_i^j \cup \{r\}$; $w_i^j \leftarrow c_i^j$;
            end
            else begin
                $U_i^j \leftarrow R^{n_i - j}$; $V_i^j \leftarrow L_i^{n_i - j} \cup \{r\}$;
                $w_i^j \leftarrow c_i^{n_i - j} + 1$;
            end
    end
3.  for $j = 0$ to $n$ do begin
        $X_1^j \leftarrow U_1^j$; $Y_1^j \leftarrow V_1^j$; $z_1^j \leftarrow w_1^j$;
    end
    for $i = 2$ to $d$ do
        for $j = 0$ to $\sum_{\ell=1}^{i} n_\ell$ do begin
            "find a $k_0$ such that $z_{i-1}^{k_0} + w_i^{j-k_0} = \min_{0 \le k \le j} \{z_{i-1}^{k} + w_i^{j-k}\}$";
            $X_i^j \leftarrow X_{i-1}^{k_0} \cup U_i^{j-k_0}$; $Y_i^j \leftarrow Y_{i-1}^{k_0} \cup V_i^{j-k_0}$;
            $z_i^j \leftarrow z_{i-1}^{k_0} + w_i^{j-k_0}$;
            end
        for j = 0 to n-1 do begin
            $L^j \leftarrow X_d^j$; $R^j \leftarrow Y_d^j$; $c^j \leftarrow z^j$;
        end
```

The proof that algorithm D is correct consists of straightforward verification that the partitions built in Steps 2 and 3 actually conform to our previous definitions of them, and that $c_i^j = \#(L_i^j;R_i^j)$, $w_i^j = \#(U_i^j;V_i^j)$, and $z_i^j = \#(X_i^j;Y_i^j)$.

Theorem 7.7. Algorithm D finds an optimal 2-partition of an n-node tree in time $O(n^3)$.

Proof. We will find an upper bound on the number of elementary steps performed by algorithm D, where an elementary step is defined to be any operation which can be executed in constant time. We assume that the set operation $S'' \leftarrow S \cup S'$ can be performed in time $O(|S| + |S'|)$, where S, S', and S'' are subsets of the n nodes. (This can be accomplished by using an ordered linked-list representation for the sets.)

Step 2 can be performed in time $O(\sum_{i=1}^{d} n_i^2) = O((n-1)^2)$ (using $\sum_{i=1}^{d} n_i = n-1$ and $n \geq 2$). Step 3 takes not more than $4\sum_{i=1}^{d}(n_1 + n_2 + \cdots + n_i)^2 + O((n-1)^2)$ elementary steps. Hence there exists a constant $c_2 > 0$ such that Steps 2 and 3 together use at most

$$c_2 \sum_{i=1}^{d}(n_1 + n_2 + \cdots + n_i)^2 \text{ steps} . \qquad (7.17)$$

At Step 1, if $n_i = 1$ we process S_i in constant time c_1'. Otherwise we assume inductively that for some $c_1 \geq \max\{c_1', c_2\}$, a recursive call for subtree S_i takes time at most $c_1 n_i^3$. Hence we can infer that Step 1 uses at most

$$c_1 \sum_{i=1}^{d} n_i^3 \text{ steps .} \tag{7.18}$$

From (7.17) and (7.18) we can bound the total running time by

$$c_1 \sum_{i=1}^{d} n_i^3 + c_2 \sum_{i=1}^{d} (n_1 + n_2 + \cdots + n_i)^2 \tag{7.19}$$

We will show that (7.19) cannot exceed $c_1 n^3$, thus justifying the inductive assumption and proving our theorem. By definition and design

$$0 \leq n_1 \leq n_2 \leq \cdots \leq n_d \quad \text{and} \quad \sum_{i=1}^{d} n_i = n - 1 \tag{7.20}$$

Let $(n_1, n_2, \ldots, n_d)$ be a vector satisfying (7.20). Assume that only the last $k+1$ n_i's are non-zero, for $0 \leq k \leq d-1$. For any fixed k the term (7.18) is maximized when $n_{d-k} = n_{d-k+1} = \cdots = n_{d-1} = 1$ and $n_d = n - k$. The term (7.17) is maximized when $n_{d-k} = n_{d-k+1} = \cdots = n_d = \frac{n}{k+1}$. Hence

$$(7.19) \leq \max_{0 \leq k \leq d-1} \{c_1[(n-k-1)^3 + k] + c_2(k+1)(n-1)^2\} \tag{7.21}$$

<u>Claim 7.11</u>. (7.21) $\leq c_1 n^3$

Proof. Fix $m > 1$ and for $k \in [0,m-1]$ define

$$f(k) \triangleq c_1(m-k)^3 + k + c_2(k+1)m^2 .$$

Then $f''(k) = 6c_1(m-k) > 0$, implying

$$\begin{aligned} f(k) &= \max\{f(0),f(n-1)\} \\ &= \max\{c_1m^3+c_2m^2,\ c_1m+c_2m^3\} \\ &= c_1m^3 + c_2m^2 \quad \text{because} \quad c_1 \geq c_2 . \end{aligned}$$

Then we have

$$(7.21) \leq c_1(n-1)^3 + c_2(n-1)^2 \leq c_1n^3 .$$

This proves the claim and the theorem. □

Dynamic programming can also be used to find k-partitions of trees for $k > 2$, but generalizing the approach used here produces an algorithm which is quite slow for all but the smallest values of k. For example, a generalized version of Step 1 must produce on the order of n^{k-1} different partitions analogous to the $n-1$ $\{L_i^j,R_i^j\}$'s algorithm D generates.

The problem of finding a general partition (defined in Chapter 2) of a tree is apparently much easier to solve. Kundu and Mistra [22] show how to find a general partition of an n-node tree in time $O(n)$.

7.4 The Case of Planar Graphs

Although the Simple Max Cut Problem (section 2.2) is NP-complete for general graphs, a polynomial-time algorithm exists for solving Simple Max Cut on planar graphs (see [10]). Whether or not there is a polynomial-time algorithm to find an optimal 2-partition of a planar graph is an open problem. However, for the case of an n-node planar graph G where the maximum node degree is held below a fixed bound, a fast algorithm is available which finds quite good 2-partitions of G, in the sense that $m^*(G) = o(n)$.

A star graph with $n-1$ arcs (defined in section 7.2) provides an example of a planar graph for which $m^* \geq \frac{N}{2}$. Hence we conclude that bounding the maximum degree of any node is necessary if we are to derive a sub-linear upper bound on m^*. In [26] Lipton and Tarjan have provided us with the means for partitioning planar graphs. Their result is:

> Let G be any n-vertex planar graph. The vertices of G can be partitioned into three sets A, B, C, such that no edge joins a vertex in A with a vertex in B, neither A nor B contains more than $\frac{2n}{3}$ vertices, and C contains no more than $2\sqrt{2}\sqrt{n}$ vertices.

Lipton and Tarjan have an algorithm which finds the desired sets A, B, and C in linear time. We have an immediate corollary:

<u>Corollary 7.2</u>. Let G be an n-node planar graph with maximum node degree d. A linear-time algorithm exists which finds a partition $\{S,\bar{S}\}$ of G such that $|S| \leq |\bar{S}| \leq \frac{2n}{3}$ and $\#(S;\bar{S}) \leq 2\sqrt{2}\, d\sqrt{n}$.

<u>Proof</u>. Given G, find A, B, and C, and suppose $|A| \leq |B|$. Assign $S \leftarrow A$ and $\bar{S} \leftarrow B$. If $|S| + |C| \leq \frac{n}{2}$ set $S \leftarrow S \cup C$. Otherwise choose any $\frac{n}{2} - |S|$ nodes in C and add them to S. Then add the remaining nodes of C to $\bar{S}$. □

We need an algorithm which can partition a planar graph into equal-sized sets. Here is an algorithm which utilizes the above algorithm as a subroutine to find a 2-partition of a planar graph. Its running time is $O(n \log n)$.

Algorithm P

```
1.  $P_1 \leftarrow P_2 \leftarrow \emptyset$;  $R \leftarrow N$  (the set of all nodes).
2.  Find a partition $\{S,\bar{S}\}$ of $R$ such that $|S| \le |\bar{S}| \le \frac{2|R|}{3}$
    and $\#(S;\bar{S}) \le 2\sqrt{2}\, d|R|$.
3.  if $|P_1| \le |P_2|$ then $P_1 \leftarrow P_1 \cup S$
    else $P_2 \leftarrow P_2 \cup S$.
4.  if $|P_1| = \frac{n}{2}$ or $|P_2| = \frac{n}{2}$ then begin
       if $|P_1| < |P_2|$ then $P_1 \leftarrow P_1 \cup \bar{S}$
       else $P_2 \leftarrow P_2 \cup \bar{S}$;
       stop;
       end
5.  $R \leftarrow \bar{S}$
6.  Go to Step 2.
```

To make it obvious that algorithm P is correct we note that at the beginning of Step 3, $n - (|P_1| + |P_2|) \ge 2|S|$, implying that $|S| + \min\{|P_1|,|P_2|\} \le \frac{n}{2}$. Hence, the cardinalities of P_1 and P_2 never exceed $\frac{n}{2}$.

Theorem 7.8. Let G be an n-node planar graph with maximum node degree d. Algorithm P finds a 2-partition of G having at most $16d\sqrt{n}$ cross arcs.

Proof. First note that the size of the set R shrinks by at least a third after each iteration of Steps 2-5. Hence the number of cross arcs generated during Step 2 on the i^{th} iteration is at most

$$2\sqrt{2}\ d\sqrt{n(\tfrac{2}{3})^{i-1}} \qquad (7.22)$$

After $t = \lceil \log_{3/2} n \rceil$ iterations $|R| \leq n(\frac{2}{3})^t \leq n \cdot n^{-1} = 1$, so at most $O(\log n)$ iterations of Steps 2-5 can occur. From (7.22) we know that the total number of cross arcs in the final 2-partition $\{P_1, P_2\}$ is at most

$$\begin{aligned} 2\sqrt{2}\ d\sum_{i=0}^{t-1}\sqrt{n(\tfrac{2}{3})^i} &\leq 2\sqrt{2}\ d\sqrt{n}\sum_{i=0}^{\infty}\sqrt{\tfrac{2}{3}}^{\,i} \\ &= 2\sqrt{2}\ d\sqrt{n}(1-\sqrt{\tfrac{2}{3}})^{-1} \\ &\leq 16d\sqrt{n}\ . \end{aligned}$$

□

A k-partition of a planar graph having $o(n)$ cross arcs can be found by combining algorithm P with the splitting method (described in Chapter 6). Suppose k is a power of 2. Using Theorem 7.8, it is easy to calculate a limit on the number of cross arcs which are produced by this method. There will be at most

$$\begin{aligned} &16d[\sqrt{n} + 2\sqrt{\tfrac{n}{2}} + 4\sqrt{\tfrac{n}{4}} + \cdots + \tfrac{k}{2}\sqrt{\tfrac{2n}{k}}] \\ &= 16d\sqrt{n}[\sqrt{2}^0 + \sqrt{2}^1 + \cdots + \sqrt{2}^{\log_2 k - 1}] \\ &= 16d\sqrt{n}[\frac{\sqrt{k}-1}{\sqrt{2}-1}] \\ &= O(d\sqrt{kn}) \ \text{ cross arcs} \end{aligned}$$

7.5 Concluding Remarks

A person in possession of a real-life k-partition problem would be for the most part encouraged by the results we have obtained in this chapter. Our random-graph analysis in Chapter 4 indicated that for values of k much over two almost all of the arcs in a k-partition will be cross arcs. However, we have shown that for certain graphs with special structure, k-partitions with relatively few cross-arcs exist.

Looking at Chapter 1, we find several applications which might exhibit the special structure we are looking for. A graph which represents the flow-of-control of a computer program might be expected to be tree-like, if we neglect those branches to basic system subroutines (e.g. to sin( ), abs( ), time-of-day( ), etc.). The accessing paths for some database applications form a tree. A large electrical network which must be broken into pieces for analysis may correspond to a nearly planar graph. Finally, if no other structure is apparent, the maximum node degree of the graph to be partitioned may be small. The notion that a small maximum node degree often implies the existence of a k-partition with relatively few cross arcs has pervaded this chapter.

While graphs exhibiting special structure are often more tractable with respect to partitioning than unstructured graphs, it appears that only specially-tailored partitioning algorithms can exploit this structure. Recall that Theorem 3.2 indicates that the performance of an iterative-improvement algorithm is very poor on certain trees. It would be an interesting experiment to test the performance of such an algorithm on a variety of trees and planar graphs.

Given an n-node graph G, let us label the class of all k-partitions of G $\underline{EQ(\frac{n}{k})}$, and let $\underline{LE(\frac{n}{k})}$ be the class of partitions $\{P_1, P_2, \ldots, P_\ell\}$ (for arbitrary ℓ) such that $|P_i| \leq \frac{n}{k}$ for $i = 1$ to ℓ. $EQ(\frac{n}{k})$ corresponds to the solution space for the k-Partition Problem; $LE(\frac{n}{k})$ represents the partitions which are feasible for a General Partition Problem (defined in section 2.1) with $W = \frac{n}{k}$. $EQ(\frac{n}{k})$ is properly contained in $LE(\frac{n}{k})$ when $n > k$, and an optimal k-partition of G may have many more cross arcs than a corresponding optimal general partition. For many applications the problem to be solved more closely resembles the General Partition Problem than the k-Partition Problem.

The two problems are similar enough that all of the iterative-improvement algorithms for partitioning a graph which have been discovered can be adapted to solve either problem (in section 6.2 we show how to convert all of our k-partition algorithms so that they produce general partitions). We predict as a consequence that on most graphs the solutions found by an iterative-improvement algorithm working from the whole space $LE(\frac{n}{k})$ will be little better than those found by an algorithm which only considers partitions in $EQ(\frac{n}{k})$.

The k-Partition and General Partition Problems lose their similarity when the only graphs to be partitioned are trees. The algorithm developed in [22] to find an optimal general partition of a tree in linear time apparently can't be adapted to find k-partitions. Furthermore, it can be easily shown that if the maximum degree of our tree is d, then an optimal general partition (from the space $LE(\frac{n}{k})$) always has fewer than kd cross arcs in its solution. This bound is substantially lower than the $O(d \log n)$ bound for k-partitions of a tree.

Bibliography

[1] Aho, A., Hopcroft, J.E. and Ullman, J.D., The Design and Analysis of Computer Algorithms, Addison-Wesley (1974), 372-374.

[2] Angluin, D. and Valiant, L., "Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings," Ninth Annual Symposium on Theory of Computing (May 1977).

[3] Brown, J.A. et al., "Design Automation and the Wrap System," Proc. Fifth Design Automation Workshop (1968).

[4] Edmonds, J. and Karp, R., "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," JACM (April 1972).

[5] Erdös, P. and Renyi, A., "On Random Graphs, I," Publications Mathematicae 6 (1959), 290-297.

[6] Even, S. and Shiloah, Y., "NP-Completeness of Several Arrangement Problems," Tech. Report #43, Department of Computer Science, Technion, I.I.I., Haifa, Israel (June 1975).

[7] Garey, M.R. et al., "Some Simplified NP-Complete Problems," Proc. Sixth Annual ACM Symp. on Theory of Computing (1974), 47-63.

[8] George, J.A., "Nested Dissection of a Regular Finite Element Mesh," SIAM J. Numerical Analysis (1973), 345-363.

[9] Grimmett, G.R. and McDiarmid, C., "On Coloring Random Graphs," Math. Proc. Cambridge Phil. Soc. 77 (1975), 313-324.

[10] Hadlock, F., "Finding a Maximum Cut of a Planar Graph in Polynomial Time," SIAM J. Computing 4, 3 (September 1975), 221-225.

[11] Hanan, M. et al., "Iterative-Interactive Technique for Logic Partitioning," IBM J. Research & Development (July 1974), 328-337.

[12] Hoel, P., Port, S. and Stone, C., Introduction to Probability Theory, Houghton Mifflin Co., Boston (1971).

[13] Johnson, D.S., "Approximation Algorithms for Combinatorial Problems," JCSS 9 (1974), 256-278.

[14] Karp, R.M., "Reducibility Among Combinatorial Problems," in Complexity of Computer Computations (eds. R.E. Miller and J.W. Thatcher), Plenum Press (1972), 85-103.

[15] ______, "The Probabilistic Analysis of Some Combinatorial Search Algorithms," Memo No. ERL-M581, University of California, Berkeley (1976).

[16] Kernighan, B., "Some Graph Partitioning Problems Related to Program Segmentation," Ph.D. Thesis, Princeton University (January 1969).

[17] Kernighan, B. and Lin, S., "An Efficient Heuristic Procedure for Partitioning Graphs," Bell Sys. Tech. J. 49, 2 (February 1970), 291-307.

[18] Knuth, D.E., The Art of Computer Programming, Vol. III, Addison-Wesley (1973), 99.

[19] Kodres, U., "Partitioning and Card Selection," Chapter 4 of Design Automation of Digital Systems: Theory and Techniques (ed. M.A. Breuer), Prentice Hall, Vol. 1 (1972).

[20] Kozyrev, V. and Korshunov, A., "On the Size of a Cut in a Random Graph," Problemy Kibernet. 29 (1974), 27-62 (Russian).

[21] Kral, J., "To the Problem of Segmentation of a Program," Information Processing Machines (1965), 140-149.

[22] Kundu, S. and Misra, J., "A Linear Tree Partitioning Algorithm," SIAM J. Computing 6, 1 (March 1977), 151-154.

[23] Lawler, E.L., "Electrical Assemblies with a Minimum Number of Interconnections," IEEE Trans. Electronic Computers (correspondence), EC-11, 1 (February 1962), 86-88.

[24] ______, "Cutsets and Partitions of Hypergraphs," Networks 3 (1973).

[25] Lin, S., "Computer Solutions to the Travelling Salesman Problem," BSTJ 44, 10 (December 1965), 2245-2270.

[26] Lipton, R. and Tarjan, R., "Applications of a Planar Separator Theorem," Eighteenth Annual Symp. on Foundations in Computer Science (August 1977).

[27] Lukes, J.A., "Combinatorial Solutions to Partitioning Problems," Ph.D. Thesis, Stanford University (1972).

[28] ______, "Efficient Algorithm for the Partitioning of Trees," IBM J. Research & Development 18, 3 (1974), 217.

[29] ______, "Combinatorial Solution to the Partitioning of General Graphs," IBM J. Research & Development (1975), 170.

[30] Margulis, G.A., "Explicit Construction of Concentrators," Problemy Peredachi Informatsii 9, 1 (1973), 71-80 (Russian).

[31] Olver, F.J.W., Asymptotics and Special Functions, Academic Press (1974), 180.

[32] Sahni, S. and Gonzalez, T., "P-Complete Approximation Problems," JACM 23 (1976), 555-565.

[33] Schweikert, D. and Kernighan, B., "A Proper Model for the Partitioning of Electrical Circuits," Proc. Eleventh Design Automation Workshop (1974), 57-62.