An Algorithm for Designing Rings for Survivable Fiber Networks

Ondria J. Wasem, Member IEEE
Bellcore, Red Bank

Key Words — Cycles in graphs, Network topology

Reader Aids — Purpose: Present an algorithm for finding cycles in graphs containing specific nodes
Special vocabulary needed for explanations: Graph theory, fiber optic network technology
Special math needed to use the results: None
Results useful to: Network planners, graph theoreticians

Abstract — The introduction of SONET technology opens opportunities for survivable network architectures, such as self-healing rings, which can improve telecommunication reliability. This paper presents an algorithm for routing fiber around a ring in a network, when the network nodes, links, connectivity, and which offices are to be placed on that ring together are known. The algorithm aids automated survivable network planning.

The algorithm was programmed in C, and run on a SPARCstation. Computation times on 47 examples of feasible and infeasible rings were reasonable. Overall, the average, minimum, and maximum runtimes were 0.41 sec., 0.06 sec., and 2.93 sec., respectively. Since the largest example network used in these results, 167 offices and 240 links, is the size of a typical large LATA network, the algorithm runs fast enough for the intended application.

In most cases, the ring routing problem cannot be solved by traveling salesman algorithms. However, under certain conditions, the problem degenerates to the traveling salesman problem, and the ring routing algorithm degenerates to the nearest neighbor method of solving that problem.

1. INTRODUCTION

Planning for survivability in a fiber network increases telecommunication reliability by providing for recovery from cable cuts and central office failures. Such planning should take into account survivable network architectures and the technologies that support them. SONET technology is being deployed, and this deployment enables the implementation of self-healing ring (SHR) architectures [1-3]. The importance of SHRs stems from their ability to maintain all communication in the event of a cable cut, and a portion of communication in the event of an office failure. In addition to increasing the survivability of a network, appropriate placement of rings can also lower capital costs from the cost of conventional schemes [1,3].

One issue in network planning is topology design, and one aspect of topology design is determining how to route fibers around a ring, once the offices to be multiplexed on the ring (the ring offices) have been chosen. It may be necessary to route fiber through offices which are not on the ring, in order to include all of the ring offices. The algorithm presented in this paper determines how to route a ring, given the offices in the network, which offices are on the ring, and the links available for the fiber. Routing rings is only a part of the larger problem of survivable network planning; therefore, this algorithm should be used in conjunction with other network-planning algorithms to design a complete network.

Section 2 describes the ring routing algorithm, while section 3 presents results of how fast it runs. Section 4 compares the stated problem with the traveling salesman problem. Finally, section 5 summarizes the work.

2. DESCRIPTION OF ALGORITHM

This is a high level overview of how the ring routing algorithm, BUILD_RINGS, works. Ref [4] contains more detail. Figure 1 illustrates how BUILD_RINGS routes a ring in a simple example. Figure 1a, shows a network with seven offices. Offices 1 & 5 are hubs. The example problem is to route a ring through offices 1-4.

First, BUILD_RINGS uses a simple method to find a ring. If it fails, then a longer more reliable algorithm is used. For the simple method, BUILD_RINGS finds a hub on the ring and the ring office furthest from it, and then uses two iterations of the Dijkstra shortest-path algorithm to find the two shortest, link disjoint paths (paths sharing no links) between the hub and the office. If the two paths are node disjoint (share no nodes), and the cycle formed by them contains all of the ring offices, then the ring is routed that way.

If the paths are not disjoint, or the cycle does not contain all of the ring offices, then BUILD_RINGS begins a more
intensive search for a ring routing, beginning with a depth-first search for paths between all pairs of ring offices, and containing only two ring offices. There are some rules for when to stop looking for these paths. For one thing, there is a threshold which upperbounds the lengths of paths which will be considered in constructing the ring. At first, this threshold is low, so that BUILD\_RINGS will look for short paths first, but the threshold increases if necessary. There is also a limit on the number of hops (links) in a path. The default value of the hop limit is three, but the user can override this. Figure 2 shows four depth-first search trees which together contain all paths with fewer than three hops between ring offices in the network of figure 1. The root and a subset of the leaves of each tree are shaded to indicate the paths found. Figure 1b shows links, corresponding to the paths in figure 2, as dotted lines.

3. COMPUTATIONAL EFFICIENCY

This section describes both the computational complexity of BUILD\_RINGS and the time required to run the algorithm on a set of example networks.

3.1 Computational Complexity

There are two parts to BUILD\_RINGS:

- Finding paths between ring nodes
- Choosing a subset of these paths to form a ring.

The complexities of these two parts of the algorithm are examined separately.

Notation

d \quad \text{degree (number of neighbors) of each office}

h \quad \text{hop limit}

m \quad \text{number of ring offices}

K \quad \text{maximum number of paths that can be found between ring offices}

First, consider the complexity of finding all paths within the hop limit between all pairs of ring offices. Sample depth-first search trees used in constructing the paths are illustrated in figure 2. In each step of the algorithm, a node of a tree is either created or visited. Assuming the root of a tree is at depth 0, the depth of a tree cannot exceed h. At depth i of a tree, there are at most \( d^i \) nodes. As the algorithm proceeds, each node is created once, and each node at depths 0 through \( h-1 \) is visited \( d \) times. There are \( m \) trees. The complexity, \( C \), of the path-finding algorithm in a network in which each office has degree \( d \) is computed as follows:

\[
C = \text{number of steps} = m \sum_{\text{nodes in tree}} \text{number of times node visited}
\]

\[
= m \left[ (d+1) \sum_{i=0}^{h-1} d^i + d^h \right] = m \left[ \frac{d+1}{d-1} (d^h - 1) + d^h \right].
\]

Therefore, the complexity of finding the paths emanating from one ring office is \( O(md^h) \).

Between any pair of ring offices, there is at most 1 1-hop path, \( d \) 2-hop paths, \( d(d-1) \) 3-hop paths, \( d(d-1)^2 \) 4-hop paths, etc. There are \( m(m-1)/2 \) ring office pairs. Therefore,

\[
K = \frac{m(m-1)}{2} \left( 1 + d \sum_{i=0}^{h-2} (d-1)^i \right)
\]

\[
= \frac{m(m-1)}{2} \left( \frac{d(d-1)^{h-1} + d - 2}{d-2} \right).
\]
As explained in section 4, the ring routing problem is a generalization of the traveling salesman problem, which is an NP-complete problem. One can therefore conjecture that the problem of finding the shortest ring route is also NP-complete. The algorithm in section 2 does not always find the shortest route. The following derivation provides an approximation to the complexity of the algorithm.

Of the $K$ paths, $m$ must be chosen to construct a ring; therefore, at most $\binom{K}{m}$ combinations must be tried. Normally, not every combination is tried, since BUILDINGS does not seek the shortest route, but rather stops at the first route found. In addition, some combinations prove infeasible before they are complete, due to paths in the combination looping (closing the ring before all of the ring offices are included) or not being node disjoint. Nonetheless, since each combination requires that $m$ paths be chosen, there are at most $m \binom{K}{m}$ choices to be made. As each path is chosen, it must be checked to see if it is node disjoint from other chosen paths. Since that path has at most $h$ hops, there are at most $h - 1$ nodes in it that must be checked to see whether they have been used in other paths. This yields at most $h - 1$ steps for each choice. Therefore, complexity, $C$, of building up the ring from the paths is:

$$C = (h - 1)m \binom{K}{m}$$

$$= (h - 1)m \left( \frac{m(m - 1)}{2} \left( \frac{d(d - 1)^{h - 1} + d - 2}{d - 2} \right) \right)$$

(4)

3.2 Runtime Results

BUILDINGS was programmed in C, and run on a SPARCstation. This section presents results of running the program on 47 example rings; 26 of the rings are feasible, and 21 are not. It was important to test BUILDINGS on infeasible rings as well as feasible rings, since the program must terminate in a reasonable amount of time in either case.

The examples are drawn from 7 networks, of sizes ranging from 11 offices and 16 links, to 167 offices and 240 links. The feasible rings have from 4-7 nodes, and can all be constructed within a 3-hop limit. The infeasible rings all have 4 nodes, and cannot be constructed even with a 5-hop limit. The average time to find a ring was 0.41 seconds. Among feasible rings, the average time to find a ring was 0.24 seconds, and among infeasible rings, the average time to find a ring was 0.61 seconds.

Figures 3-5 show scatterplots of results for the examples. Figure 3 shows runtime in seconds vs the number of offices in an example ring for the feasible rings. Based on this plot, runtime does not seem to depend on the size of the ring.

Figure 4 displays runtime vs number of offices in the network for both feasible and infeasible rings. Except for the 2 data points circled, this figure indicates that the runtime generally increases with the number of offices in the network. Since, in these examples, networks with more offices also have more links, runtime also increases with the number of links. All of the example networks were sparse (0.9-1.8 links per office); therefore, the effect of density on runtime was not examined.

Figure 5 (runtime vs maximum number of hops between ring offices) shows that for infeasible rings, in which the upper limit on the number of hops is reached, increasing this bound can increase the runtime.

Overall, the average, minimum, and maximum runtimes were 0.41 sec., 0.06 sec., and 2.93 sec., respectively. The minimum runtime was for a 6-node, feasible ring, in a network with 29 offices and 28 links. The maximum runtime was for a 4-node infeasible ring, in a network with 167 offices and 240 links, in which a 5-hop limit was employed. Among examples using the suggested 3-hop limit, the maximum runtime was 0.76 sec., for a 4-node, infeasible ring, in a network with 167 offices and 240 links.
Traveling salesman algorithms are designed not to stop until the least cost cycle (or an approximation to it in the case of approximate algorithms) is found. The cycle always exists in the classic version, because the graph is complete. In contrast, BUILD_RINGS stops before a cycle is found if there is no useful cycle in the network, even if some cycle containing all of the ring offices exists. This behavior is desirable. It is better for the algorithm to find quickly that there is no practical ring than to spend time finding one that will not be used. Naturally, if the algorithm stops due to insufficient room in the data arrays, or insufficient computer memory, it may have missed a ring. In the runtime examples presented here, all infeasible examples really did not have a practical ring, so that the results did not depend on array sizes and computer memory.

One approximation algorithm for the traveling salesman problem is the nearest-neighbor method [5,6]. In this method, at each step the shortest edge leading out of the current node that will keep the path acyclic is chosen as the next edge in the cycle. This is continued until finally edge \( n \) (if there are \( n \) nodes) leads back to node 1. If this method is generalized, by letting links be replaced by paths between required nodes, and requiring that the paths be disjoint in addition to requiring that they combine acyclically until the last path is added, the resulting algorithm is BUILD_RINGS without the distance and hop limits. In other words, when the distance and hop limits are infinite, the graph is complete, and all nodes are required, BUILD_RINGS degenerates to the nearest-neighbor method of approximating the traveling salesman solution.

5. CONCLUSIONS

This paper presents an algorithm, BUILD_RINGS, for routing rings in a network, when the network nodes, links, connectivity, and which offices are to be placed on rings together are known. This algorithm is intended to be used with other algorithms for survivable network planning, in order to increase telecommunication reliability at the planning stage. The program was written in C, and run on a SPARCstation. Under certain conditions, the ring routing problem degenerates to the traveling salesman problem, and BUILD_RINGS degenerates to the nearest neighbor method of solving that problem.

Fast computation is necessary, so that network planners can arrive at an optimal network solution quickly, or test and compare many strategies in terms of cost and survivability. Computation times on 47 examples of feasible and infeasible rings were reasonable. Overall, the average, minimum, and maximum runtimes were 0.41 sec., 0.06 sec., and 2.93 sec., respectively. In the examples, computation time increased with the size of the network, but not with the size of the ring. Therefore, within a given LATA network, computation times should not impose much restriction on the sizes of rings which can be considered. However, computation times increase with the size of the LATA network. The largest example network used in these results, 167 offices and 240 links, is the size of a typical large LATA network. Computation times were short for the examples; therefore, BUILD_RINGS is anticipated to
run fast enough for the intended application. Computation time also increased with the hop limit. Based on the empirical results, this increase in computation time should not impose stringent limits on what the network planner can try.

Currently, in research prototype software, BUILD_RINGS has been integrated with routines from Fiber Options [7,8], a software package for planning survivable fiber networks using conventional architectures. Together with these other algorithms, BUILD_RINGS enables planning of survivable fiber networks using rings in an economical manner.

ACKNOWLEDGMENT

I thank Richard H. Cardwell and Steve Okun for answering questions about the C language and the C compiler. Thanks also to Tsong-Ho Wu for indicating valuable references.

REFERENCES


Copies of past Proceedings are available (at the prices given below) at each Symposium or by mail. Copies of recent proceedings are also available from the IEEE at prices set by the IEEE.

c/o Evans Associates
804 Vickers Avenue
Durham, North Carolina 27701 USA

Reliability & Maintainability Conference
First-Tenth (1962-1971) not available

Annual Symposium on Reliability
1966-1971 not available

Annual Reliability & Maintainability Symposium
1972, 1979 not available
1990, 1991 $50 each
1989, 1990 Tutorial Notes $20 each
1991 Tutorial Notes $25 each

National Symposium on Reliability & Quality Control (in Electronics)
First-Eleventh (1954-1965) not available

Enclosing payment will speed delivery. Non-USA purchasers, please write for details about payment. Price includes postage by slow surface mail.