FROM THE PROCEEDINGS OF THE 13TH INT'L CONFERENCE ON COMPUTERS AND THEIR APPLICATIONS, HONOLULU, HAWAII (MARCH 25-27, 1998)

An Optimal Physically-Disjoint Path Algorithm for Real-Life Telecommunication Networks

Ramesh Bhandari
AT&T Laboratories
Room 3F-208, 307 Middletown-Lincroft Road
Lincroft, New Jersey 07738, USA
(732) 576-5858
rbhandari@att.com

Abstract

In today's highly competitive telecommunication industry, guaranteeing continuity of service to a business customer has become extremely important to a service provider. One way to ensure continuity of service is to provide a pair of disjoint paths such that if one path fails, the other path is automatically invoked to carry the affected traffic. In this paper, we consider the physical structure of telecommunication fiber networks, which can be complicated by span-sharing topologies, and provide an algorithm for finding an optimal pair of physically-disjoint paths between a given pair of nodes in the network. Optimality helps to reduce the network's cost in providing diverse routes to a customer, and in an overall design of a survivable communication network.

Key words: *network survivability, facility networks, disjoint paths, optimization, graph theory*

1. Introduction

As customer services increase to take advantage of the large bandwidth offered by fiber, survivability of a network is receiving a great deal of attention. Because of the large choice of service providers available in today's highly competitive market, a customer has come to expect the highest quality of service. A minimal expectation is the continuity of service, since any disruption of service due to a node or link failure within a given communication network can cause a customer loss of revenue, not to mention the bad publicity for the service provider and subsequent erosion of its customer base. Thus, guaranteeing continuity of service at all times has become a matter of paramount importance within the telecommunication industry.

One way to guarantee continuity of service to a customer is to provide service over disjoint paths such that if one path fails due to a node or link failure, the service can be continued over the other path. Algorithms for finding the shortest set of disjoint paths between a pair of nodes in a network described by a graph of nodes (or vertices) and links (or edges) have been given in the past [1-3]. However, telecommunication networks are more complicated than the traditional graph-theoretic networks of nodes and links. This is due to the fact that a communication link connecting two nodes is actually composed of a number of physical connections called spans, some of which may be common to other links in the network. Consequently, two paths disjoint at the link-level may not necessarily be disjoint at the span (or physical)-level. Thus, the algorithms given in the past for graphs described only by nodes and links can fail. This leads to the requirement for new algorithms. One such algorithm based on the requirement of optimality has been given in the past [4] for certain types of span-sharing configurations found within the telecommunication network. However, this algorithm is not adequate to deal with the general case of arbitrary span-sharing topologies.

In this paper, we present a radically-different approach which permits construction of an algorithm for finding optimal physically-disjoint paths, when an arbitrary set of span-sharing configurations within the network is present. Optimality is desirable to reduce the cost of diverse provisioning of business services, and in the design of survivable networks based on disjoint paths [1]. Section 2 describes the fiber network in detail. Section 3 gives the approach and the construction of the optimal algorithm for finding diverse routes.

2. Physical Fiber Network Description

The physical layout of a telecommunication fiber network consists of links, where each link (also sometimes called a logical connection) connects a pair of network nodes ; a link is in general a series of contiguous physical connections, called spans. A span normally is a buried conduit carrying the communication fiber between two network points, called span nodes. Figure 1 depicts a link (dashed line) with three spans: AO, OO', and O'B. A, O, O', and B are called the span nodes. Note that span nodes A and B are also the end point nodes of the link AB, and the intermediate span nodes, O and O', are indicated by dark circles. The communication fiber connecting nodes A and B, and forming the link AB, traverses physically the spans, AO, OO', and O'B. The length of link AB = AO + OO' + O'B. In general, depending upon the physical construction of the network, a given link may be

composed of any number of spans. Thus, a telecommunication network is describable by a graph, $G = (N, L, N_s, L_s)$, where N, L, N_s, L_s are the sets of nodes, links,

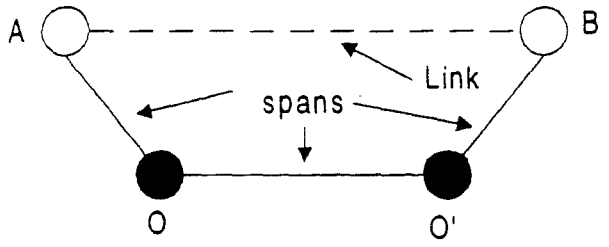

Fig. 1 Link AB composed of 3 spans: AO, OO', O'B.

span nodes and spans, respectively. In what follows, we indicate the network at the link-level by dashed lines and at the span (or physical)-level by continuous lines. Figure 2 is an example of a network at the link-level.

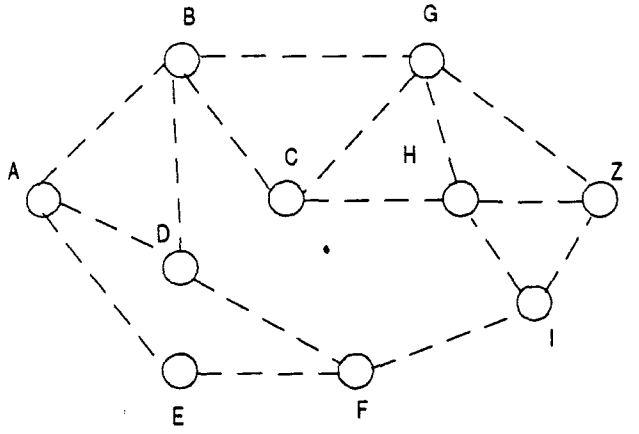

Fig. 2 A network of nodes and links (dashed lines).

Because of practical and economic considerations, the network may be so constructed that several of its links may share spans with each other. Figure 3 depicts a possible span structure for links CH and DF of Figure 2. Link CH is composed of spans CO, OO', and O'H, while link DF is composed of spans DO, OO', and O'F; span OO' is being shared by links CH and DF. In general, an innumerable span-sharing configurations can be possible within the network.

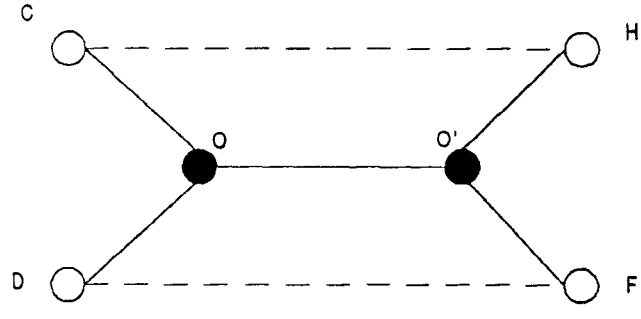

Fig. 3 Links CH and DF sharing span OO'.

Because of the presence of the span-sharing configurations within the network, the traditional shortest pair of node-disjoint path algorithm (SPVP) [1-3] run at the link level can fail. Consider, for example, the application of this algorithm for finding disjoint paths between nodes A and Z in Figure 2. Suppose the SPVP algorithm finds (ABCHZ, ADFIZ) as the shortest pair. Although the two paths are disjoint (excluding the end point nodes A and Z) at the link-level, they are clearly not physically-disjoint, since at least span OO' (see Figure 3) is common to the paths (clearly, more spans can be in common, depending upon the span-structure of the other links comprising the two paths).

Ref. [4] successfully constructed an algorithm to find the shortest pair of physically-disjoint paths in a network containing a certain class of span-configurations. The algorithm was based on network transformations and the modification of the existing SPVP algorithms. However, when additional configurations such as Figure 3 are included, the problem appears to be intractable and NP-complete [5]. We adopt a new approach here. The new approach leading to an optimal physical-disjoint path algorithm is described in Section 3. Because node and span failures are the most common in telecommunication networks, physical disjointness here implies node disjointness as well as span disjointness (although we note that in the most general sense physical disjointness implies span node disjointness (excluding the end points A and Z between which the pair of paths is desired)).

3. Optimal Algorithm for Physical-Disjointness

Instead of invoking the details of span-sharing configurations, as was done in Ref. [4], we construct the algorithm from the knowledge of span-overlaps between all pairs of links in the network. For example, in Fig. 3, the span overlap between links CH and DF is the common span OO'. In what follows, we denote the length of the span-overlap between links i and j of a given network by S_{ij}.

We therefore assume the following are given:

1. The basic graph of nodes and links, $G = (N, L)$; the length assigned to a link may be the physical length (in distance units such as miles) or may simply be the dollar cost of providing service over the link.
2. An $|L| \times |L|$ span-overlap matrix S_{ij}, specifying the amount of span-overlap between links i and j; the matrix is symmetric ($S_{ij} = S_{ji}$).

Next sections describe the construction of an algorithm that calculates two paths between a pair of nodes (A and Z) that are node-disjoint and have a minimum span overlap. The algorithm first ensures node-disjointness, and then optimizes with respect to span overlap. If more than one pair of paths exists with the same amount of span-overlap, it chooses the pair with the least total length.

3.1 General Construction Concept

We illustrate the basic ideas behind the algorithm by reference to Figure 3, which shows the network in the neighborhood of node A (assumed to be the source node). There are three links at node A (of degree 3). The (italicized) number on each link identifies the link within the network; it is *not* the length of the link. The aim is to find the shortest pair of node-disjoint paths between nodes A and Z, while satisfying the constraint of minimal span-overlap. The search

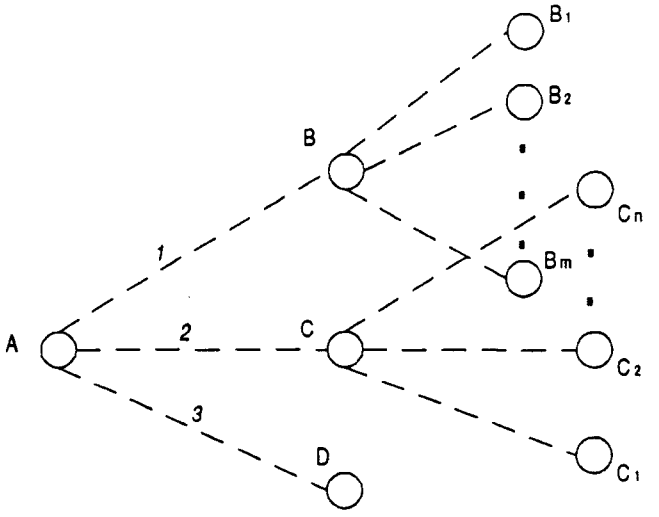

Fig. 4 Italicized numbers 1, 2, 3 denote the number assignments for links AB, AC, and AD, respectively.

for the pair starts by forming pairs of links emanating from the source node A, and adding a single link at the end of each link in a given pair to augment the pair of paths. There are three link pairs at node A in Figure 4: (1, 2), (2, 3) and (1, 3). Focusing on link pair (1, 2), one augments this pair of paths by adding a link to each of the link (or path) end points B and C; the link added at end point B is selected from the set of links incident at B, excluding the preceding link AB; similarly the link at end point C is augmented. In Figure 4, there are m possible links for node B and n for node C. Thus, the number of new possible pairs of paths generated is n x m. However, not all of these pairs of paths may be node-disjoint, since some of the end point nodes in the group of m links may be common to the end point nodes in the group of n links. Thus, the number of possible node-disjoint pairs generated $\leq$ n x m. These possible pairs are each composed of 2 links. Clearly, the possible pairs of node-disjoint paths in this augmentation process mushrooms with the addition of links (one to each path in a given pair), increasing rapidly with each augmentation. Of course, the process of augmentation terminates when the destination node Z is reached on both paths; when node Z is reached on one path, the link augmentation process ceases for this path, while continuing for the other path until destination node Z is also reached on this path. In a similar way one constructs pairs of paths ending at destination node Z starting with the remaining link pairs: (2, 3) and (1, 3) at source node A. Thus, the search performed is over all possible pairs of paths between A and Z.

Each time a pair of paths is augmented, information is stored about its current length and the total span-overlap between the two paths. Such information storage permits selection of the pair of paths between A and Z with minimum overlap; if there are ties, the pair with minimum total length is selected. We illustrate the span-overlap calculation with reference to Figure 5a, which is a network of 5 nodes and 8 links:

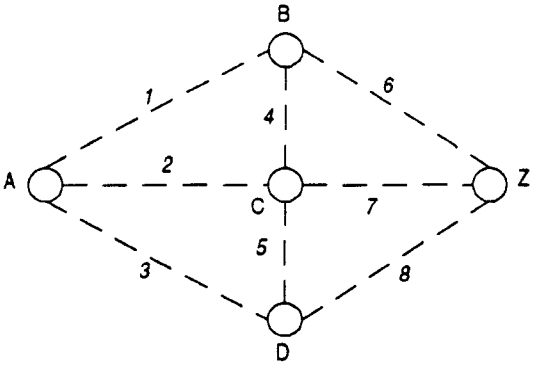

Fig. 5a A network of 5 nodes and 8 links

1. Find neighbors of source node A: B,C, and D, which are the end points of links 1, 2, and 3 in Figure 5a.
2. Form pairs of links: {(1, 2), (2, 3), (1, 3)}.
3. Obtain overlaps from the given overlap matrix: S_{12}, S_{23}, S_{13}

Pair Augmentation

4. Start augmentation with one of the 3 pairs of links, say (1, 2)

a) Find the neighbors of the forward end point of link 1 (node B), excluding its predecessor (node A): C and Z; find the associated links: 4 and 6.

b) Repeat for link 2: forward neighbors of C are B, Z, and D, and the associated links are 4, 7, and 5, respectively.

c) Form link pairs that, in conjunction with link pair (1, 2), produce node-disjoint paths (except for node commonness at node Z): (6, 7), (6, 5).

d) Form pairs of node-disjoint paths resulting from the addition of link pairs (6,7) and (6,5): (ABZ, ACZ) and (ABZ, ACD), respectively.

Overlap Calculation

i) Calculate increment in overlap due to the addition of link pair (6,7): $S'_{67} = S_{67} + S_{62} + S_{71}$.

The three components of S'_{67} arise in the following manner: S_{67} is due to the overlap between link 6 and 7. But

link 6 can also have an overlap with the predecessor links of the other path, which is link 2 in the case under consideration. Similarly, link 7 can have an overlap with link 1 of the other path. Thus, to cover all overlap possibilities, the additional overlap terms, S_{62} and S_{71}, must appear in the expression for S'_{67}. The total overlap for this pair of paths (ABZ, ACZ), each composed of two links = $S_{12} + S'_{67} = S_{12} + S_{67} + S_{62} + S_{71}$.

At this stage, we discuss a mapping in which the pair of paths (ABZ, ACZ) appears as a sequence of links of lengths S_{12} and S'_{67} in a span-overlap graph shown in Figure 5b. The

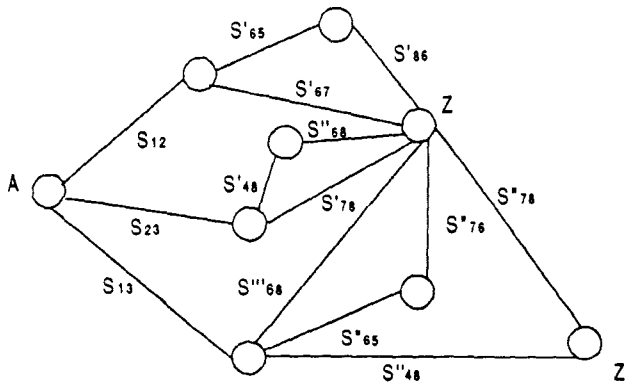

Fig. 5b A mapping of the pairs of paths possible in Fig. 5a; the path length between A and Z in this graph is the amount of span overlap for the corresponding pair in Fig. 5a.

interpretation is that traversing this pair of links (of length S_{12} and S'_{67}) in the graph of Figure 5b is equivalent to selecting the pair of paths (ABZ, ACZ) in Figure 5a.

ii) Now consider path augmentation by link pair (6, 5) and calculate the increment in the overlap of the two paths: $S'_{65} = S_{65} + S_{51} + S_{62}$.

In this case, one of the paths (with link 6) has reached the destination node Z before the other path. Nevertheless augmentation should continue for the other path, which then requires the addition of link 8, yielding an additional overlap expression $S'_{86} = S_{86} + S_{81}$. Note that along with the consideration of overlap of links 8 and 6 is the overlap of links 8 and 1. Since links 8 and 6 have the same common end point Z (the destination node), the augmentation process ceases. The total overlap for the pair of paths constructed (ABZ, ACDZ) = $S_{12} + S'_{65} + S'_{86}$. Choosing this pair of paths is equivalent to traversing links S_{12}, S'_{65}, and S'_{86} in Figure 5b.

5. Repeat Step 4 above for augmentations with link pairs: (2, 3) and (1, 3) of Figure 5a.

Augmentation from link pair (2, 3)

Starting with link pair (2, 3), one obtains node-disjoint paths: (ACZ, ADZ) and (ACBZ, ADZ), which have overlaps equal to $S_{23} + S'_{78}$ and $S_{23} + S'_{48} + S''_{68}$, respectively with $S'_{78} = S_{78} + S_{73} + S_{82}$, $S'_{48} = S_{48} + S_{43} + S_{82}$ and $S''_{68} = S_{68} + S_{63}$.

Augmentation from link pair (1, 3)

This case yields node-disjoint pairs: (ABZ, ADZ), (ABZ, ADCZ), and (ABCZ, ADZ), with overlaps $S_{13} + S'''_{68}$, $S_{13} + S''_{65} + S''_{76}$, and $S_{13} + S''_{48} + S''_{78}$, respectively; $S'''_{68} = S_{68} + S_{63} + S_{81}$; $S''_{65} = S_{65} + S_{51} + S_{63}$; $S''_{76} = S_{76} + S_{71}$; $S''_{48} = S_{48} + S_{43} + S_{81}$; $S''_{78} = S_{78} + S_{73}$.

When all the above possibilities are considered, we have a complete span-overlap graph shown in Figure 5b with

$S'_{67} = S_{67} + S_{62} + S_{71}$,
$S'_{65} = S_{65} + S_{51} + S_{62}$,
$S'_{86} = S_{86} + S_{81}$,
$S'_{48} = S_{48} + S_{43} + S_{82}$,
$S''_{68} = S_{68} + S_{63}$,
$S'_{78} = S_{78} + S_{73} + S_{82}$,
$S'''_{68} = S_{68} + S_{63} + S_{81}$,
$S''_{65} = S_{65} + S_{51} + S_{63}$,
$S''_{76} = S_{76} + S_{71}$,
$S''_{48} = S_{48} + S_{43} + S_{81}$,
$S''_{78} = S_{78} + S_{73}$.

Thus, finding a pair of disjoint paths with minimum span overlap between nodes A and Z in Figure 5a reduces to finding the shortest path between nodes A and Z in Figure 5b.

We note above that overlap between a given pair of links can occur more than once. For example, overlap between links 5 and 6 occurs twice once in S'_{65} and the other time in S''_{65}. The difference is that S'_{65} has predecessor link pair (1, 2) or equivalently predecessor nodes (B, C) different from the predecessor links (1, 3) or predecessor nodes (B, D) in the S''_{65} case.

Avoidance of Loops

When the paths are being augmented, it is possible that the new node reached on one of the paths was already a node that had been reached earlier, i.e., the path being augmented loops back on itself. We illustrate this possibility in the network of Figure 6, which consists of 7 nodes and 12 links. The two disjoint paths to be generated between nodes A and Z start with the only available link pair (1, 2) at node A. Suppose the augmentation in three iterations leads to the paths (ACEZ, ABFD). Clearly, the first path has reached the end point Z, while the second path needs to be augmented from node D. While reaching node E or node C is prevented by the requirement of node-disjointness, node B can be accessed (forming loop BFDB) unless measures are built into the algorithm to prevent it. Thus, it is also necessary to check that when the path is augmented, the node reached is not already in the list of nodes forming the path.

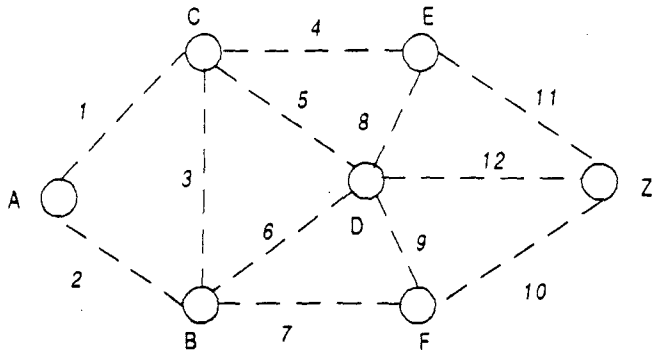

Fig. 6 A path, say ABFD, generated at the end of three iterations may augment to ABFDB, forming loop BFDB.

Implementation

The algorithm is easily implemented through the use of data structures such as structs in C language. Each move in the path augmentation process (leading to a new node) is modeled as a structure with four elements: node number, node's label, the length of the path accumulated up to that point, and a pointer to the node's predecessor, which is also a data structure with the above four elements. The last element helps to trace the path generated at any point during the search.

Refer to Figure 7 which shows two paths under construction with current nodes at the end of the paths being 4 and 5; since in implementation it is convenient to associate a number with a node, we indicate the nodes in this graph via numbers. The inputs for the algorithm are a network

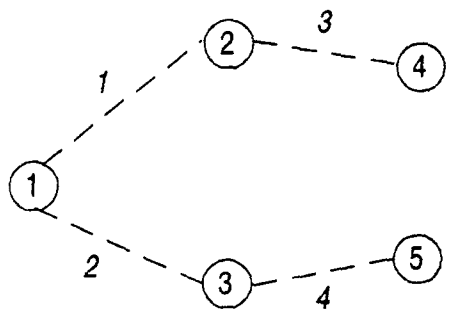

Fig. 7 A pair of paths under construction; nodes are indicated by numbers.

data base and an overlap matrix; for Figure 7, the input may look like the following:

Node	Neighbor	Link #	Length
1	2	1	10
1	3	2	8
2	4	3	5
3	1	2	8
3	5	4	3
⋮			

Link #1	Link #2	Overlap S_{ij}
1	2	3
1	3	2
1	4	3
2	3	0
2	4	3
3	4	1

The two paths up to nodes 4 and 5 in Figure 7 would be specified via structs (in C language) as indicated in Figure 8.

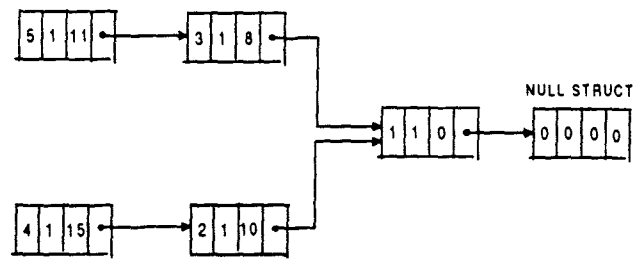

Fig. 8 Each struct is constructed of four elements, which (from left to right) are node number, node label, path length, and address to the predecessor of the node, which is also a struct with similar elements; refer to Fig. 7.

As mentioned earlier, each struct is composed of four elements, which (from left to right) are node number, node label, length of the path up to that node, and pointer to the predecessor, which is also a struct. Figure 8 is consistent with the input data given above for Figure 7. For the pair of paths in Figure 11b, the length of the path up to node 5 is 11 and the length of the other path up to node 4 is 15. Up to this point, the total overlap of the two paths is equal to $S_{12} + S_{34} + S_{23} + S_{14} = 3 + 1 + 0 + 3 = 7$, using the overlap matrix given above. This overlap information along with the information on the current nodes (nodes 4 and 5 in Figure 7) is stored so that the current state of the two paths is known before the next augmentation move. Since an innumerable pairs of paths may be possible between a given pair of end point nodes A and Z, a situation will often occur whereby a given node of the network is visited more than once during the construction of path pairs. The second element in the struct identifies the number of times that particular node has been reached by all considered pairs so far. In the example of Figure 8, each node has been reached only once so far. But if the path of another pair being constructed were to pass through one of these nodes, say node 4, this node will be assigned another struct with the second element (node's label) equal to 2, i.e., the node's label is updated by 1 each time it is reached. Similarly, in Figure 5a node C would receive the label of 1 when the pair of paths AB and AC is generated

from the source node A. Further, it will receive a new struct when it is reached from node D, forming the node-disjoint pair ADC and ABZ; the new struct will have a node label of 2. The node label helps to trace back to the source node, when a given node has struct assignments with identical path lengths.

Augmentation Process

i) Determine the current nodes' neighbors (excluding the predecessors of the current nodes).
ii) Find all possible pairs.
iii) Put pairs passing the node-disjoint and no-cycle tests into an array.
iv) Trace back for each pair to find the total length and the total overlap up to this point; tracing back is done via the 4th element which points to the predecessor node.

3.2 Algorithm

An algorithm is easily constructible based on a Dijkstra type search [6-7]:

1) Augment from the start node.
2) Put all the current nodes of the pairs into two arrays; each pair of paths has to be node-disjoint ands should satisfy the no-cycle test.
3) Select the pair of nodes from the above arrays which correspond to the pair of paths with the minimum. overlap, and remove this pair from the arrays in Step 2.
4) Augment from the selected pair and repeat Step 2 until the destination node Z is reached. Note that destination node Z in general will not be reached simultaneously; when Z is reached by one of the two paths in the pair being constructed, the augmentation on this path of course ceases but it continues from the second path until destination node is also reached on this path.
5) Terminate when the selected pair has each of its current nodes as the destination node and has the minimum overlap in the array of pairs of paths generated in Step 2; otherwise continue from Step 2. If there are several pairs with the same amount of overlap, then the pair with the least total length is selected.

Note that wherever the term node appears above, its 4-element structure is implied; thus all arrays are structs.

Efficiency: Because the manner in which the number of pairs of possible paths increases, the algorithm compared to the previous algorithms [1-4] is slow. However, given today's high speed, powerful computers with large memories, the algorithm can be implemented and used effectively. We have already tested a working code for a network of up to 270 nodes.

4. Summary

We have described the construction of an algorithm for finding an optimal pair of physically-disjoint paths for real-life telecommunication networks. Such networks are described by not only nodes and links, but also span nodes and spans, which comprise the physical part of the network. The networks become complicated due to the sharing of spans among links in the network. Consideration of arbitrary span-sharing topologies makes the problem of finding disjoint paths hard. The approach adopted here is one of seeking the desired paths simultaneously from a knowledge of a given span-overlap matrix which contains information on the amount of span overlap for every pair of links in the network. The algorithm provides a pair of paths that are node disjoint and have a minimum amount of span overlap, and chooses further (in case of ties) a pair of paths with minimum total length. The algorithm has been coded and tested for networks as large as 270 nodes. It can be very useful for providing diverse routes to customers who cannot afford to lose any amount of data in the event of a network failure. Further, it can be utilized in an overall design of survivable networks by splitting traffic demands over disjoint paths [1].

5. Acknowledgement

The author thanks Trac-Duy Tran who invoked and used the structs to create a working C code for the above algorithm.

REFERENCES

1. R. Bhandari, "Optimal Physical Diversity Algorithms and Survivable Networks", Proc. of the Second IEEE Symposium, Alexandria, Egypt (1997).
2. J.W. Suurballe, "Disjoint paths in a Network", Networks, 4 (1974) 125-145.
3. J.W. Suurballe and R.E. Tarjan "A Quick Method for Finding Shortest Pairs of Disjoint Paths", Networks, 14 (1984) 325-336.
4. R. Bhandari, "Optimal Diverse Routing in Telecommunication Fiber Networks", Proc. of IEEE/INFOCOM (1994) 1498-1508.
5. For NP-complete algorithms, see M.R. Garey and D.S. Jackson, *Computers and Intractability - A Guide to the Theory of NP-completeness*, W.H. Freeman, 1979.
6. E.W. Dijkstra, "A Note on Two Problems in Connexion with Networks", Numer. Math. 1 (1959) 269-271.
7. M. Gondran and M. Minoux, *Graphs and Algorithms*, John Wiley (1984).