FROM
THE PROCEEDINGS OF THE COMMUNICATION NETWORKS
MODELING AND SIMULATION CONFERENCE (THE SOCIETY FOR COMPUTER SIMULATION), SAN DIEGO, CA.
(JAN 14-17, 1996).

PHYSICAL DIVERSITY VERSUS COST ALGORITHM FOR NETWORKS

Ramesh Bhandari
AT&T Bell Laboratories, Rm. 2B-410A
Holmdel, New Jersey 07733
r2b@hostare.att.com

Key Words: Telecommunications; Optimization; Network; Disjoint Paths; Algorithms

ABSTRACT

In this paper, we give the development of an algorithm that calculates the cost of providing business services over two paths as a function of physical diversity. Physical diversity refers to physical-disjointness between the pair of paths connecting the source and the destination nodes in a network described by nodes and links. The algorithm is derived from a shortest pair of node-disjoint paths algorithm, which is first modified via assignment of two large parameters to obtain a maximally-disjoint paths algorithm. This algorithm then yields the number of common nodes and links when full disjointness is not possible. When the two parameters - initially fixed at large values - are now allowed to vary freely, an algorithm results which permits physical-diversity violations, i.e., it permits the two paths to have common nodes and links, depending upon the values of the parameters. The algorithm provides evaluation of savings to a customer when such diversity violations are allowed.

1. INTRODUCTION

As optical technology continues to be deployed, and the massive bandwidth of the fiber is beginning to be utilized on a large scale for provisioning of different types of services, physical diversity assumes greater importance, and may become the minimal requirement for robustness in telecommunication networks. For networks considered here, physical diversity refers to two disjoint paths between a given pair of nodes in the network. For a business customer who cannot afford any delay in transmission of his data, physical diversity ensures continuity of service in the event of a link or node failure. However, physical diversity also implies increased cost to the customer since two physically-disjoint circuits must be reserved for the customer. If, for the moment, we assume that the cost of providing service is proportional to the length of the path, then the cost to the customer is likely to be at least doubled. This is due to the fact that the sum of the length of two disjoint paths is equal to or greater than twice the length of the single (shortest) path between the two given nodes in the network. What happens if the customer cannot afford to pay such increased costs, yet desire a high level of physical-diversity, i.e., can we provide him with two paths that are nearly physically-disjoint at a cost significantly reduced in comparison to the cost for full diversity?

This paper gives the development of an algorithm that calculates cost as a function of physical-diversity. The algorithm then permits evaluation of savings when the two paths sought are allowed to have partial overlaps. Clearly, the cost of two paths is maximal when complete (100%) diversity is desired, and is minimal in the extreme case of the two circuits traversing the same (shortest) path, which is a case of zero diversity. Therefore, when the level of diversity is less than 100%, the cost lies between the above two extreme cases. In fact, situations can arise where, depending upon the network characteristics, the reduction in cost can be significant for a very small violation in physical diversity. This fact is illustrated with reference to Figure 1, which shows a network in which physically-disjoint

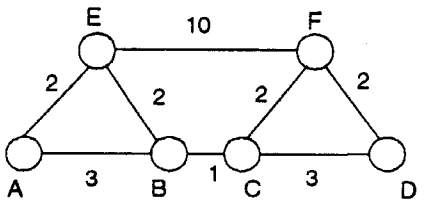

Figure 1 A network of nodes and links; the numbers represent the cost of the links.

paths are desired between various pairs of nodes. For the pair of nodes, A and D, ABCD is the shortest (least cost) path with a cost of 3+1+3=7 units. The two physically-disjoint paths between nodes A and D are ABCD and AEFD with a total cost of 7+14=21 units. If we permit diversity violation, then provisioning of two circuits, one on the path ABCD and the other on the path AEBCFD, would result in the short link BC in common with a total cost of 7+9=16 units, which is 5 units less than the 21 unit cost for full diversity. Similarly, from B to F, the cost of the shortest pair of disjoint paths (BCF,BEF) is 3+12 =15 units, while the two paths (BCF,BCDF), having the short link BC in common, cost only 3+6=9 units. Thus, provisioning of two circuits on the latter pair of paths is cheaper by (15-9)/15 x 100% = 40%.

The construction of the physical-diversity versus cost algorithm is based on the generalization of the algorithm for the shortest pair of node-disjoint paths between a given pair of nodes in the network. Given the network graph, $G(N,L)$, where N and L denote the sets of nodes and links in the network, node-disjointness automatically implies link disjointness, and thus physical-disjointness between the two paths (except for the common end point nodes). In Section 2, we review the node-disjointness algorithm given earlier by the author (Bhandari 1994), and develop an implementation scheme in Section 3. This implementation scheme for the node-disjointness algorithm then automatically leads to the algorithm for maximally-disjoint paths, useful in situations where physical-diversity is not achievable due to the inherent nature of the network. For example, deletion of link EF in Figure 1 leads to the absence of

complete disjointness in paths between pairs of nodes such as (A,D), (A,F), etc. For such pairs of nodes, link BC (also called a *bridge*) is necessarily common to the two paths found. In general, when total physical disjointness does not exist, the developed algorithm finds two paths that have the least total cost and minimum number of common nodes and links. Finally, in Section 4, we show how the physical-diversity versus cost algorithm naturally evolves from it. The algorithm is fast and works in polynomial time over a mesh network.

2. SHORTEST PAIR OF NODE-DISJOINT PATHS ALGORITHM

Because the network under consideration is bidirectional, the graph representing the network is undirected, i.e., each link of the network graph is equivalent to two oppositely directed arcs of length equal to the length of the given link. For example, in Figure 1, link AB is equivalent to two arcs, AB and BA, each of length equal to 3, the length of link AB.

Algorithm 1 *: For the graph, G(N,L), the shortest pair of node disjoint paths between a given pair of nodes may be found as follows*:

1. For the given pair of nodes, A and Z, under consideration, find the shortest path using the algorithm given in Appendix A. As an example, refer to Figure 2a. Denote the length of the shortest path found by d(AZ), where A is the source node and Z the destination node.

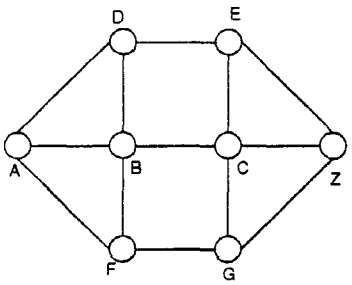

Figure 2a Network of nodes and links; the shortest path is assumed to be ABCZ with A and Z the source and destination nodes, respectively.

2. Delete the forward arc of each link of the shortest path from A to Z, i.e., delete arcs AB, BC, and CZ in Figure 2a. This deletion leaves arcs pointing in the backward direction, i.e., in the direction towards the source node A. Make the length of these arcs (BA, CB, ZC in Figure 2b) negative

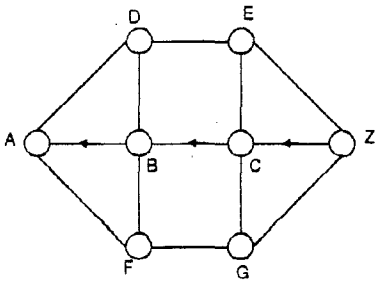

Figure 2b Network with shortest path links replaced with negative arcs directed towards source node, A.

3. Split each node on the shortest path (except the end point nodes) into two colocated subnodes, joined by an arc of length zero and directed towards the source node (in Figure 2c, arcs BB' and CC' are each of zero length). Replace external links connected to nodes on the shortest path by two arcs of the same and original length, and connected to the two subnodes as shown in Figure 2c; e.g., link DB in Figure 2b is split into arcs DB and DB', and, similarly, links FB, GC, and EC.

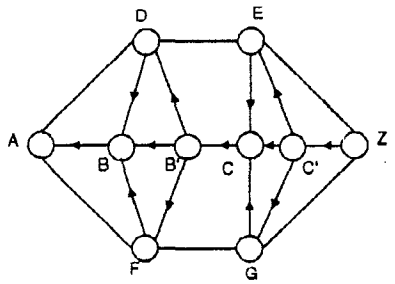

Figure 2c Network graph modified by node-splitting

4. Find the shortest path from A to Z in the modified graph (see Figure 2c), using the algorithm given in Appendix A; denote its length by d'(AZ).

5. Remove the zero length arcs, i.e., coalesce the subnodes into their parent nodes. Replace the negative arcs (see Figure 2b) with the original links (of positive length). Remove overlapping links of the two paths found (the first in Step 1 and the second in Step 4) to obtain the shortest pair of node-disjoint paths in the original graph (Figure 2a).

Step 2 above permits interlacing of the second path (found in Step 4) with the first (found in Step 1); e.g., if the second path determined in Step 4 is ADECB'FGZ (see Figure 2c), it reduces to path ADECBFGZ in the original graph (see Figure 2a), and is said to interlace with the part BC of the first path, ABCZ, found in Step 1; erasure of the overlapping part (or interlacing part), BC, then leads to the pair, (ADECZ, ABFGZ), as the shortest pair of node-disjoint paths.

Note that it is the process of node-splitting described in Step 3 that engenders node-disjointness between the two paths. If this step is omitted, the ensuing algorithm corresponds to the algorithm for the shortest pair of link-disjoint paths, with the second path now determined in a modified graph, illustrated by Figure 2b.

Algorithm 1 given above (Bhandari 1994) is different from the previous versions (Suurballe 1974; Suurballe and Tarjan 1984). In sharp contrast to the Suurballe's algorithm, our algorithm requires no canonic transformation or the use of a general

shortest path algorithm like Ford's algorithm for Step 4 above. Rather, we modify the traditional Dijkstra algorithm (Dijkstra 1959) slightly for the special graph of Figure 2c. This modified Dijkstra algorithm (Bhandari 1994), described in Appendix A, reduces to the standard Dijkstra algorithm for nonnegative graphs; thus, it is utilized also in Step 1 of Algorithm 1.

3. IMPLEMENTATION AND MAXIMAL DISJOINTNESS ALGORITHM

Figure 2c shows the modified graph in which the second path is found. All arcs along the shortest path found in Step 1 of the algorithm point in the backward direction, i.e., in the direction from destination node Z to source node A. This modification is due to Steps 2 and 3 of the algorithm. In practice, instead of removing the forward arcs in Step 2, it is convenient to leave them there, and also provide for each inserted arc of zero length, an arc in the forward direction, as shown in Figure 3. These

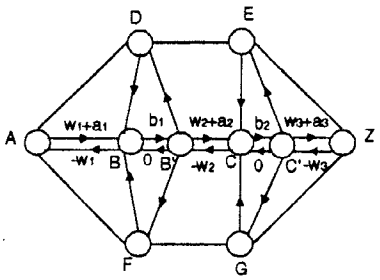

Figure 3 Network graph of Figure 2c modified with forward arcs.

forward arcs are assigned large lengths such that they are not traversed during the determination of the second path. In Figure 3, we show these lengths to be $w_i + a_i$, i =1,2,3 for arcs AB, B'C, C'Z, respectively, and equal to b_i, i =1,2 for the forward arcs complementing the split-node arcs of zero length; w_i are the original (nonnegative) lengths of the arcs, and the parameters a_i (>0) and b_i (>0) are large enough such that these arcs are not traversed during the search of the second path (Step 4 of Algorithm 1); therefore, these forward arcs are just spectator arcs. Note also that because $a_i > 0$ and $b_i > 0$, no negative cycle arises between a given forward arc and the corresponding backward arc. Thus, the introduction of these arcs causes no change in Algorithm 1. However, when complete physical-disjointness is absent, some of these forward arcs would necessarily be traversed. A suitable assignment of values to parameters a_i and b_i then leads to a maximally-disjoint paths algorithm, which determines the number of common nodes and links when full-disjointness is absent.

3.1 Absence Of Disjointness

To keep our analysis simple and useful, we assume $a_i = a_0$ for all i, and further set $b_i = b_0$ for all i. Thus, we need to determine only a_0 and b_0.

Consider the case of a graph which is a chain (a string of links). See Figure 4, which illustrates its modified form after node-

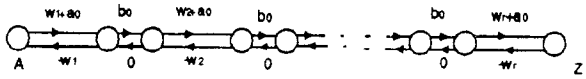

Figure 4 All |L| links of the graph form a chain from A to Z; r = |L|

splitting and assignment of forwards arcs. It corresponds to the worst-case scenario because

i) the shortest path (from A to Z) involves traversal of all the links of the graph.

ii) it is a case of zero disjointness, since both paths traverse the same set of links in the given graph.

Analysis of this extreme case helps to set the values of a_0 and b_0, which can then be applied to any arbitrary graph. Refering to Figure 4,

$$d(AZ) = W, \tag{1a}$$

where

$$W = \Sigma w_i; \tag{1b}$$

w_i is the length of the ith link, and the sum is over all the links in the graph, *G (N,L)*. Furthermore, the second run of the shortest path algorithm yields

$$d'(AZ) = W + |L|\, a_0 + (|L| - 1)\, b_0, \tag{1c}$$

We now require that parameter a_0 be such that

$$d'(AZ)/\, a_0 = |L| + \text{Remainder}(=(|L|-1)\, b_0 + W), \tag{2}$$

where |L| is the number of overlapping links of the two paths. In other words, we want

$$a_0 > (|L|-1)\, b_0 + W. \tag{3}$$

Denoting the right-hand side of Eq. (3) by r, we determine b_0 by the requirement that

$$r/b_0 = (|L|-1) + \text{Remainder } (=W), \tag{4}$$

where (|L|-1) is the number of common nodes in the two paths. The foregoing result implies

$$b_0 > W, \tag{5}$$

or

$$b_0 = W + \varepsilon, \tag{6}$$

where $\varepsilon > 0$, and W is given by Eq. (1b). We now find that the assignment:

$$a_0 = |L|\, b_0 \tag{7}$$

satisfies the inequality, Eq. (3). Thus, the parameters a_0 and b_0 are defined by the above analysis.

Note that the values assigned clearly do not permit the forward arcs to be part of the shortest path determined in the modified graph of a given graph, since alternate routes not involving the forward arcs are always available with total length smaller than b_0; in the event complete physical-disjointness does not exist, traversal of at least one of the forward arcs would automatically occur, and the length of the path found greater than b_0. The above analysis sets up the values of a_0 and b_0 (Eqs. (7) and (6)) in such a way that not only the selection of the forward arcs by the shortest path algorithm is discouraged, but the number of common links and the number of common nodes are separately determined, when physical-disjointness does not exist (see Corollary 1 below).

The analysis given above is an instance where parameter $a_0 > b_0$; parameter b_0 is considered *embedded* in parameter a_0. We could have chosen to perform the analysis where the parameter a_0 is considered embedded in parameter b_0; this would have involved division of d'(AZ) by b_0, instead of a_0 as in Eq. (2). However, we choose the definition we have established for a_0 and b_0 via Eqs. (7) and (6) because this definition permits us to find the common links first when full-disjointness does not exist (see Corollary 1 below); link-commonness appears to be a more serious concern than node commonness in telecommunication networks.

Algorithm 2 *The node-disjoint shortest pair algorithm (Algorithm 1) may be restated as follows:*

1. For the given pair of nodes, A and Z, under consideration, find the shortest path from A to Z; denote the length of the shortest path by d(AZ).

2. Modify the given graph by splitting the nodes as in Steps 2 and 3 of Algorithm 1. Add forward arcs (direction A to Z). See Figure 3 for illustration.

3. Assign a length of $b_i = b_0$, given by Eq.(6), to each one of the forward arcs connecting the pairs of split nodes; assign a length of 0 to each one of their counterparts in the backward direction.

4. Assign a length of $a_0 + w_i$ to the forward arc i, originally of length w_i; a_0 is defined in Eq. (7). To its counterpart in the backward direction, assign a length of $-w_i$.

5. Run the Modified Dijkstra algorithm (Appendix A) in the above modified graph; denote the length of the shortest path found by d'(AZ).

6. Transform to the original graph. Remove any overlapping links of the two paths traversing the shortest path links in the *opposite* direction. The desired optimal pair of paths with the pair path length = d(AZ) + d'(AZ) results.

Comments: Except when the shortest path (see Step 1 above) is a single link path, assignment of a special large parameter, a_0, is not really needed to achieve node-disjointness in Algorithm 2. Our interest in assigning a_0 a large length defined, e.g., as in Eq. (7), is motivated by the need to obtain maximally disjoint paths with direct information on the number of common nodes and common links in the event full-disjointness was not possible (see below).

Maximally Disjoint Paths

If d'(AZ) in Step 5 of Algorithm 2 is greater than b_0, full disjointness does not exist, and d'(AZ) is not the true length of the second path; consequently, d(AZ) + d'(AZ) in Step 6 of Algorithm 2 is not the true length of the optimal pair found.

Corollary 1 *Algorithm 2 finds a pair of paths that are maximally disjoint, i.e., a pair of paths with a minimum number of common nodes and links, and whose sum, d(AZ) + d_0, is a minimum; d_0 is the true length of the second path found in the modified graph, and is determined as follows:*

$$d'(AZ)/\, a_0 = m + \text{remainder } (=r), \qquad (8)$$

$$r/b_0 = n + \text{remainder } (=d_0), \qquad (9)$$

where m is the number of common links, n is the number of common nodes; m=0 and n=0 correspond to the existence of the shortest pair of node-disjoint paths.

Further Remarks:

a) If contiguous bridges present in a graph (see Figure 5a) are reduced to a single bridge of length equal to the sum of the

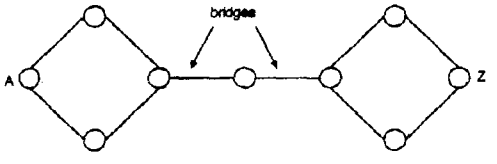

Figure 5a Graph with contiguous bridges between A and Z.

contiguous bridges, which is commonly done in the analyses of telecommunication networks, then n=2m +n', where the 2m common nodes are due to the m bridges and n' is the number of other common nodes not due to bridges. Inherent in the above expression for n is the assumption that bridges do not occur at the end point nodes. If they do, n=2m-1+n' or n=2m-2+n',

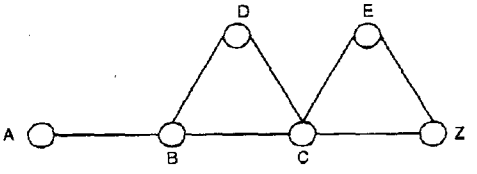

Figure 5b Two maximally disjoint paths (ABCZ, ABDCEZ) between A and Z have common nodes B and C (n=2); common node B is due to bridge AB (m=1) at the end point A, and common node C is not due to a bridge (n'=1).

accor ing as a bridge occurs at one of the 2 end point nodes or at both the end point nodes. Thus, knowing n and m along with the knowledge of whether bridges are present at the endpoint nodes determines n', the number of common nodes not due to bridges. Figure 5b is an illustration of the absence of disjointness. n=2, m=1, and n'=1 in accord with n=2m-1+n'.

b) Note that if $b_i = 0$ for all i in Step 3 of Algorithm 2, Algorithm 2 reduces to the algorithm for the shortest pair of link-disjoint paths; setting $b_i = 0$ permits the second path found in Step 5 of Algorithm 2 to pass through a node of the first path found in Step 1; e.g., in Figure 3, the second path may run as ADBB'FGCC'EZ, which reduces to path ADBFGCEZ in Figure 2a; setting $b_i = 0$ is equivalent to not having nodes split. When full link-disjointness does not exist, $d'(AZ) > a_0$. Corollary 1 then refers to maximally link-disjoint paths, with Eq. (8) determining the number of common links with $r = d_0$; Eq. (9) is not applicable. One also notes that, if one wishes, one can also improve the definition of parameter a_0 here by repeating the previous analysis (Eqs. 1a-7) with $b_0 = 0$; then from Eq. (3), one sees that one can set $a_0 = W + \varepsilon$, where $\varepsilon > 0$.

4. PHYSICAL-DISJOINTNESS VERSUS COST

In the previous sections, we gave prescriptions for the implementation of the physical-disjointness algorithms. The key parameters were a_i and b_i, assigned to the forward arcs (see Figure 3). Each of the b_i parameters were set equal to a large value, b_0, defined by Eq. (6), and similarly, the a_i were set equal to a large value, a_0, defined by Eq. (7). The largeness of these values ensured physical-disjointness. In this section, instead of fixing their values to large numbers, a_0 and b_0, we let them vary freely, i.e., we let $0 \le a_i \le a_0$; $0 \le b_i \le b_0$. This permits node-commonness as well as link commonness even when physically-disjoint paths exist. As a result, these paths are smaller (or less costlier) than fully physically-disjoint paths.

***Corollary 2**: Calling b_i the extra cost (penalty) for common node i, and a_i the extra cost (penalty) for overlap with link i, and assuming that $0 \le b_i \le b_0$ and $0 \le a_i \le a_0$, Algorithm 2 finds a pair of paths with minimum total cost; the paths may have common nodes and links, depending upon the coefficients a_i and b_i.; the total cost includes the values of a_i and b_i corresponding to the common links and nodes; the true cost (or length) of the second path found is $d_0 = d'(AZ) - \Sigma a_i - \Sigma b_i$, where the sum is over the common links and nodes.*

Special Cases of Corollary 2:

a) If $a_i \rightarrow 0$, $b_i \rightarrow 0$, the second path found is identical to the first path, i.e., it is also the shortest path in the original graph.

b) If $a_i \rightarrow a_0$, $b_i \rightarrow 0$ for all i, the shortest pair of link-disjoint paths algorithm results.

c) If $a_i \rightarrow a_0$ and $b_i \rightarrow b_0$ for all i, the shortest pair of node-disjoint paths results; except when the first path is a single link path, $a_i \ge 0$ suffices.

4.1 Cost Of Link-Disjointness

Case b) above corresponds to the algorithm for link-disjointness. Refer now to Figure 6. Assume that the shortest path from A to Z in the modified graph passes through D and E. Suppose $a_i = a_0$

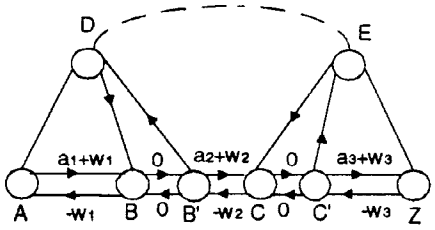

Figure 6

for all i. Suppose now parameter a_2 is reduced in value from its initial large value of a_0. Then, at some stage parameter a_2 will be such that the shortest path from A to Z may include arc BC. If and when this happens,

$$d(1+2) = d(1+2;B'C) + a_{T2}, \qquad (10)$$

where the left-hand side, d(1+2), is the length of the shortest link-disjoint pair of paths 1 and 2, and d(1+2;B'C) is the length of the shortest pair of paths which is link-disjoint, except for link B'C in common to the paths between A and Z; a_{T2} is the threshold value of parameter a_2 for the onset of link commonness. If we assume that the length of a link represents the dollar cost of provisioning service over that link, then the following interpretation arises: The cost of providing two paths between A and Z (in Figure 6) to a customer is cheaper by a_{T2} dollars, if link B'C (or link BC in the original graph) is in common. It is important to note here that there will be situations where regardless of the amount of reduction in the value of a given parameter, a_i, there will be no change in status quo. Refer to Figure 7, which is a modified graph corresponding to the

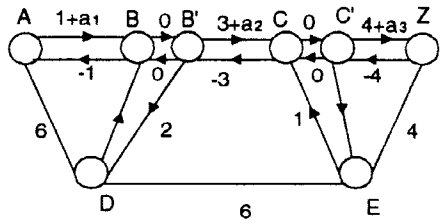

Figure 7

shortest path, ABCZ. $b_i = 0$. Using Eqs. (6) and (7) and setting $\varepsilon = 1$, $a_0 = 224$. Algorithm 2 gives correctly the shortest pair of link-disjoint paths as (ABCZ, ADEZ) with length, d(1+2) = 8 + 16 = 24. Now if $a_2 = a_3 = a_0 = 224$, and a_1 is reduced from its initial value of a_0, then when $a_1 < 3$, the second path found by the shortest path algorithm (Appendix A) is ABDEZ, i.e., link AB is common to the two paths. $a_{1T} = 3$, i.e., the customer saves 3 units if he accepts the physical-diversity violation of link AB

commonness. Now suppose BD = 6, instead of 2, in the graph of Figure 7. Then, no amount of reduction of parameter a_1 (lowest permissible value = 0), will result in link AB commonness. In this situation, the customer cannot have any savings by accepting link AB commonness; rather, such a pair of paths is more costly; thus the better choice is the link-disjoint pair, (ABCZ, ADEZ).

4.2 Node-Disjointness Versus Link-Disjointness

Refer to Figure 8, where $b_1 = 0$ initially and $a_1 = a_2 = a_0$; this is the case of the algorithm finding the shortest pair of link-disjoint paths. Assume the second path in this modified graph is ACBB'DZ. Then the shortest link-disjoint pair is (ABZ, ACBDZ), as seen in the original graph.. In Figure 8, if b_1 is now cranked up from its initial value of zero, a stage would be

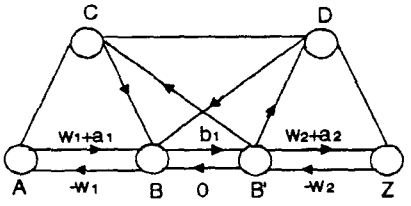

Figure 8

reached when the shortest path between A and Z in the modified graph would turn out to be ACDZ. If b_{1T} is the threshold value when this occurs, then in the original graph, the following relation holds:

$$\text{path ACDZ - path ACBDZ} = b_{1T}. \qquad (11)$$

Equivalently,

$$d(1+2) - d(1+2;B) = b_{1T}, \qquad (12)$$

where d(1+2;B) is the length of the pair of paths with node B in common, and d(1+2) is the length of the node-disjoint pair of paths.

Thus, as before, if the length of the link represents the cost of providing service, then b_{1T} represents cost savings when commonness at node 1 is permitted. Clearly, if the link-disjoint shortest pair had been (ABZ, ACDZ) to start with, then incrementing b_1 from its initial zero value would be redundant.

4.3 Possible Studies

A. Start with $a_i = b_i = 0$ (the two paths are completely identical); keep $b_i = 0$, and, setting $a_i = a$, increase a from zero to study how link-disjointness between the two paths evolves.

B. Keep $a_i = a_0$, and setting $b_i = b$, increase b from zero to see how node-disjointness evolves from link-disjointness.

Hitherto, the parameters, a_i and b_i were either all zero or equal to each other ($a_i = a$ and $b_i = b$). This restriction can be relaxed, allowing for studies pertaining to the variation of a single parameter or simultaneous variation of groups of parameters selected from the sets, a_i and b_i.

C. Keep all parameters, a_i and b_i, fixed at a_0 and b_0, respectively, except for one, say, b_j corresponding to the jth node on the shortest path. The value of b_j may be decreased from its initial large value to study the onset of node commonness at node j, and so on.

D. Let c be the extra cost (penalty) per unit (physical) length of link overlap; replace a_i with c times the physical length of link i. Instead of varying a_i above, vary c ($c \geq 0$).

4.4 Physical Disjointness and Cost Measures

1. Link-disjointness (LD) for a pair of paths may be defined as

$$LD = 1 - \Sigma w_j / \Sigma w_i, \qquad (13)$$

where the j sum is over the common links of the two paths (counted twice) and the i sum is over the links of the two paths.

2. One also defines an increased cost fraction (icf) parameter:

icf = (Cost of 2 paths obtained (minus the artificial cost due to the penalty parameters) - Identical case cost (a_i=0, b_i=0))/ Identical case cost (a_i=0, b_i=0), (14)

and study icf or simply the cost differential versus LD, where LD is defined in Eq. (13). The identical case cost = 2 times the cost of the single (shortest) path, while the two path cost = $d(AZ) + d_0$, defined in Corollary 2.

When single links (or bridge cases) exist, no matter how much one may vary the corresponding a_i, the two paths will have the single links in common, i.e., two completely link-disjoint paths are not possible. Then maximum value of LD is less than unity, and it may be defined as

$$LD_{max} = 1 - 2 \Sigma w_j / \Sigma w_i, \qquad (15)$$

where the j sum is over the bridges between the end point nodes, A and Z, and the i sum is over the two paths found when $b_i = 0$ and $a = a_0$.

3. Similarly, parameters measuring node-disjointness can be defined, and studied as a function of the cost savings.

5. SUMMARY

We have derived a physical-diversity versus cost algorithm from the shortest pair of node-disjoint paths algorithm; this node-disjoint pair algorithm (Bhandari 1994) is different from that of the Suurballe's algorithm (Suurballe 1974; Suurballe and Tarjan 1984) in that it require neither a canonic transformation nor the use of Ford's algorithm; rather, the standard Dijkstra algorithm is modified slightly in the second run of the shortest path algorithm to yield the shortest pair of node-disjoint paths. This node-disjoint algorithm is subsequently modified by the

assignment of forward arcs with appropriately large lengths. Assignment of these fixed large lengths to the forward arcs then results in the algorithm for maximally-disjoint optimal paths, which permits the determination of the number of common nodes and links in the event physical-disjointness is not possible due to the nature of the network. Subsequently, allowing the forward arc parameters to vary leads to the desired physical-disjointness versus cost algorithm. This algorithm can be very useful in assessing cost savings to customers who cannot afford the cost of full physical diversity.

The phyiscal-diversity versus cost algorithm was first given by the author in 1991 as "A Least Cost Algorithm for a Pair of Paths" (February 21, 1991, unpublished), and since then has been extended to deal with physical-diversity problems pertaining to the more complicated real-life networks (see Bhandari 1994).

References

Bhandari, R. 1994. "Optimal Diverse Routing in Telecommunication Fiber Networks." in *Proceedings of IEEE/INFOCOM),* Toronto, Canada, 1498-1508.

Dijkstra, E.W. 1959. "A Note on Two Problems in Connexion with Networks", *Numer. Math.* 1, 269-271.

Suurballe, J.W. 1974. "Disjoint Paths in a Network." *Networks* 4, 125-145.

Suurballe, J.W. and R.E. Tarjan. 1984. "A Quick Method for Finding Shortest Pairs of Disjoint Paths." *Networks* 14, 325-336.

Appendix A

The Modified Dijkstra Algorithm

This algorithm is a slight variant of the Dijkstra algorithm , and is specially tailored to find the shortest distance between a pair of nodes in the modified graph for the construction of the shortest pair of paths. For nonnegative graphs, it reduces to the standard Dijkstra algorithm.

Let d(i) denote the distance of node i from source node A. Let P(i) denote its predecessor.

1. Start with
   d(A)=0, d(i)=l(A,i), if $i \in \Gamma_A$
   =INF, otherwise (INF= Right-hand-side of Eq. (1c))
   $\Gamma_i \equiv$ set of first neighbor nodes of node i, l(i,j)= length of arc from node i to node j.
   Set S=V-{A}.
   Set P(i)=A $\forall$ i $\in$ S.

2. Find j $\in$ S such that d(j)=min d(i), i $\in$ S.
   Set S=S-{j}.
   If j =Z (the destination node), END; otherwise, go to 3.

3. $\forall$ i $\in \Gamma_j$, if d(j)+l(j,i)<d(i), set
   a) d(i)=d(j)+l(j,i), P(i)=j
   b) S=S$\cup${i};
   go to 2.

The algorithm above is different from the Dijkstra algorithm (in Step 3 above) in that it scans all the neighbors of the node selected (or permanently labeled) in Step 2. Furthermore, the algorithm, via step 3b), ensures the inclusion in set S of any node that has been relabeled in step 3a). This particular step is important for the type of graphs (with negative arcs) encountered in the disjoint path construction problem, but is redundant for the nonnegative graphs. Thus, for the latter type of graphs in which step 3b is not needed, this algorithm essentially reduces to the Dijkstra algorithm.

Author's Biography

Ramesh Bhandari obtained his Ph.D. degree in Theoretical Particle Physics from Carnegie-Mellon University in 1978. Thereafter, he performed research in diverse areas such as particle phyiscs, light scattering by particles, and synthetic aperture radar imaging before entering the field of telecommunications. Since joining AT&T Bell Laboratories in 1988, Dr. Bhandari has performd design work for survivable networks, and developed novel optimal graph-theoretic algorithms for physical diversity in real-life networks. Presently, he is constructing new teletraffic models for international networks.

Dr. Bhandari has over 20 research publications. One of his papers was selected and published in a special SPIE volume of outstanding papers on *Light Scattering;* another paper in the area of *Networks* was specially translated into Hungarian by an international scientific committee. Dr. Bhandari has also taught for over 6 years at a number of universities.