A Constrained Minimum Cost *s-t* Cutset Problem

Ramesh Bhandari and Daniel Short
Laboratory for Telecommunications Sciences
8080 Greenmead Drive, College Park, MD 20740
drbhandari617@gmail.com

Abstract

Given an undirected, unweighted graph $G = (V,E)$, what is the minimum number of edge cuts required to separate a source vertex $s \in V$ from the destination vertex $t \in V$, provided the cutset includes a given edge $pq \in E$? A solution set implies that if the cut edge pq is replaced back in the given graph, a path between vertex s and vertex t is reestablished. This problem has potential applications in the telecommunication world where robustness of a network might be tested against multiple link failures. Because the problem is hard to solve, we provide here a heuristic comprising two new techniques called the Graph Collapse and Path Revival.

Keywords: undirected, unweighted graph, robustness, network, constrained, minimum cost, *s-t* cutset, graph collapse, path revival, edge disjoint paths, closure.

1. Introduction

In a data communication network, links connecting routers are periodically brought down for maintenance [1]. Additionally, links fail occasionally, despite maintenance. Consequently, the robustness of network must take into account the possibility of links being out of commission due to maintenance or simply normal failures. With the increase in complexity and size of the data communication networks, multiple failures within a network at any given moment are a real possibility.

Due to the varying technologies of the network components, the failure rate of different links is expected to vary considerably. In this paper, we assume there is one (high capacity) robust link that practically never fails (and does not

require maintenance) while some of the other network links can at any time be down because of maintenance or technical problems. Clearly, some links being down will force the affected network traffic to be rerouted on to other links within the network, which must have enough capacity to carry the extra traffic. As one of the tests for robustness of such a network, we address the problem of identifying the minimum number of simultaneous down links that force the traffic between a given pair of nodes in the network to traverse the given single robust link. We assume that all traffic flows within the network do not necessarily follow the shortest path route. As a result, the past techniques [2-3] cannot be applied. In fact, the problem at hand translates into a *constrained minimum cost s-t cutset problem* described below. To our knowledge, such a problem has not been addressed before.

2. Problem Description

Let $G=(V, E)$ denote an undirected graph, representing a bidirectional network; V is the set of vertices (or nodes), and E is the set of edges (or links); each edge is assigned a weight of unity; we assume loops and multiple edges are absent in the graph. We also make the assumption that graph G is biconnected (or 2-connected), i.e., there always exist a pair of vertex-disjoint paths for any pair of vertices in the graph. This has the implication that there always exists a simple path, which connects a given pair of vertices, s and t, and includes a given arc pq. A simple path refers to a path in which a given vertex is not visited more than once. Unless otherwise stated, in what follows, the term *path* refers to a simple path; the terms *network* and *graph* will be used interchangeably, and so will the terms *node* and *vertex*, and *link* and *edge*.

Let $P(s,t)$ denote a path from vertex s to vertex t in the given graph $G = (V,E)$. Let Γ_a denote an s-t cutset that includes a given edge pq; s-t cutset refers to a set of edges which when cut separate vertex s from vertex t. Clearly, replacing edge pq back in the given graph reconnects vertex s and vertex t.

The problem to solve can be stated as:

Given an undirected, unweighted graph $G = (V,E)$, and a pair of vertices $s,t \in V$, and an edge $pq \in E$,

$$\text{minimize } |\Gamma_a| \tag{1}$$

subject to arc pq (or arc qp) $\in P(s,t)$.

Eq. (1) may be referred to as a constrained minimum cost s-t cutset problem, which seeks the minimum number of edges to cut in a given network graph

(with edge weights set to unity) such that there is no path connecting node *s* to node *t*. In contrast to the earlier problems [2-3], we do not make the assumption of traffic flow following the shortest paths.

An algorithm to find *all* minimum cost *s-t* cutsets within a given (weighted) graph [4] exists. One might think that one way to solve the *constrained minimum cost s-t cutset* problem is to find all minimum cost *s-t* cutsets in the graph (with the weights set to unity each), and then identify those that include edge *pq*. But there is no guarantee that the minimum cost *s-t* cutsets obtained would include edge *pq* as one of the cuts. Furthermore, especially with each edge weight set to unity, the number of all cutsets can grow exponentially with the size of the graph, making the problem intractable.

In this paper, to solve the constrained problem, we employ a heuristic procedure comprising two new techniques: Graph Collapse technique, which reduces the problem to a minimum-cost cutset *s-t* problem, and the Path-Revival algorithm, which then solves the minimum cost *s-t* cutset problem; the advantage of this latter new algorithm over the *all* minimum *s-t* cutsets technique [4] (where multiple solutions are found simultaneously) is that it allows us to find multiple solutions sequentially, permitting us to stop when a certain desired number of solutions is reached. In Sections 3 and 4, we describe the Graph Collapse technique and the Path Revival algorithm.

3. Graph Collapse

Through the Graph Collapse technique, we can reduce the constrained problem to a minimum-cost cut set problem. The Graph Collapse technique takes two node-disjoint paths, the first one from the source node *s* to one end of the diversion link and the second one from the other end of the diversion link to the destination node *t*, and collapses them into one node each. By collapsing both paths, the diversion link then stretches across the source node *s* and the destination node *t*, and the problem reduces to finding the minimum cost *s-t* cutset in the remaining graph (given graph *G* minus the diversion edge) to force the *s-t* traffic to traverse the diversion edge. Note that, because of the path collapse, no edge belonging to the collapsed paths (the nodes-disjoint paths) is cut[1].

[1] Alternatively, instead of collapsing the two node-disjoint paths, one could assign artificial weights of "infinity" to the edges of these two paths, and apply the all minimum cost *s-t* cutset technique to solve the constrained minimum cost *s-t* cutset problem. The advantage of the Graph Collapse technique is that the resulting graph becomes immediately amenable to application of the new Path Revival technique employed in this paper.

The Graph Collapse technique applicable to the two node-disjoint paths is described fully in a pseudocode below:

Pseudocode for Graph Collapse technique (for one path of the node-disjoint pair):
Let Path = {nodes on the path to be collapsed}
Let Path(*i*) = {ith node on the path}
For each Path(*i*), do:
  For all node(*j*) with an arc connected to Path(*i*) in both directions
    if node(j) is in Path, remove the link from the graph.
For each Path(*i*) except for Path(1) do:
  For all nodes(j) with an arc connected to Path(*i*), do:
    if arc (*i*,*j*) exists, then create a new arc to (Path(1),*j*), delete arc (*i*,*j*).
    if arc(*j*,*i*) exists, then create a new arc to (*j*, Path(1)), delete arc (*j*,*i*).
    In both cases, establish a list of form (*j*, *i*, Path(1)) in order to track changes.

In the example below, Figure 2.1, we have a graph with two shortest node-disjoint paths (in bold lines)[2] from *s* to *p*, and *q* to *t*. We desire to collapse both paths in order to be able to reduce the problem to a minimum cost *s*-*t* cutset.

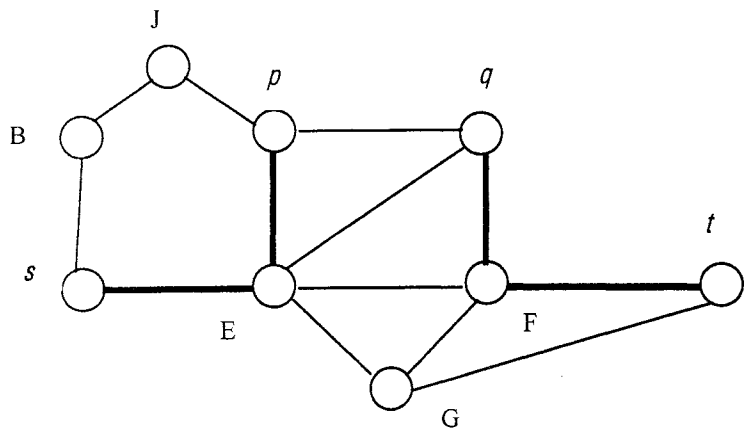

Figure 2.1 Given graph before collapsing paths *s*E*p* and *q*F*t*.

[2] There are standard algorithms to determine such disjoint paths; see, e.g., Ref. [5].

Following the above pseudocode, we find that the graph after the collapse of the two node-disjoint paths looks like the graph in Figure 2.2. The three nodes *s*, E, and *p* of one path are coalesced into a single node *s*E*p*, and the nodes *q*,F, and *t* of the second path are coalesced also into a single node *t*F*q*. Multiple edges appear, which can be taken care of in the implementation process by inserting dummy nodes (see Appendix A). The set of three multiple edges includes the link *pq*. The cardinality of the minimum cost *s-t* cutset here is 4. Replacing the edge *pq* back in the graph, we find the minimum number of edges to cut to force the *s-t* traffic over edge *pq* is 3 (as determined from the chosen node-disjoint paths; see Section 6 for further discussions).

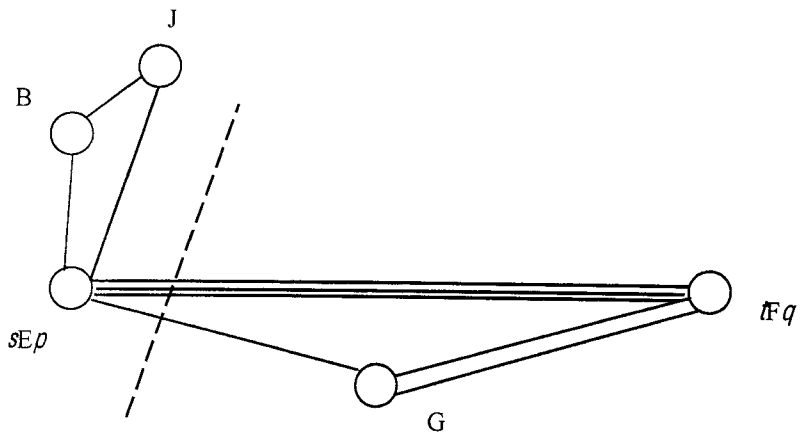

Fig 2.2 Graph after collapsing paths *s*E*p* and *q*F*t*; the dashed line indicates the minimum cost *s-t* cutset cut that separates node *s* from node *t*.

4. Path Revival

After the graph is collapsed into a minimum cost *s-t* cut set problem, there are various algorithms [4] that can be used to solve the problem, but as telecommunication networks tend to be sparse, a new minimum cost cut set algorithm, called the Path Revival Algorithm, is developed. It is a unique method via which the multiple solutions, if present, are obtained sequentially, allowing one to terminate the search when desired. It is predicated on the well known result that the cardinality of the minimum cost *s-t* cutset is equal to the maximum number of edge-disjoint paths between nodes *s* and t [6-7].

Instead of applying maximal flow algorithm [4], one starts with a *k* edge-disjoint path algorithm to find all possible edge-disjoint *s-t* paths. Once the *k* edge-disjoint paths are found, they are removed from the network and then revived

one by one. The initial removal of the edge-disjoint paths disconnects node *s* from *t*. As the first path is revived, we search for a path maximally edge-disjoint from the revived path. The idea is to determine which edges of the revived path are likely to be a candidate for solution. Clearly, only the overlapping edges of the two paths (the revived path and the maximally edge-disjoint path) are candidate solutions, since cutting any of the overlapping edges disconnects node *s* from node *t*; non-overlapping edges form cycles embedded in the path from source to sink and edges belonging to these cycles cannot be candidates because there is only one cut per path from source to the sink (the number of cuts = number of edge-disjoint paths). Once a cut candidate is established, it is removed from the graph and the next path is revived. Once the path is revived, the procedure is repeated until all *k* edge disjoint paths have been revived and the solution set composed of selected cut candidates is established as a final solution. Because there can be more than one cut candidate/path revival in the above iterative method, there can be multiple solutions. Below, we give the pseudocode for Path Revival:

Psuedocode for Path Revival Algorithm

```
Given a graph with source (s) and sink (t)
Run k edge-disjoint algorithm from s to t and record all paths
Remove all links on all paths from the graph
Revive one of any paths found from k edge-disjoint algorithm
Begin Recursive Procedure: Path Revival

Recursion: Path Revival
Find Maximal edge-disjoint path (see Appendix B)
For each edge on the revived path,
  If edge(i) is on the revived path and the maximally edge-disjoint path
    edge(i) is a cut candidate, add to Cut List
    If there is a path remaining,
      remove edge(i)
      revive next path
      Run Path Revival
      add edge(i)
    Else
      Add Cut List to Solution Set
    Remove edge(i) from the Cut List
```

Let us illustrate the Path Revival algorithm with the graph in Figure 3.1. There are three shortest edge-disjoint paths from source to the sink; these are *s*AB*t*, *s*CDEF*t*. and *s*G*t*. The listed paths are deactivated, and the result is shown in the Figure 3.2.

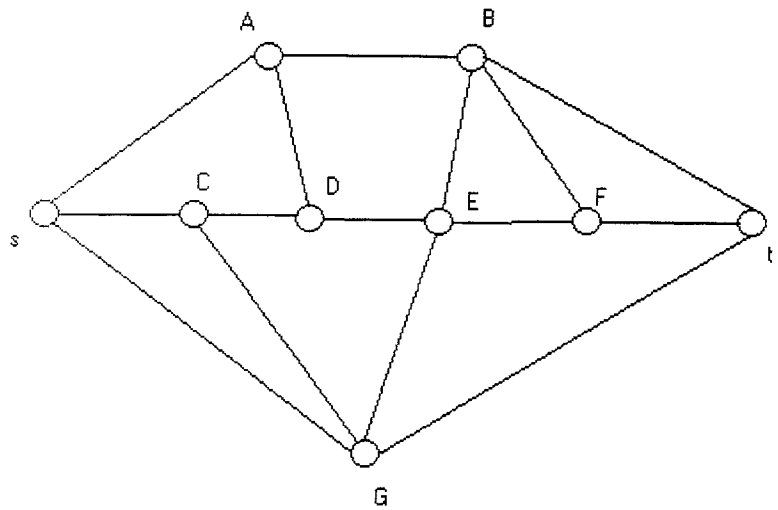

Figure 3.1 Example Graph

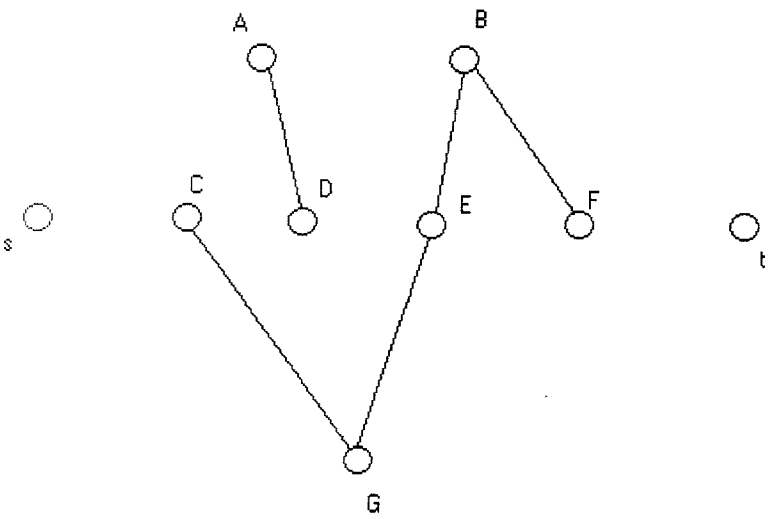

Figure 3.2 Graph after deactivating paths from s to t

Let path *s*CDEF*t* be revived. We now search for a maximally edge-disjoint path in the resulting graph (Figure 3.3). The maximally edge-disjoint path is *s*CGEBF*t*. The overlapping edges *s*C and F*t* are identified as cut candidates. We choose one of them, say *s*C, remove it from the graph, and revive a new path *s*AB*t* (belonging to the edge-disjoint path set) as shown in Figure 3.3.

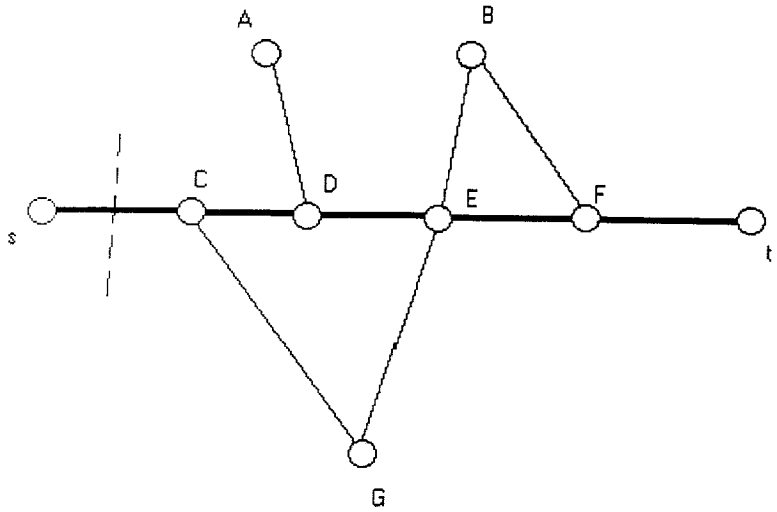

Figure 3.3 Reviving *s*CDEF*t*, and then cutting *s*C

This graph shows the new path, *s*AB*t*, and with a maximal edge disjoint path being *s*ADEF*t*, we have one cut candidate, *s*A (the only non-overlapping edge between paths *s*AB*t* and *s*ADEF*t*)[3] We then remove *s*A and revive *s*G*t*, leaving us with *s*G as a cut candidate. Now that all paths from *k*-edge disjoint algorithm are exhausted, we obtain a solution set of {*s*G, *s*C, *s*A}. With this solution set in place, we can continue with Path Revival algorithm and search for any more solutions, so we go back to when there was a choice amongst cut candidates, *s*C and F*t* in the first iteration of the path revival process, and now try F*t* as a cut candidate. Trying F*t* as a candidate will yield us {F*t*, B*t*, G*t*} as a different possible solution set. So the resulting optimal solutions are {*s*A, *s*C, *s*G} and {B*t*, F*t*, G*t*}.

[3] Note that there are additional maximally edge-disjoint paths (*s*ADEBF*t*, *s*ADCGEF*t*, etc), but the answer is independent of which path is chosen.

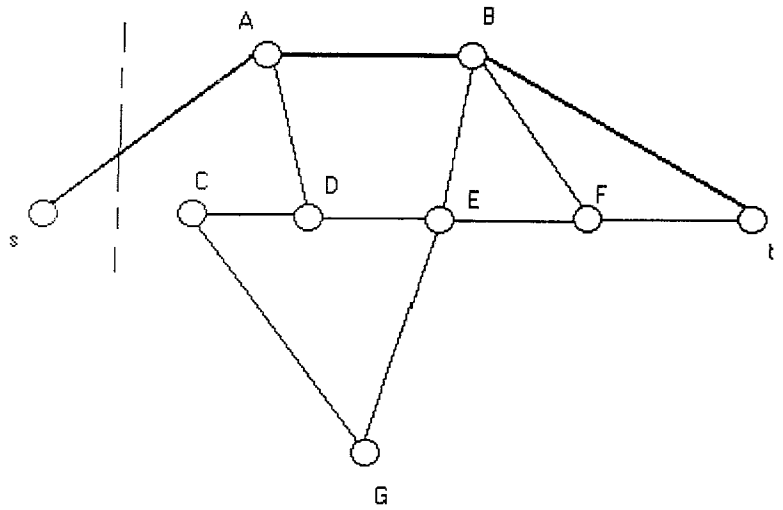

Figure 3.4 Reviving *s*AB*t*, then cutting *s*A

Proposition: Path Revival algorithm will always yield optimal and valid minimum *s-t* cutsets in a given graph.

Proof: Since *k*-edge disjoint path algorithm is used, it will give us *k* paths (the maximum number of edge-disjoint paths). There cannot be any more paths, otherwise *k*-path edge disjoint algorithm would have found those extra paths. Because there are *k* edge-disjoint paths, *k* is the minimal number of edge cuts needed to separate *s* from *t* (one particular edge cut on each of the *k* edge-disjoint paths yields the cutset) [6-7].

In the Path Revival Algorithm, we delete all paths found by *k*-edge disjoint path algorithm; this disconnects node *s* from node *t*. We then revive one path at a time. Suppose, when we search for the maximally-disjoint path, we discover a disjoint path from source to sink. This discovery then would imply that nodes *s* and *t* were already connected, leading to a contradiction. So there cannot be any edge-disjoint paths from source to the sink after reviving a path. Thus, an application of maximally edge-disjoint path algorithm [5] after path revival will always yield a set of overlapping edges. All edges in this set are cut candidates because it takes only one edge-cut to separate *s* from *t*. Once this candidate cut is removed from the graph prior to the next iteration, nodes *s* and *t* are disconnected once again and any subsequent path revival yields similarly a single edge-cut (belonging to the revived path) to separate node *s* from node *t*. Thus, at the end of *k* path revivals, the solution set contains *k* edges in the cutset.

Within the given graph, the nodes s and t still remain disconnected. Because the cut of all the edges belonging to the obtained solution set keeps node s disconnected from node t, this set constitutes a correct solution. Similar arguments establish the optimality and correctness of other solutions that arise from the case of more than one overlapping edge in the path revival process. *End of Proof.*

5. Comparison of Path Revival algorithm with Other Algorithms

In this section, we will compare Path Revival to an existing algorithm, called Closure algorithm described in. [4]. Closure method uses maximum flow algorithm to find all shortest edge disjoint paths from s to t, and then uses the paths to construct all minimum cost s-t cutsets. The closure of a set in the graph is used to determine which group of nodes does not constitute a cut set.

While Closure method uses maximum flow algorithm, Path Revival method uses the maximal edge-disjoint path algorithm simply to find alternative paths for every revived path in order to discover overlapping edges, which then constitute cut candidates. That is, Path Revival checks one revived path at a time to find cut candidates, while Closure examines the entire network to discover cutsets.

In Appendix C, we provide tables for the run times using both our technique of Path Revival and the Closure technique[4] [4]. Table 1 corresponds to the case of a single unique solution, while Tables 2 and 3 correspond to the case of multiple solutions (2 solutions). These tables indicate that Path Revival is better than Closure on sparse graph with few disjoint paths from the source to the sink, or equivalently, when the cardinality of the minimum cost s-t cutset is small (maximum number of disjoint paths = cardinality of the minimum cost s-t cutset). The critical cardinality value that determines which method is more efficient lowers whenever there are more solutions (Tables 2 and 3). If desired, because of the sequential nature in which the solutions are found, Path Revival algorithm can be modified so that it terminates as soon as a certain number of solution sets is found. As a result, the run time will often be reduced by a significant amount (see Tables 4 and 5 where Path Revival terminates after finding a single solution, while the Closure technique completes its full search, yielding two possible solutions).

---

[4]In creating the computer code for the Closure technique, we use, instead of the maximum flow algorithm, the k-edge disjoint path algorithm, which in our experience is faster than the maximum flow algorithms for finding disjoint paths.

In the end, it is possible to be able to implement both procedures, since both methods start off by searching for paths from s to t. The choice depends upon how many edge-disjoint paths exist from s to t, and how many solutions we might be interested in. That is, if we want a single solution, then Path Revival is expected to perform better. If multiple solutions are desired, then Closure algorithm might be better than Path Revival, although Path Revival has the distinct advantage of terminating upon finding the desired number of solutions (one or more, but not all). Also, if there are only a few disjoint paths (one to four different paths from s to t), then Path Revival is the better choice on sparse network.

6. Further Discussions

1) While the Path Revival algorithm yields all optimal s-t cutsets for the solution of the minimum cost s-t cutset problem, it is important to note that the overall algorithm for the solution of the constrained minimum cost s-t cutset problem (Graph Collapse + Path Revival) remains a heuristic. This is due to the fact that employing a different pair of node-disjoint paths in the Graph Collapse technique can yield different cutset results. In the example of Figure 2.1, the reader can verify that choosing path sBJp, instead of path sEp, yields as the solution the cutset {sE, pE}, which is a solution of lower cardinality than the previous solution (Figure 2.2), and therefore more desirable.

   Clearly, one way to improve the current method to solve the constrained minimum cost s-t cutset problem is to try different pairs of node-disjoint paths (one connecting node s to p (or q) and the other one connecting node t to q (or p)), and select from the results corresponding to these different pairs of paths, the best solution. This, of course, implies greater computational run times, depending upon how many such node-disjoint pairs exist and are tried. Clearly, one would benefit from the construction of a k-shortest pair of node-disjoint paths, much akin to the algorithm for the k-shortest path algorithm (which lists shortest paths between a pair of nodes in an ascending order) [8]; the k-shortest pair of node-disjoint paths algorithm would similarly list all the shortest pairs of node-disjoint paths in an ascending order. To the authors' knowledge such an algorithm does not exist in literature. While the solution to the constrained minimum cost s-t cutset problem can be improved with the use of multiple disjoint pairs of paths, there is still no guarantee that the best solution selected would be the optimal solution, unless all existing node disjoint pairs of paths are utilized, which would make the problem intractable for large graphs.

2) It should be noted that Graph Collapse technique is not just limited to minimum-cost cut set problem. It could be used to exclude certain (forbidden) parts of the networks while searching for cut sets using algorithms. In the example of Figure 2.1, if link E*q* is a forbidden link, we would redefine the given network graph by collapsing the edge E*q* into a single node E*q*. This way the forbidden link is never a candidate link in the cutset determination process.

7. Summary

In this paper we have addressed the mathematical problem of finding a minimum *s-t* cutset that includes a specific edge in an undirected graph (with each edge having a weight =1). This problem arises in designing and checking for robustness in networks. Because the problem is hard to solve, we devise a heuristic procedure comprising two new techniques: Graph Collapse technique, which reduces the problem to the standard minimum cost *s-t* cutset, and the Path Revival algorithm, which allows us to obtain multiple solutions in a sequential manner; this has the advantage that the algorithm process can be stopped after a desired number of solutions have been obtained, which is in sharp contrast to an existing method where all the solutions are obtained simultaneously at the very end of the algorithm process. As a result, the new Path Revival algorithm is fast for sparse networks, clearly winning out when a single solution is desired, as numerical results show.

The above heuristic procedure uses as input the shortest pair of node-disjoint paths, one connecting the source node to one end of the desired constraint edge *pq* and the other path connecting the destination node to the other end of the constraint edge. Each path of this shortest pair of node-disjoint paths is collapsed into a single node by the Graph Collapse technique. In principle there can be several node-disjoint pairs of paths. Since the results of the Graph Collapse technique are dependent upon which node-disjoint pair is used, the procedure presented to solve the constrained minimum cost *s-t* cutset problem is a heuristic. Optimality of the solutions obtained can be improved by running the heuristic procedure a number of times against different selected pairs of node disjoint paths, and selecting the best solution set. In large graphs, this can become very time consuming. Clearly, a *k*-node-disjoint pair of paths algorithm would be highly desirable here. To our knowledge, such an algorithm does not exist in literature.

Appendix A: Taking care of multiple edges

This can be accomplished by inserting a dummy node for each additional edge connecting a given pair of nodes in the collapsed graph. Figure below illustrates this concept:

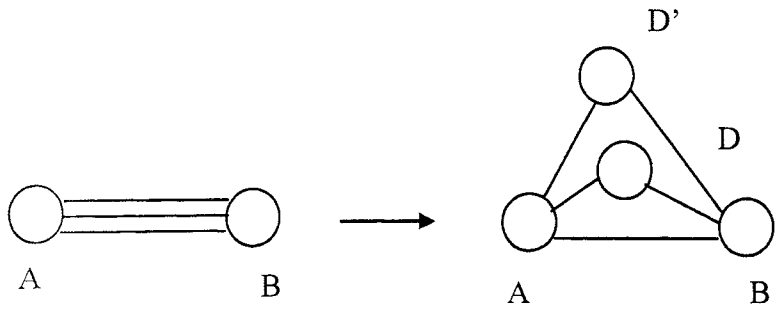

Figure A.1 Two nodes A and B connected by multiple edges; introduction of dummy nodes D and D' eliminates the multiple edges.

Appendix B: Maximal Edge-disjoint Path Pair Algorithm for Path Revival Algorithm.

In this Maximal Edge-disjoint path pair for this problem, we are only interested in looking for any overlaps between two different paths; the overlapping edges then become cut-candidates. We are not interested in shortest path length, etc.

Pseudocode for Maximal Edge-disjoint path pair:

```
With a path given, remove all forward arcs on the path.
Also, let List = {all members on the path}
For each node on the path, label it 0.
Let ListPos = {1} /* We will be trying to get to highest possible position on the
                     list when searching. If we cannot get to the end of the
                     list, then top position and next position will be the cut
                     candidate. */
For first node on the path, label it 1 /* We are starting search from first node.
                                         Each node found will be labeled 1.*/
While(label of last list item == 0)
  Let List2 = path(ListPos)
  While(List2 is not empty)
    Let CurrentNode = List2(1)
    Let Label(CurrentNode) = 1
    Remove first item from List2.
    Search all outward arcs from CurrentNode.
    If last node on the path is found, terminate the algorithm, return all cutsets.
    If the outward arcs reaches nodes not searched (label == 0),
      Label the unsearched node with 1, and add to list2.
    If the outward arcs reaches nodes that has label == 1, ignore the node.
  end

  Select highest positioned labeled node on the List. Highest labeled node and
  the next node is established as a cut candidate. Set ListPos to the position of
  node after the highest labeled node.
end
```

Appendix C: Comparison of Run Times (sec) between Path Revival Algorithm and Closure Algorithm; N is the number of nodes in the graph; higher the value of k, larger the connectivity, and denser the graph; k is the number of edge-disjoint paths from the source (s) to the sink (t), which also equals the cardinality of the minimum cost s-t cutsets found.

Table 1. Sample Data of 1-solution Path Revival vs. Closure; $N = 500$.

PathRev	Closure	Difference	k
0.0309	0.088	-0.0571	1
0.02575	0.08335	-0.0576	2
0.05435	0.10955	-0.0552	2
0.09105	0.1012	-0.01015	5
0.064	0.11505	-0.05105	6
0.179	0.1666	0.0124	9
0.2097	0.1802	0.0295	10
0.29435	0.2197	0.07465	14

Table 2. Sample Data of 2-solution Path Revival vs. Closure; $N = 100$

PathRev	Closure	Delta	k
0.01615	0.02125	-0.0051	2
0.0212	0.02245	-0.00125	3
0.0279	0.02505	0.00285	4
0.0326	0.02705	0.00555	5
0.03735	0.0282	0.00915	5
0.0497	0.0484	0.0013	8
0.055	0.02865	0.02635	9
0.06415	0.04	0.02415	11

Table 3. Sample Data of 2-solution Path Revival vs Closure; $N = 200$

PathRev	Closure	Delta	k
0.02305	0.03715	-0.0141	2
0.0327	0.0375	-0.0048	3
0.04605	0.04645	-0.0004	4
0.0516	0.04815	0.00345	5
0.06295	0.04805	0.0149	6
0.0705	0.053	0.0175	7
0.07985	0.0581	0.02175	8

0.1016	0.06885	0.03275	10
0.1132	0.06805	0.04515	11

Table 4. Sample Data of 2-solution Closure vs One-solution Path Revival; N = 100

OnePR	Closure	Difference	k
0.00985	0.02125	-0.0114	2
0.0128	0.02245	-0.00965	3
0.01635	0.02505	-0.0087	4
0.02185	0.02705	-0.0052	5
0.0215	0.0282	-0.0067	5
0.0298	0.0484	-0.0186	8
0.0323	0.02865	0.00365	9
0.0399	0.04	-1E-04	11

Table 5. Sample Data of 2-solution Closure vs. One-solution Path Revival; N = 200

OnePR	Closure	Difference	k
0.01465	0.03715	-0.0225	2
0.0211	0.0375	-0.0164	3
0.0289	0.04645	-0.01755	4
0.03145	0.04815	-0.0167	5
0.03695	0.04805	-0.0111	6
0.0431	0.053	-0.0099	7
0.0485	0.0581	-0.0096	8
0.0607	0.06885	-0.00815	10
0.0642	0.06805	-0.00385	11

References

[1] P. Francois, M. Shand, and O. Bonaventure, "*Disruption Free Topology Reconfiguration in OSPF Networks*", Proc. of IEEE International Conference on Computer Communications (INFOCOM), Anchorage, Alaska (2007).

[2] R. Bhandari, *Sliding Shortest Path Algorithms*, Proc. of the Cologne-Twente Workshop on Graphs and Combinatorial Optimization, Paris, France (2009), pp. 97-101.

[3] R. Bhandari, *The Improved Sliding Shortest Path Algorithm*, Congressus Numerantium, 203 (2010), pp.175-192, a refereed journal of the 41st Southeastern International Conference on Graph Theory, Combinatorics, and Computing, Florida Atlantic University, Boca Raton, Florida, March 2010.

[4] Norm Curet, Jason DeVinney, and Mathew Gaston, *An Efficient Network Flow Code for Finding All Minimum Cost s-t Cutsets*, Computers & Operations Research 29, 205-219 (2002)

[5] Ramesh Bhandari, *Survivable Networks: Algorithms for Diverse Routing*, Kluwer Academic Publishers (1998).

[6] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, The MIT Press (1990).

[7] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows: Theory, Algorithms and Applications*, Englewood Cliffs, NJ Prentice Hall (1993).

[8] J. Yen, *Finding the k Shortest Loopless Paths in a Network*, Management Science, 17(11), 712-716 (1971)