21

Graph Theory for Survivability Design in Communication Networks

Daryoush Habibi and Quoc Viet Phung
Edith Cowan University
Australia

1. Introduction

Design of survivable communication networks has been a complex task. Without establishing network survivability, there can be severe consequences when a physical link fails. Network failures which may be caused by dig-ups, vehicle crashes, human errors, system malfunctions, fire, rodents, sabotage, natural disasters (e.g. floods, earthquakes, lightning storms), and some other factors, have occurred quite frequently and sometimes with unpredictable consequences. To tackle these, survivability measures in a communication network can be implemented at the service layer, the logical layer, the system layer, and the physical layer.

The physical layer is the base resource infrastructure of the network, and to be able to protect it, we need to ensure that the physical topology of the network has sufficient link and node diversity. Without this, protection at higher layers will not be feasible. With the implementation of Dense Wavelength Division Multiplexing (DWDM) in the optical backbone of metropolitan and long-haul networks, greater flexibility is achieved in providing alternate routes for light-path connections. However, the survivability problem at the physical layer remains the same. In fact, it becomes even more critical, because each link of a backbone network carries huge amounts of traffic and the failure of an optical component, such as a fiber cut or a node failure, may cause a very serious problem in terms of loss of data and profit.

The physical topology of a network is considered to be survivable if it can cope with failure scenarios occurring at network components. In other words, the physical topology must remain connected under the failure scenario. For example, to cope with single link failures in the network, the physical topology must be at least 2-connected, meaning that there is at least 2 link-disjoint paths between any two nodes in the network. Generally, to protect against the failure of any set of k links in a network, the physical topology of that network must be $(k+1)$-connected. Menger's theorem (Menger, 1927) gives the necessary and sufficient condition for survivability of networks at the physical layer, using the connectivity between network's cut-sets. However, the computational complexity of this model grows exponentially with the size of the network, since a network with $|V|$ nodes would yield $2^{|V|}-2$ cut-sets. Therefore, the cut-set technique cannot efficiently deal with even moderate size networks of say 40 nodes, and larger networks are out of computational reach of this technique. Testing for survivability of large networks can be done using a technique called bi-connected components of a graph introduced by W. D. Grover (Grover, 2004). This technique can determine vulnerable links and nodes of the network. However, verifying network survivability is just the first step in

network planning, after which we need to apply appropriate protection routing schemes using such techniques as Shared Backup Path Protection (SBPP), Pre-configured Protection Cycles (p-cycle), or ring protection. It is therefore very helpful if the algorithm used for determining the physical survivability of the network can also provide additional information which is of benefit to protection design.

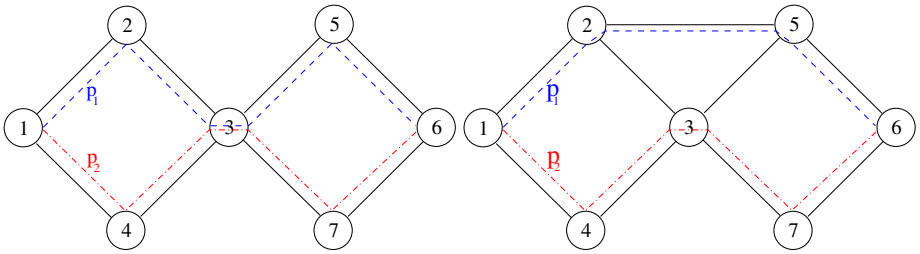
2. Chapter outline

It is generally understood that research in transport networks has a close connection to graph theory. A graph can be used to present key aspects of a network, such as its topology and/or the associated capacity. Indeed, a network consists of many more elements such as the physical equipment (e.g. OADMs, OXCs, etc.), fibre cables, and so on. The objective of this chapter is to provide an overview of network connectivity in relation to network protection design. In addition, this chapter also aims to introduce and analyse the advantages and disadvantages of methods and algorithms for searching network connectivity as well as sets of disjoint and distinct paths for protection design.

Given these objectives, the chapter is organized as follows. Section 3 discusses the connectivity property of graph in relation to network protection design. Section 4 discusses two approaches to establishing the survivability of physical topology. In Section 5, we present the problem of diverse routing and graph algorithms. Finally, we close this chapter in Section 6 by summarising the key aspects of network connectivity in network protection design.

3. Graph connectivity in relation to network protection design

Practically, different traffic requirements over a network would require different connectivity between nodes. Some traffic demands may require no protection, or may only need to be carried when possible. In contrast, other traffic demands may ask for a full protection against either single-link failure, dual-link failures or other types of failure. Hence, the required connectivity between nodes would be different for these varied protection requirements. This section first presents the general requirement for network connectivity in which the network can at least be protected against any single failure scenario, e.g. the failure of any single link or node of the network. In this case, the physical topology of the network must be 2-connected.



(a) Two-connected graph (b) Biconnected graph

Fig. 1. Two connected graph versus biconnected graph

The protection mechanism is designed to maintain the continuity of services under network failures. The failure can be minor such as a channel failure, or be more significant such as failure of groups of links or nodes. As far as the connectivity of the physical topology is

concerned, the failure may be in a single link, a single node, a group of links, a group of nodes. Protection design cannot cover all failure scenarios. Instead, it will consider the set of specific (or predetermined) failures. For example, protection design against single link failures needs to determine the recovery routes for services so that they can maintain their services under the failure of any single link in the network. To do that, the network connectivity must provide at least 2 *disjoint* paths between any source and destination nodes. The term "disjoint" here is with respect to the failure scenario, meaning that the "disjoint" paths must not suffer from the same failure. For example, for single link failures, the 2 paths must be link-disjoint. Similarly, for single node failures, the 2 paths must be node-disjoint.

Fig. 1 shows an example of a pair of link-disjoint paths (Fig. 1(a)) and node-disjoint paths (Fig. 1(b)). It can be easily seen that link-disjoint paths may share nodes along their paths, e.g. node 3 on paths p_1 and p_2 in Fig. 1(a). Hence, link-disjoint paths may not be node-disjoint. On the other hand, node-disjoint paths are always link-disjoint.

A graph which provides at least 2 link-disjoint paths between any two nodes is 2-*connected*. With stronger connectivity, a *biconnected* graph is able to provide at least 2 node-disjoint paths between any two nodes.

Generally, a network must provide at least K link-disjoint paths between any node-pairs to be able to protect against simultaneous failure of $K-1$ links. The graph of such networks is said to be K-*connected*. The rest of this chapter will mainly discuss network connectivity which supports single link and single node failures, known as 2-connected and biconnected.

4. Establishing physical survivability

This section presents the procedures for establishing network survivability against single-link failures which is one of the most common failure scenarios occurring in practical networks. As discussed, the physical topology of a network can only be survivable under single-link failures if and only if it is 2-connected. Manual verification for survivability is only suitable with small networks where designers can perform the verification in just few seconds or few minutes. However, manual verification may take hours or even days for large scale networks. Furthermore, it is prone to human errors. Therefore, automatic survivability verification is important in both theory and practice. The concept of survivable networks is more complex than the concept of connectivity in graph theory. In addition, efficient automation algorithms based on graph theory can help designers to reduce the computational time and avoid human errors. This section presents techniques for evaluating the physical survivability of networks. Firstly, we outline and analyse the strengths and weaknesses of a popular method, namely the cut-set method. Then, we introduce two comparable techniques that can deal with network sizes of many thousand nodes (Habibi et al., 2005). One technique is based on Depth-First Search (DFS) and the other uses properties of 2-connected graphs.

4.1 Survivability via cut-sets

A network is survivable if the size of every cut-set of the network is equal to or larger than 2. At a glance, this definition leads to a view that the network has nodal-degree of two, meaning that every node in the network is connected to at least two other nodes. Since every node is connected to at least two other nodes in the network, on the surface this property seems to be able to offer two disjoint paths between any two nodes in the network. In fact, this is a misconception. If a network is 2-connected then the nodal degree of all nodes in the

network is equal to or larger than 2. The reverse does not hold, however, in that a network in which the nodal degree of all its nodes equal to or larger than 2 is not always 2-connected. The topology in Fig. 2 illustrates this concept. In that Figure we can see that path $(3-5-6)$ is a bridge that connects two subsets of network nodes $X = \{1,2,3,4\}$ and $Y = \{6,7,8,9\}$. As a result, all paths between nodes $x \in X$ and $y \in Y$ must share the same path $(3-5-6)$. Hence, although all nodes in this network have a nodal degree equal to or larger than 2, it is not a 2-connected network.

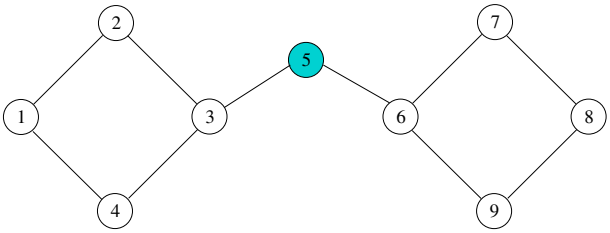


Fig. 2. An illustration of the failure of the nodal degree technique

Therefore, all algorithms for verification of network survivability based on node-degree of two may yield undesirable and inaccurate results and hence are not reliable. The cut-set assumption described below has been preferred for the accuracy of network survivability verification.

Let $G = (V, E)$ be a network topology. A cut in G is a partition of V into parts S and $\bar{S} = V \setminus S$. Each cut defines a set of edges consisting of those edges in E with one end-point in S and the other in $\bar{S}$. This edge set is referred as the cut-set $CS(S, V \setminus S)$ associated with the cut $\langle S, V \setminus S \rangle$. Let $|CS(S, V \setminus S)|$ be the size of the cut-set, being the number of links between S and $V \setminus S$. Thus, according to the cut-set assumption, **a network is 2-connected if** $|CS(S, V \setminus S)| \geq 2, \forall S \subset V$. If S is a subset of only a single node in the network, then the cut-set assumption is essentially the same as the node-degree assumption.

Since the cut-set assumption is related the number of links connected between two subsets of a cut, it can assure the network to offer **link-disjoint paths**, but **not node-disjoint paths**. In other words, a configuration of the network that satisfies the condition of the cut-set assumption can provide at least one link-disjoint path-pair between any distinct pair of source node and destination node.

The implementation of the cut-set assumption is not complex but its computational time for large scale networks is its biggest disadvantage. The number of cut-sets increases exponentially with the number of network nodes and is calculated as (Grover, 2004):

$$N_{cutset} = 2^{|V|} - 2$$

where N_{cutset} is the number of cut-sets in the network, and $|V|$ is the number of network nodes.

Table 1 shows the example of the number of the possible cut-sets, N_{cutset}, versus the number of network nodes $|V|$. The number of cut-sets doubles with an increase of one node in the network. For instance, N_{cutset} in a network of 20 nodes is over 1 million; and it is over 32 million with $|V| = 25$; which is $32(= 2^5)$ times larger than $|V| = 20$; and the number of

cut-sets in the networks of $|V| = 30$ nodes is up to 1 billion cut-sets. So, the cut-set technique becomes intractable even with moderate scale networks ($20 \leq |V| \leq 30$).

$\|V\|$	20	25	30
N_{cutset}	1,048,574	33,554,430	1,073,741,822

Table 1. The number of cut-sets versus the number of network nodes

In summary, the node-degree assumption is simple but not reliable for the verification of network survivability. Meanwhile, the cut-set assumption is only applicable for link-survivable networks, and it is intractable with large scale networks. The verification ability of these two assumptions for different connectivities of the physical topology are summarised in Table 2. The node-degree assumption cannot verify any type of physical topology that has potential to support network survivability (namely 2-connected and biconnected networks) whereas the cut-set assumption can verify the survivability of a network that is 2-connected but cannot identify exactly a 2-connected topology or verify a biconnected topology. Next, we propose an approach that can classify network topologies, and determine if they are unconnected, (1−)connected, 2-connected or biconnected.

	Network connectivity			
	Disconnected	(1-)Connected	2-connected	Biconnected
Node-degree assumption	Yes	Yes	No	No
Cut-set Assumption	Yes	Yes	Yes	No

Table 2. Performance of two common assumptions in terms of network connectivity

4.2 An approach to verifying physical topology for designing network survivability

Let $G(V, E)$ be the graph presenting the physical topology of a network. If the degree of any node in G is zero, the graph is disconnected. If the degree of any node in G is 1, the graph cannot be 2-connected or biconnected. For the sake of network survivability design, from here after, we assume that the degree of every node in the graph is at least 2. This condition allows every node to belong to a biconnected component or a bridge. Let G' and G'' be two biconnected components of the graph G. The relationship between G' and G'' determines the connectivity of the graph:

- If G' and G'' have at least 2 common nodes, then G is a 2-connected graph with no cut-node (i.e. node bridge) or cut-link (i.e. link bridge). In other words, G is a biconnected graph.
- If G' and G'' only have one common node, then G is a 2-connected graph with an *articulation node* which is the common node.
- If G' and G'' are separated by a cut-link (or a cut-path), then G is not a 2-connected graph, and the cut-link (or cut-path) cannot be protected.
- If G' and G'' have no common links or nodes, then G is a disconnected graph.

The key point for determining the connectivity of a graph is to find all biconnected components in the graph and check the relationship between these components. Biconnected components can be found using either Depth-First Search (DFS) or the properties of the 2-connected graph.

4.2.1 Finding biconnected components using Depth-First Search (DFS)

Algorithm 1 outlines the pseudo-code for verifying a 2-edge-connected graph. The principle is to find a bridge (edge) via articulation nodes - a cut at one of these nodes would disconnect the graph. A bridge is an edge or a path segment that connects two biconnected component neighbourhood. A graph is 2-edge-connected if it contains no bridge. The DFS algorithm for finding biconnected components was proposed by Robert Tarjan (Tarjan, 1971), and subsequently, revised for a particular study in survivable mesh networks in (Grover, 2004). Algorithm 1 is a modified version of the original DFS algorithm in (Grover, 2004; Tarjan, 1971) that allows for the verification of the 2-edge connectivity of a physical topology.

The underlying concept behind this search is to firstly "depth" explore unvisited nodes, called "depth search", via a walk through incident edges from the current node (v). At any depth search, the current visited node is pushed to a stack. The depth search continues until there are no unvisited nodes reachable from the current node. The search is "backtracking" by examining visited nodes in the stack and continuing depth search at these nodes. The algorithm is terminated when there are no nodes in the stack.

Algorithm 1 Depth-First Search for 2-connected graph

Require: A graph $G(\mathbf{V}, \mathbf{S})$ presented the physical topology of a given network.
Ensure: Return **true** if G is two connected, otherwise **false**

$count \leftarrow 1$;
Set current node $v \leftarrow v_0$; //start at node v_0
Set $dfs[v] = btk[v] \leftarrow count++$;
repeat
  if (there is an unvisited incident edge e of v) **then**
    if ($\exists w \leftarrow v$ is unvisited) **then**
      $dfs[v] = btk[v] \leftarrow count++$;
      Push $stack \leftarrow v$;
      Current node $v \leftarrow w$;
    else
      $btk[w] = min(btk[w], btk[v])$
    end if
  else
    Pop $w \leftarrow stack$;//parent of node
    if ($btk[v] \geq dfs[w]$) **then**
      Record w is an articulation node
    else
      $btk[w] = \min btk[w], btk[v]$;
    end if
  end if
  Current node $v \leftarrow w$;
until (there is no node in the stack)
if (there is no edge bridge between any two articulation nodes) **then**
  return true;
else
  return false;
end if

Two parameters are assigned to each visited node v, the $dfs[v]$ and the $btk[v]$, to discover articulation nodes. The $dfs[v]$ identifies the "depth first search" order of the node v when it is visited for the first time. The $btk[v]$ is determined using "back tracking" as follows:

- Initially, $btk[v]$ is assigned a value equivalent to the value of $dfs[v]$ when the node is first visited.
- $btk[v]$ is then updated as $btk[v] = \min(btk[v], dfs[w])$ when it has a neighbour w through an unvisited edge, and w is an ancestor of v.
- The value of $btk[w]$, where w is the parent of v, is also updated during the backtracking step as $btk[w] = \min(btk[w], btk[v])$.

The calculation of the value btk helps determine if a node v is a cut node (or articulation node) when $btk[v]$ is larger than or equal to $dfs[w]$, where w is the parent of v.

4.2.2 Finding biconnected components using property of 2-connected graphs

Alternatively, based on the properties of 2-connected graphs, biconnected components can be built from a simple and small cycle by adding the so-called H-paths to the cycle. Let H be a biconnected component on the graph G. A H-path is a non-trivial path on G that meets H exactly at its end nodes. The following proposition and proof are adopted from (Diestel, 2000).

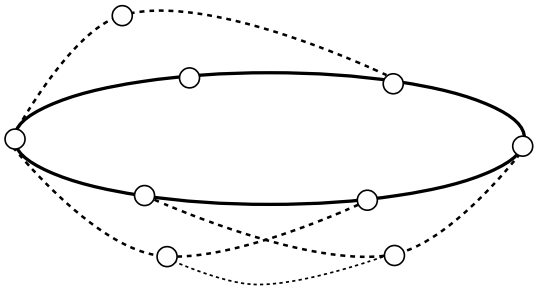


Fig. 3. The construction of 2-connected graphs

Proposition 4.1. *A graph is 2-connected if and only if it can be constructed from a cycle by successively adding H-paths to graph H already constructed.*

Proof. Clearly, every graph constructed as proposed is 2-connected. Conversely, let G be a 2-connected graph, then G contains a cycle, and hence a maximal subgraph H is constructable, as evident in Fig. 3. Any edge $x, y \in E(G) \setminus E(H)$ with $(x, y) \in H$ defines a H-path. Then, H is an induced sub-graph of G. If $H \neq G$, then by the connectedness of G, there is an edge vw with $v \in G - H$ and $w \in H$. As G is 2-connected, $G - w$ has a $v - H$ path P. Then wvP is a H-path in G, and $H \cup wvP$ is a constructable sub-graph of G. □

Based on this proposition, we can use the relationship between network's cycles or 2-connected graphs to verify the survivability of its physical topology.

An undirected graph is thus seen as the combination of all fundamental cycles. Using Algorithm 2, these fundamental cycles can be found from a spanning tree $T(V, E')$, $E' \subset E$ of a graph $G(V, E)$ (eg. the spanning tree highlighted by thick lines in Fig. 4).

Algorithm 2 Finding cycles

Require: A tree T and and edge e whose end-nodes is in T.
Ensure: A cycle P formed by T and e.

```
1: (s, d) ← end-nodes of e;
2: queue ← [node.s, node.P]; check ← 0;
3: while (check = 0 & queue ≠ ∅) do
4:    [v] ← head(queue);
5:    queue ← queue − head(queue);
6:    if (v.s = d) then
7:       check = 1; P ← v.P;
8:    else
9:       for (all v_k is neighbour of v_s) do
10:         node.s ← v_k; node.P ← P ∪ v_k;
11:         push node into queue;
12:      end for
13:   end if
14: end while
```

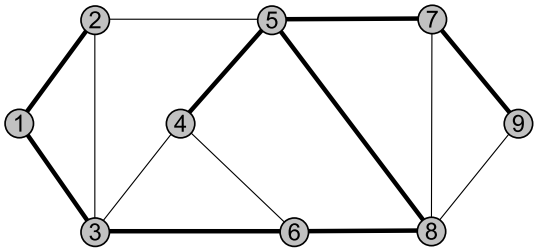


Fig. 4. Spanning tree on an arbitrary graph

However, it is not easy to find all of the fundamental cycles in the graph. For instance, in Fig. 4, the edges represented with thin lines are not part of the spanning tree (shown by thick lines). If any of these edges are added to the tree, it will form a unique cycle, but such cycle is not necessarily a fundamental cycle (eg. consider adding edge $2-5$). Any set of cycles found from the spanning tree can be used to verify the survivability of the topology from which it is generated. An algorithm for finding a set of cycles through spanning tree of a graph is represented in Algorithm 2. An efficient method for finding fundamental cycles of a graph, referred to as *Paton's* algorithm, is outlined in (Paton, 1969). Further discussion of this topic is outside the scope of this chapter.

The connectivity of the graph then can be determined using Algorithm 3. Note that the **set of remaining links** contains links in the graph but not in the tree.

5. Graph algorithms for diverse routing in protection design

Previously in this chapter, we have discussed the connectivity of the physical topology to support the problem of designing multiple quality of protections. However, the assurance of connectivity at the physical layer is only the first step to ensure traffic demands can be conveyed from the source to the destination. Once a physical layer is given, the main goal of the Logical Topology Design (LTD) is to determine routes for traffic flow in the most efficient

Algorithm 3 Construct biconnected components

Require: A spanning tree T of the graph G and the set of remain links.
Ensure: Set of biconnected components.
1: Firstly, a biconnected subgraph $G_i, i = 1$ is found by picking up a link e in the set of remain links and joining it with a unique path in the tree between end nodes of e.
2: a new biconnected subgraph $G_j, j = i + 1$ is constructed from a remain link and a unique path between end nodes of that link. If G_j have more than one node in common with a component $G_k, k = 1 \ldots i$, join $G_k \leftarrow G_k \cup G_j$. Otherwise, a new biconnected subgraph G_j is recorded, $i \leftarrow i + 1$.
3: Repeat step 2 until there is no remain links left.
4: The set of constructed biconnected components $G_1, G_2, \ldots, G_i$ is return.

way, that is to minimise the amount of physical resources utilised, particularly the amount of capacity utilisation. One approach is to enumerate the set of best candidates for each demand. In network protection design, these candidates can be distinct paths or disjoint paths. The disjoint paths are to ensure the success of the recovery process from network failures. On the other hand, the distinct paths provide the flexibility in selecting the most efficient working and backup routes. This section presents graph algorithms to support the diverse routing problem including finding K shortest (least cost) paths, finding two disjoint paths and finding K shortest disjoint path-pairs between any two nodes in a network.

5.1 Algorithm for finding K shortest paths between two nodes in the network

A set of all paths between two nodes can be used to select the best paths on which traffic flows between these nodes may be carried. All paths between two nodes in the graph can be found using either the Breath-First Search or the Depth-First Search. However, the number of all paths increases exponentially with the size of the network. Hence, finding all paths for routing is not a practical approach for real-time provisioning. In addition, the number of best selected paths is very small compared to all possible paths. Therefore, finding a small set of best eligible paths would be more practical and computationally efficient.

This section discusses and analyses the complexity of algorithms presented in the literature. We outline an efficient graph algorithm for finding K shortest paths between any two nodes in the network. This algorithm is an adaptation from (Martins et al., 1998) for directed graph to undirected graph.

Model selection used in survivable network design employs a set of potential candidates as inputs and selects optimal candidates as outputs. Input candidates differ depending on the protection techniques used. This can be either a set of paths for routing working flows, backup routes (in non-joint optimisation approaches), a set of disjoint path-pairs or a set of cycles in p-cycle design. This section introduces a basic algorithm used as a subroutine for finding input candidates for routing working flows and backup routes in non-joint optimisation approaches; the K shortest (minimum) paths algorithm.

K shortest paths is a classical graph problem that has been widely studied (Eppstein, 1998; Martins et al., 1998; Martins & Santos, 2000). Eppstein *et al.* (Eppstein, 1998) proposed a k shortest path algorithm between any two nodes in a digraph in time complexity of $O(|E| + |V| \log |E| + K)$, where $|V|$ is number of nodes, $|E|$ is the number of edges and K is number of paths required. Martins *et al.* (Martins et al., 1998) present two algorithms for

Algorithm 4 K shortest paths algorithm

Require: An undirected graph $G(V, E)$, a pair of source and destination nodes and the number of shortest paths required.

Ensure: A set of K - shortest paths over graph G from s to d.

1: To assure the possible repetition of the algorithm in a path between a pair of source-destination nodes (s, d), the given network is enlarged with a super source node S and super destination D, with zero cost edges $(s*, s)$ and $(d, d*)$. Following this, the shortest tree from source node s to other nodes in the network is obtained, and the first shortest path is marked as $p_1 = \{s_0(= s), s_1, \ldots, s_{r-1}, s_r(= d)\}$ from s to d.

2: Determine the first node s_h in p_1 such that s_h has more than a single incoming edges. If a node s'_h , of which the incoming edges are the incoming edges of s_h except those coming from s_{h-1}, does not exist, then generate the node s'_h, else determine the next node s_i in p_1 that has not alternate yet. The cost $d(s, s'_h)$ of shortest path from s to s'_h is calculated as:

$$d(s, s'_h) = \min_x \left(d(s, x) + d(x, s'_h)\right)$$

where (x, s'_h) are incoming edges of s'_h.

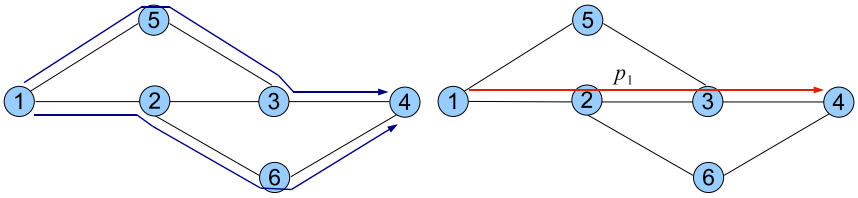
3: For each $s_j = \{s_i, \ldots, s_{r-1}\}$, generate s'_j following the same rules as s'_h, but with one more incoming edge of (s'_{j-1}, s'_j). Clearly, the shortest path from s to s'_j is the second shortest path from s to s_j. Therefore $p_2 = \{s_0, \ldots, s_i, \ldots, s'_{r-1}, s_r(= d)\}$ is the second shortest path. Repeat step 2 to determine next shortest path $p_k(k = 2, 3, \ldots)$ until $k = K$.

the K shortest paths problem, one based on a label setting algorithm and another based on a label correcting algorithm. The results show that these algorithms perform better than the algorithm in (Eppstein, 1998). The K-shortest path algorithm employed in this study is adapted from (Martins & Santos, 2000), which is proposed over a directed graph approach. Building on this idea, this study has developed an algorithm (Algorithm 4) which is relevant for an undirected approach as well. This algorithm runs with the time complexity of $O(K \times |E|)$, where K is the number of shortest paths required and $|E|$ is the number of undirected edges in the graph.

5.2 Algorithm for finding two disjoint paths (disjoint path-pair) between nodes in the network

Finding a disjoint path-pair between two nodes in the network is a basic solution for protection design in which the primary and secondary paths must not suffer from the same failure. Basically, finding a pair of disjoint paths can be done in two steps. In the first step, the first path is determined from the original graph. Then, all edges contained in the first path are removed from of the graph. The second disjoint path is determined from the residual graph. Both paths are usually determined using any shortest path algorithm such as Dijkstra's algorithm (Dijkstra, 1959) or Bellman-Ford algorithm (Bellman, 1958). The two step approach is simple in both concept and implementation. There is, however, no guarantee that the total cost of the found disjoint path-pair is minimum. In addition, this approach may fail to find a solution in some cases even when a disjoint path-pair exists between the two nodes. An example of a trap topology is shown in Fig. 5. It is easy to see in Fig. 5(a) that there are two disjoint paths between nodes 1 and 4. However, by using the two-step approach, the first shortest path may be path p_1 ($1 \rightarrow 2 \rightarrow 3 \rightarrow 4$) as shown in Fig. 5(b). The second

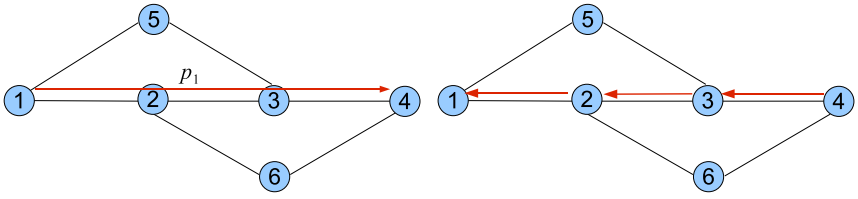
path is determined after all spans contained in the first path (edges $\{(1-2),(2-3),(3-4)\}$) are removed. It can easily be seen that the second path can not be found since the graph is disconnected between nodes 1 and 4.



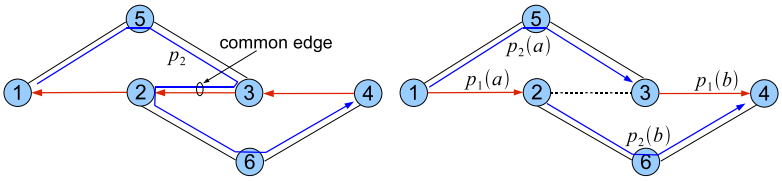
(a) A pair of edge-disjoint paths is created (b) Find the first shortest path p_1

Fig. 5. An example of trap topology

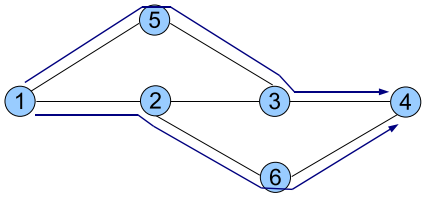
To resolve these two drawbacks, a graph algorithm, known as the one step approach, for determining a shortest disjoint path-pair was first proposed by Surballe (Surballe, 1974) and modified by Bhandari (Bhandari, 1994; 1999) to adapt to the negative weight of edges. The algorithm, as its name implies, determines the first and the second disjoint path simultaneously.



(a) Find the first shortest path p_1 (b) Create negative reverse-directed edges from p_1



(c) Find the second shortest path p_2 (d) Remove the common edge $(2-3)$



(e) A pair of edge-disjoint paths is created

Fig. 6. An illustration of the one-step approach to resolve trap topology

Let us assume that a shortest disjoint path-pair needs to be determined between the source node s and the destination node d. The one step algorithm is outlined as follows:

- Find a shortest (least cost) path between the source node s and the destination nodes d, and denote it as p_1 (Fig. 6(a)).
- Mark the direction of each edge traversed in p_1 from s toward d as positive.
- Remove all directed edges on the shortest path p_1 and replace them with reverse direction edges by multiplying -1 to the original edge cost (Fig. 6(b)).
- Find the least cost path from s to d in the modified graph using the modified Dijkstra's algorithm, and denote it as path p_2 (Fig. 6(c)).
- Remove any edge in the original graph traversed by both p_1 and p_2 (Fig. 6(d)). These are called interlacing edges.
- Identify all path segments remaining after the process of the edge removal (Fig. 6(e)) which forms a pair of disjoint paths from the source node s to the destination node d.

5.3 Algorithm for finding K shortest disjoint path-pairs between nodes in the network

In path protection techniques such as path restoration, or shared backup path protection, there are two sets of candidates needed for provisioning of the working flows and the backup routes. Finding and employing these two sets independently may compromise the protection conditions as the backup and working paths must be disjoint. There is no guarantee that each working route has a disjoint path in the corresponding set of backup routes. In addition, using two different sets of candidates also increases the number of decision variables used in model selection which increases the complexity of both space requirement and computational time. Another approach is to determine one set of candidates in which each candidate is a disjoint path-pair serving as a primary and a backup route. This set would always satisfy the disjointness requirements for protection and reduce the complexity in both time and space of model selections such as Integer Linear Programming (ILP) and Mixed ILP (MILP). By combining algorithms in Sections 5.1 and 5.2, this section introduces an algorithm for finding K shortest (least cost) disjoint path-pairs between two nodes in the networks.

Algorithm 5 K disjoint-paths pairs (KDPPs)

Require: An undirected graph $G(V, S)$ source node s, destination node d, and K, the number of shortest disjoint-paths pairs required.

Ensure: A set of K shortest disjoint-paths pairs.

1: Find a shortest path between s and d, denoted by p.
2: Define the direction of each edge traversed in p from s to d as positive 1.
3: Remove all directed edges on the shortest path p and replace them with reverse direction edges by multiplying the original edge cost with -1.
4: Find K shortest path (least cost) paths from s to d in the modified graph using the K shortest path algorithm (Algorithm 4). Denote them as the set of paths $P = \{p_1, p_2, \ldots, p_K\}$.
5: For each pair of paths $(p, p_i), i = 1 \ldots K$, remove any edge of the original graph traversed by both p and p_i. These are called interlacing edges. Identify all path segments from the rest of edges. These from disjoint-paths pairs, denote as $\{(w_1, r_1), (w_2, r_2), \ldots, (w_K, r_K)\}$.

The two-step and one-step approaches can be used to find the set of disjoint path-pairs. They are simple and computationally efficient. However, since these approaches provision traffic streams sequentially without back-tracking, sometimes no solution may be found even when a feasible solution does exist. In fact, the graph theory approach is more applicable

in online provisioning than in offline provisioning. These approaches have been extensively studied in the literature (Patre et al., 2002; Sen et al., 2001; Xin et al., 2002; Zhang et al., 2003). Authors in (Xin et al., 2002) investigated survivable routing based on both the two-step and the one-step approaches, namely Separate Path Selection (SPS) and Joint Path Selection (JPS) respectively. These approaches aim to optimise the network resource utilisation of each traffic connection by minimising the total cost of the primary and backup paths. The performance of these approaches is evaluated for different protection path cost functions. The results have shown that JPS performs substantially better than SPS and also scales very well in terms of the network resource abundance. In addition, JPS can utilise network resources better than SPS. In (Zang et al., 2003), the two-step and one-step approaches are also investigated, but with a different objective function based on the blocking probability. The simulation results have shown that the one-step approach significantly outperforms the two-step approach in this case. The improvement in blocking probability is significant, around 10%-20%, especially when fixed routing is used.

In this section, the one-step approach is used as a sub-function in an algorithm for finding K shortest disjoint path-pairs between any two nodes in the graph. The pseudo code of this algorithm is shown in Algorithm 5. This algorithm is then proved to yield K distinct disjoint-path pairs. Since a path pair found from one step algorithm was proved to be disjoint in (Bhandari, 1999) , it is only necessary to prove that the K found path pairs do not coincide with each other.

Proof. Let $P = \{P_k | k = [1 \dots K]\}$ be the set of K pairs of disjoint paths yielded from Algorithm 5, where $P_k = \{w_k, r_k\}$ is the k^{th} pair.

Let $P_i = \{w_i, r_i\}$ and $P_j = \{w_j, r_j\}$ be any two disjoint-path pairs yielded from the first shortest path p and paths p_i and p_j $(i, j \in [1 \dots K])$ respectively. This study proves that P_i and P_j do not coincide.

Since p_i and p_j do not coincide, there exists at least one edge e_k contained in p_i but not contained in p_j. Following this, it is proved that e_k can only belong to either P_i or P_j. If e_k is contained in p then e_k is obviously not contained in P_i. However, since e_k is not contained in p_j, e_k is not an interlacing edge of (p, p_j) and hence e_k is in P_j. Conversely, if e_k is not contained in p, then e_k is in P_i. In addition, since e_k is not contained in both p and p_j, e_k is not contained in P_j. Therefore, P_i and P_j do not coincide and the K found disjoint-path pairs are distinct from each other. □

6. Conclusion

In this chapter we have investigated and discussed the connectivity of graphs in relation to the requirements of protection of traffic demands in practical networks. We have discussed and analysed the complexity of several methods for verifying the connectivity of the physical topology of a network. Diverse routing was then addressed as an important problem in designing network protection, followed by the introduction of three algorithms for finding distinct and disjoint paths. These algorithms address the challenges for network protection design using graph theory.

7. References

Bellman, R. (1958). On a routing problem, *in Quarterly of Applied Mathematics* 16(1): 87–90.

Bhandari, R. (1994). Optimal diverse routing in telecommunication fiber networks, *Proceedings of the 13th IEEE Networking for Global Communications.*, Vol. 3, Toronto, Ont. , Canada, pp. 1498 –1508.

Bhandari, R. (1999). *Survivable Networks: Algorithms for Diverse Routing*, Kluwer Academic Publishers.

Dijkstra, E. (1959). A note on two problems in connection with graphs, *Numerische Mathe-matik* 1: 269–271.

Diestel, R. (2000). *Graph Theory*, Springer-Verlag.

Eppstein, D. (1998). Finding the *k* shortest paths, *SIAM Journal on Computing* 28(2): 652–673.

Grover, W. D. (2004). *Mesh-based Survivable Networks: Options and Strategies for Optical, MPLS, SONET/SDH, and ATM networking*, Prentice Hall PTR.

Habibi, D., Nguyen, H., Phung, Q. & Lo, K. (2005). Establishing physical survivability of large networks using properties of two-connected graphs, *TENCON 2005 2005 IEEE Region 10*, IEEE, pp. 1–5.

Martins, E. D. Q. V., Pascoal, M. M. B. & Santos, J. L. E. D. (1998). The *k* shortest paths problem, *Technical report*, CISUC.

Martins, E. D. Q. V. & Santos, J. L. E. D. (2000). A new shortest paths ranking algorithm, *Investigação Operacional* 20(1): 47–62.

Menger, K. (1927). Zur allgemeinen kurventheorie, *Fund. Math* 10(95-115): 5.

Paton, K. (1969). An algorithm for finding a fundamental set of cycles of a graph, *Communications of the ACM* 12(9): 514–518.

Patre, S. D., Maier, G. & Martinelli, M. (2002). Design of static WDM mesh networks with dedicated path protection, *Infocom*.

Sen, A., Shen, B., Bandyopadhyay, S. & Capone, J. (2001). Survivability of lightwave networks - path lengths in WDM protection scheme, *Journal of High Speed Networks* 10(4): 303–315.

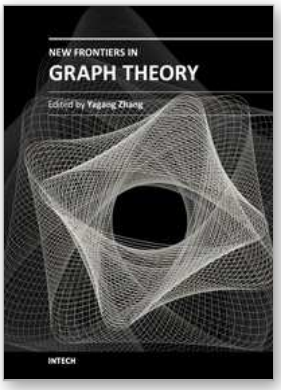
Surballe, J. (1974). Disjoint paths in a network, *Networks* 4: 125–145.

Tarjan, R. (1971). Depth-first search and linear graph algorithms, *Proc. th Annual Symposium on Switching and Automata Theory*, pp. 114–121.

Xin, C., Ye, Y., Dixit, S. S. & Qiao, C. (2002). A joint lightpath routing approach in surviavble optical networks, *Optical Networks Magazine* 3(3): 13–20.

Zang, H., Ou, C. & Mukherjee, B. (2003). Path-protection routing and wavelength assignment (RWA) in WDM mesh networks under duct-layer constraints., *IEEE/ACM Transactions on Networking* 11(2): 248–258.

Zhang, J., Zhu, K., Sahasrabuddhe, L. & Yoo, S. B. (2003). On the study of routing and wavlength assignment approaches for survivable wavelength-routed WDM mesh networks, *Optical Networks Magazine* 4(6): 16–28.



New Frontiers in Graph Theory

Edited by Dr. Yagang Zhang

ISBN 978-953-51-0115-4
Hard cover, 526 pages
Publisher InTech
Published online 02, March, 2012
Published in print edition March, 2012

Nowadays, graph theory is an important analysis tool in mathematics and computer science. Because of the inherent simplicity of graph theory, it can be used to model many different physical and abstract systems such as transportation and communication networks, models for business administration, political science, and psychology and so on. The purpose of this book is not only to present the latest state and development tendencies of graph theory, but to bring the reader far enough along the way to enable him to embark on the research problems of his own. Taking into account the large amount of knowledge about graph theory and practice presented in the book, it has two major parts: theoretical researches and applications. The book is also intended for both graduate and postgraduate students in fields such as mathematics, computer science, system sciences, biology, engineering, cybernetics, and social sciences, and as a reference for software professionals and practitioners.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821