Computer Communications and Networks

Jacek Rak
David Hutchison *Editors*

Guide to Disaster-Resilient Communication Networks

Springer

Fundamental Schemes to Determine Disjoint Paths for Multiple Failure Scenarios[*]

Teresa Gomes[†‡] Luísa Jorge[§‡] Rita Girão-Silva[†‡] Jose Yallouz[¶]
Péter Babarczi[||] Jacek Rak[#]

teresa@deec.uc.pt ljorge@inescc.pt rita@deec.uc.pt jose@joseyallouz.com
babarczi@tmit.bme.hu jrak@pg.edu.pl

January 2020

[*]This chapter is based on work from COST Action CA15127 ("Resilient communication services protecting end-user applications from disaster-based failures – RECODIS") supported by COST (European Cooperation in Science and Technology). This work is funded by ERDF Funds through the Centre's Regional Operational Program and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia, I.P. under the project CENTRO-01-0145-FEDER-029312. This work was also partially supported by FCT under project UIDB/00308/2020.

[†]University of Coimbra, Department of Electrical and Computer Engineering, Rua Sílvio Lima, 3030-290 Coimbra, Portugal

[‡]INESC Coimbra, Rua Sílvio Lima, 3030-290 Coimbra, Portugal

[§]Instituto Politécnico de Bragança & CeDRI, Campus de Santa Apolónia, 5300-253 Bragança, Portugal

[¶]Technion, Israel Institute of Technology, Department of Electrical Engineering, Haifa, Israel

[||]Budapest University of Technology and Economics, MTA-BME Future Internet Research Group, H-1111 Muegyetem rakpart 3, Budapest, Hungary

[#]Gdansk University of Technology, Faculty of Electronics, Telecommunications and Informatics, G. Narutowicza 11/12, PL-80-233 Gdansk, Poland

Abstract

Disjoint path routing approaches can be used to cope with multiple failure scenarios. This can be achieved using a set of k ($k > 2$) link- (or node-) disjoint path pairs (in single-cost and multi-cost networks). Alternatively, if Shared Risk Link Groups (SRLGs) information is available, the calculation of an SRLG-disjoint path pair (or of a set of such paths) can protect a connection against the joint failure of the set of links in any single SRLG. Paths traversing disaster-prone regions should be disjoint, but in safe regions it may be acceptable for the paths to share links or even nodes for a quicker recovery. Auxiliary algorithms for obtaining the shortest path from a source to a destination are also presented in detail, followed by the illustrated description of Bhandari's and Suurballe's algorithms for obtaining a pair of paths of minimal total additive cost. These algorithms are instrumental for some of the presented schemes to determine disjoint paths for multiple failures scenarios.

1 Introduction

The capability of a network to deliver services in the presence of failures of network elements, known as *survivability*, is undoubtedly one of the critical issues in the design of resilient communication systems. Concerning the multi-hop routing, survivability is typically assured by additional communication paths called *backup* (or *alternate*) paths used to restore the network traffic affected by a failure of network elements traversed by a *primary* (*working*) path [1, 2]. As failures in communication networks can refer to either network links or nodes, backup paths should be link- (or node-) disjoint with the respective working paths, i.e., have no common links (transit nodes) with the associated working paths, accordingly (see Fig. 1).

An important observation is that nodal-disjointness includes link-disjointness. Therefore, a scheme protecting against a failure of a network node automatically provides protection against a link failure, but the reverse relation is not true.

In some cases, establishing a pair of end-to-end disjoint paths may be not possible. It can happen for instance due to topological constraints of the network graph, or because of the network segmentation issues (i.e., when different parts of the network are owned by different operators not willing to share the intra-network information among networks). In such cases, establishing a set of communication paths traversing joint links/nodes can be the only solution. Similarly, in the case of large problem instances (e.g., for large networks and distant end-nodes of a demand), to reduce the computational time, it may

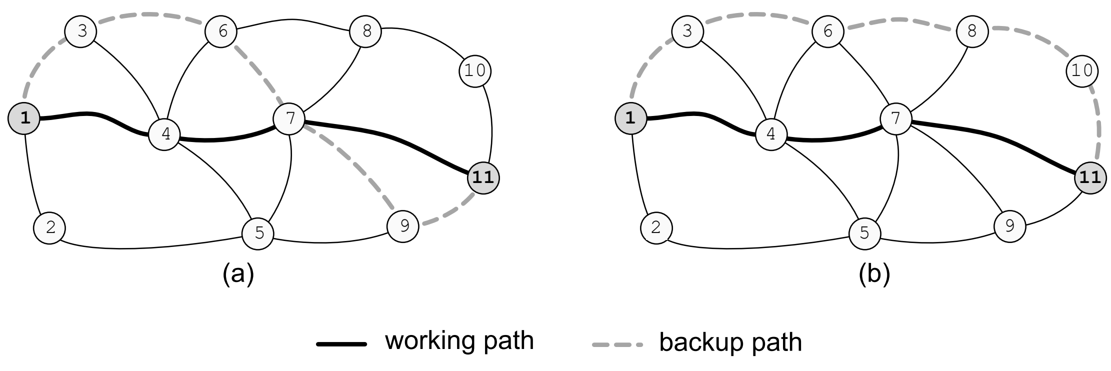


Figure 1: Examples of (a) link- and (b) nodal-disjointness of working and backup paths for a demand between nodes `1` and `11` to assure protection against a failure of a network link/node

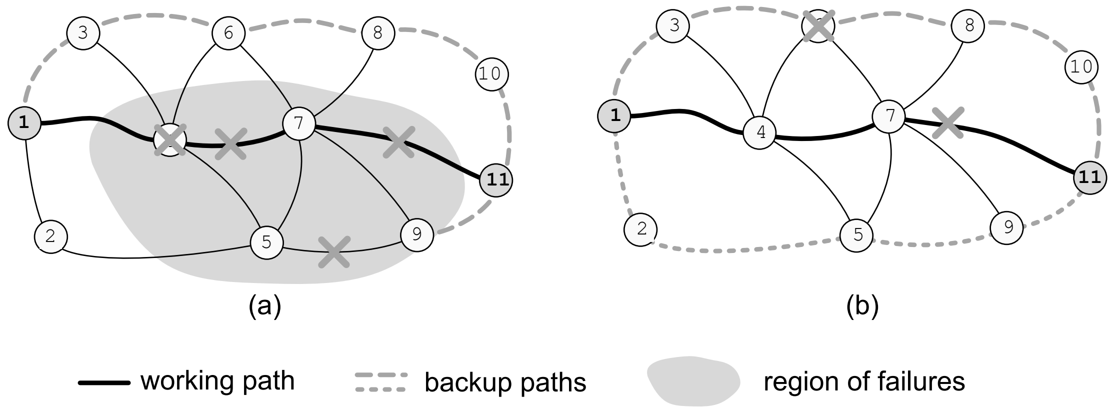


Figure 2: Example survivability schemes for a demand (`1`, `11`) for multiple failures of network elements (a) occurring in a given region, and (b) without region-dependent correlation

be reasonable to introduce segmentation of the network graph and calculate a set of disjoint paths for each such segment.

For a given demand, a pair of link-(node-) disjoint paths is sufficient to provide protection against a failure of a single link (node), which, based on statistics, covers over 70% of all failure scenarios [3]. Also, based on meteorological observations related to the frequency of occurrence of natural disasters, as well as statistical results related to malicious human activities, the risk of occurrence of simultaneous failures of multiple network elements is raising [4]. In the former case, a backup path should be *region-disjoint / geographically-diverse* with the associated working path, while in the latter case of non-correlated multiple (k-1) failures, a scheme of k disjoint paths seems necessary, as shown in Fig. 2.

Other causes of multiple failures include physical co-location of multiple links in the same physical concrete conduit defining a *Shared Risk Link Group* (SRLG), i.e., a group of links subject to a simultaneous failure after a conduit cut.

If a cost of a network link is the same for each of k paths of a demand, this scenario is referred to as a *single-cost network* case. Otherwise, if differentiated costs of a link are applied when finding each of k paths, the scheme is known to be the *multi-cost network* case [1], as, e.g., under backup path sharing, where the cost of a backup path link is only a part of its cost from working path computations [5].

A common objective function of the optimization problem related to the calculation of disjoint paths is the minimization of the total cost of communication paths of a given demand (assuming a given metric of link costs), or for the entire set of demands. In the first case, an optimal solution for a given demand can be obtained in a fast way (i.e., in polynomial time) for some problem instances (e.g., for single-cost networks). However, for minimization of the total path cost for all demands jointly, no time-efficient solution has been proposed so far, and the problem is, therefore, still considered as a non-easy one requiring sub-optimal heuristic schemes.

In our opinion, all issues highlighted above form the set of fundamental concepts necessary to understand before deploying any routing scheme resistant to multiple failures. Therefore, the objective of this chapter is to discuss in detail all mentioned aspects as well as to provide description of characteristics of major schemes of path calculations available in the literature designed to enable restoration of the the affected traffic after simultaneous failures of multiple network elements.

The remaining part of this chapter is structured as follows. In Sect. 2, the basic optimization problems related to resilient routing are defined. Section 3 highlights the concept of SRLG and its role in determination of a set of paths resistant to multiple failures. Section 4 presents and illustrates two algorithms for shortest path determination (Dijkstra's and Modified Dijkstra's algorithms). Further sections provide description of representative schemes to establish a set of disjoint paths, including Suurballe's and Bhandari's algorithms (Sect. 5) for single-cost networks, k-Penalty scheme for multi-cost networks (Sect. 6), and schemes to obtain the partially disjoint paths (Sect. 7). Finally, Sect. 8 concludes the chapter.

2 Algorithms for Disjoint Routing

Several disjoint routing problems can be formulated as optimization problems. Some of them can easily be solved by effective algorithms, which are presented in the next subsections.

Consider the network to be represented by a *directed* graph $G(V, A)$, where $V = \{v_1, v_2, \ldots v_n\}$ is the set of nodes and $A = \{a_1, a_2, \ldots, a_m\}$ is the set of directed arcs, where n and m are the size of sets V and A, respectively. Each element a of A is an ordered pair of elements of V, hence $a = (v_i, v_j)$, and arc a may also be represented as a_{ij}. In a single cost network, each arc is assigned a cost represented by c_a or c_{ij}, usually positive. Often the cost represents an additive metric, e.g., length, cost or delay. In those cases the cost of a path is the sum of the cost of the arcs in the path.

When bandwidth is the relevant cost, then the metric is no longer additive, because a path bandwidth is determined by its arc with the smallest bandwidth. However, the algorithm in Sect. 4.1, with a suitable modification [6], can be used to calculate the widest path from a source to a destination node.

The most common disjoint path calculations are the *min-sum*, *min-min* and *min-max* problems. In the *min-sum* problem, one seeks a pair of paths such that the sum of their cost is minimal. The *min-min* problem seeks the minimum cost path for which a disjoint path can be found. Finally the *min-max* problem seeks to minimize the cost of the most expensive path of the pair. Only the *min-sum* can be solved in polynomial time. However the maximization of the sum of the bandwidths of the paths is $\mathcal{NP}$-complete [7].

Variants of these three problems can be obtained if additional constraints are considered, namely if the paths must be SRLG-disjoint [8], or geodiverse (to make them less prone to geo-correlated failures) [9–11]. For security reasons, it may also be necessary that the paths visit some specified nodes [12, 13]. Algorithms for calculating the most available path pairs have also been proposed [14]. A description of an algorithm for calculating the most available path pair with geodiversity constraints can be found in [15] and in Chapter 3.2.

To illustrate how to formalize these problems, we use the *min-sum* problem. Let P_A designate the feasible design space of path pairs (p_1, p_2) from a source node s to a destination node d $(s, d \in V)$ such that p_1 and p_2 are arc-disjoint paths. The solution to the *min-sum* problem for the arc-disjoint case is (p_1^*, p_2^*) given by:

$$(p_1^*, p_2^*) = \arg \min_{(p_1,p_2) \in P_A} c(p_1) + c(p_2) \tag{1}$$

with $c(p_k) = \sum_{a \in p_k} c_a$ with $k = 1, 2$.

For presenting an Integer Linear Problem (ILP) formulation some additional notation is necessary:

$$x_{ij}^k = \begin{cases} 1 & \text{if path } \ p_k \text{ traverses an arc } (v_i, v_j) \in A \\ 0 & \text{otherwise} \end{cases} \qquad k = 1, 2 \tag{2}$$

The formulation is as follows, with $c_{ij} > 0$:

$$\min \sum_{(v_i, v_j) \in A} c_{ij} (x_{ij}^1 + x_{ij}^2) \tag{3a}$$

subject to:

$$\sum_{j:(v_i,v_j)\in A} x_{ij}^k - \sum_{j:(v_j,v_i)\in A} x_{ji}^k = \begin{cases} 1 & \text{if } v_i = s \\ -1 & \text{if } v_i = d \\ 0 & \text{otherwise} \end{cases}, \quad \forall v_i \in V, \quad k \in \{1, 2\} \tag{3b}$$

$$x_{ij}^1 + x_{ij}^2 \leq 1, \quad \forall (v_i, v_j) \in A \tag{3c}$$

$$x_{ij}^k \in \{1, 0\}, \quad \forall (v_i, v_j) \in A, \ k \in \{1, 2\} \tag{3d}$$

where Eq. (3b) are the flow conservation constraints and Eq. (3c) ensures the paths are arc-disjoint. Because it is assumed $c_{ij} > 0, \forall (v_i, v_j) \in A$, the minimization of the total cost, ensures no cycles will be present in p_k, with $k = 1, 2$. If the paths were to be node-disjoint, then constraints in Eq. (3c) would have to be written as $\sum_{j:(v_i,v_j)\in A} (x_{ij}^1 + x_{ij}^2) \leq 1, \forall v_i \in V : v_i \neq s \wedge v_i \neq d$, which ensures arcs from p_1 and p_2 do not share an intermediate (head) node. In this case an additional constraint ($x_{sd}^1 + x_{sd}^2 \leq 1$) is also needed to avoid using link (s, d) in both paths.

3 SRLG-disjoint Routing

A *Shared Risk Link Group* (SRLG) is a group of links subject to simultaneous failure due to a common risk of failure, such as co-location of multiple links in the same physical concrete conduit. In Fig. 3(a) the optical links (lightpaths) are represented by e_i, $i = 1, 2, \ldots, 5$, and in Fig. 3(b) the conduits that contain them are represented by g_η, $\eta = a, b, c, d, e, f, i$. A failure at a lower network layer can appear as a set of multiple failures at an upper layer (e.g., the cut of conduit g_a at the physical layer in Fig. 3(b) will result in the failure of

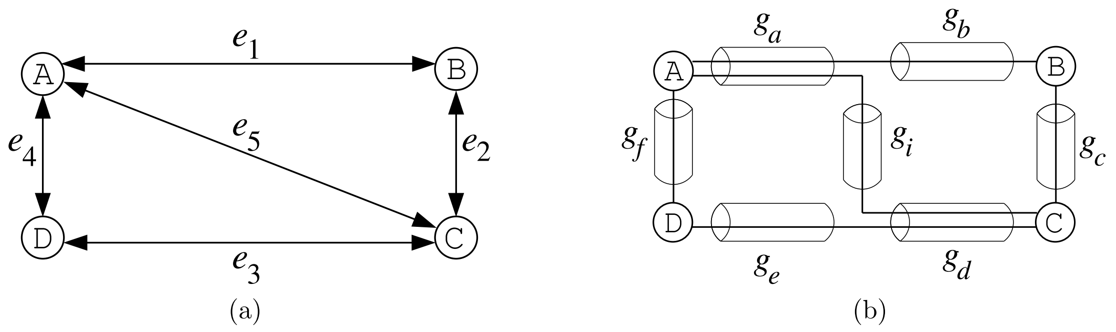


Figure 3: Layered architecture of an optical network – adapted from [16]; (a) Lightpath Layer, (b) Physical Layer

two optical links e_1 and e_5 at the optical lightpath layer in Fig. 3(a)). Hence SRLGs can be used to represent multiple failures due to the multi-layered nature of communication networks. Moreover, SRLGs can be used to represent geographically correlated failures, as shown in [17] and in Chapter 1.4.

Let R be a set of risks that may cause faults in the network. Let A_r represent the subset of network optical lightpaths (corresponding to a graph where arcs have an assigned capacity) that can be affected by risk $r \in R$. Hence A_r is an SRLG associated with r. For example, assume each edge in Fig. 3(a) is represented by a pair of arcs (in opposite directions) linking its end nodes. Then A_1 (associated with risk $r = 1$ of an event cutting conduit g_a and causing the failure of optical links e_1 and e_5, as previously mentioned) is given by $\{$`(A,B)`,`(B,A)`,`(A,C)`,`(C,A)`$\}$ for the directed representation of the network (where e_1 is given by $\{$`(A,B)`,`(B,A)`$\}$ and e_5 is given by $\{$`(A,C)`,`(C,A)`$\}$).

Let

$$R_p = \{r \in R : \text{ path } p \text{ contains elements of } A_r\} \tag{4}$$

Based on [8] the SRLG problem can be defined as follows.

Definition 1 *Find two paths p_1 and p_2 between a pair of nodes s and d, such that $R_{p_1} \cap R_{p_2} = \emptyset$. We also say that p_1 and p_2 are two SRLG-diverse paths (with respect to R).*

To formalize the *min-sum* version of the SRLG-disjoint routing problem, the following notation is used:

– h_{rij} with $r \in R$ and $(v_i, v_j) \in A_r$ indicates if SRLG A_r contains arc (v_i, v_j).

$$h_{rij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in A_r \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

– z_r^k is the decision variable of the SRLG associated with risk r affecting path p_k $(k = 1, 2)$

$$z_r^k = \begin{cases} 1, & \text{if } p_k \text{ contains an arc of } A_r \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

The *min-sum* version of the SRLG-disjoint routing problem is then given by (with $c_{ij} > 0$) [8]:

$$\min \sum_{(v_i,v_j)\in A} c_{ij}(x_{ij}^1 + x_{ij}^2) \tag{7a}$$

subject to:

$$(3b) - (3d) \tag{7b}$$

$$\sum_{(v_i,v_j)\in A} h_{rij} x_{ij}^k \leq |A| z_r^k, \quad r \in R,\ k = 1, 2 \tag{7c}$$

$$z_r^1 + z_r^2 \leq 1, \quad r \in R \tag{7d}$$

$$h_{rij}, z_r^k \in \{0, 1\}, \quad \forall (v_i, v_j) \in A,\ k \in \{1, 2\}, \quad r \in R \tag{7e}$$

Equation (7c) ensures that if risk r affects p_k then some arc from A_r must be present in the path, otherwise no arc in A_r can be selected for path p_k. The coefficient $|A|$ is used because p_k may contain more than an arc affected by a given risk r. Equation (7d) ensures no element from R affects simultaneously both paths.

In [8] it is shown that finding an SRLG-disjoint path pair is $\mathcal{NP}$-complete, hence several heuristics have been proposed for solving this problem: in [16,18] for the *min-sum* version and in [19,20] for the *min-min* version. Note, however, that the optimal stopping condition proposed in [18] is not valid for general SRLGs, as shown in [21].

4 Shortest Path Algorithms

The objective of the original version of Dijkstra's algorithm proposed in 1959 in [22] is to find a single path between a pair of source s and destination d nodes in a network graph characterised by the lowest cost defined as a total cost of all traversed links. Other variants of the algorithm introduced later on focus, e.g., on finding the shortest-path tree from a specific source node to all the other nodes in the graph. Although this algorithm, and others presented in this section, consider directed graphs, these algorithms also apply to networks with undirected edges, if each of those edges are represented by a pair of arcs in opposite directions with the same cost (provided the cost is non-negative). Dijkstra's algorithm is currently widely used by routing protocols such as IS-IS [23], or OSPF [24].

Next, in Sect. 4.1, Dijkstra's algorithm is presented together with an illustrative example. This is followed, in Sect. 4.2, by a variant which can be used in a network with negative costs, but without negative cycles. A negative cycle is a path where the only repeated nodes are the source and destination node, such that its total cost is negative. This variant is necessary for presenting Bhandari's algorithm in Sect. 5.2.

4.1 Dijkstra's Algorithm

When finding a path between a given pair of source s and destination d nodes, execution of the original Dijkstra's algorithm is started at node s. All network nodes v_i $(i = 1, \dots, n)$ are initially marked as *unvisited* (i.e., with unknown optimal cost of paths from node s to these nodes v_i). The *labels* which store t_{si}, the cost of paths from node s to all the other nodes $v_i \in V$, are initially set to infinity. Of course the distance of s to itself (t_{ss}) is zero, and s will be marked as the first *current* node. Let $S = V$ be the initial set of all *unvisited* nodes. Associated with each node v_i is also τ_i, its predecessor node in the best known route from s to v_i; this predecessor is initially set to s for all nodes.

Each iteration of Dijkstra's algorithm preforms the following tasks:

1. set the status of *current* node to node v_i $(v_i \in S)$ characterized by the lowest current non-definitive label cost t_{si} of a path from s to this node v_i $(v_i = \arg\min_{v_h \in S} t_{sh})$;

2. mark the *current* node v_i as a *visited* node (i.e., remove it from set S because it has "*definitive*" minimal label);

3. update the values of the labels t_{sj} of paths from s to all v_j ($v_j \in S$) adjacent to the *current* node v_i ($(v_i, v_j) \in A$), according to formula (8) if $t_{si} + t_{ij} < t_{sj}$,

   $$t_{sj} = t_{si} + t_{ij} \tag{8}$$

   and set $\tau_j = v_i$, i.e., node v_i becomes the predecessor of v_j for the path with the updated label t_{sj}.

The algorithm is executed until node d is marked as visited or, if necessary, until all the nodes receive a definitive label – creating a full minimum-cost tree for the network from node s to all nodes in $V \setminus \{s\}$ (assuming all nodes can be reached from s). The cost of the optimal path from s to d is given by label t_{sd} and the path is obtained from the sequence of predecessors of d, starting with τ_d until reaching s.

The network in Fig. 4(a) is used to exemplify the Dijkstra's algorithm. Figures 4(b)-(g) show the subsequent steps to find the shortest path tree from node `A` to all the other nodes. Nodes with definitive labels are marked with an asterisk in the figures. The thicker lines signal arcs in the shortest path tree rooted at node `A`.

At the start, node `A` is labelled with 0 and all other nodes are labelled with an infinite value (not shown for simplicity). Hence the *current* node `A` label becomes definitive because is the smallest label value. The next step is to update the label for nodes connected to `A` (since `A` was the *current* node). The label of `B` changes to $t_{AB} = 2$ and the label of `D` changes to 3 (Fig. 4(b)). For simplicity, the possible successive updates of the predecessor of each node are omitted, but each predecessor in the shortest path is shown in the figures as the tail node of the (thicker) arcs depicting the current shortest path.

Next, `B` becomes the *current* node, and its label is marked as definitive because it is the smallest non-definitive label value. Once again it is necessary to check which non-definitive labels need upgrading. Connected to node `B` are nodes `C` and `D`. Node `C` gets a new non-definitive label, it is now 6. Node `D` does not change its label because it would become 6 which is greater than its present cost t_{AD} (Fig. 4(c)).

Node `D` is now the *current* node and gets a definitive label (because it has the smallest non-definitive value among nodes `C` and `D`). Next, both nodes `C` and `E` change their labels because they are connected to node `D`. Node `C` gets a new non-definitive label of value 5. Node `E` changes its label to 10.

Afterwards node `C` becomes the *current* node and to gets a definitive label. This in turn will cause node `E` to update its label to 9. Notice only node `E` could get a new label,

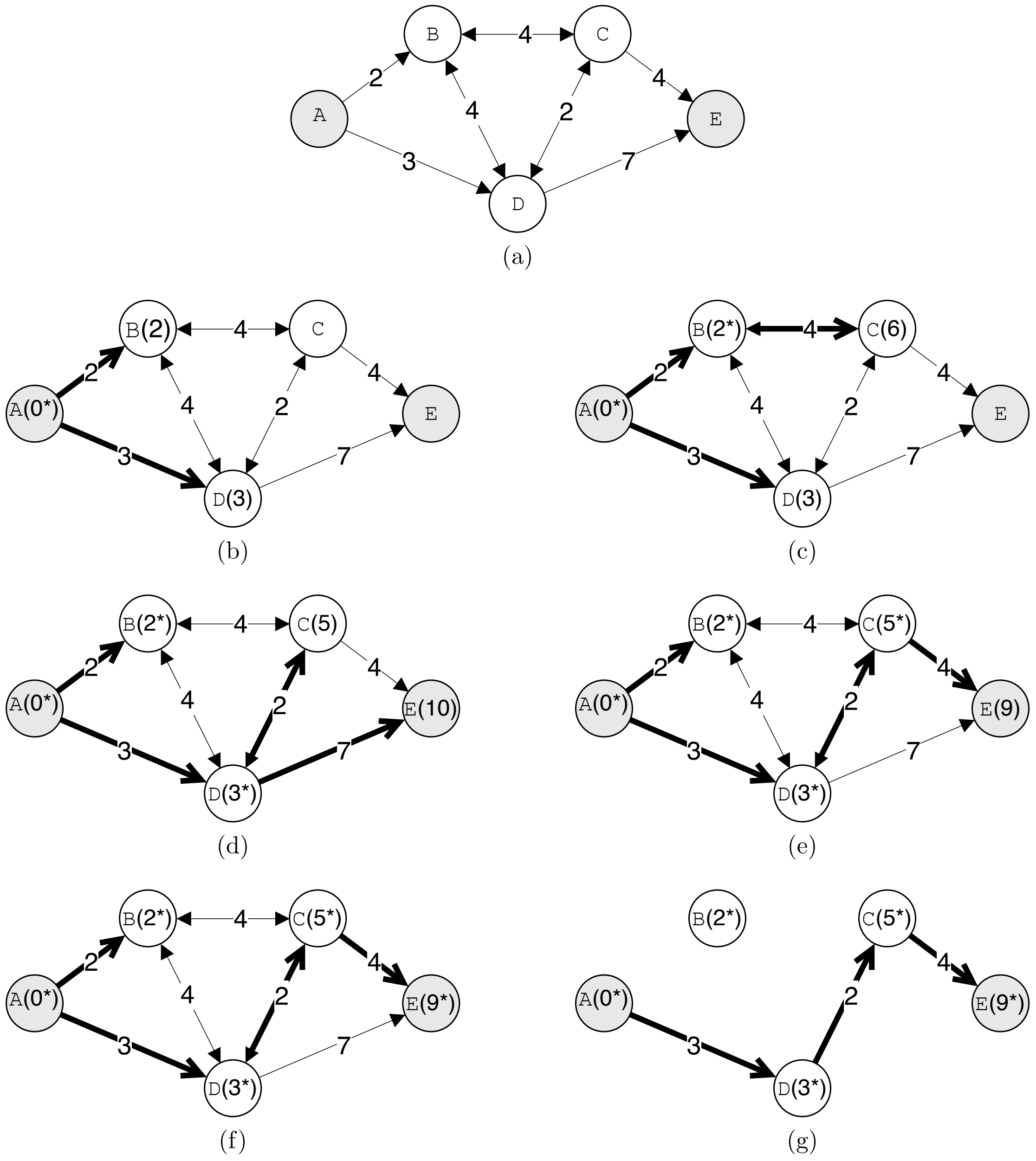


Figure 4: Dijkstra's algorithm example: (a) Network used; (b)-(g) Dijkstra's algorithm steps

because all other nodes already have a definitive label (Fig. 4(e)). Finally, node `E` gets a definitive label (Fig. 4(f)) and the execution of the algorithm ends.

The running time of the original Dijkstra's algorithm is bounded from above by $\mathcal{O}(|V|^2)$, where $|V|$ is the number of nodes in a graph, but can be reduced to $\mathcal{O}(|A| + |V| \log |V|)$, where $|A|$ is the number of arcs in the graph when using a Fibonacci heap in implementation of the algorithm [25].

Dijkstra's algorithm operating in polynomial time has been shown to return the optimal path, i.e., the minimal cost path. The proof of optimality can be based on the invariant hypothesis that for all nodes v_i with definite label, the costs of paths from s to v_i (t_{si}) are the minimal ones, i.e., they cannot be decreased by using any other path. Indeed, it is clear to see that according to the description of the algorithm above, a path from s to any v_i traverses only the nodes with definitive label. Therefore, for any node with non-definitive label v_w at a given stage of the algorithm we have:

$$t_{sw} \geq t_{si} \tag{9}$$

If we consider a possibility of existence of a shorter path to node v_i via node v_w then we would have:

$$t_{si} = t_{sw} + t_{wi} \tag{10}$$

which is contradictory to formula (9).

4.2 Modified Dijkstra's Algorithm

The Dijkstra's algorithm is very efficient but it has the aforementioned limitation that it only works with non-negative arc (and edge) costs. A modified Dijkstra's algorithm can support negative arc costs (but only if there are not any negative loops).

The algorithm is similar to the previous one, but in this algorithm definitive labels may not be final: they must be checked in each round (in its third step) and if a lower cost is found, these labels will become non-definitive again and their values will be substituted with new (i.e., smaller) non-definitive labels. If all arc costs are non-negative no *definitive* $\rightarrow$ *non-definitive* change can occur, and the modified Dijkstra's algorithm behaves just like the regular Dijkstra, albeit slower. In the next example, to illustrate the modified Dijkstra's algorithm, the previous network topology is reused, with certain arcs reversed and with symmetrical costs (Fig. 5(a)).

We start labelling node `A` with 0 and all other nodes with an infinite value, as with the

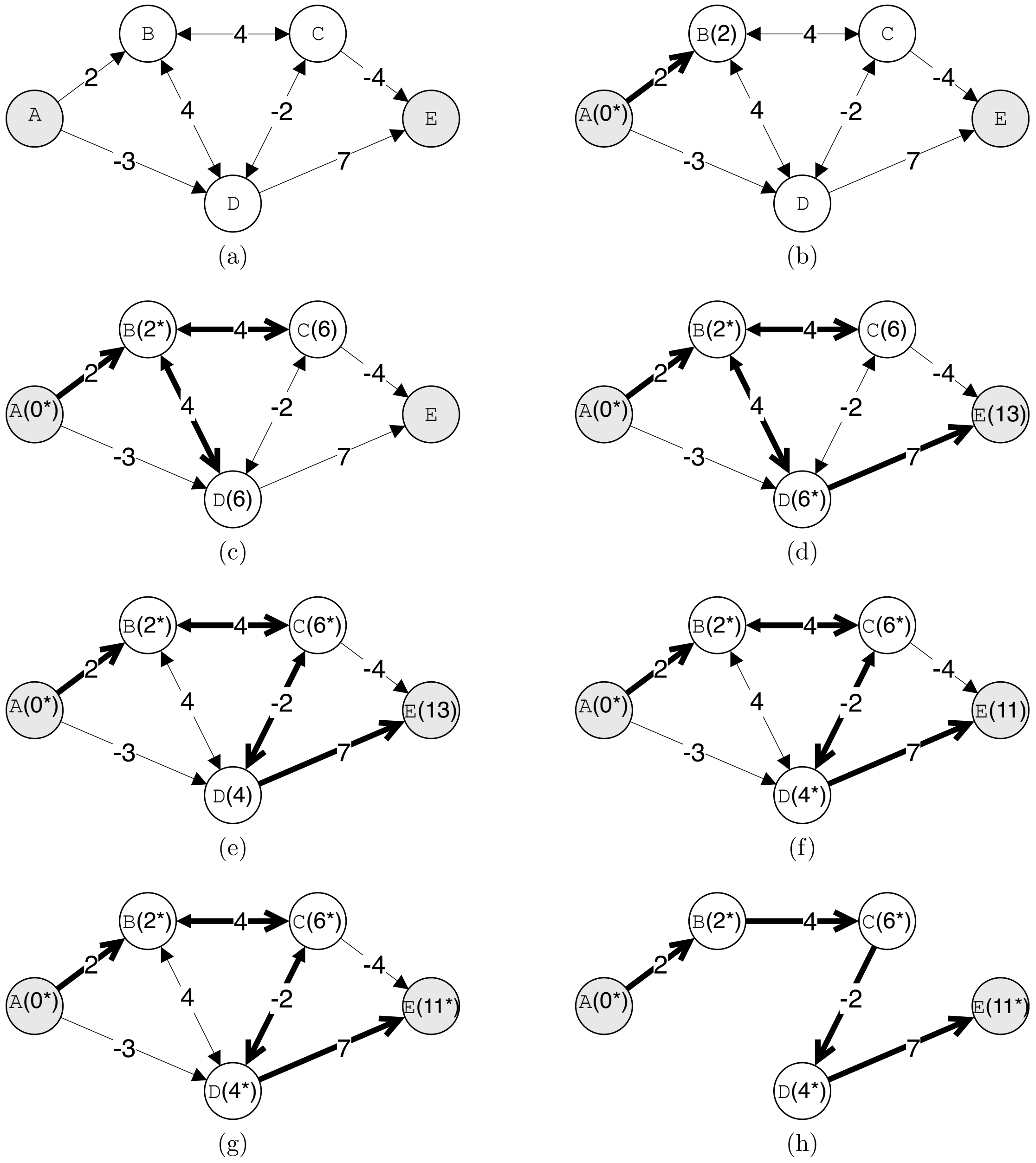


Figure 5: Modified Dijkstra's algorithm example: (a) Network used; (b)-(g) Modified Dijkstra's algorithm steps

original Dijkstra's algorithm. Then node `A` label becomes definitive. The next step is to upgrade the label for nodes connected to node `A`. Node `B` is the only one directly connected to node `A`. Label of `B` changes to 2 (Fig. 5(b)).

Next, `B` label becomes definitive. Once again we need to check which non-definitive labels need to be upgraded. Nodes connected to `B` are node `C` and node `D` (Fig. 5(c)). Node `C` gets a new non-definitive label, with value 6. Node `D` gets a new non-definitive label as well, also with value 6.

Any of the two nodes with minimum cost value, could be chosen to become the *current* node. Node `D` is chosen and gets a definitive label. Connected to `D` are nodes `A`, `B` and `E` (Fig. 5(d)). Only node `E` will change its label value to 13.

The node which will next become definitive is `C` (Fig. 5(e)). Connected to `C` is node `D`. Despite label of `D` having been considered definitive, it must become non-definitive again, because a smaller cost is found as shown in Fig. 5(e).

Label of `D` becomes definitive (Fig. 5(f)). Connected to node `D` is node `E`, and it gets a new non-definitive label, with value 11. Now, label of `E` becomes definitive, and the algorithm ends (Fig. 5(g)). All nodes now have definitive labels, defining the Shortest Path Tree (Fig. 5(h)).

5 Suurballe's and Bhandari's Algorithm

Considering the *min-sum* problem, two general algorithms are commonly used to determine two link-disjoint paths for this problem, the Suurballe's algorithm [26] and the Bhandari's algorithm [27].

5.1 Suurballe's Algorithm

For the Suurballe's algorithm [26], the steps of the basic algorithm are:

1. Find the minimum cost path from source s to destination d.
2. Transform the network, by changing the direction for the arcs in the computed minimum-cost path and the cost for all network arcs, as described below.
3. Find a new minimum-cost path from s to d in the changed network.
4. Remove the common arcs with opposite directions in the computed paths. The remaining arcs form two minimum cost disjoint paths.

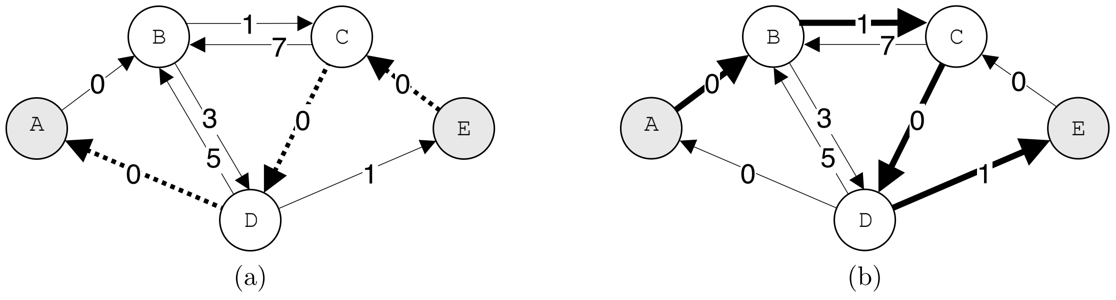


Figure 6: Suurballe's algorithm: (a) after transformation and (b) new/second shortest path

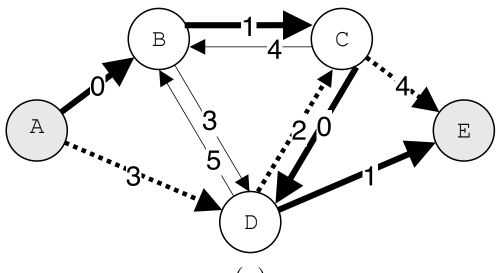
In Suurballe's algorithm, the transformation in Step 2 (creating what is called reduced costs) is done to avoid negative arc costs. Assuming the original cost of arc (v_i, v_j) is defined as c_{ij} and the minimum distance from source s to node v_i in the original network is computed in Step 1 as t_{si}, the arc costs c_{ij} are replaced by costs c'_{ij} as given in formula (11). This assures that the cost of a path between any two particular nodes is changed by the same amount that all other paths between those nodes, since any non-terminal node along the path contributes twice with some value (distance from s to that node), but once as negative and once as positive, with a null sum overall. Therefore, any optimal solution for the network with reduced costs is also the optimal solution for the network using the original costs.

$$c'_{ij} = c_{ij} + t_{si} - t_{sj} \tag{11}$$

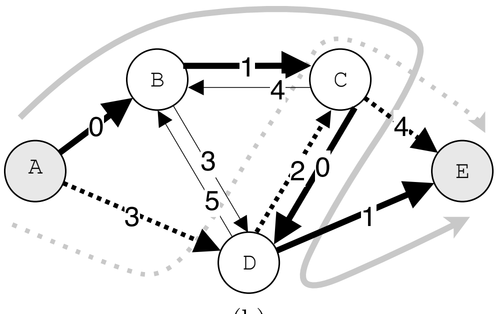
The cost c'_{ij} thus reflects the additional cost (above the minimum cost previously computed) induced by including arc (v_i, v_j) in the path to node v_j. After that, the arcs on the original path (which have a reduced null cost) will have their direction reverted, and any duplicate arcs coalesce into the null cost arc thus created. Since every c'_{ij} is either positive or null, the Dijkstra's algorithm can be used in this new network in Step 3 of the algorithm.

Figures 6, 7 and 8 exemplify the Suurballe's algorithm, in the same network as before, to determine two disjoint paths from `A` to `E` (see Fig. 4(a)).

The algorithm begins by calculating the shortest path from `A` to `E` (already shown in Fig. 4(g)). The next step is to reverse the arcs on the shortest path (depicted by the dotted lines), and assign them a null cost, as shown in Fig. 6(a). All other arc costs change, to reflect the additional cost of that arc to the shortest path. For instance, the cost of arc

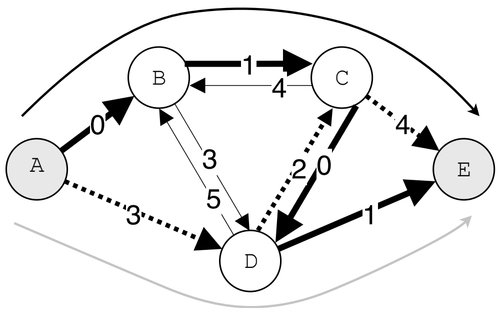


(a)

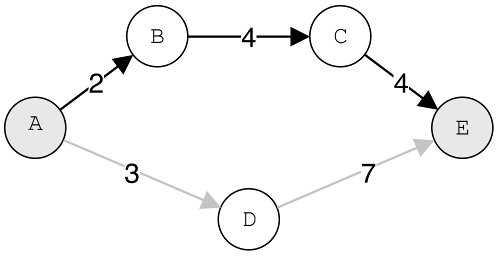


(b)

Figure 7: Suurballe's algorithm: (a) paths before deinterlacing and (b) interlaced edge



(a)



(b)

Figure 8: Suurballe's deinterlacing: (a) the two new paths and (b) after deinterlacing, disjoint paths

(C,B) becomes 7 ($4 + 5 - 2$) as stated in formula (11).

After creating the transformed network (Fig. 6(a)), the Disjkstra algorithm is used to obtain the second path (Fig. 6(b)). Both computed paths are shown in Fig. 7(a) (before deinterlacing). Figure 7(b) highlights that there is an interlaced edge (C,D) – an edge used on both directions – which consequently can be removed.

Figure 8(a) depicts the two computed disjoint paths, and Fig. 8(b) the new (disjoint) paths, obtained after removing the interlacing edges.

5.2 Bhandari's Algorithm

For the Bhandari's algorithm [27], the steps of the basic algorithm are:

1. Find the minimum cost path from source s to destination d.
2. Transform the network by changing cost and direction for the arcs in the computed minimum-cost path, as presented below.
3. Find a new minimum-cost path from s to d in the changed network.
4. Remove the common arcs with opposite directions in the computed paths. The remaining arcs form two minimum cost disjoint paths.

The Bhandari's algorithm uses the same structure as in Suurballe's algorithm [26], but a simpler network transformation in Step 2 (Bhandari's network transformation), which only affects a subset of the arcs. However, this transformation requires the new minimum-cost algorithm in Step 3 to be able to support negative arc costs.

As in Suurballe's algorithm, in Step 2 the arcs belonging to the original shortest path are reversed, but in this transformation their costs become symmetrical of the original costs. Duplicate arcs are coalesced, but now into the negative cost arcs. All remaining arcs are unchanged. This transformation creates a network with negative arc costs (although no negative cycles assuming non-negative original arc costs). Since negative arc costs prevent the use of the standard Dijkstra's algorithm, in Step 3 a suitable algorithm is required (e.g., the modified Dijkstra's algorithm).

The transformation in Step 2 ensures that arcs belonging to the shortest path of Step 1 are not used when the shortest path algorithm is run in the modified network in Step 3. In addition, the (new) arcs directed towards the source allow interlacing the path found in Step 3 with the shortest path found in the original graph (Step 1). As stated in [27], the

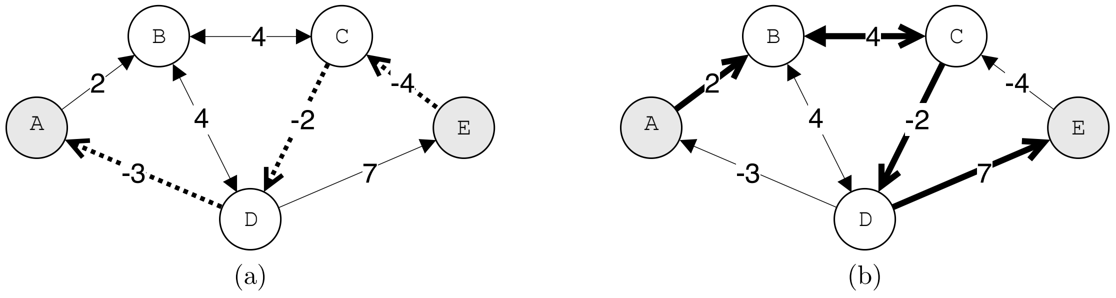


Figure 9: Bhandari's algorithm: (a) after transformation and (b) new/second shortest path

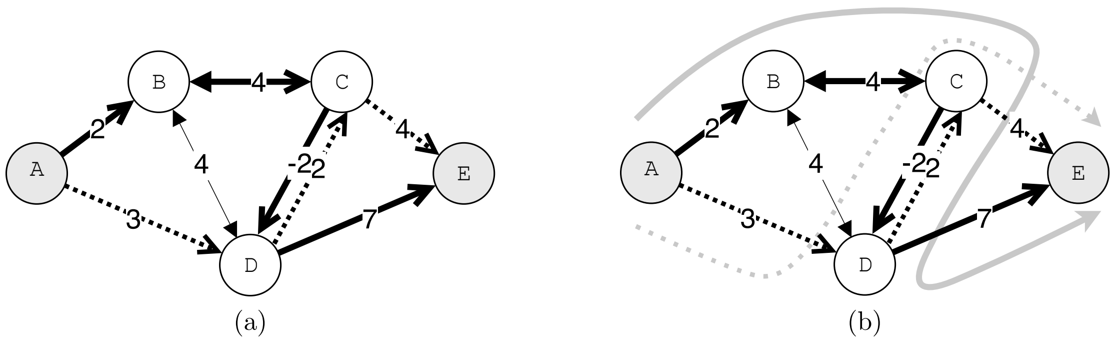


Figure 10: Bhandari's algorithm: (a) paths before deinterlacing and (b) interlaced edge

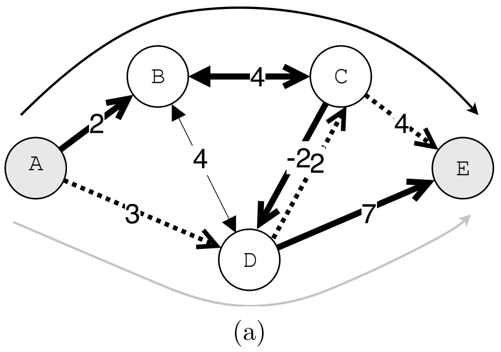
optimality of the solution results from the allowed interlacing and negativity of the arcs costs.

To exemplify the Bhandari's algorithm, we reuse again the network topology in Fig. 4(a). To get the transformed network, we need only to reverse the arcs present in the shortest path and assign them the symmetrical of their cost. Other arc costs do not change, as depicted in Fig. 9(a).

Again take the transformed network and calculate the second shortest path but now with a suitable algorithm allowing negative arc costs. This path is depicted in Fig. 9(b). Subsequently, this algorithm executes the same operations of the Suurballe's algorithm. Both paths determined are shown in Fig. 10(a). Figure 10(b) highlights the interlaced edge (pair of arcs in opposite directions) on those paths.

In Fig. 11(a), the two computed disjoint paths are presented again. In Fig. 11(b), the final disjoint paths are displayed after removing the interlacing edges.

For simplicity, the paths computed so far by the algorithms in this section are just arc-disjoint: they may share nodes other than the source and destination. However both Suurballe's and Bhandari's algorithms can be used to obtain node-disjoint paths of *min-*



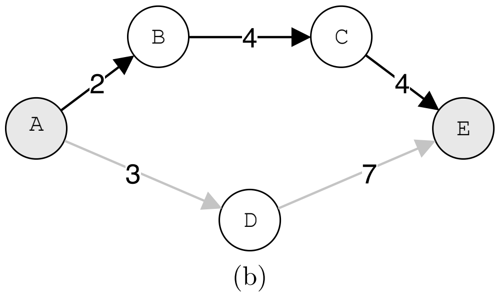


Figure 11: Bhandari's algorithm deinterlacing: (a) the two "new" paths and (b) after deinterlacing, disjoint paths

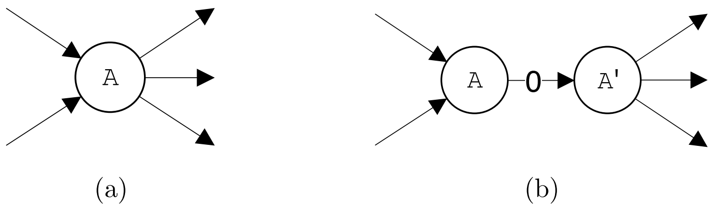


Figure 12: Examples of node splitting: (a) Before node splitting (b) After node splitting

sum cost, as any node can be split into two nodes linked by a directed arc with null cost, as is explained in the next sub-section.

5.3 *k*-Bhandari's Algorithm

To create fully arc- and node-disjoint paths, the network can be modified by replacing each node A with two nodes, A and A′, connected by a null cost arc, as shown in Fig. 12. If a set of k node-disjoint paths is required, the node transformation shown in Fig. 12 is necessary for each transit node traversed by all $k-1$ paths.

The arcs incident in A remain incident in A and the arcs emergent from A now emerge from A′. This transformation is illustrated in the k-Bhandari's algorithm. This algorithm is an extension of the previously presented Bhandari's algorithm, to obtain k disjoint paths, with $k \geq 2$. The rationale for this algorithm is to iterate the Bhandari's algorithm steps, and before each new path performing a Bhandari network transformation for the $k-1$ paths to discovered so far.

The network in Fig. 13(a) is used to exemplify the k-Bhandari's algorithm, knowing the two *min-sum* disjoint paths from A to H, shown in Fig. 13(a) To get the third disjoint path we need to transform the network considering these two paths.

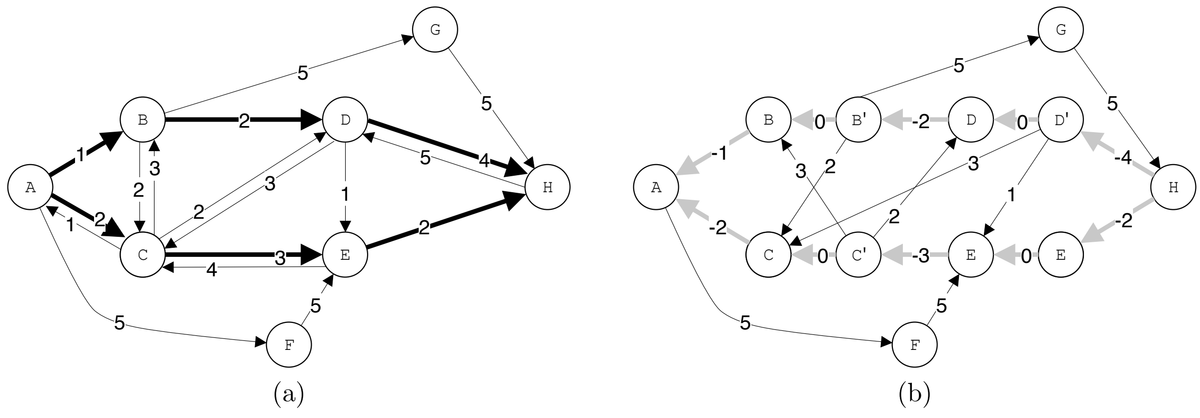


Figure 13: Bhandari's transformation: (a) Original paths, no node splitting (b) Transformed network, with node splitting

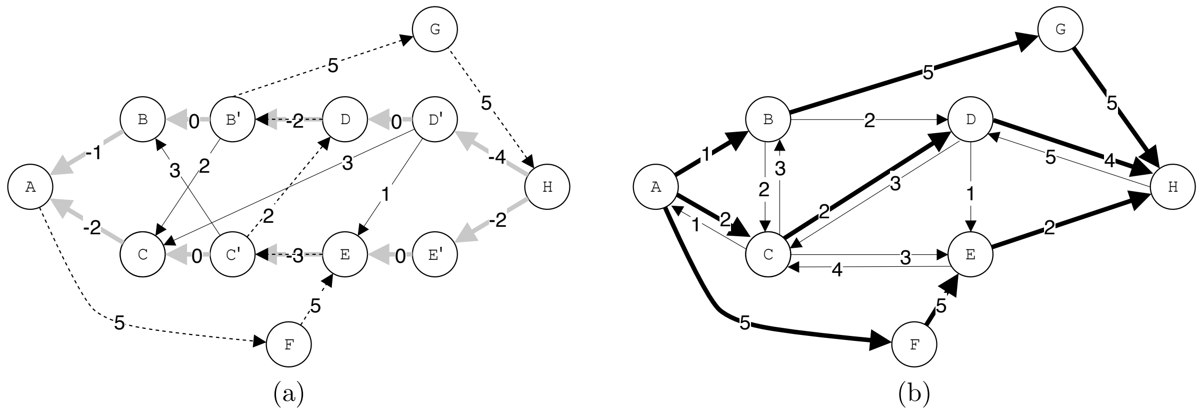


Figure 14: Example of k-Bhandari's algorithm: (a) New shortest path on transformed network (b) three disjoint paths, after merging and deinterlacing

Figure 13(b) shows the transformed network, considering the two previous paths. All arcs of the two paths were reversed and their cost changed to the symmetrical. No other cost was changed. Notice that to enforce node dis-junction, node splitting was performed in this figure.

In the transformed network, using the modified Dijkstra's algorithm, a new shortest path was obtained (shown by a dashed line in Fig. 14(a)). After removing the interlaced edge, we obtain three *min-sum* node-disjoint paths shown in Fig. 14(b).

6 Establishing a Set of k Disjoint Paths for a Multi-cost Network Scenario

Concerning the problem to establish a set of k disjoint paths, as already mentioned in this chapter, there are two distinct scenarios, referred to as *single-cost*, and *multi-cost* networks, accordingly. The former case refers to the same costs c_h of arcs a_h applied when finding each of k disjoint paths of a demand [2]. In general, the problem of establishing a set of k disjoint paths for a demand is $\mathcal{NP}$-complete [2, 28]. However, for single-cost networks, the exact solution can be found in polynomial time for a particular case of the *min-sum* problem with the objective to minimize the total cost of k disjoint paths for a demand (e.g., using Bhandari's algorithm [29]).

In a *multi-cost* network case [2], the cost c_h of arc a_h may be, in turn, different for each of k paths of a demand. It can occur, e.g., in the case of a backup path sharing scheme, where the cost of arc a_h is frequently only a fraction of its cost used in computations of a primary path [30]. However, as shown in [2], the problem of establishing a set of k disjoint paths for the multi-cost network case is $\mathcal{NP}$-complete even concerning the *min-sum* objective function.

In order to find a sub-optimal solution to the *min-sum* problem of establishing a set of k disjoint paths in a multi-cost scenario, we can apply e.g., k-Penalty algorithm from [1] being similar to the *active path first* (APF) approach [2] of establishing each next disjoint path in a single consecutive iteration. The main idea of the algorithm is presented in Fig. 16 for a demand to establish a set of k=3 node-disjoint paths between nodes `1` and `10`.

k-Penalty utilizes Dijkstra's algorithm [29] to calculate each of k paths of a demand. Similar to APF, in k-Penalty the primary path is determined first (Fig. 16(a)). However, next steps for k-Penalty are different. In APF, when calculating each next (j-th) disjoint path for a demand, the costs of links referring to the already calculated (j-1 paths) are set to infinity to identify the *forbidden arcs* needed to be excluded from further computations. A disadvantage of APF is its sensitivity to the *trap problem* denoting the inability to establish a next disjoint path despite a real possibility to obtain it for a graph of a given network topology, as shown in Fig. 15.

To prevent from triggering the trap problem, when calculating each next (j-th) path by k-Penalty, the costs of forbidden arcs (i.e., links traversed by previous j-1 paths or links incident to transit nodes of previous j-1 paths in the case of a link- and nodal-disjointness,

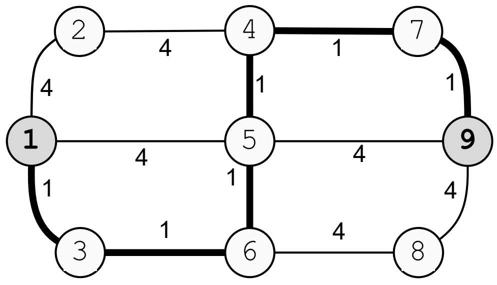


Figure 15: Example of a trap scenario for APF approach despite the topological possibility to establish a set of three node-disjoint paths for a demand between nodes `1` and `9`

accordingly) are not set to infinity. They are increased in the beginning of a given iteration by the total a cost of all j-1 already calculated disjoint paths (Fig. 16(b)). Traversing these forbidden links will thus cost more to try to prevent the next (j-th path) from traversing links already used by previous j-1 paths of a demand (i.e., to provide disjointness).

However, if the next j-th path happens to be not disjoint with previous j-1 paths (as in Fig. 16(b)), the costs of all conflicting arcs (i.e., links jointly traversed by previous j-1 paths or links incident to transit nodes jointly traversed by previous j-1 paths in the case of a link- and nodal-disjointness, accordingly) are permanently increased by the total cost of j-th path, all calculated paths are removed and the algorithm starts its execution from the beginning (Fig. 16(c)). k-Penalty terminates after calculating all k disjoint paths possibly after several conflicts (Fig. 16(c)-(g)) or returning no solution after reaching the maximum number of allowed conflicts.

7 Minimum-cost Path-pairs with Common Arcs and Nodes

We have seen in the previous sections that finding a suitable set of failure-disjoint paths for the connection minimising a certain arc metric (e.g., delay) might be a challenging problem. However, network operators are willing to provide their services with minimal effort and resources while a certain level of survivability is maintained for the connections. We argue in this section that end-to-end path disjointness is often an unnecessary and strict requirement to achieve this goal.

For example, paths might traverse disaster-prone regions, where disjointness is a desired property, while in safe regions the two paths might share some common arcs to minimize network resource usage. This routing problem boils down to find a minimum-cost path-pair

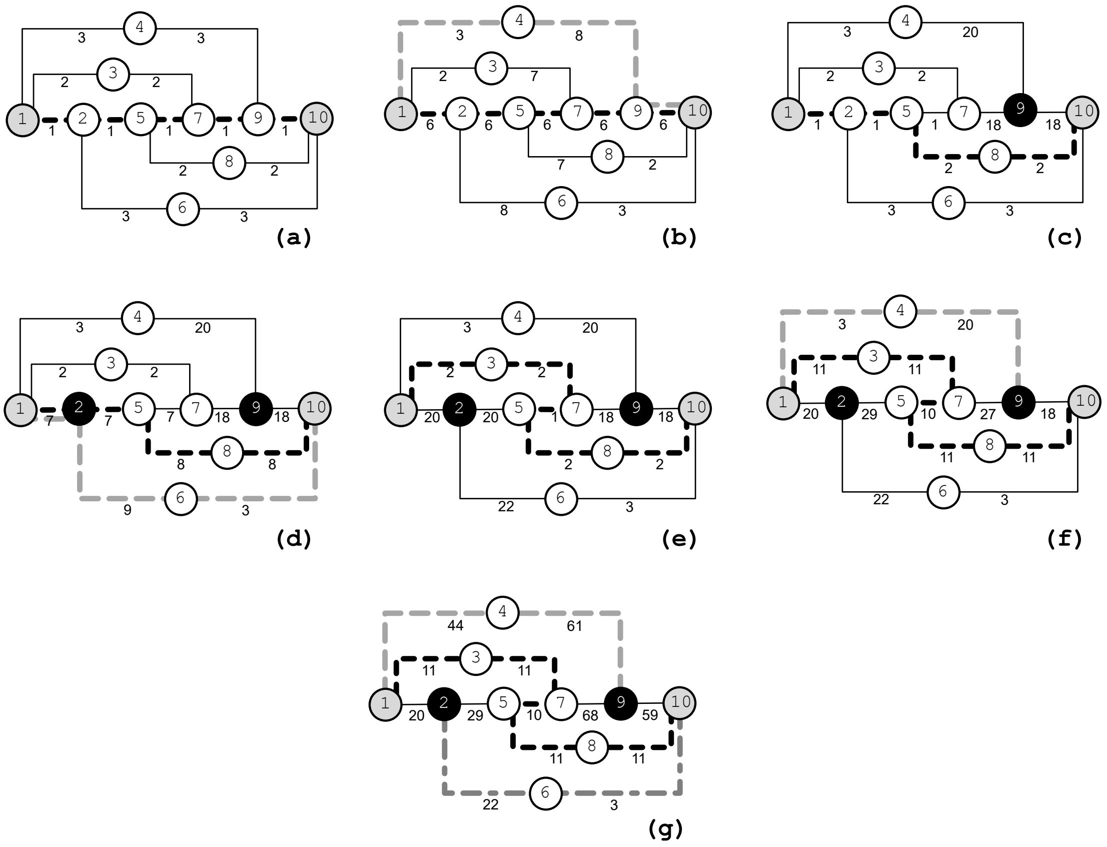


Figure 16: Example execution steps of k-Penalty algorithm for a demand between nodes `1` and `10`

while a given level of availability is maintained [31], discussed in Sect. 7.1. Furthermore, common nodes along an established arc-disjoint pair of paths might have also beneficial implications. For instance, in cases that the recovery time is critical, the traffic might be retransmitted from the closest common node to the faulty arc rather than from the source. Accordingly, the problem of allowing that the pair of arc-disjoint paths share a bounded number of common nodes [32] will be discussed in Sect. 7.2.

7.1 Tunable Availability-aware Routing

In [31], the concept of *tunable survivability* was developed, which provides a quantitative measure to specify the desired level of survivability. This concept allows any degree of survivability in the range 0% to 100%, thus transforming survivability into a quantifiable metric. In addition to the cost, each arc $a \in A$ is associated with a failure probability value ρ_a. We define a *survivable connection* as a pair of paths (p_1, p_2), which are not necessary disjoint. Accordingly, we quantify the level of survivability of connections as the probability that all common arcs are operational is at least $\rho = \prod_{a \in p_1 \cap p_2}(1 - \rho_a)$.

The cost of a survivable survivable connection (p_1, p_2) is defined as follows.

Definition 2 *For connection (p_1, p_2) its cost* CT-cost *$W_{CT}(p_1, p_2)$ is the sum of its arc costs counting the common arcs* twice *($W_{CT}(p_1, p_2) = \sum_{a \in p_1} w_a + \sum_{a \in p_2} w_a$).*

Counting common arcs twice suits well for QoS metrics like average delay, while counting arcs once is good to represent monetary costs, giving rise to several tunable survivability optimization problems [31]. Here, we establish an interesting structural property and present an algorithmic solution to the following problem.

Problem 1 CT-Constrained QoS Max-Survivability (CT-CQMS): *Given a network $G(V, A)$, a source node $s \in V$, a destination node $d \in V$ and a QoS bound B, find a survivable connection (p_1, p_2) from s to d such that:*

$$\max \prod_{a \in p_1 \cap p_2} (1 - \rho_a) \quad \textit{subject to} \quad W_{CT}(p_1, p_2) \leq B. \tag{12}$$

A. The Structure of Optimal Solutions

We proceed to show that the arcs that may affect the survivability level of the optimal solution are restricted to a (typically small) subset of the network's arcs.

Definition 3 *Given a survivable connection (p_1, p_2), a* critical arc *is an arc $a \in A$ that is common to both paths p_1 and p_2. Accordingly, the set of critical arcs of a survivable connection is defined as $\mathbb{C}(p_1, p_2) = \{a | a \in p_1 \cap p_2\}$.*

Definition 4 *Given a source s and a destination d, $L^{(s,d)}$ is the set of all the cost-shortest paths between s and d.*

Definition 5 *Given a source node $s \in V$ and a destination node $d \in V$, an* in-all-cost-shortest-paths arc *is an arc $a \in A$ that is common to all paths in $L^{(s,d)}$. Accordingly, the set of in-all-cost-shortest-paths arcs is defined as $\mathbb{L} = \{a |\ a \in \bigcap_{p \in L^{(s,d)}} p\}$.*

Note that if there is a unique cost-shortest path between s and d, i.e., $|L^{(s,d)}| = 1$, then $\mathbb{L}$ precisely consists of its arcs. Moreover, $\mathbb{L}$ is a subset of the set of arcs of any cost-shortest path. We are ready to present the main result of [31].

Theorem 1 *For* any bound *B on the additive end-to-end QoS, a (any) survivable connection (p_1, p_2) that is an optimal solution of the respective CT-Constrained QoS Max-Survivability Problem (per Def. 1) is such that* **all its critical arcs are in-all-cost-shortest-paths arcs**. *That is, $\mathbb{C}(p_1, p_2) \subseteq \mathbb{L}$.*

B. Establishing QoS-aware Survivable Connections

It was proved [31] that the optimization problem for Prob. 1 is $\mathcal{NP}$-hard. However, an exact solution of pseudo-polynomial complexity is still possible. The approach is based on a graph transformation that reduces our problem to a standard *Restricted Shortest Path* (RSP) problem. We recall that RSP is the problem of finding a cost-shortest path while obeying an additional (additive) constraint, as follows.

Definition 6 Restricted Shortest Path (RSP) Problem*: Given is a network $G(V, A)$ where each arc $a \in A$ is associated with a length l_a and a time t_a. Let T be a positive integer and $s, d \in V$ be the source and the destination nodes, respectively. Find a path p from s to d such that:*

$$\min \sum_{a \in p} l_a \quad \textit{subject to} \quad \sum_{a \in p} t_a \leq T. \tag{13}$$

Although the RSP problem is known to be $\mathcal{NP}$-hard [33], the literature provides several pseudo-polynomial solutions [34] as well as ϵ-optimal Fully Polynomial Time Approximation Schemes (FPTAS) [35], which we employ to solve Prob. 1. Moreover, we use the findings of Sect. 7.1A in order to further reduce the complexity of the solutions for the *CT* problem. Next, we present the main ideas behind a pseudo-polynomial algorithmic scheme for solving the CT-CQMS problem (refer to [31] for full details). The method employs two well-known algorithms: (i) to find two arc-disjoint paths with *min-sum* between two nodes (e.g., Sect. 5.1); (ii) to solve the $\mathcal{NP}$-hard RSP problem (e.g., using a pseudo-polynomial scheme from [34]). The CT-CQMS scheme includes the following stages:

Stage 0 The algorithm first finds a cost-shortest path p_{min} in the network $G(V, A)$ by employing a well-known shortest path algorithm (e.g., Sect. 4.1). According to Thm. 1, in an optimal solution of a CT problem, each of the critical arcs is included in any cost-shortest path. Therefore, we can have the algorithm focus on just nodes and arcs that belong to some (any) cost-shortest path.

Stage 1 Construction of a transformed network $\tilde{G}(\tilde{V}, \tilde{A})$ constituting an input for an RSP algorithm in the next stage. Specifically, the transformed network consists of two types of arcs: (i) *simple arcs*, which consist of the original network arcs in the cost-shortest path p_{min}; (ii) *disjoint arcs*, which consist of additional arcs representing possible *min-sum* path-pairs between pairs of nodes in the cost-shortest path p_{min}. Each arc is associated with two metrics: a length $l_{\tilde{a}}$ and a time $t_{\tilde{a}}$ according to its arc type.

Stage 2 Given the above transformed network $\tilde{G}(\tilde{V}, \tilde{A})$, the second stage calculates a restricted shortest path. Here, we may employ any pseudo-polynomial time algorithm or FPTAS, e.g., [34,35], for solving the RSP problem.

Stage 3 Accordingly, in the third stage, we construct the sought pair of paths of a survivable connection (p_1, p_2) out of the arcs of the RSP solution, i.e., path $\tilde{p}$. Then, the algorithm outputs the optimal survivable solution (p_1, p_2).

7.2 Min-Cost Arc-disjoint Path-pairs with Common Nodes

Considering single arc failures, a common approach for failure protection is the usage of a (fully) arc-disjoint path-pair (p_1, p_2) between given source and destination nodes. While the two paths must be arc-disjoint to guarantee surviving failures, they still might share common nodes, i.e., a node, other than the source and destination, that belongs to both paths. We define $\mathcal{C}(p_1, p_2)$ as the number of common nodes of (p_1, p_2). Accordingly, a *node-disjoint path-pair* (p_1, p_2) is a pair of arc-disjoint paths between these nodes with no common nodes, i.e., $\mathcal{C}(p_1, p_2) = 0$. In [32], the total cost of the two paths (*min-sum*) is used to measure the quality of the solution (as in Def. 2). We are looking for a *minimum cost arc-disjoint path-pair* between the source and destination with different restrictions on the number of their common nodes, e.g., resulting in the following optimization problem:

Problem 2 *Given are a network $G(V, A)$, a source node $s \in V$, a destination node $d \in V$ and an integer k. Find an arc-disjoint path-pair (p_1, p_2) such that:*

$$\min W(p_1, p_2) \quad \textit{subject to} \quad \mathcal{C}(p_1, p_2) \leq k. \tag{14}$$

This problem is referred to as *Minimum-Cost Arc-Disjoint Paths restricted Upper Bound Common Nodes* (called **MWLD-Upper-BCN** in [32]).

A. MWLD-Upper-BCN in General Graphs

It was proved that the problem versions with lower and tight bound are $\mathcal{NP}$-hard in general graphs [32]. However, note that in directed acyclic graphs (DAGs) a polynomial-time algorithmic scheme exists, which provides solutions for all three bounds [32]. We summarize here only the polynomial-time solution for the MWLD-Upper-BCN problem in general graphs, which consists of the following stages:

Stage 1 Comprising the construction of an auxiliary network $\tilde{G}(\tilde{V}, \tilde{A})$ with identical sets of nodes $\tilde{V} = V$ and arcs representing possible node-disjoint shortest path-pairs in the (original) network (e.g., calculated with Suurballe's algorithm [26] (see Sect. 5.1). Therefore, the cost of each arc in $\tilde{G}$ is equal to the cost of the disjoint path-pair between its end-nodes in G.

Stage 2 Calculating a cost-shortest path with at most $k + 1$ arcs between s and d in the new auxiliary network $\tilde{G}(\tilde{V}, \tilde{A})$ by applying the L-Link Bellman-Ford algorithm [32] (which is a dynamic programming approach that iteratively finds a cost-shortest path with at most L arcs), where L is set to be $k+1$. By stopping the algorithm execution at $L = k+1$, we will have an optimal path $\tilde{p}$ with at most k common nodes, where the arcs of the optimal path $\tilde{p}$ represents node-disjoint path-pairs in the original graph.

Stage 3 Finally, constructing the pair of paths (p_1, p_2) in the original network from the solution of the L-Link Bellman-Ford algorithm.

The time complexity of the algorithm is $\mathcal{O}(|A| \cdot |V|^2 + |V|^3 \cdot \log(|V|))$. We note here that in problem instances where the optimal arc-disjoint solution is already node-disjoint, or the bound is larger than the number of common nodes in the optimal arc-disjoint solution, the MWLD-Upper-BCN algorithm can not improve the cost of traditional arc-disjoint *min-sum* algorithms.

B. MWLD-Upper-BCN with Selective Common Nodes

From a practical perspective, the common nodes of the desired arc-disjoint path pair might consist of a special property, e.g., having high storage capacity (data centre node) or lying in a disaster-safe area. Therefore, it can be required to restrict the common nodes to be selected out of this smaller subset. Such a solution can be obtained with a slight modification of the MWLD-Upper-BCN algorithm.

Problem 3 *Given are a network $G(V, A)$, a source s and destination d, an integer k and a* set of allowable common nodes $V_c \subsetneq V \setminus \{s, d\}$. *Find an arc-disjoint path pair (p_1, p_2) such that in Prob. 2, while its common nodes* must *be allowable common nodes, i.e., $\{c | c \in p_1 \wedge c \in p_2\} \subseteq V_c$.*

In Prob. 3 the arc-disjoint path pair (p_1, p_2) also is a node-disjoint path pair in terms of the specific set $V \setminus V_c$. The proposed reduction [32] is shown in Fig. 12, which denies every node $v \in V \setminus (V_c \cup \{s, d\})$ from being a common node.

C. Minimum-Cost Arc-Disjoint Second Path with Common Nodes

Finally, we consider the problem of optimizing a second arc-disjoint path given one path, where a restriction on their number of common nodes has to be satisfied.

Problem 4 *Given are a graph $G(V, A)$, a source node $s \in V$, a destination node $d \in V$, a first established path $p_1 = s \rightsquigarrow d$. Find an arc-disjoint path p_2 such that:*

$$\min W(p_2) \quad \textit{subject to} \quad \mathcal{C}(p_1, p_2) \leq k. \tag{15}$$

It was shown that the optimal solution of a minimum-cost path is tractable for the upper-, tight- and lower-bound case [32]. The algorithm runs on the original graph G and follows a dynamic programming approach by maintaining 3-dimensional arrays for the path costs and path nodes. Thus, it has a time complexity of $\mathcal{O}(|V|^3)$.

8 Conclusions

The great majority of network failures are caused by a single node or link failure [3]. To protect against this type of events link-(node-) disjoint path pairs can be used. An Integer Linear Problem (ILP) Formulation for the calculation a min-sum disjoint path pair was presented. Then effective algorithms for solving the min-sum link-disjoint path pair problem were described, together with illustrative examples. A simple graph transformation, that allows to transform a node into an arc, was explained, thus showing how the previous algorithms can be used to obtain node-disjoint solutions. Moreover a detailed description of Dijkstra's algorithm, a basic sub-routine required by many of the discussed protection schemes, was given.

To further improve network survivability one must also consider strategies to make the network resilient to multiple failures. This can be addressed through the concept of SRLGs. Alternatively a set of k ($k > 2$) disjoint paths can be calculated. Hence k-Bhandari's algorithm is briefly described followed by an approach to determine a set of k-disjoint paths in a multi-cost network scenario.

Finally, and in order to reduce the cost required by extra resources to increase network resilience, a more conservative approach was discussed: instead of requiring a pair of paths to be fully disjoint, it would suffice to ensure the paths are disjoint in disaster-prone areas, while being allowed to share links and/or nodes in safer network regions. To conclude, we would like to recall that the algorithms presented in this chapter are often used as sub-routines for solving problems related to network resilience and/or to disaster-resilience such as in [10, 15].

References

[1] J. Rak. k-Penalty: a novel approach to find k-disjoint paths with differentiated path costs. *IEEE Communications Letters*, 14(4):354–356, April 2010.

[2] D. Xu, Y. Chen, Y. Xiong, C.Qiao, and X. He. On the complexity of and algorithms for finding the shortest path with a disjoint counterpart. *IEEE/ACM Transactions on Networking*, 14(1):147–158, Feb. 2006.

[3] J. Rak. *Resilient Routing in Communication Networks.* Computer Communication and Networks. Springer, 2015.

[4] T. Gomes, J. Tapolcai, C. Esposito, D. Hutchison, F. Kuipers, J. Rak, A. de Sousa, A. Iossifides, R. Travanca, J. André, L. Jorge, L. Martins, P. O. Ugalde, A. Pašić, D. Pezaros, S. Jouet, S. Secci, and M. Tornatore. A survey of strategies for communication networks to protect against large-scale natural disasters. In *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pages 11–22, Sept 2016.

[5] J. Tapolcai, Pin-Han Ho, D. Verchere, T. Cinkler, and A. Haque. A new shared segment protection method for survivable networks with guaranteed recovery time. *Reliability, IEEE Transactions on*, 57(2):272 –282, June 2008.

[6] M. Pollack. The maximum capacity through a network. *Operations Research*, 8(5):733–736, 1960.

[7] B. H. Shen, B. Hao, and A. Sen. On multipath routing using widest pair of disjoint paths. In *2004 Workshop on High Performance Switching and Routing*, pages 134 – 140, 2004.

[8] J. Q. Hu. Diverse routing in optical mesh networks. *IEEE Transactions on Communications*, 51(3):489–494, March 2003.

[9] Y. Cheng, J. Li, and J. P. G. Sterbenz. Path geo-diversification: Design and analysis. In *2013 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 46–53, Sept 2013.

[10] Y. Cheng, D. Medhi, and J. P. G. Sterbenz. Geodiverse routing with path delay and skew requirement under area-based challenges. *Networks*, 66(4):335–346, 2015.

[11] A. de Sousa, D. Santos, and P. Monteiro. Determination of the minimum cost pair of D-geodiverse paths. In *The 2017 International Conference on Design of Reliable Communication Networks (DRCN 2017)*, Munich, March 8-10 2017.

[12] R. C. de Andrade. New formulations for the elementary shortest-path problem visiting a given set of nodes. *European Journal of Operational Research*, 254(3):755 – 768, 2016.

[13] L. Martins, T. Gomes, and D. Tipper. Efficient heuristics for determining node-disjoint path pairs visiting specified nodes. *Networks*, 70(4):292–307, 2107.

[14] T. Gomes and J. Craveirinha. Efficient calculation of the most reliable pair of link disjoint paths in telecommunication networks. *European Journal of Operational Research*, 182(3):1055–1064, 2007.

[15] A. de Sousa, T. Gomes, R. Girão-Silva, and L. Martins. Minimization of the network availability upgrade cost with geodiverse routing for disaster resilience. *Optical Switching and Networking*, 31:127–143, 2019.

[16] T. Gomes, C. Simões, and L. Fernandes. Resilient routing in optical networks using SRLG-disjoint path pairs of min-sum cost. *Telecommunication Systems Journal*, 52(2):737–749, 2013.

[17] J. Tapolcai, L. Rónyai, B. Vass, and L. Gyimóthi. List of shared risk link groups representing regional failures with limited size. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, May 2017.

[18] A. Todimala and B. Ramamurthy. IMSH: An iterative heuristic for SRLG diverse routing in WDM mesh networks. In *13th International Conference on Computer Communications and Networks, ICCCN'2004*, pages 199–204, October 2004.

[19] M. J. Rostami, S. Khorsandi, and A. A. Khodaparast. CoSE: A SRLG-disjoint routing algorithm. In *Proceedings of the Fourth European Conference on Universal Multiservice Networks (ECUMN'07)*, Toulouse, France, 2007.

[20] D. Xu, Y. Xiong, C. Qiao, and G. Li. Trap avoidance and protection schemes in networks with shared risk link groups. *Journal of Lightwave Technology*, 21(11):2683–2693, November 2003.

[21] T. Gomes, M. Soares, J. Craveirinha, P. Melo, L. Jorge, V. Mirones, and A. Brízido. Two heuristics for calculating a shared risk link group disjoint set of paths of min-sum cost. *Journal of Network and Systems Management*, 23(4):1067–1103, 2015.

[22] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269 – 271, 1959.

[23] D. Oran. OSI IS-IS intra-domain routing protocol. ITEF RFC 1142, February 1990.

[24] C. Moy. Ospf version 2. ITEF RFC 2328, February 1998.

[25] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms and applications*. Prentice Hall, 1993.

[26] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984.

[27] R. Bhandari. Optimal physical diversity algorithms and survivable networks. In *Computers and Communications, 1997. Proceedings., Second IEEE Symposium on*, pages 433–441. IEEE, 1997.

[28] A. Sen, B. H. Shen, and S. Bandyopadhyray. Survivability of lightwave networks – path lengths in WDM protection scheme. *Journal of High Speed Networks*, 10(4):303–315, Oct. 2001.

[29] R. Bhandari. *Survivable Networks, Algorithms for Diverse Routing*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1999.

[30] J. Rak. Fast service recovery under shared protection in WDM networks. *Journal of Lightwave Technology*, 30(1):84–95, Jan 2012.

[31] J. Yallouz and A. Orda. Tunable QoS-aware network survivability. *IEEE/ACM Transactions on Networking*, 25(1):139–149, 2017.

[32] J. Yallouz, O. Rottenstreich, P. Babarczi, A. Mendelson, and A. Orda. Minimum-weight link-disjoint node-“somewhat disjoint” paths. *IEEE/ACM Transactions on Networking*, 26(3):1110–1122, June 2018.

[33] M. R. Garey and D. S. Johnson. *”Computers and Intractability: A Guide to the Theory of NP-Completeness”*. W. H. Freeman & Co., 1979.

[34] HC Joksch. The shortest route problem with constraints. *Journal of Mathematical analysis and applications*, 14:191–197, 1966.

[35] D. H. Lorenz and D. Raz. A simple efficient approximation scheme for the restricted shortest path problem. *Operations Research Letters*, 28(5):213 – 219, 2001.