Divide And Conquer For Fast SRLG Disjoint Routing

Abstract—**Ensuring transmission survivability is a crucial problem for high-speed networks. Path protection is a fast and capacity-efficient approach for increasing the availability of end-to-end connections. The emerging SDN infrastructure makes it feasible to provide diversity routing in a practical network. For more robust path protection, it is desirable to provide an alternative path that does not share any risk resource with the active path. We consider finding the SRLG-Disjoint paths, where a Shared Risk Link Group (SRLG) is a group of network links that share a common physical resource whose failure will cause the failure of all links of the group. Since the traffic is carried on the active path most of time, it is useful that the weight of the shorter path of the disjoint path pair is minimized, and we call it Min-Min SRLG-Disjoint routing problem. We prove this problem is NP-complete. The key issue faced by SRLG-Disjoint routing is the trap problem, where the SRLG-disjoint backup path (BP) can not be found after an active path (AP) is decided. Based on the min-cut of the graph, we design an efficient algorithm that can take advantage of existing search results to quickly look for the SRLG-Disjoint path pair. Our performance studies demonstrate that our algorithm can outperform other approaches with a higher routing performance while also at a much faster speed.**

Index Terms—**Shared Risk Link Groups (SRLG), SRLG-Disjoint Routing, Max-Flow Min-Cut Theorem**

I. Introduction

With the rapid increase of application data and the deployment of high speed network, a failure in the network infrastructure (e.g. a fiber cut or a router shutdown) may lead to a vast amount of data loss. This makes it more important to exploit diversity routing with additional paths to increase the survivability [1] of end-to-end transmissions.

Besides the difficulty and complexity in finding the additional paths, it is also difficult to enable transmission through the alternative paths in conventional networks. The emerging Software Defined Networking (SDN) paradigm [2], [3] separates the data plane from the control plane, and applies centralized control for more efficient network monitoring and management. Compared to the decentralized routing in conventional networks, the centralized control in SDN allows advanced routing service such as diversity routing to be easily implemented [4], [5], [6]. For example, some simple extensions on OpenFlows allow nodes to autonomously react to failures by switching to a pre-computed end-to-end backup path [7], [8]. This makes the diversity routing a viable and increasingly important method for network survivability.

To protect a mission-critical connection from a single link (node) failure, a common solution for path protection [9] is to find a link (node) disjoint pair of paths from a source (ingress) node to a destination (egress) node. When there are no faults, the active path (AP, also called working path or primary path) is used to carry the traffic. When a fault occurs, the traffic is re-routed along the other path, called backup path (or BP). For a network to be reliable, both AP and BP paths must not share a common risk of failure.

A Shared Risk Link Groups (SRLG) is a group of network links that share a common physical resource (cable, conduit, node or substructure) whose failure will cause the failure of all links of the group. To ensure that AP and BP paths do not fail at the same time, they should be SRLG-disjoint without links sharing any common risks. Since the traffic is carried on AP most of the time, it is useful that the cost of one path of the disjoint path pair is minimized to use the path as AP. In order to enable SRLG disjoint routing and facilitate other failure risk management, it is important to know which links form SRLG. Some recent studies [10], [11] propose to automatically discover and collect the SRLG information. In this paper, we focus on Min-Min SRLG-disjoint routing to find two SRLG-disjoint paths such that the smaller weight of these two paths is minimized.

The finding of backup paths that are not coupled with the active path yet also efficient has been a big challenge and a problem that has attracted many attentions from academia and industries. Although several link/node-disjoint routing algorithms are proposed so far [12], [13], [14], [15], [16], [17], [18], [19], the number of SRLG-disjoint routing algorithms is very limited. A link-disjoint or node-disjoint routing problem is only a simple and specific case of SRLG-disjoint routing problems. Among the limited studies on SRLG-disjoint routing, one type of solution is to formulate an Integer Linear Programming (ILP) [19] problem to jointly optimize the selection of both AP and BP, and then solve the formed ILP problem using the branch-and-bound [20] search techniques. With a high time complexity, the ILP-based algorithms are not feasible for large networks.

To reduce the complexity, several heuristic algorithms following active-path-first (APF) are proposed [21], [22], [23]. However, this type of methods faces a big challenge. For a specific AP, it may not be able to find an SRLG disjoint BP even though a pair of disjoint paths do exist. This is the so-called "trap" problem [24], which can exist even if the network is highly connected [25], and becomes more severe when the network is sparse. Although some attempts are made to address the trap issue in a simple scenario where one link belongs to only one SRLG, a large number of paths need to be tested in order to find a disjoint path pair, which incurs a high computational complexity. As a network link may belong to several SRLGs, the problem can become intractable.

In this work, based on the graph theory, we provide the insights for the trap problem when looking for the SRLG-disjoint paths. Specifically, for an AP found with the trap problem, we observe that there exists a sub-set of links on the AP path that no AP going through all these "problematic" links can find an SRLG-disjoint BP. We call this set as a SRLG Conflicting Link Set. Once encountering a trap problem, instead of searching through all possible alternative paths, we propose to look for the SRLG Conflicting Link Set based on max-flow min-cut theorem [26]. We further propose a divide-and-conquer min-min SRLG disjoint routing algorithm to partition the original routing problem into several sub-problems which can be executed in parallel to find the viable AP and BP pairs. Our main contributions are summarized as follows:

- We propose a novel scheme to construct a new flow graph with a clever setting of the link capacity to facilitate the finding of the SRLG Conflicting Link Set.
- We propose an algorithm to find a minimum SRLG Conflicting Link Set, which helps to reduce the complexity of searching for the alternative SRLG-disjoint path pair.
- Based on the risk sharing features of the SRLG graph, we transform the minimum SRLG Conflicting Link Set finding problem to a set cover problem, which allows us to apply general algorithms to find the minimum SRLG Conflicting Link Set under different complex SRLG scenarios (including a link belonging to one or multiple SRLG with various SRLG patterns).
- We propose a novel divide-and-conquer algorithm which can partition the original min-min SRLG disjointing routing problem into multiple sub-problems for parallel executions upon encountering a trap problem. Compared to existing techniques, such a solution searching process can take advantage of the existing AP search results and parallel executions for significantly faster path finding.
- We have done extensive simulations on a multi-core CPU platform to evaluate the proposed algorithms. The simulation results demonstrate that our algorithm can find the best solutions in different network scenarios at much faster speed.

The rest of the paper is organized as follows. We introduce the related work in Section II and background in Section III. We introduce our problem and our basic solution to address the trap issue in Section IV and Section V, respectively. In Section VI and VII, we present in details our algorithms for finding the SRLG Conflicting Link Set and the SRLG-disjoint routing paths. Finally, we conclude our work in Section IX.

II. Related work

A shared risk link group (SRLG) is a group of links that share a component whose failure causes the failure of all links in the group. For path protection, although some link/node-disjoint routing algorithms have been proposed [12], [13], [14], [15], [16], [17], [18], [19], finding paths for SRLG-disjoint routing is more intractable and there are very limited studies on SRLG-disjoint routing. When the SRLG contains only one link, the SRLG-disjoint routing problem can be reduced to a link-disjoint routing problem, while node-disjoint routing problem could be transformed into a link-disjoint routing problem through node splitting [26]. Since a SRLG group generally includes more than one link and a network link may belong to several SRLGs, finding a pair of SRLG-disjoint paths is much more difficult than finding a pair of link/node disjoint paths.

To solve a SRLG-disjoint routing problem, one possible way is to form an Integer Linear Programming (ILP) [19] problem to optimally select both AP and BP paths through the branch-and-bound search. This method incurs a high time complexity, and is not feasible for large networks. To reduce the complexity, APF-based heuristics are shown to be able to achieve near-optimal solutions to the Min-Min SRLG-disjoint routing problem [21], [22], [23]. In these APF-based heuristics, an AP is found first by using the Dijkstra algorithm (or any other shortest path algorithms) without considering the need to find a corresponding BP, and the BP is found (again using the Dijkstra algorithm for example) after removing the links or nodes along the AP or share the risk with the AP.

However, as a major challenge in using the APF heuristic, once an AP is found, a SRLG disjoint BP may not be able to be found even though a pair of disjoint paths do exist in the network. This is called the "trap" problem, which can happen even if the network is highly connected [25], and certainly cannot be ignored in a sparsely-connected network. There are two kinds of trap, unavoidable and avoidable trap. Unavoidable traps are constraints forced by the topology, and cannot be solved by any algorithm. If a network is not 2-edge connected, then there is no algorithm that can guarantee the presence of two SRLG disjoint pathes in the topology. On the other hand, an avoidable trap occurs when an SRLG-disjoint path pair exists between two nodes but cannot be found due to the shortcomings of the routing algorithms. In this paper, we consider the avoidable trap.

As an extension to a simple-APF, the KSP (K-shortest path) algorithm [23] is proposed to deal with the trap problem for node/link disjoint path. Although it is one of the most effective algorithms to deal with the trap problem, its performance suffers in a large network as KSP may involve multiple path-searching-tests (K tests) until it finds the disjoint paths. After the current candidate AP encounters the trap problem, the next candidate AP to be tested is selected solely based on the path length, without considering which link (or links) along the current candidate AP has caused the failure in finding disjoint BP. As a result, a large number of paths often need to be tested in order to find a disjoint path pair, which introduces a large time complexity with a large K. Different from KSP, for an AP encountering the trap problem, we apply the SRLG Conflicting Link Set derived from this AP path to guide the future AP testing. This helps to largely reduce the time complexity in finding the alternative paths.

Other SRLG-disjoint routing algorithms, including [27], [28], [29], [30], [31], search for the maximal SRLG disjoint routing paths that share the minimum number of common links. As AP and BP may share same risks, the solutions found through this type of approaches are not reliable. Our algorithm targets to find complete SRLG disjoint paths. The work in [32] attempts to find the complete SRLG disjoint paths. It cuts down the problem search space to speed up the path searching process. However, it may return paths with large cost as the space after cutting down may lose the optimal solution. Instead, to largely speed up the searching process, we exploit the conflicting link set found to partition the original problem into multiple sub-problems which can be executed in parallel. As a result, our algorithm can run much faster and return the paths with very low cost.

The work in [29] transforms the SRLG disjoint routing problem to the link-disjoint routing problem and then exploits the link-disjoint routing algorithm to solve the problem. However, only certain SRLG styles (e.g, star-style) can be transformed to the link-disjoint, which limits the application of this scheme. Rather than calculating the conflicting link set, when AP encounters a trap problem, CoSE [28] tries to find a SRLG set that no AP going through the SRLG set can find the SRLG BP path. CoSE first looks for the SRLG set commonly shared by APs through multiple rounds of search, then partitions the original problem based on the SRLG set to search for a SRLG-disjoint path pair. Without utilizing the risk sharing feature in SRLG, this exhaustive search in CoSE incurs very high computational overhead.

Besides the long computation time, most of current SRLG-routing algorithms consider a single scenario that one link belongs to only one SRLG. Different from existing studies [28], [29], this paper tries to guide the search of alternative AP paths by analyzing the links on the path of the AP caught into the "trap". More specifically, this paper proposes a divide and conquer algorithm to partition the original problem into multiple sub-problems which can be executed in parallel to largely speed up the searching process. Moreover, our SRLG-routing algorithm does not have any constraint on the SRLG-style and can efficiently handle complex SRLG situation that a link belongs to multiple SRLGs.

III. PRELIMINARIES

This paper designs SRLG disjoint routing algorithm based on the max-flow min-cut theorem. This section introduces the preliminaries on the theorem.

A. Maximum flow

Let $G = (\mathbb{V}, \mathbb{E})$ be a network (where $\mathbb{V}$ is the set of $|\mathbb{V}|$ nodes and $\mathbb{E}$ is the set of $|\mathbb{E}|$ links) with $s \in \mathbb{V}$ and $d \in \mathbb{V}$ being the source and the destination respectively.

The **capacity** of link e_i, denoted by c_{e_i}, represents the maximum amount of flow that can pass through the link. A **flow** of link, denoted by f_{e_i}, should meet the following two constraints:

1) Capacity Constraint: $\forall e_i \in \mathbb{E}$: $f_{e_i} \le c_{e_i}$.

2) Conservation of Flows: $\forall u \in \mathbb{V} - \{s, d\}$: $\sum\limits_{v\in\mathbb{V}} f_{(v,u)} = \sum\limits_{v\in\mathbb{V}} f_{(u,v)}$, where (v,u) and (u,v) denote the links $e(v,u)$ and $e(u,v)$.

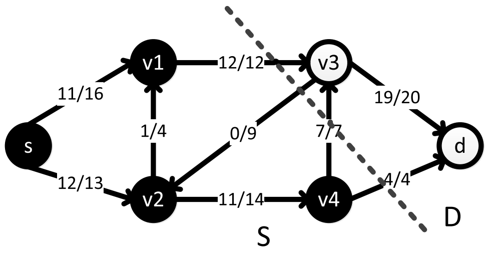


Fig. 1. An example to illustrate max-flow min-cut theorem. Each link e_i is labeled by f_{e_i}/c_{e_i}, where f_{e_i} and c_{e_i} denote the flow and capacity of link e_i, respectively.

The **value of flow** is defined by $|f| = \sum\limits_{v\in\mathbb{V}} f_{(s,v)}$, where s is the source. It represents the amount of flow passing from s to d.

Maximum Flow Problem: The goal is to maximize $|f|$ by routing as much flow as possible from s to d.

B. Minimum cut

An **s-d cut** $\Phi = (\mathbb{S}, \mathbb{D})$ is a partition of $\mathbb{V}$ such that $s \in \mathbb{S}$ and $d \in \mathbb{D}$. The **cut-set** $\mathbb{L}_\Phi$ of Φ is the link set

$$\mathbb{L}_\Phi = \{(u,v) \in \mathbb{E} : u \in \mathbb{S}, v \in \mathbb{D}\}. \tag{1}$$

Note that if the links in the cut-set $\mathbb{L}_\Phi$ are removed, then $|f| = 0$. That is, no flow can pass from s to d. In this paper, we try to find the SRLG Conflicting Link Set based on the feature of the cut-set.

The capacity of an s-d cut Φ is defined by $c(\Phi) = \sum\limits_{e_i\in\mathbb{L}_\Phi} c_{e_i}$.

Minimum s-d Cut Φ Problem: Minimize $c(\Phi)$, that is, to determine $\mathbb{S}$ and $\mathbb{D}$ such that the capacity $(c(\Phi))$ of the s-d cut $\Phi = (\mathbb{S}, \mathbb{D})$ is minimal.

C. Max-flow min-cut theorem

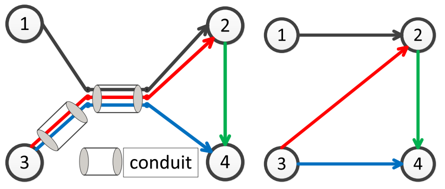
Max-flow min-cut theorem: The maximum value of an $s-d$ flow is equal to the minimum capacity over all $s-d$ cuts. In Fig.1, max-flow f in G with its value $|f| = f_{(s,v_1)} + f_{(s,v_2)}$. The cut $\Phi(\mathbb{S}, \mathbb{D})$ with $\mathbb{S} = \{s, v_1, v_2, v_4\}$ and $\mathbb{D} = \{v_3, d\}$ is the min-cut with its capacity $c(\Phi) = c_{(v_1,v_3)} + c_{(v_4,v_3)} + c_{(v_4,d)} = 12 + 7 + 4 = 23$. Obviously, $|f| = c(\Phi)$, that is, the maximum value of an s-d flow is equal to the minimum capacity over all s-d cuts.

IV. PROBLEM DESCRIPTION AND ANALYSIS

A. Problem description

A shared risk link group (SRLG) is a group of links that share a component whose failure causes the failure of all links in the group. A link can belong to multiple SRLGs.

An example of such a component is the fiber conduit [33] in optical networks, where several optical links may be placed side-by-side in one single conduit, as illustrated in Fig.2. Links (1,2), (3,2) and (3,4) are placed inside a single conduit, while links (3,2) and (3,4) also share another single conduit. If a conduit gets cut, the corresponding links will fail. Each conduit corresponds a SRLG. Other example applications of SRLGs are the correlated congestion of transportation networks and cascading failures of power grid networks [34].



(a) Physical topology with conduit (b) Network Graph

Fig. 2. Example of shared risk link group(SRLG)

Let $\mathbb{R}$ be the risk (failure) set in the network. Each risk may correspond to a conduit cut, a fiber cut, a card failure at a node, a software failure, or any combination of these factors. For $r_i \in \mathbb{R}$, its SRLG is the link set associated with risk r_i, denoted as $\mathbb{R}_{r_i}$, where $1 \leq i \leq \chi$ and $\chi = |\mathbb{R}|$ is the number of risks/SRLGs. In Fig.3(a), the network includes five SRLGs $\mathbb{R}_{r_1} = \{e_1, e_9\}$, $\mathbb{R}_{r_2} = \{e_2, e_3, e_{19}\}$, $\mathbb{R}_{r_3} = \{e_2, e_4, e_{11}, e_{17}\}$, $\mathbb{R}_{r_4} = \{e_5, e_{13}\}$, $\mathbb{R}_{r_5} = \{e_{15}, e_{18}\}$. In this example, link e_2 is in two SRLGs $\mathbb{R}_{r_2}$ and $\mathbb{R}_{r_3}$.

A network is often represented as a graph $G(\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ is the set of $|\mathbb{V}|$ nodes (which for instance represent routers) and $\mathbb{E}$ is the set of $|\mathbb{E}|$ links (which for instance represent optical fiber lines or radio channels). Links may be characterized by weights representing for instance their delay, length, or cost. For a link e_i, we denote the weight of the link as w_{e_i}. The weight of a path P is denoted as $w_P = \sum\limits_{e_i \in \mathbb{P}} w_{e_i}$, which is the sum of the weight of each link in the path.

Let r_P denote the risk set that impacts a path P, that is $r_P = \{r \in \mathbb{R}$: path P contains links in $\mathbb{R}_r\}$. In Fig.3(c), the link set on the AP is $\mathbb{AP} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$ with $e_1 \in \mathbb{R}_{r_1}$, $e_2 \in \mathbb{R}_{r_2}$, $e_2 \in \mathbb{R}_{r_3}$, $e_3 \in \mathbb{R}_{r_2}$, $e_4 \in \mathbb{R}_{r_3}$, $e_5 \in \mathbb{R}_{r_4}$, the risk set of AP is $r_{AP} = \{r_1, r_2, r_3, r_4\}$. $\mathbb{ER}$ denotes the set of links not belonging to AP that share the common risks with AP. In Fig.3(c), $\mathbb{ER} = \{e_9, e_{11}, e_{17}, e_{13}, e_{19}\}$.

SRLG-disjoint paths share no common risks among themselves, that is, the failure of a path due to a risk would not affect other paths. Fig.3(b) shows two SRLG-disjoint paths, denoted as AP and BP. As these two paths share no common risk, if AP fails, BP can still work. This paper focuses on finding two SRLG-disjoint paths for path protection, which can be described as follows.

Min-Min SRLG-disjoint routing problem. Given a graph $G(\mathbb{V}, \mathbb{E})$, a weight w_{e_i} associated with each link $e_i \in \mathbb{E}$, a source s and a destination d, find a pair of SRLG disjoint paths from s to d (denoted as AP and BP), thus that the smallest path weight of the two disjoint paths is minimized, that is,

$$\begin{array}{ll} \underset{AP,BP}{\mathit{minimize}} & \min\left(w_{AP}, w_{BP}\right) \\ \mathit{subject\ to} & r_{AP} \cap r_{BP} = \phi \\ & \mathbb{AP} \cap \mathbb{BP} = \phi \end{array} \tag{2}$$

where w_{AP} and w_{BP} are the path weights of AP and BP, respectively, $\mathbb{AP}$ and $\mathbb{BP}$ are the link sets on paths AP and BP, respectively, r_{AP} and r_{BP} are the risk sets that impacts AP and BP, respectively.

B. Problem analysis

Theorem 1. *The Min-Min SRLG-disjoint path problem is NP-complete.*

Proof. According to [16], the Min-Min link-disjoint routing problem is NP-complete problem. It is commonly known that the Min-Min link-disjoint routing problem is a sub-problem of Min-Min SRLG-disjoint routing problem. That is, after we remove the constraint $r_{AP} \cap r_{BP} = \phi$ in (2), the Min-Min SRLG-disjoint routing problem is transformed to a Min-Min link-disjoint routing problem. Therefore, according to the reducibility theorem [35] in computer complexity field, it is easy to conclude that Min-Min SRLG-disjoint routing problem is also NP-complete. □

V. Trap Problem and Solution Overview

In this section, we first introduce the trap problem, and then given an overview of our solution.

A. Trap problem

As discussed earlier, APF-based heuristic algorithms may get caught into the "trap" problem. That is, when an AP is found, it may not be able to find a SRLG disjoint BP path even though a pair of disjoint paths do exist in the network.

Fig.3.(c),(d) illustrates the trap problem. The dotted line denotes an AP with its link set $\mathbb{AP} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$. After removing the links on AP and also the links that share the common risks with AP, the remaining graph shown in Fig.3.(d) is obviously disconnected, so BP can not be found.

Although KSP is known as an effective algorithm to handle the trap problem, it may face the problem of inefficiency. Fig.4 shows an example to illustrate why the KSP is extremely inefficient. In this graph, suppose the link weights of e_1, e_2, e_3, e_4 are much larger than other links. Moreover, among e_1, e_2, e_3, e_4, the link weights of e_1 and e_2 are much smaller than those of e_3, e_4. Then the first K smallest weight paths from s to d found in the first K tests will always contain the shortest path segment e_1, e_2 denoted by the dotted line. The first K shortest APs will suffer from the trap problem, as e_1 and e_4 share the same risk and BP can not be found as a result. To avoid the trap problem, K have to be set as a large value, which brings a high time complexity to KSP.

When a trap problem happens and there is no SRLG-disjoint BP can be found for a given AP, there may exist a sub-set of links in the $\mathbb{AP}$ such that no AP going through all these "problematic" links can find a SRLG-disjoint BP. In this paper, we call this set as a SRLG Conflicting Link Set. Different from KSP, when the shortest AP encounters the trap problem, we will address the issue through two major steps. We will first find the set of SRLG conflicting links (Section VI) like the set $\{e_1, e_2\}$ in the example of Fig.4, and then apply a divide

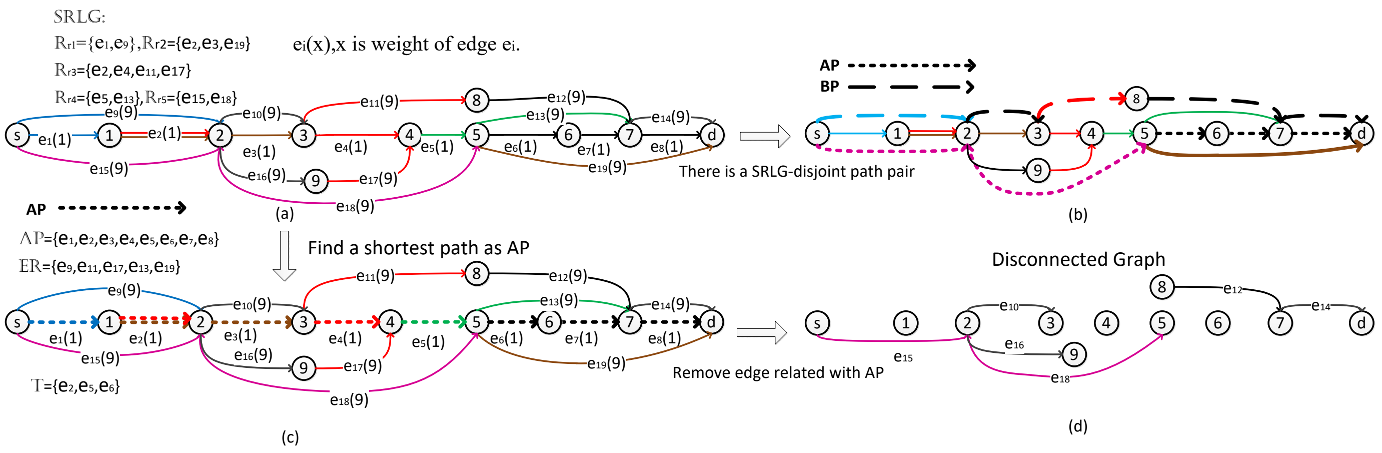

SRLG:
Rr1={e1,e9},Rr2={e2,e3,e19}
Rr3={e2,e4,e11,e17}
Rr4={e5,e13},Rr5={e15,e18}
ei(x),x is weight of edge ei.
There is a SRLG-disjoint path pair
AP
BP
(a)
(b)
AP
AP={e1,e2,e3,e4,e5,e6,e7,e8}
ER={e9,e11,e17,e13,e19}
Find a shortest path as AP
T={e2,e5,e6}
Disconnected Graph
Remove edge related with AP
(c)
(d)

Fig. 3. (a) A graph with five SRLGs: $\mathbb{R}_{r_1} = \{e_1, e_9\}$,$\mathbb{R}_{r_2} = \{e_2, e_3, e_{19}\}$,$\mathbb{R}_{r_3} = \{e_2, e_4, e_{11}, e_{17}\}$,$\mathbb{R}_{r_4} = \{e_5, e_{13}\}$,$\mathbb{R}_{r_5} = \{e_{15}, e_{18}\}$. (b)AP and BP in the graph. (c) The shortest weight path AP in the graph. $\mathbb{AP} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$, $\mathbb{ER} = \{e_9, e_{11}, e_{17}, e_{13}, e_{19}\}$. (d) Graph after deleting the links in $\mathbb{AP}$ and $\mathbb{ER}$.

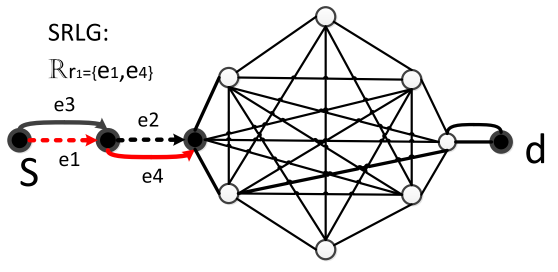


Fig. 4. An example to illustrate the inefficiency of KSP

and conquer algorithm (Section V-B) to partition the original problem into two sub-problems $\mathcal{P}(\emptyset, \{e_1\})$ and $\mathcal{P}(\{e_1\}, \{e_2\})$. The two sub-problems can be executed in parallel on a multi-core CPU platform to quickly search for the SRLG disjoint path pair.

Although together all the links along the AP form such a SRLG Conflicting Link Set, we are interested in a set with as few links as possible. This is because the number of sub-problems to partition in our divide-and-conquer algorithm (in section V-B) is determined by the size of the SRLG Conflicting Link Set. In Section VI-C, we will present our solution to find the minimum SRLG Conflicting Link Set.

B. Divided-and-Conquer

After we obtain the SRLG Conflicting Link Set, we design a divide-and-conquer algorithm to partition the original Min-Min SRLG-disjoint routing problem into multiple sub-problems which can be executed in parallel to speed up the progress of SRLG disjoint path pair finding.

To facilitate the problem partition, we first define two disjoint link sets $\mathbb{I}$ and $\mathbb{O}$ with $\mathbb{I} \cap \mathbb{O} = \emptyset$, where $\mathbb{I}$ is called the inclusion set and $\mathbb{O}$ is called the exclusion set. Denoted by $\mathcal{P}(\mathbb{I}, \mathbb{O})$ the sub-problem (of the Min-Min SRLG-disjoint Problem) for finding a pair of AP and BP, where the AP is the shortest among all possible APs that **must** use the links in $\mathbb{I}$ but **not** use the links in $\mathbb{O}$.

Originally, let $\mathbb{I} = \emptyset$ and $\mathbb{O} = \emptyset$, the original Min-Min SRLG-Disjoint Routing Problem can be represented by $\mathcal{P}(\emptyset, \emptyset)$. Note that, for any given link, the solution to $\mathcal{P}(\emptyset, \emptyset)$, if it exists at all, will use an AP that either contains the link or not. Given the SRLG Conflicting Link Set $\mathbb{T}$ with $|\mathbb{T}|$ links denoted by $e_1, e_2, \cdots, e_{|\mathbb{T}|}$, the original problem can be partitioned sequentially as follows.

Step 1, $\mathcal{P}(\emptyset, \emptyset)$ can be divided into two sub-problems $\mathcal{P}(\emptyset, \{e_1\})$ and $\mathcal{P}(\{e_1\}, \emptyset)$.

Step 2, similarly, $\mathcal{P}(\{e_1\}, \emptyset)$ can be further divided into two sub-problems $\mathcal{P}(\{e_1, e_2\}, \emptyset)$ and $\mathcal{P}(\{e_1\}, \{e_2\})$.

The partition process continues until in the step $|\mathbb{T}|$, we have that $\mathcal{P}(\{e_1, e_2, \cdots, e_{|\mathbb{T}|-1}\}, \emptyset)$ can be further divided into two sub-problems $\mathcal{P}(\{e_1, e_2, \cdots, e_{|\mathbb{T}|-1}, e_{|\mathbb{T}|}\}, \emptyset)$ and $\mathcal{P}(\{e_1, e_2, \cdots, e_{|\mathbb{T}|-1}\}, e_{|\mathbb{T}|})$. Note that, as the sub-problem $\mathcal{P}(\{e_1, e_2, \cdots, e_{|\mathbb{T}|-1}, e_{|\mathbb{T}|}\}, \emptyset)$ has $\mathbb{I} = \{e_1, e_2, \cdots, e_{|\mathbb{T}|-1}, e_{|\mathbb{T}|}\} = \mathbb{T}$ and $\mathbb{O} = \emptyset$, it has no solution.

We will try to find an optimal solution for each sub-problem except the sub-problem $\mathcal{P}(\{e_1, e_2, \cdots, e_{|\mathbb{T}|}\}, \emptyset)$, and then select the best one (i.e., the path pair with the shortest AP) to be the final (optimal) solution to the original problem $\mathcal{P}(\emptyset, \emptyset)$. If there is no solution to any of these sub-problems, we can guarantee that there is no solution to the original problem, as the way we divide the problem has included all possible paths.

In terms of the complexity, it should take less time to solve each sub-problem than the original problem itself as at least one link (which is from $\mathbb{T}$) will be removed from any further path computation for an AP, which also ensures that a different AP will be found and tested for the existence of a SRLG-disjoint BP.

When encountering the trap problem, our solution partitions the original problem and tests each sub-problem to look for the final solution. In our divide-and-conquer solution, the sub-

$$\mathcal{P}(\emptyset,\emptyset)\begin{cases}\mathcal{P}(\{e_2\},\emptyset)\begin{cases}\mathcal{P}(\{e_2,e_5\},\emptyset)\begin{cases}\mathcal{P}(\{e_2,e_5,e_6\},\emptyset)\\ \boxed{\mathcal{P}(\{e_2,e_5\},\{e_6\})}\end{cases}\\ \boxed{\mathcal{P}(\{e_2\},\{e_5\})}\end{cases}\\ \boxed{\mathcal{P}(\emptyset,\{e_2\})}\end{cases}$$

Fig. 5. Example to illustrate divide-and-conquer solution

problem is tested based on the SRLG Conflicting Link Set found from the AP encountering the trap problem. Compared to existing algorithms which search for alternative AP paths without considering the existing results and problems, our algorithm can largely reduce the computation cost. For the example in Fig.5, the SRLG Conflicting Link Set is $\mathbb{T} = \{e_2, e_5, e_6\}$. The partition process can be shown in Fig.5. According to the SRLG Conflicting Link Set, we should try the total 3 marked sub-problems $\mathcal{P}(\{e_2, e_5\}, \{e_6\})$, $\mathcal{P}(\{e_2\}, \{e_5\})$ and $\mathcal{P}(\emptyset, \{e_2\})$, and among which select the best one with the lowest AP path weight to be the final (optimal) solution to the original problem $\mathcal{P}(\emptyset, \emptyset)$.

Note that we do not need to solve the sub-problems $\mathcal{P}(\{e_2, e_5\}, \emptyset)$ and $\mathcal{P}(\{e_2\}, \emptyset)$, as their solutions have already been included in the sub-problems above. The solution space of the first one consists of two sub-problems $\mathcal{P}(\{e_2, e_5, e_6\}, \emptyset)$ and $\mathcal{P}(\{e_2, e_5\}, \{e_6\})$. As the SRLG Conflicting Link Set is $\mathbb{T} = \{e_2, e_5, e_6\}$, obviously, the sub-problem $\mathcal{P}(\{e_2, e_5, e_6\}, \emptyset)$ does not have a solution. Thus the solution space of $\mathcal{P}(\{e_2, e_5\}, \{e_6\})$ is equal to that of $\mathcal{P}(\{e_2, e_5\}, \emptyset)$. Similarly, the solution space of $\mathcal{P}(\{e_2\}, \emptyset)$ includes that of $\mathcal{P}(\{e_2\}\{e_5\}, \emptyset)$ and $\mathcal{P}(\{e_2\}, \{e_5\})$.

When a trap problem happens, the number of sub-problems to test is $|\mathbb{T}|$, which is equal to the size of the SRLG Conflicting Link Set. To reduce the complexity, in Section VI-C, we focus on finding the minimum SRLG Conflicting Link Set given an AP.

VI. Find SRLG Conflicting Link Set

In this section, we describe how to find a SRLG Conflicting Link Set for a given AP for which there is no SRLG-disjoint BP in a network G.

A. Build a new graph with a novel capability setting principle

As introduced in Section III, if all the links in the cut-set $\mathbb{L}_\Phi$ are removed, then $|f| = 0$. That is, no flow can pass from s to d. In this paper, we try to find the SRLG Conflicting Link Set based on the concept of the cut-set. If an AP flow passes from s to d through a path that shares the risk with all links in the cut-set, then no SRLG-disjoint BP can be found, as no link in the cut-set can be selected for a BP path to pass through the cut.

To facilitate finding the SRLG Conflicting Link Set based on the cut-set, we construct a new graph G^* as follows.

1) G^* uses the same $\mathbb{V}$ and $\mathbb{E}$ of G.

2) The link weight w_{e_i} associated with each link e_i is the same as the corresponding link weight of G.

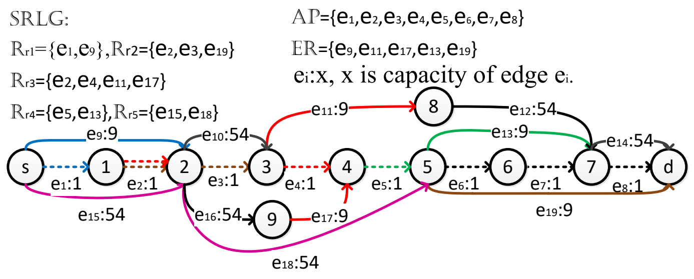

SRLG:
$\mathbb{R}_{r1}=\{e_1,e_9\}$,$\mathbb{R}_{r2}=\{e_2,e_3,e_{19}\}$
$\mathbb{R}_{r3}=\{e_2,e_4,e_{11},e_{17}\}$
$\mathbb{R}_{r4}=\{e_5,e_{13}\}$,$\mathbb{R}_{r5}=\{e_{15},e_{18}\}$
$\mathbb{AP}=\{e_1,e_2,e_3,e_4,e_5,e_6,e_7,e_8\}$
$\mathbb{ER}=\{e_9,e_{11},e_{17},e_{13},e_{19}\}$
e_i:x, x is capacity of edge e_i.
s
1
2
3
4
5
6
7
8
9
d
e_1:1
e_2:1
e_3:1
e_4:1
e_5:1
e_6:1
e_7:1
e_8:1
e_9:9
e_{10}:54
e_{11}:9
e_{12}:54
e_{13}:9
e_{14}:54
e_{15}:54
e_{16}:54
e_{17}:9
e_{18}:54
e_{19}:9

Fig. 6. New Graph G^*

3) We adopt the following principle to set the capacity c_{e_i} associated with each link $e_i \in \mathbb{E}$:

$$c_{e_i} = \begin{cases} 1 & e_i \in \mathbb{AP} \\ |\mathbb{AP}| + 1 & e_i \in \mathbb{ER} \\ |\mathbb{AP}| + (|\mathbb{AP}| + 1) \times |\mathbb{ER}| + 1 & otherwise \end{cases} \qquad (3)$$

where $\mathbb{AP}$ contains the set of links forming the smallest weight path AP in the graph G, and $\mathbb{ER}$ denotes the set of links not belonging to AP that share the common risks with AP.

In Fig.3(c), the link set of AP is $\mathbb{AP} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$, $\mathbb{ER} = \{e_9, e_{11}, e_{17}, e_{13}, e_{19}\}$. We have $|\mathbb{AP}| = 8$, $|\mathbb{ER}| = 5$, $|\mathbb{AP}| + 1 = 9$ and $|\mathbb{AP}| + (|\mathbb{AP}| + 1) \times |\mathbb{ER}| + 1 = 54$. We generate a new graph in Fig.6, with the capacity of the links in the graph set according to the principle in Equ.(3).

In Section VI-B, we will demonstrate that when an AP encounters the trap problem, such principles make the min cut-set in the new graph belonging either to $\mathbb{AP}$ or $\mathbb{ER}$, which further provides the possibility of obtaining the SRLG Conflicting Link Set with the links on the AP.

B. Properties on the min-cut in the new graph G^*

The max-flow min-cut theorem states that in a network, the maximum amount of flow passing from the source s to the destination d is equal to the total capacity of the links in the minimum cut, i.e. the smallest total capacity of the links which if removed would disconnect the destination d from the source s. Our algorithm will find the minimum SRLG Conflicting Link Set based on the min-cut of the graph.

We will first show that some good properties of our rebuilt graph can serve as a base to find the minimum SRLG Conflicting Link Set.

Let Φ=($\mathbb{S}$, $\mathbb{D}$) be a min-cut of G^* and $\mathbb{L}_\Phi$ represent the **cut-set**. Fig.7 shows the min-cut Φ=($\mathbb{S}$, $\mathbb{D}$) with $\mathbb{S} = \{s, 1, 2, 3, 4, 5, 9\}$, $\mathbb{D} = \{6, 7, 8, d\}$, and $\mathbb{L}_\Phi = \{e_{11}, e_{13}, e_{19}, e_6\}$.

Lemma 1. *Any path from s to d in G^* must pass through at least one link in $\mathbb{AP}$ or $\mathbb{ER}$.*

Proof. We prove the lemma by the way of contradiction. Assume that for the AP, there exists another path from s to d in G^* that does not share the risks with AP, that is, the path does not pass through the links in $\mathbb{AP}$ or $\mathbb{ER}$. We can easily conclude that this path is the SRLG-disjoint BP for the

AP, which contradicts with the claim that there is no SRLG-disjoint BP for the AP in the network. □

Lemma 2. *The value of any max flow of G^* is at most $|\mathbb{AP}|+(|\mathbb{AP}|+1)\times|\mathbb{ER}|$.*

Proof. Assume the value of a max flow f of G^* is $|f|=k$. f can be partitioned into k 1-unit-flows from s to d in G^*. According to Lemma.1, each of these 1-unit-flows must pass through at least one link in $\mathbb{AP}$ or $\mathbb{ER}$. Note that the capacity of the links in $\mathbb{AP}$ or $\mathbb{ER}$ are in 1 and $|\mathbb{AP}|$+1, respectively. According to the capacity set for the links in $\mathbb{AP}$ and $\mathbb{ER}$, a link in $\mathbb{AP}$ can carry only one unit-flow traffic, while the link in $\mathbb{ER}$ can be shared by at most $|\mathbb{AP}|$+1 unit-flows. Therefore, there can be at most $|\mathbb{AP}|+(|\mathbb{AP}|+1)\times|\mathbb{ER}|$ such unit-flows. □

Lemma 3. *All links in the* ***cut-set*** *$\mathbb{L}_\Phi$ of the min-cut Φ of G^* belong to $\mathbb{AP}$ or $\mathbb{ER}$.*

Proof. According to the max-flow min-cut theorem, the capacity of the min-cut Φ, denoted by $c(\Phi)$, should be equal to the max-flow value, which is at most $|\mathbb{AP}|+|\mathbb{ER}|\times(|\mathbb{AP}|+1)$ according to lemma.2. According to the capacity setting principle defined in Eq.(3), for the links that are neither in $\mathbb{AP}$ nor in $\mathbb{ER}$ (that is $e_i\notin\mathbb{AP}$ and $e_i\notin\mathbb{ER}$), their capacity is $c_{e_i}=|\mathbb{AP}|+(|\mathbb{AP}|+1)\times|\mathbb{ER}|+1$, which is larger than $|\mathbb{AP}|+(|\mathbb{AP}|+1)\times|\mathbb{ER}|$. Thus, none of them can be a link in the **cut-set** $\mathbb{L}_\Phi$. So all the links in the **cut-set** $\mathbb{L}_\Phi$ must belong to $\mathbb{AP}$ or $\mathbb{ER}$. □

In the network with SRLG, if a link is selected by an AP or shares the same risk with the links selected in the AP, this link can not be selected as the SRLG-disjoint BP of the AP. We call this link "blocked" by the AP.

Theorem 2. *If a path of a unit-flow of G blocks all the links in* ***cut-set*** *$\mathbb{L}_\Phi$, then no more flow can pass through the cut of the graph.*

Proof. If a path of a unit-flow blocks all the links in **cut-set** $\mathbb{L}_\Phi$, then no flow can use the links in $\mathbb{L}_\Phi$ and thus no more flow can pass the cut Φ. □

Theorem 2 provides us the possibility to find the SRLG Conflicting Link Set. That is, when an AP encounters a trap problem, we can find the sub-set of the $\mathbb{AP}$ that can block all the links in **cut-set** $\mathbb{L}_\Phi$, and this sub-set of links can form the SRLG Conflicting Link Set. When any a path contains all links in the SRLG Conflicting Link Set, no more flow can pass the cut Φ, thus no SRLG-disjoint BP can be found.

C. Set cover problem for SRLG Conflicting Link Set

Although together all links on the AP path form a SRLG Conflicting Link Set, we are interested in a set that has as few links as possible, as the size of the SRLG Conflicting Link Set determines the number of sub-problems partitioned according to Section V-B.

According to theorem 2, **the minimum SRLG Conflicting Link Set finding problem** can be described as: find the minimum subset of links on AP that can block all the links in the **cut-set** $\mathbb{L}_\Phi$.

For any link e_i, let $\mathbb{SR}_{e_i}$ denote the link set that shares risks with e_i. Obviously, $\mathbb{SR}_{e_i}$ covers e_i itself and all the links in the SRLG that includes e_i. For example in Fig.7, as e_2 is in two SRLGs $\mathbb{R}_{r_2}=\{e_2,e_3,e_{19}\}$, $\mathbb{R}_{r_3}=\{e_2,e_4,e_{11},e_{17}\}$, therefore, $\mathbb{SR}_{e_2}=\{e_2,e_3,e_{19},e_4,e_{11},e_{17}\}$.

For each link e_i on AP, we define its cut-block-link set as $\mathbb{B}_{e_i}=\mathbb{SR}_{e_i}\cap\mathbb{L}_\Phi$, which is the sub-set of **cut-set** $\mathbb{L}_\Phi$ that can be blocked by e_i.

Therefore, the minimum SRLG Conflicting Link Set finding problem can be formulated as a Set Cover Problem: given $\mathbb{AP}$ (the link set of path links of AP), the **cut-set** $\mathbb{L}_\Phi$, and a collection of the cut-block-link sets $\mathbb{B}_{e_1},\mathbb{B}_{e_2},\cdots,\mathbb{B}_{e_{|\mathbb{AP}|}}$, we want to find the fewest sets whose union is $\mathbb{L}_\Phi$, that is, the smallest $\mathbb{T}\subseteq\{e_i|e_i\in\mathbb{AP}\}$ such that $\cup_{e_i\in\mathbb{T}}\mathbb{B}_{e_i}=\mathbb{L}_\Phi$.

The Set Cover Problem is usually an NP-hard problem with its complexity depending on the size of the elements (denoted by n). In our minimum SRLG Conflicting Link Set finding problem, $n=|\mathbb{L}_\Phi|$, that is, the edge number of cut set $\mathbb{L}_\Phi$. As this paper's focus is not to improve the solution of the set cover problem, we apply the algorithm proposed in [36] with the complexity $O(log(|\mathbb{L}_\Phi|))$ to find the minimum SRLG Conflicting Link Set. Usually, even for a large scale network, the AP with the shortest path does not have a large number of $n=|\mathbb{L}_\Phi|$. Therefore, the cost to calculate the minimum SRLG Conflicting Link Set through the set cover is not large.

To illustrate how to find the minimum SRLG Conflicting Link Set, we show an example in Fig.7. For the min-cut $\Phi(\mathbb{S},\mathbb{D})$, $\mathbb{S}=\{s,1,2,3,4,5,9\}$ and $\mathbb{D}=\{d,6,7,8\}$, the **cut-set** is $\mathbb{L}_\Phi=\{e_{11},e_{13},e_{19},e_6\}$. For links in $\mathbb{AP}$, cut-block-link sets are: $\mathbb{B}_{e_1}=\emptyset$, $\mathbb{B}_{e_2}=\{e_{11},e_{19}\}$, $\mathbb{B}_{e_3}=\{e_{19}\}$, $\mathbb{B}_{e_4}=\{e_{11}\}$, $\mathbb{B}_{e_5}=\{e_{13}\}$, $\mathbb{B}_{e_6}=\{e_6\}$, $\mathbb{B}_{e_7}=\emptyset$ and $\mathbb{B}_{e_8}=\emptyset$. To cover $\mathbb{L}_\Phi$, the fewest cut-block-link sets are $\{\mathbb{B}_{e_2}$, $\mathbb{B}_{e_5}$, $\mathbb{B}_{e_6}\}$. Therefore, the minimum SRLG Conflicting Link Set is $\mathbb{T}=\{e_2,e_5,e_6\}$.

In the example of Fig.7, although $|\mathbb{L}_\Phi|=4$, the size of the minimum SRLG Conflicting Link Set $|\mathbb{T}|=|\{e_2,e_5,e_6\}|=3$ is even smaller than $|\mathbb{L}_\Phi|=4$. This is because e_2 belongs to SRLGs $\mathbb{R}_{r_2}$ and $\mathbb{R}_{r_3}$, and can block two links in e_{11} and e_{19} in the **cut-set** $\mathbb{L}_\Phi$.

According to the graph style of SRLG, there are two categories [29]: star-style and none star-style. For star-style SRLG, all the links start from the same node or ends at the same node. For example, in Fig.7, e_1 and e_9 start from the the same node s, $\mathbb{R}_{r_1}$ is a star-style SRLG. For none star-style SRLG, not all links in the SRLG starting from the same node or ending at the same node. In Fig.7, $\mathbb{R}_{r_2}$, $\mathbb{R}_{r_3}$, $\mathbb{R}_{r_4}$ and $\mathbb{R}_{r_5}$ are none star-style SRLGs. Moreover, even though Fig.7 includes both star-style SRLG and none star-style SRLG, our algorithm works efficiently and effectively to find the conflicting set through the solving of the set cover problem.

Therefore, different from some existing studies [29] which can only handle a single SRLG style or a simple scenario such as one link belonging to only one SRLG, our algorithm works

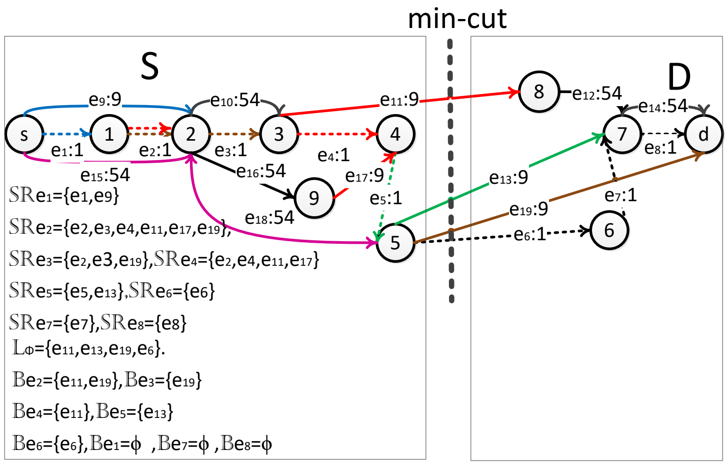


Fig. 7. Min cut of the network graph, $\mathbb{SR}_{e_1} = \{e_1, e_9\}, \mathbb{SR}_{e_2} = \{e_2, e_3, e_4, e_{11}, e_{17}, e_{19}\}, \mathbb{SR}_{e_3} = \{e_2, e_3, e_{19}\}, \mathbb{SR}_{e_4} = \{e_2, e_4, e_{11}, e_{17}\}, \mathbb{SR}_{e_5} = \{e_5, e_{13}\}, \mathbb{SR}_{e_6} = \{e_6\}, \mathbb{SR}_{e_7} = \{e_7\}$ and $\mathbb{SR}_{e_8} = \{e_8\}$. $\mathbb{L}_\Phi = \{e_{11}, e_{13}, e_{19}, e_6\}$. $\mathbb{B}_{e_1} = \emptyset, \mathbb{B}_{e_2} = \{e_{11}, e_{19}\}, \mathbb{B}_{e_3} = \{e_{19}\}, \mathbb{B}_{e_4} = \{e_{11}\}, \mathbb{B}_{e_5} = \{e_{13}\}, \mathbb{B}_{e_6} = \{e_6\}$,$\mathbb{B}_{e_7} = \emptyset$ and $\mathbb{B}_{e_8} = \emptyset$.

effectively in a more general scenario that a link can belong to one or multiple SRLGs with more diverse SRLG styles.

VII. The complete min-min SRLG disjoint routing Algorithm

In this section, we first present our complete solution, then analyze its complexity.

A. Complete solution

Algorithm 1 shows the complete min-min SRLG disjoint routing Algorithm. The input parameter in the algorithm includes the network graph (G), the source (s), the destination (d), the inclusion link set should be included in AP ($\mathbb{I}$), and the exclusion link set should not be included in AP ($\mathbb{O}$). The output of the algorithm is the SRLG disjoint path pair (AP, BP).

To look for the SRLG disjoint paths, the smallest weight AP in the network is searched first through FIND_AP$(G, s, d, \mathbb{I}, \mathbb{O})$ in step 2 with $\mathbb{I} = \phi, \mathbb{O} = \phi$, and then BP is searched through FIND_SRLG_Disjoint_BP (G, s, d, AP) in step 4. Specially, the AP path can be found through the Dijkstra algorithm. To calculate BP, FIND_SRLG_Disjoint_BP (G, s, d, AP) includes two steps. First, for all the links on the AP, remove the links that share a common risk with these links. Second, the Dijkstra's algorithm runs over the remaining links of the network again to compute the second shortest path BP from s to d.

If we can find a SRLG-Disjoint BP, the Min-Min SRLG disjoint routing problem is solved and the path pair found is returned as shown in step 6. Otherwise, a trap problem happens. To handle the trap problem, step 8 first finds the SRLG Conflicting Link Set $\mathbb{T}$, step 10 further divides the original Min-Min SRLG disjoint routing problem into $|\mathbb{T}|$ sub-problems based on the conflict set $\mathbb{T}$. All the sub-problems can be executed in parallel. Step 11 utilizes the set $\mathbb{F}$ to store the feasible solutions satisfying that both $AP_i \neq \phi$ and $BP_i \neq \phi$. Among all the feasible solutions in the $\mathbb{F}$, the path pair with the lowest AP path weight will be selected as the optimal solution for the original Min-Min SRLG disjoint routing problem.

We take the graph in Fig.3 as an example to illustrate our Algorithm 1. According to step 2, our algorithm first searches for a path $\mathbb{AP} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$ with the smallest weight through the dijkstra algorithm, with the path shown as the dotted line in Fig.3(c). After removing the links on AP and also the links that share the common risks with AP, we have the graph in Fig.3(d), which is a disconnected graph and no BP can be found. However, as shown in Fig.3(b), there exists a SRLG-disjoint path pair in the topology. Therefore, a trap problem happens. After finding the SRLG conflict link set $\{e_2, e_5, e_6\}$ in step 8, we apply divide and conquer to divide the original problem $\mathcal{P}(\emptyset, \emptyset)$ into three sub-problems, $\mathcal{P}(\emptyset, \{e_2\})$, $\mathcal{P}(\{e_2\}, \{e_5\})$ and $\mathcal{P}(\{e_2, e_5\}, \{e_6\})$ according to step 10. After executing these sub-problems in parallel, the path pair with the minimum AP weight is returned.

B. Complexity analysis

To find a SRLG disjoint path pair when an AP encounters a trap problem, our algorithm first calculates the SRLG Conflicting Link Set, and then solves the original problem by partitioning it into $|\mathbb{T}|$ sub-problems. As introduced in Section VI-C, the edge number of cut set $\mathbb{L}_\Phi$ is usually not large, therefore, finding the SRLG Conflicting Link Set in our paper does not introduce much cost. Therefore, we focus on the computation cost on the path finding process.

Generally, for a network with $|\mathbb{E}|$ links and $|\mathbb{V}|$ nodes, the complexity of finding the least weight path is $(|\mathbb{E}| + |\mathbb{V}|) \times log(|\mathbb{V}|)$. To solve the trap problem, our path finding problem is a little bit different from the original least weight path finding problem. We introduce some constraints for path finding, for example, an AP path must pass through a link set or cannot pass a set. As these link sets are usually not large, the constraints bring little difference in the cost calculation. As

Algorithm 1 Min-Min

```
Require: $G$: the network graph
    $s$: the source node
    $d$: the destination node
    $\mathbb{I}$: the inclusion link set should be included in AP
    $\mathbb{O}$: the exclusion link set should not be included in AP
Ensure: AP: the active path
    BP: the backup path
 1: $AP = \emptyset$, $BP = \emptyset, \mathbb{I} = \emptyset, \mathbb{O} = \emptyset$
 2: $AP \leftarrow$ FIND_AP($G, s, d, \mathbb{I}, \mathbb{O}$)
 3: if $AP \neq \emptyset$ then
 4:     return $BP \leftarrow$ FIND_SRLG_Disjoint_BP($G, s, d, AP$)
 5:     if $BP \neq \emptyset$ then
 6:         return path pair $(AP, BP)$
 7:     else
 8:         find SRLG Conflicting Link Set $\mathbb{T}$
 9:         $\mathbb{T} \leftarrow \mathbb{T} - (\mathbb{I} \cup \mathbb{O})$
10:         divide and conquer for execution in parallel
            $\left\{\begin{array}{l} (AP_1, BP_1) = Min - Min\,(G, s, d, \mathbb{I}, \mathbb{O} \cup \{t_1\}), \\ (AP_2, BP_2) = Min - Min\,(G, s, d, \mathbb{I} \cup \{t_1\}, \mathbb{O} \cup \{t_2\}), \\ (AP_3, BP_3) = Min - Min\,(G, s, d, \mathbb{I} \cup \{t_1, t_2\}, \mathbb{O} \cup \{t_3\}), \\ \cdots \\ \left(AP_{|\mathbb{T}|}, BP_{|\mathbb{T}|}\right) = Min - Min\left(G, s, d, \mathbb{I} \cup \{t_1, t_2, \cdots, t_{|\mathbb{T}|-1}\}, \mathbb{O} \cup \{t_{|\mathbb{T}|}\}\right) \end{array}\right.$
11:         $F \leftarrow$ FIND_FEASIBLE($(AP_1, BP_1)), \cdots, (AP_{|\mathbb{T}|}, BP_{|\mathbb{T}|})$
12:         if $F \neq \emptyset, \emptyset$ then
13:             return path pair $(AP, BP)$ satisfying that $AP = \arg\min_{AP}\{F\}$
14:         end if
15:     end if
16: end if
```

different sub-problems have different link sets thus different complexity, to make the description simple and clear, we still use $(|\mathbb{E}|+|\mathbb{V}|)\times log(|\mathbb{V}|)$ as one time of path searching. As our algorithm divides the original problem into $|\mathbb{T}|$ sub-problems, the complexity of our algorithm is $|\mathbb{T}| \times (|\mathbb{E}|+|\mathbb{V}|)\times log(|\mathbb{V}|)$.

For complexity comparison, we also show the complexity of path finding process in CoSE [28] and KSP [23].

CoSE tries to find a conflicting SRLG set instead of a Conflicting Link Set as we do. Although their search for conflicting SRLG set is exhaustive with a high cost, in this paper, we focus on the cost analysis of path finding process and do not take into account this cost in the comparison. As our SRLG Conflicting Link Set is derived from min-cut and the set cover problem, the $|\mathbb{T}|$ is the minimum size of SRLG Conflicting Link Set. Therefore, the output of the conflicting SRLG set in CoSE is at least $|\mathbb{T}|$, and we denote the SRLG set as $\{SRLG_1, SRLG_2, \cdots, SRLG_{|\mathbb{T}|}\}$. As each SRLG path includes multiple links, the sub-problems to be partitioned should be much larger than ours. In the process of the problem partition for an SRLG including $|SRLG|$ links, the inclusion or exclusion of an SRLG link in an AP would create $|SRLG|$ sub-problems. Thus a S-RLG set $\{SRLG_1, SRLG_2, \cdots, SRLG_{|\mathbb{T}|}\}$ will introduce $\prod_{i=1}^{|T|}|SRLG_i|$ sub-problems in CoSE, as a link combination (with each link extracted from a SRLG to get it out of the trap) corresponds a sub-problem. Therefore, the complexity of CoSE is $\prod_{i=1}^{|\mathbb{T}|}|SRLG_i| \times (|\mathbb{E}|+|\mathbb{V}|)\times log(|\mathbb{V}|)$, which is much larger than ours.

For KSP [23], the path finding complexity is $K \times ((|\mathbb{E}| + |\mathbb{V}|) \times log(|\mathbb{V}|))$, where K is the number of first K shortest paths that should be tested before finding the SRLG disjoint BP. However, as KSP does not borrow any information from the previous path searching process, in the worst case, KSP should try all the paths from the source s to the destination d. Therefore, the worst K would be $2^{|\mathbb{E}|}$, which brings very large computation cost.

Therefore, compared with CoSE [28] and KSP [23], our algorithm exploits the min-cut theory to reduce the number of paths searched thus having the smallest computation cost. In Section VIII-B3, we will further provide extensive simulations to demonstrate that our algorithm can achieve very high computation speed to find the SRLG disjoint paths using the topology trace data.

VIII. PERFORMANCE EVALUATION

We first describe the simulation setup, then present the simulation results.

A. Simulation setup

As we could not find any topology trace data that contain SRLG links, we generate a synthesized data set by injecting SRLG into a Huawei topology trace published. Huawei topology trace has 7 different topologies with various number of nodes, links, link weights (represent link's delay or other parameter). Table I shows the basic network setting (number of nodes, links) in these 7 topologies. Based on huawei's topology trace, we randomly generate SRLG groups, with the number of SRLG graphs injected and the SRLG edge ratio (defined as $\frac{No.SRLG\ edges}{Total\ No.edges}$) shown on the 3rd and 4th rows of the table.

For performance comparisons, besides our scheme (named by SCLS), we also implemented other four SRLG-disjoint routing algorithms as follows:

1) ILP: The work in [19] aims to find SRLG-disjoint paths through an ILP formulation such that the total weight of the two paths are minimized. We are not aware that other studies look for the Min-Min SRLG-disjoint paths through ILP formulation. Therefore, following [19], we formulate our Min-Min SRLG-disjoint routing problem by changing the objective function.
2) IQCP: As any $0-1$ integer linear program where all variables are either 0 or 1, ILP can be formulated as a quadratically constrained program. Different from ILP, we also model Min-Min SRLG-disjoint routing problem through the Integer Quadratic Constraints Program (IQCP) [19].
3) KSP [23]: It finds the first K shortest paths between the source and the destination as candidate APs, and then tests them one by one in the increasing order of their costs to see if it has a corresponding (disjoint) BP, until such a BP is found.
4) CoSE [28]: When an AP encounters a trap problem, CoSE tries a simple and exhaustive search to find a SRLG set that no AP going through the SRLG set can find

TABLE I
7 DIFFERENT TOPOLOGIES.

Topology	1	2	3	4	5	6	7
Node	527	521	521	2023	443	519	449
Edge	4158	4052	4152	4142	2772	4050	2778
No.SRLG	133	86	89	121	34	128	88
SRLG edge ratio	15.87%	10.13%	16.67%	16.17%	8.79%	13.67%	7.12%

the SRLG-disjoint BP path. Based on the SRLG set, it partitions the original problem and designs an algorithm to search for the SRLG disjoint path pair.

The first two (ILP and IQCP) are based on integer program models. In our implementation, the tool GUROBI 7.0 [37] is employed to resolve these two integer problems. Six performance metrics are applied to evaluate the performance of different SRLG-disjoint routing algorithms:

Path weight: is the sum of the link weight in the path.

Path hop: the number of hops in the path.

Runtime: the average number of milliseconds taken for SRLG-disjoint path finding.

Algorithm speedup: Given the computation time under two different algorithms (alg_1 and alg_2), denoted as T_1 and T_2, the algorithm speedup in the computation time of the alg_2 with respect to the alg_1: $S_{1-2} = T_1/T_2$.

Core speedup: The core speedup [38] of a parallel program is typically defined as $S_P = \frac{T_1}{T_p}$, where p is the number of processor cores and T_1 and T_p denote the running time on 1 core and p cores, respectively.

Efficiency: is defined [38] as $E_p = \frac{S_p}{p} = \frac{T_1}{pT_p}$, it is typically reported as a percentage in the range (0, 1].

All simulations are run on a linux server, which is equipped with Intel(R) Xeon(R) CPU E5-2620 0 2.00GHz (24 Cores) and 32.00GB RAM. To measure the computation time, we insert a timer to all the implemented algorithms.

All algorithms are implemented using 7 topologies. After normalizing the results from each topology, we take the average of the normalized results as the final simulation results according to min-max normalization [39]. For the readers who are interested in this work, the source codes of our algorithms and the synthesized topology trace sets can be downloaded from the github website [40].

B. Performance comparison

From the analysis in Section VII-B, under KSP, the computation complexity to find the first K shortest paths is $K \times ((|\mathbb{E}| + |\mathbb{V}|) \times log(|\mathbb{V}|))$, which would be large with the worst $K = 2^{|\mathbb{E}|}$. Consistent with the analysis, when we run KSP using the 7 topologies, no simulation results can be returned within 1 hour, while others can return results within 11 seconds. Long computation time makes KSP difficult to use in practice. Therefore, we do not provide the simulation results under KSP.

1) Path Weight: Fig.8 shows the weight of AP, BP, and the sum of both AP and BP. Obviously, all implemented algorithms SCLS, CoSE, ILP and IQCP achieve the same AP

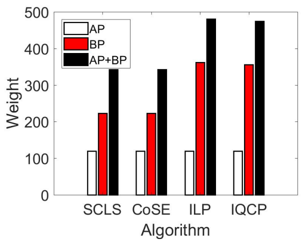


Fig. 8. Path weight

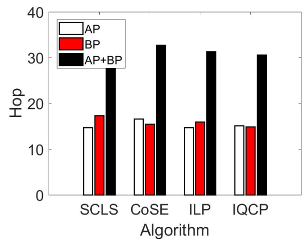


Fig. 9. Path hop

weight, but different BP weights thus different sum weights of AP and BP. As all the algorithms solve the same Min-Min SRLG-disjoint routing program, although they find different SRLG-disjoint path pairs, they can all achieve the goal of finding the AP with the same least weight. As SCLS, CoSE find the BP based on a shortest path algorithm (such as Dijkstra), they also find the BP path with the same weight. However, the two ILP-based algorithms, ILP and IQCP, focus on minimizing the weight of AP but find any BP that is SRLG-disjoint with AP, the BP paths searched by these two algorithms are different.

2) Path Hop: Fig.9 shows the path hop of AP, BP, and the sum of both AP and BP. As all algorithms target to minimize the least path weight of the SRLG-disjoint path pair instead of the number of path hops, they have the same AP weight (Fig.8) even though they have different AP path hops (Fig.9). Although the AP weights under all algorithms are smaller than the BP weights in Fig.8, in Fig.9, the AP hops may not always be fewer than the BP hops.

3) Runtime: Fig.10 shows the run time under different algorithms by varying the number of CPU cores utilized. As the runtime under CoSE is significantly larger than that under other algorithms, to more clearly show the results of other algorithms, we further plot the runtime results in Fig.11 by excluding CoSE. As ILP, and IQCP are not parallel algorithms, the runtime of these algorithms under different number of cores is approximately equal. The runtime of our SCLS and CoSE decreases with the increase of the number of processor cores because these two algorithms can partition the original problem into multiple sub-problems to execute in parallel and take advantage of the parallelism of the multi-core CPU to speed up the path searching process. Although CoSE is a parallel algorithm, the computation time is even larger than ILP and IQCP. Some possible reasons include 1) the search process to find the conflicting SRLG set in CoSE is not efficient; 2) As one SRLG usually includes multiple links,

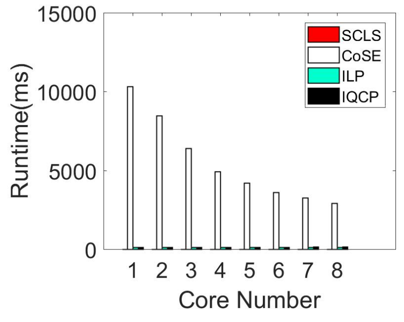


Fig. 10. Runtime

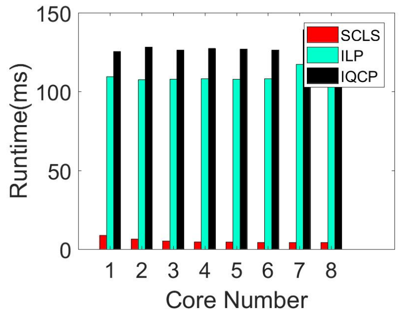


Fig. 11. Runtime without CoSE

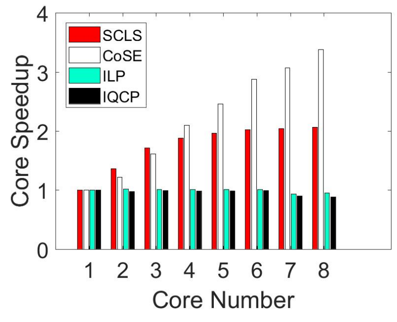


Fig. 12. Core speedup

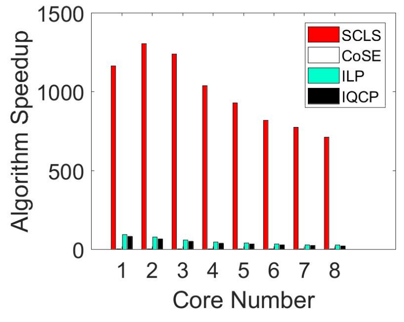


Fig. 13. Algorithm speedup

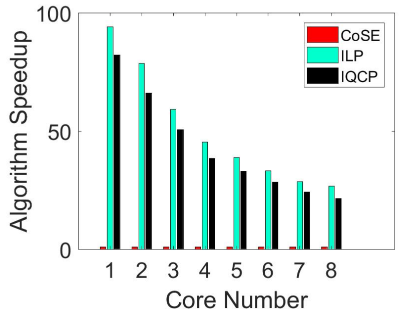


Fig. 14. Algorithm speedup without SCLS

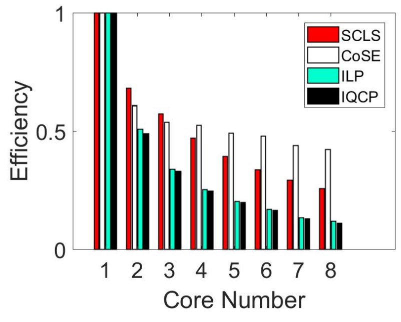


Fig. 15. Efficiency

the partitioning of problem based on conflicting SRLG will introduce a large number of sub-problems to solve, which also results in a large computation cost.

Different from CoSE, our SCLS looks for the set of conflicting links on an AP caught into the trap problem based on the min-cut theory in graph, and achieves the lowest time in Fig.10. This demonstrates that our conflicting link set finding algorithm is efficient, and moreover our divide-and-conquer algorithm and intelligent AP searching process based on SRLG Conflicting Link Set can largely reduce the computation cost.

4) Algorithm speedup: In Fig.13, we further compare their computation speeds. Specially, to find out how much speedup is gained when using different algorithms to find the required paths, we use CoSE as the baseline algorithm and set alg_1 =CoSE. Similar to the results in the Fig.10, the speed of SCLS is more than 700 times that of CoSE in Fig.13. Similar to Fig.10, as the running speed of CoSE is significantly smaller than others and can hardly be observed in Fig.13, we further plot the algorithm speedup results in Fig.14 by excluding the largest one SCLS.

5) Core speedup: Rather than using the algorithm speedup to compare the overall running speeds of all algorithms, the metric ”core speedup” is utilized to evaluate how the number of cores in the CPU impacts the running speed of a given algorithm. Fig.12 plots the core speedup under all algorithms implemented. Core speedup under algorithms ILP and IQCP is approximately equal to 1 under any core number because they are not parallel algorithms. The core speedup of our SCLS increases with increase of core number when it is less than 4, beyond which the core speedup of SCLS remains stable, which demonstrates that 4 core is sufficient for SCLS. This result is consistent with Amdahl's law[41] that the theoretical core speedup is limited to a upper bound determined by the problem size. However, the runtime under CoSE continues to increases even though the core number is equal to 8 which doubles the number 4. This result demonstrates that even 8-core CPU can not satisfy the parallelism requirement in CoSE. This is because the conflicting SRLG set found by CoSE includes a large number of links, which further results in a large number of sub-problems and thus large problem size and computation cost.

6) Efficiency: Similar to Fig.12, with more cores involved, the efficiency value decreases as a large core number brings more cost to coordinate the process. In Fig.15, the efficiency values of all algorithms decrease with the increase of the core number. Consistent with the results in Fig.12, as CoSE introduces more sub-problems than our SCLS, after the core number reaches 4, the efficiency under CoSE is larger than SCLS. However, our SCLS achieves significantly larger algorithm speedup as shown Fig.13.

All the simulation results demonstrate that our SCLS can outperform other approaches with higher routing performance while at a much higher search speed, because the conflicting link set found can facilitate efficient problem partition for parallel algorithm execution with low computation cost.

IX. Conclusion

In this paper, we propose an efficient algorithm to solve the Min-Min SRLG-Disjoint routing problem in the presence of the trap problem. To reduce the complexity of searching for the alternative pair, we propose a divide-and-conquer solution to partition the original Min-Min SRLG-Disjoint routing problem into multiple sub-problems based on a SRLG conflicting link set derived from the AP path encountering the trap problem. Our algorithm takes advantage of existing AP search results and parallel executions for significantly faster path finding. We have conducted extensive simulations using the topology trace from Huawei on a multi-core CPU platform. The simulation results demonstrate that our algorithm can outperform other approaches with higher routing performance while at a much higher search speed.

References

[1] J. Yallouz and A. Orda, "Tunable qos-aware network survivability," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 1, pp. 139–149, 2017.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[3] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[4] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, attributes, and use cases: A compass for sdn," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 210–217, 2014.

[5] L. F. Müller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspary, and M. P. Barcellos, "Survivor: an enhanced controller placement strategy for improving sdn survivability," in *Global Communications Conference (GLOBECOM), 2014 IEEE*, pp. 1909–1915, IEEE, 2014.

[6] V. Lopez, J. M. Gran, J. P. Fernandez-Palacios, D. Siracusa, F. Pederzolli, O. Gerstel, Y. Shikhmanter, J. Mårtensson, P. Sköldström, T. Szyrkowiec, *et al.*, "The role of sdn in application centric ip and optical networks," in *Networks and Communications (EuCNC), 2016 European Conference on*, pp. 138–142, IEEE, 2016.

[7] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takács, and P. Sköldström, "Scalable fault management for openflow," in *Communications (ICC), 2012 IEEE International Conference on*, pp. 6606–6610, IEEE, 2012.

[8] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "Openflow-based segment protection in ethernet networks," *Journal of Optical Communications and Networking*, vol. 5, no. 9, pp. 1066–1075, 2013.

[9] F. A. Kuipers, "An overview of algorithms for network survivability," *ISRN Communications and Networking*, vol. 2012, 2012.

[10] P. Sebos, J. Yates, G. Hjalmtysson, and A. Greenberg, "Auto-discovery of shared risk link groups," in *Optical Fiber Communication Conference and Exhibit, 2001. OFC 2001*, vol. 3, pp. WDD3–WDD3, IEEE, 2001.

[11] F. Zhang, O. G. de Dios, C. Margaria, M. Hartley, and Z. Ali, "Rsvp-te extensions for collecting shared risk link group (srlg) information," tech. rep., 2017.

[12] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, no. 2, pp. 325–336, 1984.

[13] R. Bhandari, "Optimal physical diversity algorithms and survivable networks," in *Computers and Communications, 1997. Proceedings., Second IEEE Symposium on*, pp. 433–441, IEEE, 1997.

[14] C.-L. Li, S. T. McCormick, and D. Simchi-Levi, "The complexity of finding two disjoint paths with min-max objective function," *Discrete Applied Mathematics*, vol. 26, no. 1, pp. 105–115, 1990.

[15] Y. Guo, F. Kuipers, and P. Van Mieghem, "Link-disjoint paths for reliable qos routing," *International Journal of Communication Systems*, vol. 16, no. 9, pp. 779–798, 2003.

[16] D. Xu, Y. Chen, Y. Xiong, C. Qiao, and X. He, "On finding disjoint paths in single and dual link cost networks," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 1, IEEE, 2004.

[17] A. Beshir and F. Kuipers, *Variants of the min-sum link-disjoint paths problem.* IEEE/SCVT, 2011.

[18] L. Guo and H. Shen, "On finding min-min disjoint paths," *Algorithmica*, vol. 66, no. 3, pp. 641–653, 2013.

[19] J. Q. Hu, "Diverse routing in optical mesh networks," *IEEE Transactions on Communications*, vol. 51, no. 3, pp. 489–494, 2003.

[20] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations research*, vol. 14, no. 4, pp. 699–719, 1966.

[21] E. Oki, N. Matsuura, K. Shiomoto, and N. Yamanaka, "A disjoint path selection scheme with shared risk link groups in gmpls networks," *IEEE Communications letters*, vol. 6, no. 9, pp. 406–408, 2002.

[22] G. Li, B. Doverspike, and C. Kalmanek, "Fiber span failure protection in mesh optical networks," *Optical Networks Magazine*, vol. 3, no. 3, pp. 21–31, 2002.

[23] D. Eppstein, "Finding the k shortest paths," *SIAM Journal on computing*, vol. 28, no. 2, pp. 652–673, 1998.

[24] D. A. Dunn, W. D. Grover, and M. H. MacGregor, "Comparison of k-shortest paths and maximum flow routing for network facility restoration," *IEEE Journal on selected areas in Communications*, vol. 12, no. 1, pp. 88–99, 1994.

[25] P. Laborczi, J. Tapolcai, P.-H. Ho, T. Cinkler, A. Recski, and H. T. Mouftah, "Solving asymmetrically weighted optimal or near-optimal disjoint path pair for the survivable optical networks," in *Third International Workshop On Design Of Reliable Communication Networks (DRCN01)*, 2001.

[26] L. R. Ford Jr and D. R. Fulkerson, *Flows in networks*. Princeton university press, 2015.

[27] M. J. Rostami, A. A. E. Zarandi, and S. M. Hoseininasab, "Msdp with aco: A maximal srlg disjoint routing algorithm based on ant colony optimization," *Journal of Network and Computer Applications*, vol. 35, no. 1, pp. 394–402, 2012.

[28] M. J. Rostami, S. Khorsandi, and A. A. Khodaparast, "Cose: A srlg-disjoint routing algorithm," in *Universal Multiservice Networks, 2007. ECUMN'07. Fourth European Conference on*, pp. 86–92, IEEE, 2007.

[29] P. Datta and A. K. Somani, "Graph transformation approaches for diverse routing in shared risk resource group (srrg) failures," *Computer Networks*, vol. 52, no. 12, pp. 2381–2394, 2008.

[30] D. Xu, Y. Xiong, and C. Qiao, "A new promise algorithm in networks with shared risk link groups," in *Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE*, vol. 5, pp. 2536–2540, IEEE, 2003.

[31] A. Todimala and B. Ramamurthy, "Imsh: An iterative heuristic for srlg diverse routing in wdm mesh networks," in *Computer Communications and Networks. Proceedings. 13th International Conference on*, pp. 199–204, IEEE, 2004.

[32] 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*, vol. 21, no. 11, p. 2683, 2003.

[33] R. Bhandari, "Optimal diverse routing in telecommunication fiber networks," in *INFOCOM'94. Networking for Global Communications., 13th Proceedings IEEE*, pp. 1498–1508, IEEE, 1994.

[34] D. Coudert, P. Datta, S. Pérennes, H. Rivano, and M.-E. Voge, "Shared risk resource group complexity and approximability issues," *Parallel Processing Letters*, vol. 17, no. 02, pp. 169–184, 2007.

[35] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[36] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.

[37] G. Optimization *et al.*, "Gurobi optimizer reference manual," *URL: http://www. gurobi. com*, vol. 2, pp. 1–3, 2012.

[38] A. Grama, *Introduction to parallel computing*. Pearson Education, 2003.

[39] D. Tax and R. Duin, "Feature scaling in support vector data descriptions," *Learning from Imbalanced Datasets*, pp. 25–30, 2000.

[40] SCLSAlgorithm, "github." https://github.com/SCLSAlgorithm/SCLSAlgorithm.

[41] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, pp. 483–485, ACM, 1967.