Finding Minimum-Weight Link-Disjoint Paths with a Few Common Nodes

Binglin Tao, Mingyu Xiao,* Jingyang Zhao
School of Computer Science and Engineering
University of Electronic Science and Technology of China
taobl@std.uestc.edu.cn, myxiao@gmail.com, 1176033045@qq.com

Abstract

Network survivability has drawn certain interest in network optimization. However, the demand for full protection of a network is usually too restrictive. To overcome the limitation of geographical environments and to save network resources, we turn to establish backup networks allowing a few common nodes. It comes out the problem of finding k link-disjoint paths between a given pair of source and sink in a network such that the number of common nodes shared by at least two paths is bounded by a constant and the total link weight of all paths is minimized under the above constraints. For the case $k = 2$, where we have only one backup path, several fast algorithms have been developed in the literature. For the case $k > 2$, little results are known. In this paper, we first establish the NP-hardness of the problem with general k. Motivated by the situation that each node in a network may have a capability of multicasting, we also study a restricted version with one more requirement that each node can be shared by at most two paths. For the restricted version, we build an ILP model and design a fast algorithm by using the techniques of augmenting paths and splitting nodes. Furthermore, experimental results on synthetic and real networks show that our algorithm is effective in practice.

Introduction

Dealing with network failures has become an urgent and important research topic due to the rapid development of network and the extensive establishment of network facilities in the past decades. In general, we can protect from network failures by increasing the degrees of internal nodes (Steiglitz, Weiner, and Kleitman 1969; Kerivin and Mahjoub 2005; Kuipers 2012), limiting average distances or hop counts between nodes (Gouveia, Simonetti, and Uchoa 2011), and increasing the connectivity (Cohen and Nutov 2013; Kortsarz and Nutov 2016) in the network.

One of the most effective and widely used methods to protect networks from failures is to establish a backup network (or path) for each primary network (or path) with some requirements like node degrees and connectivity (Yallouz, Rottenstreich, and Orda 2016; Yallouz and Orda 2017; Johnston, Lee, and Modiano 2015; Wei, Shen, and Bose 2014; Choudhury, Bhadra, and De 2019). It becomes a fundamental problem in network protection to pre-compute and reserve both primary and backup paths (or networks) between a specific pair of source and sink nodes in the network in advance during the configuration phase (Beshir and Kuipers 2009; Sherali, Özbay, and Subramanian 1998; Wei, Shen, and Bose 2014). Usually, the primary and backup paths need to meet some performance requirements, e.g. Quality of Service (such as disjointness, delay, cost, failure probability, and power level limits), which is typically quantified by the sum of some link weights. Nonetheless, a working path with some disjoint backup paths will not suffer from malfunction caused by only a single link failure, in reality (Wang and Doucette 2018).

The disjointness can be regarded as a constraint on nodes or links in the network. Although node-disjoint paths can protect from both node and link failures, link-disjoint paths are more widely used and studied in network protection. Recently, the concept of "partially disjoint paths" was proposed, in which the paths are link-disjoint but also have a constraint on the number of common nodes shared by at least two paths.

Hu et al. (1998) proposed a new type of switches, namely Splitter-and-Delivery Switch (SAD). It has both point-to-point and multicasting capability in networks to satisfy the demand of data transmission from a source to several destinations, which means the nodes in one source-destination path can be shared by other source-destination paths, saying *common nodes*. Common nodes can be considered as shared resources, such as power, spectrum, optical channel, buffer memory, etc. Yallouz et al. (2018) mentions that common nodes along an established link-disjoint pair of paths can save time to retransmit the data from the closest nodes instead of the source when failures occur. However, the number of common nodes usually can not be very large due to the hardware constraints in switches and the occurrence probability of node failures. Some models to save the spare resources by limiting the number of common nodes in the networks are also considered in (Le, Molnár, and Palaysi 2014; Din and Lai 2015). Therefore, the number of common nodes

*The corresponding author.
Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

in paths or networks is a critical criterion for both protecting from link and node failures.

In order to both protect networks from link and node failures and also save resources, in this paper, we consider the problem of finding k link-disjoint paths between a given pair of source and sink nodes of minimum total weight with a bounded number of common nodes. For $k = 2$, we have only one backup path. Fast polynomial algorithms for this case were developed in the literature (Yallouz et al. 2018; Guo et al. 2018). However, we may need more than one backup path or network (Wang and Doucette 2018; Johnston, Lee, and Modiano 2015). For the case $k \geq 3$, although it was mentioned in the literature, we are not aware of any nontrivial result. In this paper, we establish the computational complexity of this problem with general k. In detail, our contribution is summarized as follows.

The Contribution

We make the following primary contributions in this paper:

- We prove the NP-hardness of the problem of finding k link-disjoint shortest paths with at most δ common nodes, answering a question in previous papers (Guo et al. 2018). The NP-hardness is established by a reduction from the known NP-hard problem — the densest δ subgraph problem in bipartite graphs.
- We also study the problem with one more constraint: each node can be shared by at most two paths. This constraint is natural and useful to control node failures (Yallouz et al. 2018; Wang and Zhu 2018). We build an ILP formulation for this problem. Furthermore, inspired by the augmenting path technique for finding k link-disjoint shortest paths in (Busacker and Gowen 1960) and the splitting node technique, we design a fast algorithm with a desirable running time bound.
- Simulation study shows that the fast algorithm is effective in practice and also much faster than the ILP algorithm.

Related Work

The shortest path problem is one of the most famous and fundamental problems. The extension to find k node-disjoint paths between two nodes minimizing the total weight was first considered in (Suurballe 1974), where a polynomial algorithm was proposed. The link-disjoint version of the problem can be reduced to the node-disjoint version (Suurballe and Tarjan 1984). Several problems of finding k link-disjoint paths with different constraints were proposed. Due to load balance, Guo et al. (2013) considered the problem of finding two link-disjoint paths to minimize the cost of the shortest path, which was proved to be NP-hard. In the context of networks with a low occurrence of failure, Li et al. (1990) proposed an NP-hard problem that aims at finding a link-disjoint path pair such that the weight of the longest path is minimized. In order to prune low-capacity links, Shen et al. (2004) considered the problem of finding a widest link-disjoint path pair, which has the maximum bandwidth or capacity. The problem of finding several paths with a bounded number of nodes and links was firstly considered in (Seymour and Kar 2013), where the authors gave a heuristic algorithm in wireless sensor networks. Later, Yallouz et al. (2018) gave an $O(|E||V|^2 + |V|^3 \log |V|)$-time algorithm for finding a pair of link-disjoint paths with at most δ common nodes (minimizing the total weight) in a network $G = (V, E)$. The running time bound was further improved to $O(\delta|E| + |V| \log |V|)$ by Guo et al. (2018). This reference also inquired methods to solve the problem of finding more than two link-disjoint paths with bounded common nodes.

Problem Models

A *network* is represented by a directed graph $G(V, E)$, where V is the set of nodes and E is the set of links. We may use $\langle u, v\rangle$ to denote a link from node u to node v. Each link $e \in E$ is assigned with a positive weight $w(e) \in \mathbb{R}^+$ which represents an additive metric such as delay, distance, etc. We may also associate a cost for a link e: $c(e) \in \mathbb{R}$. A *path* in the network is an ordered set of different nodes $P = \langle s_0, s_1, ..., s_k\rangle$ such that $s_i \in V$ for $i \in [0, k]$ and $\langle s_j, s_{j+1}\rangle \in E$ for $j \in [0, k-1]$. A set of k paths are *link-disjoint* if no pair of paths have a common link. For a path or a set of paths P, we use $w(\mathrm{P})$ (resp., $c(\mathrm{P})$) to denote the total weight (resp., cost) of all links appearing in paths in P. Given a set P of paths from a source node s to a sink node t, a node different from s and t is called a *common node* if it appears in at least two paths in P and we use $C_o(\mathrm{P})$ to denote the set of common nodes in P.

Definition 1. (The minimum k link-disjoint paths with δ common nodes problem, (k, δ)-LDP). *Given a network $G = (V, E)$ with a positive weight on links $w : E \to \mathbb{R}^+$, two integers k and δ, a source node $s \in V$, and a sink node $t \in V$. The goal of (k, δ)-LDP is to find a set* P *of k link-disjoint paths from s to t such that $w(\mathrm{P})$ is minimized under the constraint $|C_o(\mathrm{P})| \leq \delta$.*

Considering that there are some memory and channel limitations on each node in the network, we also define a restricted version of (k, δ)-LDP, in which any node can be shared by at most two paths in P.

Definition 2. (Restricted (k, δ)-LDP). *Given a network $G = (V, E)$ with a positive weight on links $w : E \to \mathbb{R}^+$, two integers k and δ, a source node $s \in V$, and a sink node $t \in V$. The goal of Restricted (k, δ)-LDP is to find a set* P *of k link-disjoint paths from s to t such that $w(\mathrm{P})$ is minimized under the constraints that no node in $V \setminus \{s, t\}$ is contained in more than two paths in* P *and $|C_o(\mathrm{P})| \leq \delta$.*

NP-Hardness

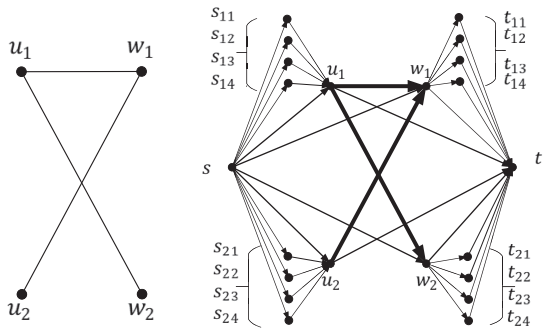
As mentioned in the introduction, $(2, \delta)$-LDP has been well studied and several fast algorithms have been proposed. However, for general k, the status of (k, δ)-LDP is not clear, although it was discussed in the literature to extend the existing algorithms for $(2, \delta)$-LDP to the problem with general k. Here, we show that (k, δ)-LDP is indeed NP-hard.

In fact, we will prove the NP-hardness of the decision version of checking whether there exist k link-disjoint paths sharing at most δ common nodes in an unweighted network. Our proof is based on a reduction from the known

NP-hard problem – *the densest δ subgraph problem in bipartite graphs*. The densest δ subgraph problem is defined as follows. Given a graph B, and two integers δ and β, the problem is to check whether the graph B has a subgraph of δ nodes containing at least β edges. The densest δ subgraph problem is a well known NP-hard problem (Feige, Kortsarz, and Peleg 2001) and it remains NP-hard even in bipartite graphs (Corneil and Perl 1984).

Given an instance $I = (B = (V_B, E_B), \delta, \beta)$ of the densest δ subgraph problem in bipartite graphs, we construct an instance I' of the decision version of (k, δ)-LDP. Note that here the graph B is a bipartite graph and we use U and W to denote the left and right parts of the nodes in B, where $U \cup W = V_B$.

The network $G = (V, E)$ in the instance I' is constructed as follows. There is a copy of graph B, where each edge in E_B is given a direction and becomes a link from the node in U to the node in W. We introduce two nodes s and t, and for each node $v \in U \cup W$, we add two new links $\langle s, v\rangle$ and $\langle v, t\rangle$. For each node $u_i \in U$, we introduce β new nodes s_{ij} ($j \in \{1, 2, \ldots, \beta\}$) and 2β new links $\langle s, s_{ij}\rangle$ and $\langle s_{ij}, u_i\rangle$ ($j \in \{1, 2, \ldots, \beta\}$). For each node $w_i \in W$, we introduce β new nodes t_{ij} ($j \in \{1, 2, \ldots, \beta\}$) and 2β new links $\langle w_i, t_{ij}\rangle$ and $\langle t_{ij}, t\rangle$ ($j \in \{1, 2, \ldots, \beta\}$). So the network G contains $|V| = (\beta+1)|V_B|+2$ nodes and $|E| = |E_B|+2(\beta+1)|V_B|$ links. See Figure 1 for an illustration of the construction with $\beta = 4$.



(a) $B = (V_B, E_B)$ (b) The graph $G = (V, E)$

Figure 1: The construction of the graph G

In instance I', we set s and t as the source and sink nodes, respectively. The problem is to check whether there are at least $k = |V_B| + \beta$ link-disjoint paths from s to t sharing at most δ nodes.

We prove the following lemma.

Lemma 1. *The instance $I = (B = (V_B, E_B), \delta, \beta)$ of the densest δ subgraph problem in bipartite graphs is a yes-instance if and only if the instance $I' = (G, \delta, k = |V_B|+\beta)$ of the problem of checking the existence of k link-disjoint paths sharing at most δ nodes is a yes-instance.*

Proof. In G, we distinguish two kinds of paths from s to t: the path passes through a link corresponding to an edge in B or not. The *node-path*, which does not pass through a link corresponding to an edge in B, is a path of length two from s to a node in V_B and then to t. In Figure 1, $\langle s, u_1, t\rangle$ is a node-path. An *edge-path* is a path from s to t passing through a link corresponding to an edge in B. In Figure 1, $\langle s, s_{11}, u_1, w_2, t_{21}, t\rangle$ and $\langle s, s_{21}, u_2, w_1, t\rangle$ are two edge-paths.

We first consider the necessary condition. Assume that G has a set P of $|V_B|+\beta$ link-disjoint paths from s to t sharing at most δ common nodes except s and t. We use S to denote the set of shared nodes. It is easy to see that S is a subset of V_B. We will show that S induces a subgraph of at least β edges in B. For each node in $V_B \setminus S$, at most one path in P contains it since it is not a shared node. So after deleting $V_B \setminus S$ from G, there are at least $|V_B| + \beta - |V_B \setminus S| = |V_B| + \beta - |V_B| + |S| = \beta + |S|$ paths in P left in the graph. We use P' to denote the set of paths left in P after deleting $V_B \setminus S$. For each node in S, there is at most one corresponding node-path. So P' contains at most $|S|$ node-paths and contains at least $\beta + |S| - |S| = \beta$ edge-paths, where each edge-path is corresponding to a different edge in B with two endpoints in S. Therefore, the node set S induces a subgraph of at least β edges in B and then instance I is a yes-instance.

On the other hand, we assume that B has a subgraph $S_B = (V_1, E_1)$ with $|V_1| = \delta$ nodes and $|E_1| \geq \beta$ edges, and then show that there is a set P' of $k = |V_B| + \beta$ link-disjoint paths from s to t sharing at most common δ nodes in G. For each node $v \in V_B$, we add the node-path $\langle s, v, t\rangle$ to P'. For each edge $uw \in E_1$, we add an edge-path $\langle s, s', u, w, t', t\rangle$ to P', where we can choose edge-paths such that all s' and t' are different nodes in these edge-paths. So P' is a set of $|V_B| + |E_1|$ link-disjoint paths from s to t. Note that no two node-paths will share a node except s and t. Thus, all shared nodes should be endpoints of some edges in E_1, which are the nodes in V_1. The number of shared common nodes is at most $|V_1| = \delta$. □

Lemma 1 shows that

Theorem 1. *It is NP-hard to check the existence of k link-disjoint paths sharing at most δ common nodes in a network.*

Theorem 1 directly implies the NP-hardness of (k, δ)-LDP.

An ILP Formulation

In this section, we build an integer linear programming for (k, δ)-LDP and Restricted (k, δ)-LDP. The ILP algorithm will be used to compare with our fast algorithm for Restricted (k, δ)-LDP in the next section.

First, we consider the ILP formulation for (k, δ)-LDP. For each link $\langle i, j\rangle \in E$, we set a binary variable x_{ij}. Link $\langle i, j\rangle \in E$ is in one path in the solution if and only if $x_{ij} = 1$. For each node $i \in V \setminus \{s, t\}$, we set a binary variable y_i to indicate whether node i is shared by at least two paths in the solution. The ILP formulation for (k, δ)-LDP is

given below:

$$\min \sum_{\langle i,j\rangle \in E} w_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \sum_{j:\langle i,j\rangle \in E} x_{ij} - \sum_{j:\langle j,i\rangle \in E} x_{ji} = \begin{cases} 0, & \forall i \in V \setminus \{s,t\} \\ k, & i = s \\ -k, & i = t; \end{cases} \tag{2}$$

$$\sum_{i \in V \setminus \{s,t\}} y_i \leq \delta; \tag{3}$$

$$\sum_{j:\langle i,j\rangle \in E} x_{ij} \leq y_i k + 1, \quad \forall i \in V \setminus \{s,t\}; \tag{4}$$

$$x_{ij}, y_i \in \{0,1\}. \tag{5}$$

Constraint (2) follows from the integer linear programming formulation for the minimum cost flow problem (Ahuja and Magnanti 2018). We regard k paths as k flows in the network G and set the link capacity to 1 to ensure the link-disjointness. Constraint (3) ensures that the number of common nodes among k link-disjoint paths is at most δ. Constraint (4) requires that the number of paths passing node i is at most 1 if $y_i = 0$ and the number of paths passing node i is at most $k+1$ if $y_i = 1$.

For Restricted (k, δ)-LDP, we only need to replace Constraint (4) with the following Constraint $(4')$ in the above ILP.

$$\sum_{j:\langle i,j\rangle \in E} x_{ij} \leq y_i + 1, \quad \forall i \in V \setminus \{s,t\}. \tag{4'}$$

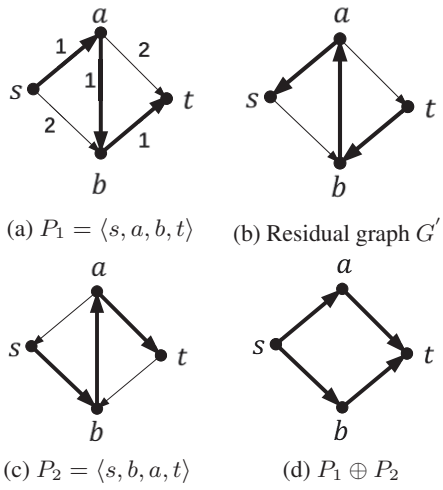
Note that Constraint $(4')$ requires that the number of paths passing node i is at most 1 if $y_i = 0$ and the number of paths passing node i is at most 2 if $y_i = 1$.

In the above ILP, the number of variables is the number of x_{ij} plus the number of y_i, i.e., $|E| + |V| - 2$. The number of constraints is $|V| + 1 + (|V| - 2) + (|E| + |V| - 2) = |E| + 3|V| - 3$.

A Fast Algorithm for Restricted Version

In this section, we design a fast algorithm for Restricted (k, δ)-LDP, which runs in $O((\delta+1)k|E||V|)$ time.

Our algorithm uses two major techniques. The first one is the augmenting path technique for the max-flow problem (Edmonds and Karp 1972), which is also used to find successive shortest paths (Busacker and Gowen 1960). So the initial idea of our algorithm is to iteratively find a path from s to t with minimum total weight. We can treat this problem as a shortest path problem by regarding the link weight as the distance. To guarantee link-disjoint, we will not allow a new path to pass through a link already used by previous paths. However, if we directly delete from the network the links used by previous paths, we may not be able to find the correct solution. In the example in Figure 2, we may not be able to find the second path if we delete the links $\langle s,a\rangle, \langle a,b\rangle, \langle b,t\rangle$ in path $P_1 = \langle s,a,b,t\rangle$. So it comes out the famous technique of residual graphs and augmenting paths. Once we find a path with a minimum total weight in the current graph, we construct a residual graph by reversing the direction of all links in the path and setting the weight of reversed link as the negative of its weight. In the next step, we find a path with a minimum total weight in the residual graph. See Figure 2 for an illustration of the construction of the residual graph. In Figure 2, path $P_1 = \langle s,a,b,t\rangle$ is the minimum weight path in G and path $P_2 = \langle s,b,a,t\rangle$ is the minimum weight path in the residual graph G'. Then $P_1 \oplus P_2$ is a pair of link-disjoint paths with minimum weight in the original graph G. We use $P_1 \oplus P_2$ to denote the combined graph such that a link $\langle a,b\rangle$ is in $P_1 \oplus P_2$ if and only if $\langle a,b\rangle$ is in one of P_1 and P_2 and the reversing link $\langle b,a\rangle$ is not in any of P_1 and P_2. It was proved in (Suurballe 1974) that by using this technique of residual graphs and augmenting paths, we can find a set of k paths between two nodes with minimum total weight. This algorithm only works for the problem without constraints on common nodes shared by the paths. We still have two more constraints: each node can be shared by at most two paths, and the number of common nodes shared by the paths is bounded by δ. We will use the following technique to guarantee these constraints.



(a) $P_1 = \langle s,a,b,t\rangle$ (b) Residual graph G'

(c) $P_2 = \langle s,b,a,t\rangle$ (d) $P_1 \oplus P_2$

Figure 2: An example

The second technique is to split a node to control the number of paths passing through it. This technique is also used in Guo et al.'s algorithm for $(2, \delta)$-LDP (Guo et al. 2018). For a node v in the network, we split it into two nodes v_1 and v_2 such that only inner links are incident on v_1 and only outer links are incident on v_2, where we add 2 parallel links from v_1 to v_2 with weight 0. See Figure 3 for an illustration of the operation of splitting nodes. In order to bound the number of common nodes, we will also set a cost to each newly added link between two split nodes, one link with cost 0 and the other one with cost 1. Thus, we need to find k link-disjoint paths with the total cost at most δ. In our algorithm, in order to make the algorithm effective, we may split a node only when a path passing through it and do not deal with other "unused" nodes in the network.

The above two techniques are intuitive. However, we need to combine them well to guarantee the correctness of the whole algorithm. For the whole algorithm, there are two im-

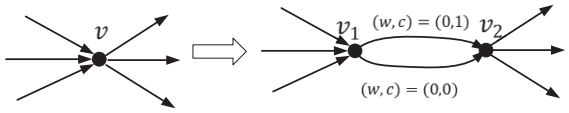


Figure 3: The operation of splitting nodes

portant sub-steps: one is to find a minimum weight path with a bounded cost; the other one is to construct proper residual graphs, called *restricted residual graphs*. The algorithm is simple and easy to understand. The main steps of the algorithm are presented in Algorithm 1.

Algorithm 1 Algorithm LDP

Require: An instance of Restricted (k, δ)-LDP: a network $G = (V, E)$ with a positive weight on links $w : E \to \mathbb{R}^+$, two integers k and δ, and two nodes s and $t \in V$.
Ensure: The minimum weight of k link-disjoint paths with at most δ common nodes in G.
1: Set the initial link cost: $c(e) = 0$ for each $e \in E$;
2: $GR \leftarrow G$; $q \leftarrow \delta$; $w \leftarrow 0$;
3: **for** $(i = 1; i = k; i++)$ **do**
4: In GR, compute a minimum weight path R_i between s and t such that the total cost is at most q;
5: Update the bound of the cost $q \leftarrow q - c(R_i)$;
6: Update the total weight $w \leftarrow w + w(R_i)$;
7: Compute the restricted residual graph with respective to R_i and update the current network GR;
8: **end for**
9: Return w.

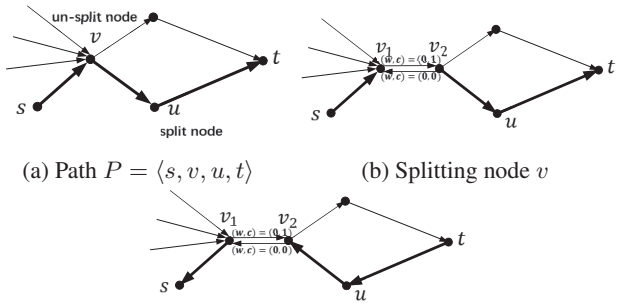
Remark 1. In Step 4, when there is no path between s and t with the cost at most q, the algorithm will stop and report $\perp$ to report that there is no solution. We will explain Step 4 and Step 7 in detail below.

Remark 2. With some modifications, the algorithm can also output the k link-disjoint paths. We use R'_i to denote the corresponding path resulted from R_i by identifying the two split nodes into the original one, where R_i is the minimum weight path computed in Step 4. Then the graph $R'_1 \oplus \cdots \oplus R'_k$ can be decomposed into k link-disjoint paths from s to t in G, which is the set of paths we are seeking for. We use $\{P_1, \ldots, P_k\}$ to denote the solution set.

Shortest Paths With Bounded Cost. In Step 4, we need to compute a minimum weight path with total cost bounded by a given value. For the minimum weight path problem without the cost constraint, the problem becomes the shortest path problem since we can regard the weight as the distance and we can use classic shortest path algorithms to solve it. When adding the cost constraint, the problem becomes hard. Joksch et al. (1966) gave a dynamic programming algorithm to solve it. To improve the running time bound, we can also modify the famous Bellman–Ford algorithm to solve it in $O((\delta + 1)|V||E|)$ time: for each node $v \in V \setminus \{s, t\}$, each integer $l \in [0, n]$ and integer $d \in [0, \delta]$, compute the minimum weight path from s to v with cost at most d and length at most l links in a dynamic programming way.

There is also a way to accelerate the algorithm. For the first time to execute Step 4, we can compute a minimum weight path P_1 without considering the cost constraint because all the link costs are zero for the current network. For this case, it is to solve the shortest path problem and we use the fast Dijkstra's algorithm with running time bound $O(|E|+|V| \log |V|)$. The idea is adopted in the experimental parts, although it can improve the theoretical running time bound.

Restricted Residual Graph. We introduce how to build our residual graph in Step 7, which is modified from the traditional residual graph for the max flow problem. In fact, we also need to split nodes and set costs for links. We mark all nodes in the initial network as "un-split". The restricted residual graph obtained from the current graph GR with respective to a path P is constructed as follows. First, for each un-split node v in P, split it into two "split nodes" v_1 and v_2 such that all inner links are incident on v_1 and all outer links are incident on v_2, add one link $e_1 = \langle v_1, v_2 \rangle$ from v_1 to v_2 with weight $w(e_1) = 0$ and cost $c(e_1) = 1$, and add one link $e_2 = \langle v_2, v_1 \rangle$ from v_2 to v_1 with weight $w(e_2) = 0$ and cost $c(e_2) = 0$. Note that we only split "un-split" nodes. Second, for each link e in P, reverse its direction and set the weight and cost as the negative of them, i.e., $-w(e)$ and $-c(e)$. We use $GR \ominus P$ to denote the operation of the above two steps, i.e., first split un-split nodes with respect to P and then reverse the direction of links in P). So the restricted residual graph is $GR \ominus P$. See Figure 4 for an illustration of the construction.



(a) Path $P = \langle s, v, u, t \rangle$ (b) Splitting node v

(c) Reversing links in P

Figure 4: The construction of the residual graph

Analysis

Next, we analyze our algorithm LDP.

Theorem 2. *Algorithm LDP runs in $O((\delta + 1)k|E||V|)$ time.*

Proof. Steps 1 and 2 of the algorithm take linear time. Step 4 uses $O((\delta + 1)|V||E|)$ time, Steps 5 and 6 can be done in constant time, and Step 7 can be done in linear time. Steps 4-7 will be executed for k iterations. So the running time bound of the algorithm is $O((\delta + 1)k|V||E|)$. □

Lemma 2. *Let I be an instance of Restricted (k, δ)-LDP and v be an arbitrary node different from s and t in the network. Let I' be the new instance after splitting v into two nodes v_1 and v_2 such that all inner links are incident on v_1 and all outer links are incident on v_2 and adding two parallel links from v_1 to v_2 with weight 0. Then I has a solution with weight at most W if and only if I' has a solution with weight at most W.*

This lemma says the equivalence of the instances before and after splitting a node. The new "split" nodes in I' are still allowed to be shared by two paths. However, this will not affect the equivalence since we are going to seek link-disjoint paths. The correctness of this lemma is easy to observe. We can take the two split nodes as a single "big" node and then the instances are identified. Note that, in the construction of the restricted residual graph in Step 7 of Algorithm LDP, after splitting a node we reverse the direction of one of the parallel links because we consider the operation of reversing links in the path together.

Lemma 3. *Let I be an instance of Restricted (k, δ)-LDP. Let $\mathrm{P} = \{P_1, \ldots, P_k\}$ be a set of k link-disjoint paths from s to t in G with $p \leq \delta$ common nodes. If* P *is an optimal solution to Restricted (k, δ)-LDP then the restricted residual graph $G' = G \ominus P_1 \cdots \ominus P_k$ does not contain a directed cycle with cost at most $\delta - p$ and negative weight.*

Proof. Assume that the residual graph G' contains a directed cycle C such that $c(C) \leq \delta - p$ and $w(C) < 0$. We will prove that the weight of paths in P is not minimum, which means there is another set P' of k link-disjoint paths having at most δ common nodes such that $w(\mathrm{P}') < w(\mathrm{P})$. Since the original graph G has no negative link weight, we know that there is no cycle of negative weight in G. So we know that the cycle C in G' must contain at least one "reversed" link, which is corresponding to a link in a path in P. So we know that $\mathrm{P}' = P_1 \oplus \cdots \oplus P_k \oplus C$ is still a set of k link-disjoint paths from s to t in G. However, the weight of P' is less than that of P:

$$w(\mathrm{P}') \leq w(\mathrm{P}) + w(C) < w(\mathrm{P}).$$

The cost of P satisfies:

$$c(\mathrm{P}') \leq c(\mathrm{P}) + c(C) < p + \delta - p < \delta.$$

□

Based on Lemma 3, we can prove the following result. The detailed proof is omitted due to space limitations.

Theorem 3. *The path set returned by Algorithm LDP is a set of minimum-weight k link-disjoint paths from s to t such that there are at most δ common nodes and each common node is shared by two paths.*

Remark. We give some explanations on why Algorithm LDP only works for Restricted (k, δ)-LDP but not for (k, δ)-LDP. In (k, δ)-LDP, more than two paths in the solution can share a common node. However, it only contributes to one common node in the constraint no matter how many paths (at least two paths) sharing it, which may cause trouble in the construction of the residual graph and the proof of Lemma 3. For example, when two paths pass through a common node, two links between the two split nodes are reversed with costs being 0 and -1 respectively; when one more path passes through the common node, we have three links between the two split nodes reversed with costs all being 0 (since removing any one path passing through the node will not decrease the number of common nodes). We will lose some cost "-1". Then the cost of a set of paths can not be distributed to links well.

Experimental Results

In order to evaluate the performance of our Algorithm LDP, we compare it with the integer linear programming algorithm presented in our previous section, which is denoted by ILP. In fact, as far as we know, our algorithm is the first nontrivial (polynomial) algorithm for Restricted (k, δ)-LDP for general k. We compare our algorithm with an integer linear programming based algorithm because these kinds of sequence problems are very suitable for integer linear programming based solvers. Note that in (Guo et al. 2018), the algorithm for $(2, \delta)$-LDP was also compared with an integer linear programming algorithm.

In our experiments, both LDP and ILP output the same result on each instance. So we will mainly compare the running time.

Environments and datasets. Both LDP and ILP are implemented in C/C++, on a PC with Intel Core i5 processor, and 4GB memory. Specifically, the implementation of ILP formulation adopts the Gurobi Optimizer of version 8.1.1[1]. In the experiments, we generate random graphs (n, m) from the $Network$X library[2] with n nodes and m links. We consider two kinds of networks, P2P networks and communication networks, from Stanford Large Network Dataset Collection within Stanford Network Analysis Project (SNAP)[3].

Evaluation on Random Graphs

In this subsection, random graphs with n nodes and m links are generated from the $Network$X library with the link weight uniformly distributed in $[1, 100]$. During the experiments, we randomly generate 200 different graphs for each fixed pair of n and m, and for each graph, we randomly choose two nodes as the source and sink. In real networks, we may not have changes to use too many backup paths due to the dynamic of the networks. So we consider k with values from 2 and 5. Also many protocols, such as TCP/IP, only allow a data package pass through at most 16 routers from source to destination (Kurose and Ross 2008). The length of the path can not be very large and the number of common nodes can not be very large. So we mainly consider δ from 5 to 12.

The running time displayed in the following figures is the average running time on 200 instances. In Figure 5, we show the average running times on two groups of instances with $(n, m) = (1000, 10000)$ and $(n, m) = (10000, 100000)$ under different values of k and δ. In fact, fluctuations in values

[1] https://www.gurobi.com/

[2] https://networkx.github.io/

[3] http://snap.stanford.edu/data/

of k and δ have little effect on the running time of ILP, and the three lines for ILP are overlapped. From Figure 5, we can see that LDP is faster than ILP. We also show more compared results on different instances in Table 1.

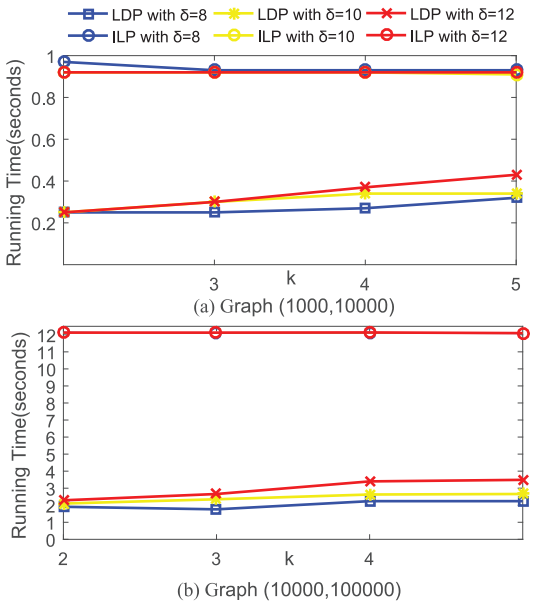


Figure 5: The results with different values of δ and k

Evaluation on Real Datasets

We also compare our algorithms on some real-world network datasets from SNAP. To be consistent with the experimental environment for simulation results, we also ran our algorithms with randomly chosen source and sink over 200 times on each instance and report the average running time, where δ is set as 10. Table 2 shows the running times in seconds of LDP and ILP on different networks, and also the number of instances having feasible solutions (there exist satisfied k link-disjoint paths) among the 200 instances in round brackets. The feasibility of the instances is caused by the choice of the source and sink nodes. In Table 2, below the instance name, (n, m) denotes the number of nodes and the number of links in the network. From this table, we can see that LDP's performance is much better than that of ILP on real-world datasets. Note that when k increases, the running time of LDP may decrease, because, for large k, the number of instances having feasible solutions may decrease.

Conclusion

In this paper, we have established the NP-hardness of (k, δ)-LDP. We prove that it is still NP-hard to check the existence of k link-disjoint paths sharing at most δ common nodes in a network, which indicates that it is also hard to find approximation solution to (k, δ)-LDP. On the other hand, the restricted version may be easier. We design a fast polynomial algorithm and an ILP for Restricted (k, δ)-LDP. Experimental results show that our polynomial algorithm is very effective.

Table 1: Running time in seconds on more random graphs

(n, m, δ)	Alg.	k=2	k=3	k=4	k=5
(50, 250, 5)	LDP	0.006	0.007	0.008	0.007
	ILP	0.06	0.06	0.048	0.046
(60, 360, 6)	LDP	0.009	0.012	0.012	0.014
	ILP	0.054	0.059	0.054	0.054
(70, 490, 6)	LDP	0.011	0.014	0.023	0.032
	ILP	0.064	0.064	0.064	0.064
(80, 640, 8)	LDP	0.016	0.023	0.024	0.030
	ILP	0.076	0.081	0.075	0.075
(90, 810, 10)	LDP	0.028	0.030	0.040	0.040
	ILP	0.086	0.085	0.085	0.085
(100, 1000, 10)	LDP	0,030	0.037	0.05	0.5
	ILP	0.112	0.115	0.098	0.101
(500, 25000, 10)	LDP	0.305	0.417	0.571	0.734
	ILP	0.937	0.935	0.943	0.944
(600, 36000, 10)	LDP	0.428	0.577	0.792	0.828
	ILP	1.26	1.242	1.265	1.265
(700, 49000,10)	LDP	0.555	0.798	1.087	1.307
	ILP	1.718	1.754	1.78	1.782
(800, 64000, 12)	LDP	1.601	2.222	2.511	3.782
	ILP	3.881	3.860	3.854	3.890
(900, 81000, 12)	LDP	1.787	2.493	3.486	4.972
	ILP	5.231	5.153	5.266	5.133
(1000, 100000, 12)	LDP	2.281	3.382	4.441	5.650
	ILP	6.920	7.151	7.040	7.013

For further study, it is interesting to determine the computational complexity of (k, δ)-LDP for each constant $k \geq 3$.

Acknowledgments

The work is supported by the National Natural Science Foundation of China, under grants 61972070 and 61802049. We would like to thank Weibo Lin, Zhenyu Guo and Yi Zhou for their discussions on this paper.

Table 2: Average running time and the number of feasible instances for SNAP networks

Networks	Alg.	k =2	k =3	k =4	k =5
p2p-Gnut.05	LDP	2.33(28)	3.29(30)	3.87(20)	3.99(7)
(8846,31839)	ILP	6.59(28)	6.66(30)	6.61(20)	6.62(7)
p2p-Gnut.06	LDP	2.35(43)	3.80(27)	4.57(22)	4.97(3)
(8717,31525)	ILP	6.57(43)	6.51(27)	6.49(22)	6.47(3)
p2p-Gnut.08	LDP	1.61(42)	2.13(25)	2.09(11)	1.76(1)
(6301,20777)	ILP	4.48(42)	4.46(25)	4.43(11)	4.41(1)
p2p-Gnut.09	LDP	1.92(38)	1.84(13)	2.54(6)	2.70(1)
(8114,26013)	ILP	5.80(38)	5.76(13)	5.75(6)	5.74(1)
p2p-Gnut.24	LDP	4.52(23)	5.59(7)	7.16(6)	5.22(1)
(26518,65369)	ILP	21.10(23)	21.02(7)	20.98(6)	20.96(1)
p2p-Gnut.25	LDP	3.46(20)	5.74(25)	4.62(3)	3.91(1)
(22687,54705)	ILP	17.44(20)	17.47(25)	17.37(3)	17.37(1)
p2p-Gnut.30	LDP	5.73(17)	6.35(12)	8.09(4)	8.34(1)
(36682,88328)	ILP	31.31(17)	31.13(12)	31.11(4)	31.04(1)
p2p-Gnut.31	LDP	10.63(18)	11.09(7)	13.66(7)	13.58(2)
(62586,147892)	ILP	64.41(18)	64.70(7)	64.26(7)	64.30(2)
Email-EuALL	LDP	26.88(13)	30.25(8)	28.80(3)	27.88(0)
(265214,420045)	ILP	469.81(13)	468.18(8)	467.16(3)	467.58(0)

References

Ahuja, R. K., and Magnanti, T. L. 2018. *Network Flows*. Franklin Classics.

Beshir, A., and Kuipers, F. 2009. Variants of the min-sum link-disjoint paths problem. In *16th Annual Symposium on Communications and Vehicular Technology in the Benelux (SCVT 2009), November 19, 2009*, 1–6. IEEE/SCVT.

Busacker, R. G., and Gowen, P. J. 1960. A procedure for determining a family of minimum-cost network flow patterns. Technical report, RESEARCH ANALYSIS CORP MCLEAN VA.

Choudhury, P. D.; Bhadra, S.; and De, T. 2019. A brief review of protection based routing and spectrum assignment in elastic optical networks and a novel p-cycle based protection approach for multicast traffic demands. *Optical Switching and Networking* 32:67–79.

Cohen, N., and Nutov, Z. 2013. A (1+ln2)(1+ln2)-approximation algorithm for minimum-cost 2-edge-connectivity augmentation of trees with constant radius. *Theor. Comput. Sci.* 489-490:67–74.

Corneil, D. G., and Perl, Y. 1984. Clustering and domination in perfect graphs. *Discrete Applied Mathematics* 9(1):27–39.

Din, D.-R., and Lai, I.-R. 2015. Multicast protection problem on elastic optical networks using segment-base protection. In *2015 International Conference on Informatics, Electronics & Vision (ICIEV)*, 1–6. IEEE.

Edmonds, J., and Karp, R. M. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM* 19(2):248–264.

Feige, U.; Kortsarz, G.; and Peleg, D. 2001. The dense k-subgraph problem. *Algorithmica* 29(3):410–421.

Gouveia, L.; Simonetti, L.; and Uchoa, E. 2011. Modeling hop-constrained and diameter-constrained minimum spanning tree problems as steiner tree problems over layered graphs. *Math. Program.* 128(1-2):123–148.

Guo, L., and Shen, H. 2013. On finding min-min disjoint paths. *Algorithmica* 66(3):641–653.

Guo, L.; Deng, Y.; Liao, K.; He, Q.; Sellis, T.; and Hu, Z. 2018. A fast algorithm for optimally finding partially disjoint shortest paths. In *IJCAI*, 1456–1462.

Hu, W. S., and Zeng, Q. J. 1998. Multicasting optical cross connects employing splitter-and-delivery switch. *IEEE Photonics Technology Letters* 10(7):970–972.

Johnston, M.; Lee, H.; and Modiano, E. 2015. A robust optimization approach to backup network design with random failures. *IEEE/ACM Trans. Netw.* 23(4):1216–1228.

Joksch, H. C. 1966. The shortest route problem with constraints. *Journal of Mathematical analysis and applications* 14(2):191–197.

Kerivin, H., and Mahjoub, A. R. 2005. Design of survivable networks: A survey. *Networks* 46(1):1–21.

Kortsarz, G., and Nutov, Z. 2016. A simplified 1.5-approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *ACM Trans. Algorithms* 12(2):23:1–23:20.

Kuipers, F. 2012. An overview of algorithms for network survivability. *ISRN Communications and Networking* 2012:1–24.

Kurose, J. F., and Ross, K. W. 2008. *Computer Networking: A top-down approach*, volume 4. Pearson/Addison Wesley Boston USA.

Le, D. D.; Molnár, M.; and Palaysi, J. 2014. Multicast routing in WDM networks without splitters. *IEEE Communications Magazine* 52(7):158–167.

Li, C.; McCormick, S. T.; and Simchi-Levi, D. 1990. The complexity of finding two disjoint paths with min-max objective function. *Discrete Applied Mathematics* 26(1):105–115.

Seymour, Z., and Kar, D. 2013. Finding partially link-disjoint paths in wireless sensor networks. In *European Wireless 2013; 19th European Wireless Conference*, 1–6. VDE.

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

Sherali, H. D.; Özbay, K.; and Subramanian, S. 1998. The time-dependent shortest pair of disjoint paths problem: Complexity, models, and algorithms. *Networks* 31(4):259–272.

Steiglitz, K.; Weiner, P.; and Kleitman, D. J. 1969. The design of minimum-cost survivable networks. *IEEE Transactions on Circuit Theory* 16(4):455–460.

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

Suurballe, J. W. 1974. Disjoint paths in a network. *Networks* 4(2):125–145.

Wang, W., and Doucette, J. 2018. Availability optimization in shared-backup path protected networks. *Journal of Optical Communications and Networking* 10(5):451–460.

Wang, L., and Zhu, D., eds. 2018. *Computing and Combinatorics - 24th International Conference, COCOON 2018, Qing Dao, China, July 2-4, 2018, Proceedings*, volume 10976 of *Lecture Notes in Computer Science*. Springer.

Wei, Y.; Shen, G.; and Bose, S. K. 2014. Span-restorable elastic optical networks under different spectrum conversion capabilities. *IEEE Trans. Reliability* 63(2):401–411.

Yallouz, J., and Orda, A. 2017. Tunable qos-aware network survivability. *IEEE/ACM Trans. Netw.* 25(1):139–149.

Yallouz, J.; Rottenstreich, O.; Babarczi, P.; Mendelson, A.; and Orda, A. 2018. Minimum-weight link-disjoint node-"somewhat disjoint" paths. *IEEE/ACM Trans. Netw.* 26(3):1110–1122.

Yallouz, J.; Rottenstreich, O.; and Orda, A. 2016. Tunable survivable spanning trees. *IEEE/ACM Trans. Netw.* 24(3):1853–1866.