INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. 2003; **16**:779–798 (DOI: 10.1002/dac.612)

Link-disjoint paths for reliable QoS routing

Yuchun Guo[1], Fernando Kuipers[2] and Piet Van Mieghem[2,*,†]

[1] *School of Electrical and Information Engineering, Northern Jiaotong University, Beijing 100044, People's Republic of China*
[2] *Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands*

SUMMARY

The problem of finding link/node-disjoint paths between a pair of nodes in a network has received much attention in the past. This problem is fairly well understood when the links in a network are only specified by a single link weight. However, in the context of quality of service routing, links are specified by multiple link weights and restricted by multiple constraints. Unfortunately, the problem of finding link/node disjoint paths in multiple dimensions faces different conceptual problems. This paper presents a first step to understanding these conceptual problems in link-disjoint quality of service routing and proposes a heuristic link-disjoint QoS algorithm that circumvents these problems. Copyright © 2003 John Wiley & Sons, Ltd.

KEY WORDS: QoS routing; link-disjoint paths; restorable/reliable routing

1. INTRODUCTION

The problem of finding disjoint paths in a network has been given much attention in the literature due to its theoretical as well as practical significance to many applications, such as layout design of integrated circuits, survivable design of telecommunication networks and restorable/reliable routing. Paths between a given pair of source and destination nodes in a network are called *link disjoint* if they have no common (i.e. overlapping) links, and *node disjoint* if, besides the source and destination nodes, they have no common nodes. With the development of optical networks and the deployment of MPLS or GMPLS networks, the disjoint paths problem is receiving renewed interest as fast restoration after a network failure is crucial in such kind of networks. In robust communication networks, a connection usually consists of two link- or node-disjoint paths: one active path, and one backup path. A service flow will be redirected to the backup path if the active path fails. Load balancing, another important aspect for communication networks to avoid network congestion and optimize network throughput, also

*Correspondence to: P. Van Mieghem, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands.
† E-mail: P.VanMieghem@ewi.TuDelft.nl

Contract/grant sponsor: NUFFIC

Received September 2002
Revised April 2003
Accepted June 2003

requires disjoint paths to distribute flows. Robustness and load balancing are, among others, both aspects of quality of service (QoS) routing.

In this paper we will focus on finding QoS-aware *link-disjoint* paths. In general a link-disjoint paths algorithm can be extended to a node-disjoint algorithm with the concept of *node splitting*, i.e. replacing one node with two nodes that are linked together by a link with zero weights [1]. Throughout this paper, we use the following notation. A network is denoted by a directed graph $G(V,E)$, where V is the set of nodes and E is the set of links. A directed link from node u to node v is represented as $u \rightarrow v$, $u, v \in V$. Each link is characterized by a link weight vector $\mathbf{w}$ consisting of M link metrics $w_m(u \rightarrow v)$, for $m = 1, \ldots, M$. We assume that only non-negative link metrics are assigned to each link. However, in the process of computing disjoint paths, negative link weights may be assigned to links. QoS metrics can be (a) *additive*, e.g. delay, jitter, in which case the path-weight vector consists of summing the link-weight vectors of the links defining the path, (b) *multiplicative*, e.g. one minus the packet loss probability, which can be considered as additive after taking the logarithm and (c) *min/max*, e.g. bandwidth, and policy flags, in which case the minimum (or maximum) link weight defines the weight of a path. Min/max links that do not obey the constraints can be pruned from the topology, which is called topology filtering. Additive metrics cause more difficulties and therefore without loss of generality, we assume all metrics to be additive [2]. In the context of QoS routing or multi-constrained routing, a path is called feasible when its weight vector does not violate the constraints specified by the vector $\mathbf{L}$.

Since we mainly focus on finding link-disjoint paths, a path P, between a source s and destination t is considered to be a set of links that compose this path. With a slight abuse of notation, we choose P to denote the path as well as its link set. If path P_1 is link-disjoint with P_2, there is no common link element in the link set representing each path and $P_1 \cap P_2 = \emptyset$, else $P_1 \cap P_2 \neq \emptyset$.

Definition of path length: Given a graph $G(V,E)$ with M metrics per link, the *non-linear length* of a path P from source node s to destination node t is defined as [3]

$$l(P) = \max_{m=1,\ldots,M} \left(\frac{w_m(P)}{L_m} \right) \tag{1}$$

where $w_m(P) = \sum_{(u \rightarrow v) \in P} w_m(u \rightarrow v)$.

The normalization in (1) by the constraints $\mathbf{L}$ ascertains that if $l(P) > 1$, then one of the constraints has been violated. For $M = 1$, the non-linear length of a path as defined in (1) reduces to a *linear* one, and the link weight vector $\mathbf{w}$ reduces to a scalar $w(u \rightarrow v)$. When no constraint is required, as in the LPP problem stated below, the linear length of a path is computed as $\sum_{(u \rightarrow v) \in P} w(u \rightarrow v)$, i.e. $L_1 = 1$. For simplicity of representation, the above notation of path length $l(P)$ is still used.

If path P_1 is link-disjoint with P_2, i.e. $P_1 \cap P_2 = \emptyset$, we have $l(P_1 \cup P_2) = l(P_1) + l(P_2)$ for $M = 1$. But for $M > 1$, we have $l(P_1 \cup P_2) \leqslant l(P_1) + l(P_2)$. Our target in this paper is to find a set of two link-disjoint paths that both obey multiple constants. We define the *total length* of two paths as

$$l(P_1) + l(P_2) \tag{2}$$

for $M \geqslant 1$.

Link-disjoint path pair (*LPP*) *problem*. Given a graph $G(V, E)$ with 1 metric per link ($M = 1$), for a source-destination pair (s, t), find a set of two paths P_1 and P_2, such that $P_1 \cap P_2 = \emptyset$, and the total length $l(P_1) + l(P_2)$ is minimized.

The LPP problem can be solved in polynomial time [1, 4, 5].

Multiple constrained path (MCP) problem. Given a graph $G(V, E)$ with $M > 1$ metrics per link and a constraint vector $\mathbf{L}$, for a source-destination pair (s, t), find a path that obeys the constraint vector $\mathbf{L}$,

$$w_m(P) \leqslant L_m \quad \text{for } m = 1, \ldots, M.$$

where $w_m(P) = \sum_{u \to v \in p} w_m(u \to v)$, for $m = 1, \ldots, M$.

The MCP problem is NP-complete [6, 7].

Multiple constrained link-disjoint path pair (MCLPP) problem. Given a graph $G(V, E)$ with $M > 1$ metrics per link and a constraint vector $\mathbf{L}$, for a source-destination pair (s, t), find a pair of link-disjoint paths P_1 and P_2, such that $P_1 \cap P_2 = \emptyset$, and both paths obey the constraint vector $\mathbf{L}$.

Theorem 1

MCLPP is NP-complete.

Proof

Given a chain topology with $n + 1$ nodes and $2n$ links, each with a two-component weight vector as depicted in Figure 1 and a set of numbers $a_i \in A$, $0 \leqslant a_i \leqslant S$, for $i = 1, \ldots, n$, where $S = \sum_{i=1}^{n} a_i$. The constraints are chosen as follows: $L_1 = nS - (S/2)$, and $L_2 = (S/2)$.

To solve the MCLPP problem, we need to find two paths P and P' from node 1 to node $n + 1$ that obey the constraints. Since, for all link weight vectors, the sum of the components equals S, we have that $w_1(P) + w_2(P) = nS$ and $w_1(P') + w_2(P') = nS$. Accordingly, a solution satisfying the constraints is only found if $w_1(P \text{ and } P') = nS - (S/2)$ and $w_2(P \text{ and } P') = (S/2)$. The problem has now become an instance of the well-known NP-complete partition problem [8] and can only be solved by finding the set $A' \subseteq A$, for which $\sum_{a_i \in A'} a_i = (S/2)$. A feasible path P exists if the set A' exists. A feasible path P consists of the lower link if $a_i \in A'$ and the upper link if $a_i \notin A'$. The path P' then follows the remaining links. $\square$

In this paper we focus on solving the MCLPP problem. Related work on finding disjoint paths in one dimension between a source and a destination will be reviewed in Section 2 and a simple link-disjoint algorithm LBA will be explained in Section 3. In Section 4 an extension of LBA to multiple dimensions is discussed and shown to be difficult. Therefore, a heuristic

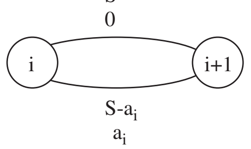

Figure 1. The assignment of link weights to the links in the chain topology between nodes i and $i + 1$.

algorithm DIMCRA for solving the MCLPP problem is proposed in Section 5. We conclude this article in Section 6.

2. RELATED WORK

2.1. Link-disjoint paths routing in one dimension

An intuitive method to determine two shortest link-disjoint paths between a pair of source and destination nodes consists of two steps. The first step retrieves the shortest path between a given pair of nodes in a graph. The second step is to remove all the links of that path from the graph, and to find the shortest path in the pruned graph. We will refer to this method as the remove-find (RF) method. Although the RF method is direct and simple, it has at least two disadvantages due to the removal of links belonging to the first shortest path: (a) provided that two link-disjoint paths exist, there is no guarantee that they will be found as illustrated in Section 3.1 and (b) the second link-disjoint shortest path may have a significantly larger length.

To surmount the disadvantages of the RF method, other methods have been devised to find a pair of shortest link-disjoint paths with minimal total length [1, 4, 5, 8–12]. In Reference [5], Suurballe proposes an algorithm, referred to as Suurballe's algorithm, to find K *node-disjoint* paths with minimal total length using the *path augmentation* method. The path augmentation method is originally used to increase the size of a matching with an augmenting path [13] and to find a maximum flow or a minimum cost flow in a network [14, 15]. The problem to find link/node disjoint paths can be viewed as a special case of the minimum cost flow problem as demonstrated in References [1, 4, 5]. The basic idea of Suurballe's algorithm is to construct a solution set of two disjoint paths based on the shortest path and a shortest augmenting path. K disjoint paths can be obtained by augmenting the $K-1$ optimal disjoint paths with this algorithm. In 1984, Suurballe and Tarjan [1] improved Suurballe's algorithm such that pairs of *link-disjoint* paths from one source node to n destination nodes could be efficiently obtained in a single Dijkstra-like computation. This algorithm is referred to as the S–T algorithm. To find n pairs of disjoint paths, the S–T algorithm requires $O(E \log_{(1+E/n)} n)$ time and Suurballe's algorithm $O(n^2 \log n)$, where n is the number of destination nodes and E is the number of links. Kar *et al*. [16] and Kodialam and Lakshman [17, 18] incorporated the S–T algorithm into their algorithms to find a pair of link-disjoint paths serving as active and backup paths for routing bandwidth guaranteed connections. Liang [19] extended the S–T algorithm to find two link-disjoint paths between a pair of nodes with optimization in both network load and routing cost.

In 1994, Bhandari [4] proposed an algorithm to find a pair of *span-disjoint* paths between two nodes in optical-fibre networks. The disjoint paths algorithm used by Bhandari is a modified version of Suurballe's algorithm [5] that requires a special link weight transformation to facilitate the use of Dijkstra's. Bhandari made a simplification to Suurballe's algorithm by directly setting all the link weights on the first shortest path negative. Shaikh [20] made an extension to Bhandari's algorithm [4] to solve the span-disjoint paths problem in more complicated structured optical networks.

It is proved in References [21, 22] that the LPP problem will be NP-complete if it is required that the maximal length of the two disjoint paths, i.e. $\max(l(P_1), l(P_2))$, is minimized. In addition, Ho and Mouftah [23] proposed another optimal object function $\alpha \cdot l(P_1) + l(P_2)$, where P_1 and P_2 are the active path and the backup path, respectively. The parameter α can be set large for a shared protection scheme ($1:N$ or $M:N$) and could be as small as unity for a dedicated protection scheme (1:1). When $\alpha = 1$, it reduces to the object function used in References [1, 4, 5].

Heuristic algorithms based on matrix calculation [24] or recursive matrix-calculation [25] to solve the *K-shortest* link-disjoint paths problem with a bounded hopcount have been proposed as well. There are also some algorithms for finding *K-best* paths, i.e. K disjoint or maximally disjoint paths with minimum total length between a pair of nodes, in a *trellis* graph [8, 12]. An optimal algorithm for finding *K-best* paths without hopcount limitation between a pair of nodes is given by Lee and Wu in Reference [26], where they transfer the *K-best* paths problem into a maximum network flow and minimum cost network flow algorithm via some modifications to the original graph. Distributed algorithms for the link/node-disjoint paths algorithms can be found in References [9–11].

2.2. Disjoint paths routing in multiple dimensions

To the best of our knowledge there is no literature on the MCLPP problem. Recently some papers on disjoint paths in QoS routing have emerged. However, they only considered bandwidth and/or delay as their QoS metrics [18, 27–30]. The maximally disjoint shortest and widest paths (MADSWIP) algorithm from Taft-Plotkin *et al.* [31], involves a modified version of the S–T algorithm to find a pair of disjoint paths. MADSWIP can produce a pair of widest or shortest maximally link-disjoint paths from a source node to all other nodes. Moreover it tries to find two paths simultaneously to satisfy the maximally link-disjointness to each other in a QoS routing context. However the link metrics used in their algorithm are bandwidth and delay, where only the latter metric is additive.

3. PATH AUGMENTATION FOR SOLVING LPP

In this section we will present a simplified variant of Bhandari's Algorithm [4], referred to as LBA (link-disjoint version of Bhandari's algorithm), which can produce an optimal solution for the LPP problem. The basic steps of LBA are given in Section 3.1. The fundamental concepts of this algorithm are discussed in Section 3.2. The optimality is proved in Section 3.3 and in Section 3.4, LBA is shown to be loop-free.

3.1. The steps of LBA

Bhandari's algorithm [4] was designed to find a pair of span-disjoint paths in an optical network. We modify Bhandari's algorithm into a link-disjoint path pair algorithm LBA by omitting the node-splitting operation that ensures the node-disjointness and the graph transformations that ensure span-disjointness.

Before explaining the operation of LBA we first introduce some notations that will be used further. If we reverse the direction and the sign of the link weights of each link on the path P_1 between s and t, i.e. $w(v \to u) = -w(u \to v)$, $\forall\ (u \to v) \in P_1$, then we will have a path directed from t to s, denoted by $-P_1$, which consists of the reversed P_1 links. We define[‡] $l(-P_1) = -l(P_1)$.

[‡]With the definition of length in (1), we have $l(-P_1) = -l(P_1)$ only for $M = 1$.

A set, which consists of the P_1 links whose reversed links appear on P_2 and vice versa, is denoted as $P_1 \tilde{\cap} P_2 = \{(u \to v) \text{ and } (v \to u) | (u \to v) \in P_1 \text{ and } (v \to u) \in P_2\}$. In all the figures, bold lines represent links on the shortest path(s) in a graph or its corresponding modified graph, dashed lines represent reversed links which do not exist in the original graph and bold dashed lines represent such reversed links that appear on the shortest path. The steps of the LBA algorithm are as follows:

Given a directed graph $G(V, E)$, for a source-destination pair (s, t),

Step 1: Find the shortest[§] path P_1 from node s to node t;

Step 2: Replace P_1 with $-P_1$, a modified graph $G(V, E')$ is created;

Step 3: Find a shortest path P_2 from node s to node t in the modified graph $G(V, E')$; if P_2 does not exist, then stop;

Step 4: Take the union of P_1 and P_2, remove from the union the link set which consists of the P_1 links whose reversed links appear in P_2, and *vice versa*, then group the remaining links into two paths P_1' and P_2', i.e. $P_1' \cup P_2' = (P_1 \cup P_2) \backslash (P_1 \tilde{\cap} P_2)$.

We will explain the steps of LBA with an example in Figure 2. Suppose that we are required to find a set of two shortest disjoint paths between a and b. In Step 1, the shortest path from a to b is found as $P_1 = acdb$, with minimum length 4. In Step 2, a modified graph $G(V, E')$ is created by reversing the direction and the sign of the weight of each link on P_1. For instance, the link

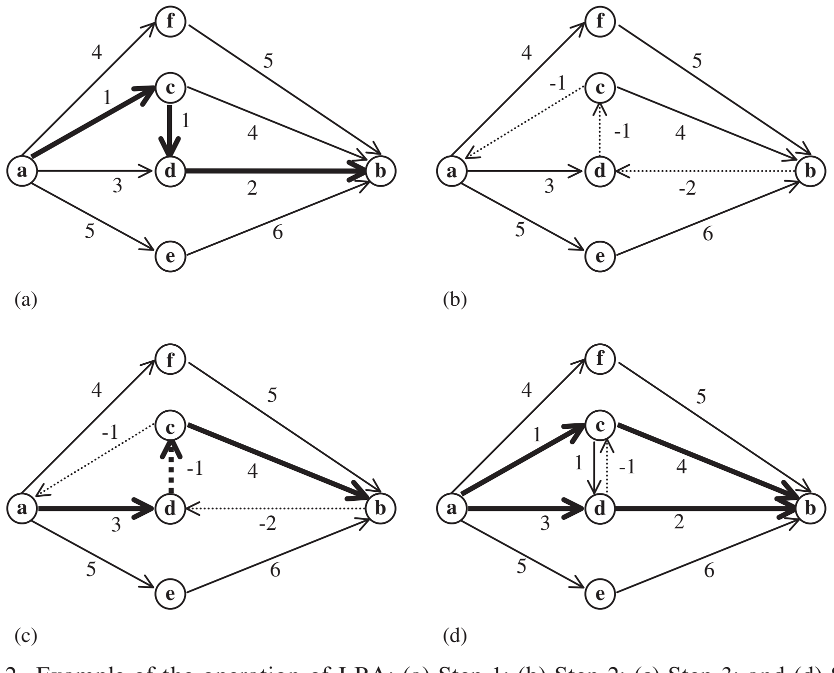

Figure 2. Example of the operation of LBA; (a) Step 1; (b) Step 2; (c) Step 3; and (d) Step 4.

[§] If there exist more than one shortest path in the original graph or in the modified graph, either one of them can be chosen. Choosing different shortest paths may lead to different solution sets. However, these solution sets will have the same minimum total length.

$c \to d$ with weight 1 is replaced by the link $d \to c$ with weight -1. In Step 3, the shortest path in the modified graph $P_2 = adcb$ has length 6. In Step 4, $P_1 \tilde{\cap} P_2 = \{c \to d, d \to c\}$ is removed from the union $P_1 \cup P_2$. The solution set of disjoint paths $\{P_1', P_2'\} = \{acb, adb\}$ is obtained. The total length of this path set equals $5 + 5 = 10$, which is exactly the minimal total length of two link-disjoint paths in this graph.

For comparison, in Figure 3, we apply the RF method on the same topology with the same requirements. In step 1 the shortest path *acdb* is retrieved. In step 2, a modified graph is created by removing all the links on *acdb*. The shortest path in the modified graph is *aeb* with length 11. Thus the set $\{acdb, aeb\}$ has a total length $4 + 11 = 15$, which is longer than 10 as found with LBA. This example illustrates that the RF method cannot guarantee to find the optimal solution. More important, in the graph shown in Figure 4(a), although there exist two link-disjoint paths between a and b, RF cannot find the second path in step 2 as shown in Figure 4(b). LBA, on the other hand, still returns the optimal set in this case.

3.2. *LBA is based on the shortest path*

In this subsection, we will clarify why the optimal solution set of LBA, as well as other path augmentation algorithms [4, 10, 30], is based on the shortest path. Although the theory presented here is based on (or can be derived from) the theory of min-cost flow [14, 15], it is instructive to give an outline.

We will first show that the optimal set for the LPP problem is based on the shortest path. Secondly, we will show that the optimal set of two link-disjoint paths has the smallest difference in length from the shortest path among all the possible sets of link-disjoint paths. Finally, we will show that the logical difference set (defined below) can be viewed as a path.

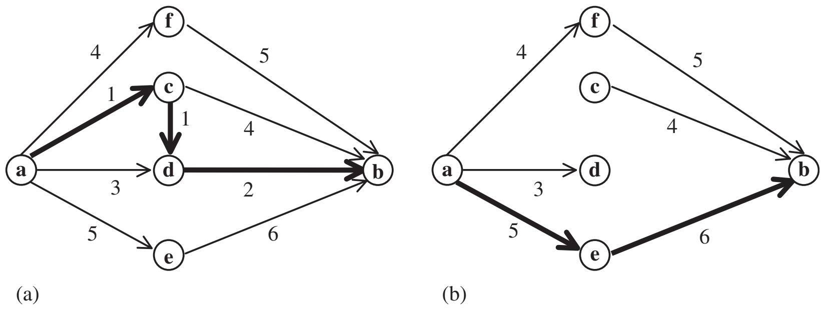

Figure 3. Example 1 of the operation of RF; (a) Step 1; and (b) Step 2.

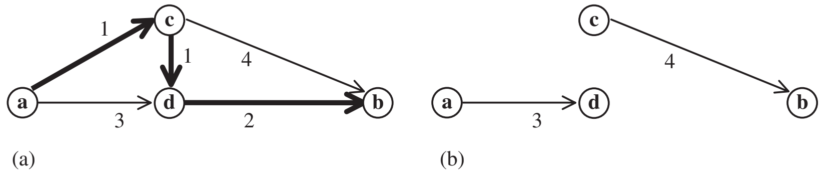

Figure 4. Example 2 of the operation of RF; (a) Step 1; and (b) Step 2.

Given a digraph $G(V,E)$ and a pair of source-destination nodes (s,t), the relation between a set of two link-disjoint paths $\{P_{d1},P_{d2}\}$ and the shortest path P_1 belongs to one of the following types:

1. P_1 itself is P_{d1} or P_{d2}, i.e. $P_1 = P_{d1}$ or $P_1 = P_{d2}$;
2. P_1 overlaps with both paths P_{d1} and P_{d2}, i.e. $P_1 \cap P_{d1} \neq \emptyset$, $P_1 \neq P_{d1}$ and $P_1 \cap P_{d2} \neq \emptyset$, $P_1 \neq P_{d2}$;
3. P_1 only overlaps with one path in the set $\{P_{d1},P_{d2}\}$, but not with the other one, i.e. $P_1 \cap P_{d1} \neq \emptyset$, $P_1 \neq P_{d1}$ and $P_1 \cap P_{d2} = \emptyset$ (or $P_1 \cap P_{d2} \neq \emptyset$, $P_1 \neq P_{d2}$ and $P_1 \cap P_{d1} = \emptyset$);
4. P_1 is link-disjoint with both paths in $\{P_{d1},P_{d2}\}$, i.e. $P_1 \cap (P_{d1} \cup P_{d2}) = \emptyset$.

Lemma 1
Given a directed graph $G(V,E)$ and a source-destination pair (s,t), if the optimal set $\{P'_1,P'_2\}$ of LPP exists, $P'_1 \cup P'_2$ must contain either the first shortest path P_1 itself or some P_1 links on each of its two paths.

Proof
If $P'_1 \cup P'_2$ is of type (4), then each path in $\{P'_1,P'_2\}$ is link-disjoint with P_1. As P_1 is the shortest path, both $\{P_1,P'_1\}$ and $\{P_1,P'_2\}$ have a total length shorter than $\{P'_1,P'_2\}$. Hence the optimal set $\{P'_1,P'_2\}$ cannot be of type (4) and $P'_1 \cup P'_2$ must contain some or all P_1 links to be the optimal set.

If $P'_1 \cup P'_2$ is of type (3), only one path in $P'_1 \cup P'_2$ contains some P_1 links, without loss of generality, suppose P'_1 contains some P_1 links, and the other path P'_2 is link-disjoint with P_1, then $\{P_1,P'_2\}$ is a set which is shorter than $\{P'_1,P'_2\}$. Hence the optimal set $\{P'_1,P'_2\}$ cannot be of type (3).

Therefore, if the optimal set $\{P'_1,P'_2\}$ exists, $P'_1 \cup P'_2$ must be either of type (1) or (2). $\square$

Property 1
The optimal set $\{P'_1,P'_2\}$ has the smallest difference in length

$$Y = l(P'_1) + l(P'_2) - l(P_1) \geqslant 0 \tag{3}$$

from the shortest path P_1, among all the possible sets of link-disjoint path pairs.

In the set $P'_1 \cup P'_2 \cup (-P_1)$, the P_1 links contained in the set $P'_1 \cup P'_2$ will form loops with the $-P_1$ links. For example, if a P_1 link $u \to v$ is contained in the set $P'_1 \cup P'_2$, then it will create a loop with the link $v \to u$ on $-P_1$ between the nodes u and v. The length of this loop is zero because $w(v \to u) = -w(u \to v)$. Let us denote $O_l = (P'_1 \cup P'_2)\tilde{\cap}(-P_1)$, which means that the set O_l consists of each P_1 link in the union of $P_2 \cup P_1$ and its corresponding $-P_1$ link. We define the logical difference set[¶] between $P'_1 \cup P'_2$ and P_1 as $(P'_1 \cup P'_2) - P_1 = P'_1 \cup P'_2 \cup (-P_1) \setminus O_l$. In fact, $l(O_l) = 0$ because the set O_l consists of loops with zero length, each consisting of a pair of

[¶] The logical difference set $P_2 - P_1$ also can be computed as $P_2 - P_1 = \{(u \to v)|(u \to v) \in P_2\backslash(P_2 \cap P_1)\} \cup \{(v \to u)|(u \to v) \in P_1\backslash(P_2 \cap P_1)\}$, which means that if a link $u \to v$ of P_2 does not appear on P_1, then this link belongs to the difference set $P_2 - P_1$, and if a link $u \to v$ of P_1 does not appear on P_2, then its direction reversed link $v \to u$ belongs to the difference set $P_2 - P_1$, with a link weight $w(v \to u) = -w(u \to v)$. In set theory, the difference operation is defined as $P_2 - P_1 = P_2\backslash(P_1 \cap P_2)$, and the symmetric difference operation is defined as $P_2 - P_1 = (P_2 \cup P_1)\backslash(P_1 \cap P_2)$. The concept of logical difference set in this paper resembles the symmetric difference set but it is not the same.

opposite P_1 and $-P_1$ links. With $l(-P_1) = -l(P_1)$, we have

$$\begin{aligned} l((P_1' \cup P_2') - P_1) &= l((P_1' \cup P_2') \cup (-P_1)) - l(O_l) = l(P_1') + l(P_2') + l(-P_1) \\ &= l(P_1') + l(P_2') - l(P_1) \end{aligned}$$

which is exactly Y in (3).

Lemma 2 shows that the logical difference set forms the shortest path in the modified graph where P_1 is replaced with $-P_1$.

Lemma 2
Given a directed graph $G(V, E)$ and pair (s, t) and let P_1 be the shortest path in this graph. We define $G(V, E')$ as the graph $G(V, E)$ for which the path P_1 is replaced with $-P_1$. The logical difference set $P_1' \cup P_2' - P_1$ between the optimal set of two link-disjoint paths $\{P_1', P_2'\}$ and the shortest path P_1 forms the shortest path P_2 from node s to node t in $G(V, E')$.

Proof
We will first prove that $P_2 = P_1' \cup P_2' - P_1$ is a complete path from s to t in $G(V, E')$, then we will prove that P_2 is the shortest path in $G(V, E')$.

Part A. From Lemma 1, the optimal set of two link-disjoint paths $P_1' \cup P_2'$ must contain either the first shortest path P_1 itself or some P_1 links on each of its two paths.

If $(P_1' \cup P_2') \supset P_1$, without loss of generality, suppose $P_1' = P_1$, then $O_l = P_1 \supset (-P_1)$. With the definition of logical difference set, we have $P_2 = ((P_1' \cup P_2') \cup (-P_1)) \backslash O_l = (P_1 \cup P_2' \cup (-P_1)) \backslash (P_1 \cup (-P_1)) = P_2'$. Hence P_2 must be a complete path from s to t.

If $P_1' \cup P_2'$ contains some P_1 links on each of its two paths, as $-P_1$ is the path from t to s in $G(V, E')$, and neither P_1' nor P_2' contains any $-P_1$ links, then the union $P_1' \cup P_2' \cup (-P_1)$ contains two cycles: one cycle consists of P_1' and $-P_1$, the other consists of P_2' and $-P_1$. When the set O_l is removed from the union set, the remaining links compose the logical difference set P_2. Hence P_2 must be a complete path from s to t.

Part B. Assume that the shortest path in $G(V, E')$ is $P_3 \neq P_2$, then we must have $l(P_3) < l(P_2)$. As $l(P_2) = l(P_1') + l(P_2') - l(P_1)$ we have $l(P_3) + l(P_1) < l(P_1') + l(P_2')$, which contradicts the assumption that $\{P_1', P_2'\}$ is the optimal set. □

3.3. LBA is loop-free

Many routing algorithms assume non-negative link weights to avoid a loop of negative length appearing on a path. However, negative link weights introduced to a graph in LBA will not cause loops in the routing process.

Theorem 2
Given a digraph $G(V, E)$ and source-destination pair (s, t) and let P_1 be the shortest path in this graph. The modified graph $G(V, E')$ is defined as the graph $G(V, E)$ for which P_1 is replaced with $-P_1$. A loop containing some negative link(s) in $G(V, E')$ will not have a negative length.

Proof
Assume $sv_1 \ldots v_i v_{i+1} \ldots v_n t$ is the shortest path P_1 from node s to node t in $G(V, E)$, as shown in Figure 5(a). The corresponding path $-P_1$ in $G(V, E')$ (Figure 5(b)) has a link $(v_{i+1} \rightarrow v_i)$ which appears on loop $P_1 = u_i v_{i+1} v_i u_i$. Suppose the loop P_1 has a negative length $l(P_1) = w$.

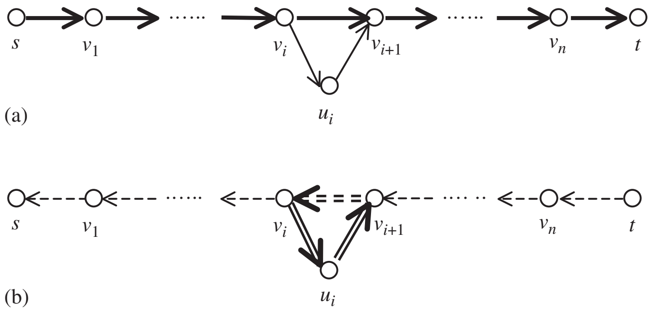

Figure 5. A loop contains some negative link; (a) The shortest path $P_1(s,t)$; and (b) A loop containing some $-P_1$ link.

$(u_i \to v_{i+1}) + w(v_{i+1} \to v_i) + w(v_i \to u_i) < 0$. Because $w(v_{i+1} \to v_i) = -w(v_i \to v_{i+1})$, we must have $w(v_i \to u_i) + w(u_i \to v_{i+1}) < w(v_i \to v_{i+1})$. Hence the sub-path $sv_1 \ldots v_i u_i v_{i+1}$ is shorter than the sub-path $sv_1 \ldots v_i v_{i+1}$. This contradicts the assumption that $sv_1 \ldots v_i v_{i+1} \ldots v_n t$ is the shortest path. □

3.4. *Optimality of the solution produced with LBA*

Theorem 3

Given a directed graph $G(V,E)$ and source-destination pair (s,t), the algorithm LBA returns the optimal set for the LPP problem.

Proof

Let P_1 be the shortest path in the original graph $G(V,E)$ found in step 1 of LBA and P_2 be the shortest path in the modified graph $G(V,E')$, found in step 3 of LBA. $\{P_1', P_2'\}$ is the solution set generated by LBA. The proof consists of three parts.

Part A (*Proof of link-disjointness*)

By construction of the solution set, we must have $P_1' \cap P_2' = \emptyset$.

Part B (*Proof of minimal total length*)

Suppose the optimal set of link-disjoint paths is $\{P_1'', P_2''\}$ instead of $\{P_1', P_2'\}$. According to Lemma 2, the logical difference set of $\{P_1'', P_2''\}$ with P_1 is the shortest path in the modified graph $G(V,E')$. This contradicts that P_2 is the shortest path in modified graph $G(V,E')$.

Part C (*Proof of loop-freeness*)

On Theorem 2, LBA is loop-free. Thus the solution set returned by LBA must be the optimal set. □

4. EXTENDING LBA TO MULTIPLE DIMENSIONS

The extension of LBA to multiple dimensions using SAMCRA [2] is called **MLBA** (Multiple-constrained LBA). A brief description of SAMCRA, which serves as the multiple-constrained shortest path routing algorithm in MLBA, is given in Section 4.1. The basic steps of MLBA (Multiple-constrained LBA) are presented in Section 4.2. The problems appearing in multiple dimensions are addressed in Section 4.3.

4.1. *Brief introduction of SAMCRA*

SAMCRA [2] is an exact multiple-constrained routing algorithm based on three concepts: (a) *non-linear path length*, (b) *k-shortest path routing*, and (c) *non-dominance*. The non-linear length function defined in (1) is necessary for exactness and implies that a sub-path of a shortest path is not necessarily a shortest itself. It is therefore necessary to keep track of *multiple* sub-paths at each intermediate node on the path between a pair of nodes. A (sub)-path P_1 is *dominated* by a (sub)-path P_2 if $w_m(P_1) \geqslant w_m(P_2)$, for $m = 1, \ldots, M$, with an inequality sign for at least one m. This operation reduces the search space and removes loops from a route when non-negative link weights are used.

4.2. *Operations of MLBA*

The basic steps of MLBA are the same as those for LBA except that the shortest path routing algorithm is replaced with SAMCRA in MLBA. We will illustrate the operation of MLBA with the example topology shown in Figure 6(a). For the sake of simplicity, we have assigned each link a two-dimensional weight vector, but it is also possible to use an M-dimensional weight vector ($M > 1$). The complexity of solving the MCLPP problem will increase with M, but as shown in Reference [2], the complexity may decrease (and even become polynomial) if M tends to infinity. To solve the MCLPP problem, we are required to find two link-disjoint paths from source node a to destination node b that both obey the constraints vector $\mathbf{L} = (20, 20)$. Among the solutions to MCLPP we prefer the one with the minimum total length. The shortest multiple-constrained path from node A to node B is the path $acdb$. Its path weight vector is

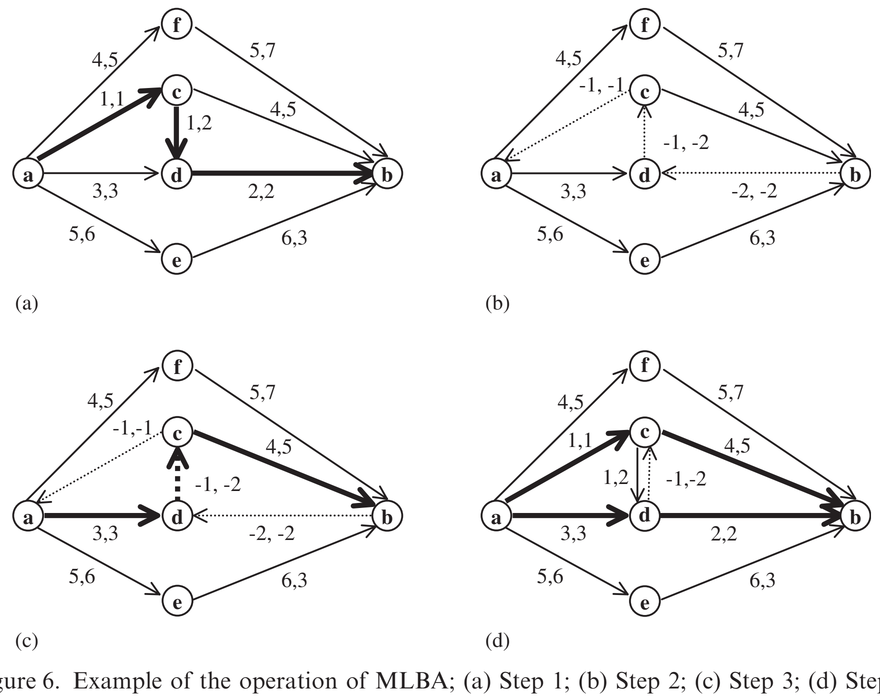

Figure 6. Example of the operation of MLBA; (a) Step 1; (b) Step 2; (c) Step 3; (d) Step 4.

(4,5). The optimal set of two shortest link-disjoint paths (according to (2)) in this topology is $\{acb, adb\}$, with path vectors (5,6) and (5,5) respectively and minimum total length $0.3+0.25=0.55$.

Now let us run MLBA on this topology. In *Step 1*, the shortest path $P_1 = acdb$ is found. In *Step 2*, the original graph is modified by replacing all the P_1 links with $-P_1$ links. In this case, each component of link weight vector of a $-P_1$ link is set negative. For instance, the link $c \rightarrow d$ with weight vector (1,1) is replaced with the link $d \rightarrow c$ with corresponding weight vector $(-1,-1)$. In *Step* 3, the shortest path in the modified graph found with SAMCRA is $P_2 = adcb$, with path weight vector $(3,3)+(-1,-1)+(4,5)=(6,6)$. In *Step 4*, the set O_l consisting of a pair of opposite P_1 and P_2 links $(c \rightarrow d)$ and $(d \rightarrow c)$ are removed from the union of P_1 and P_2. Then the optimal solution set $\{acb, adb\}$ is returned.

4.3. Problems due to the non-linear length in multiple dimensions

4.3.1. Loops caused by negative link weights. For $M = 1$, SAMCRA acts just like Dijkstra's algorithm, therefore MLBA reduces to LBA and negative link weights along $-P_1$ will not cause a loop in the routing process of MLBA. For $M > 1$, Theorem 2 still holds and a loop containing some $-P_1$ link(s) still has non-negative length. However, some of the components of the loop weight vector may be negative, causing MLBA to pass this loop a finite number of times. We will explain this looping through Figure 7, where each link possesses two link metrics. Suppose that the shortest path P_1 is *sadf*, depicted with bold lines in Figure 7(a). The link weights vector (x_1, x_2) of link $c \rightarrow d$ must be chosen to ensure that the path *sacdf* is longer than *sadf*, i.e.

$$\max \begin{bmatrix} w_1(s \rightarrow a) + w_1(a \rightarrow c) + x_1 + w_1(d \rightarrow f) \\ w_2(s \rightarrow a) + w_2(a \rightarrow c) + x_2 + w_2(d \rightarrow f) \end{bmatrix}$$

$$> \max \begin{bmatrix} w_1(s \rightarrow a) + w_1(a \rightarrow d) + w_1(d \rightarrow f) \\ w_2(s \rightarrow a) + w_2(a \rightarrow d) + w_2(d \rightarrow f) \end{bmatrix}$$

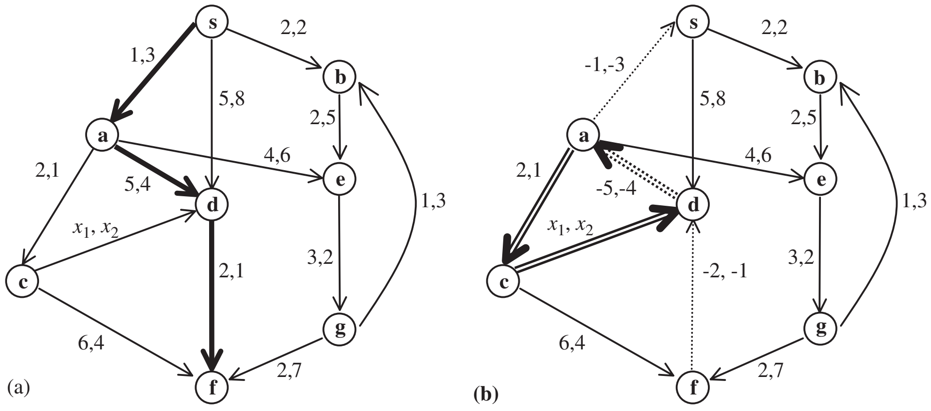

Figure 7. Non-dominance may fail to remove a loop in the case of $M > 1$; (a) The shortest path is *sadf*; (b) The loop *dacd* contains a negative link $d \rightarrow a$.

Numerically,

$$\max \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} > \max \begin{bmatrix} w_1(a \to d) - w_1(a \to c) \\ w_2(a \to d) - w_2(a \to c) \end{bmatrix} = \begin{bmatrix} 5-2 \\ 4-1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \tag{4}$$

After Step 2 of MLBA is executed, there appears a loop $P_1 = dacd$ shown with double lines in Figure 7(b), containing the link $d \to a$ with negative link weights $(-5, -4)$.

If Equation (4) holds and each component of vector (x_1, x_2) is greater than 3, then the sub-path *sdacd* will be dominated by the direct link $s \to d$ with weight vector (5,8) and will be removed by the non-dominance check in SAMCRA. However, if Equation (4) holds but one component of (x_1, x_2) is not greater than 3, say $x_1 < 3$, $x_2 > 3$, then

$$\begin{bmatrix} w_1(P_1) = w_1(d \to a) + w_1(a \to c) + w_1(c \to d) = (-5) + 2 + x_1 < 0 \\ w_2(P_1) = w_2(d \to a) + w_2(a \to c) + w_2(c \to d) = (-4) + 1 + x_2 > 0 \end{bmatrix}$$

where $(w_1(P_1),\ w_2(P_1))$ is the path vector of the loop P_1. In this case, the sub-path *sdacd* is not dominated by the link $s \to d$, although $l(P_1) > 0$. Hence, loops can occur in MLBA that continue until one of the constraints is violated. Unfortunately, checking all paths to assure that they are loop-free is computationally too expensive.

As mentioned in Section 2, in Suurballe's algorithm [5] and the S–T algorithm [1], a transformation of link weights $w'(u \to v) = w(u \to v) + d(u) - d(v)$ is applied to each link, where $d(u)$ is the distance from source node s to node u on the shortest path tree. This transformation is made to guarantee that the links on the shortest path tree have zero link weights and those links not on the tree have link weights greater than zero in the modified graph. However, an artifact of a non-linear length is that subsections of shortest paths are not necessarily shortest paths [2, 3]. Consequently, for $M > 1$, Suurballe's transformation cannot ensure non-negative link weights and loops may emerge.

4.3.2. Total length of the solutions produced with MLBA. We assume for the moment that the constraints are large enough such that all paths are feasible. If $M = 1$, it is proved in Section 3.4 that the solution set $\{P_1', P_2'\}$ produced with MLBA has the minimum total length. With the total length defined in (2), Lemma 1 in Section 3 still holds for $M > 1$. The optimal solution set of two link-disjoint multiple-constrained paths with minimum total length either contains the first shortest path P_1 itself or some P_1 links on each of its two paths. Also, the optimal set $\{P_1', P_2'\}$ still obeys Property 1. Unfortunately, the logical difference set $(P_1' \cup P_2') - P_1$ is not necessarily the shortest path P_2 in the modified graph, since $l((P_1' \cup P_2') - P_1) = l(P_1') + l(P_2') - l(P_1)$ does not necessarily hold for $M > 1$. Hence, Lemma 2 may not hold for $M > 1$ and the solution set constructed based on P_1 and P_2 is not necessarily the optimal set with minimum total length. Moreover, the solution set may also violate the constraints or a feasible solution may not be found.

5. DIMCRA

In the previous section, we have shown that it is not trivial to extend LBA to multiple dimensions. Due to the problems existing in MLBA, we propose a heuristic algorithm DIMCRA (link-disjoint multiple constraints routing algorithm) for the MCLPP problem.

5.1. Operations of DIMCRA

DIMCRA (G, s, t): Given a directed graph $G(V, E)$, a constraint vector $\mathbf{L}$ and *a source-destination pair* (s, t),

Step 1. Find the shortest path P_1 obeying $\mathbf{L}$ with SAMCRA; if P_1 does not exist, then stop;

Step 2. Reverse the direction of all the links on the shortest path P_1, and set the sign of their link weights zero, $w_m(v \to u) = 0$, $\forall (u \to v) \in P_1$ and $m = 1, \ldots, M$. A modified graph G' is created;

Step 3. Find the shortest path P_2 constrained by $2\mathbf{L}$ in the modified graph G' with SAMCRA; if P_2 does not exist, then stop;

Step 4. Make the union of P_1 and P_2, remove from the union the P_1 links whose reversed links appear on P_2, and vice versa, then group the remaining links into a set of two paths $\{P'_1, P'_2\}$, i.e. $P'_1 \cup P'_2 = (P_1 \cup P_2) \backslash (P_1 \tilde{\cap} P_2)$.

Step 5. Check the length of each path in the set $\{P'_1, P'_2\}$. If the path $P'_i (1 \leqslant i \leqslant 2)$ violates the constraints, then update the modified graph G' by removing the link set $P'_i \backslash (P'_i \cap P_1)$ from it, and go to Step 3. Otherwise stop and return the current solution set $\{P'_1, P'_2\}$.

Compared to MLBA, DIMCRA uses a different transformation to create the modified graph. In Step 2 of DIMCRA, the shortest path links are still reversed in direction but the corresponding direction-reversed links are assigned with *zero* link weight vectors instead of negative ones. Therefore the loop problem caused by negative link weights that mainly destroys the efficiency of MLBA is bypassed. In MLBA, P_2 is required to obey the constraints, *which may cause some feasible sets to be ignored by MLBA*. In fact, when $\mathbf{w}(P_2) > \mathbf{L}$, if P_2 contains no reversed P_1 link(s), then $\{P'_1, P'_2\}$ is actually $\{P_1, P_2\}$ and cannot be a feasible set. But if P_2 contains some reversed P_1 link(s), it is possible that $\{P'_1, P'_2\}$ is a feasible set, for instance, $l(P_1) = 0.6$, $l(P'_1) = 0.8$, $l(P'_2) = 0.9$, and $l(P_2) = 1.1$. However, if $\mathbf{w}(P_2) > 2\mathbf{L}$, then we must have $\mathbf{w}(P'_1 + P'_2) = \mathbf{w}(P_2 + P_1 - P_{1r}) > \mathbf{w}(P_2) \geqslant 2\mathbf{L}$, where P_{1r} denotes the set of P_1 links whose reversed links appear on P_2, and P_{1r} must be a proper subset of P_1. Therefore, in Step 3 of DIMCRA, the constraint check on path P_2 in SAMCRA is performed with $2\mathbf{L}$ as the constraints vector, otherwise a feasible solution set may not be found. We have also added an extra step, *Step 5* of DIMCRA, to check that the constraints are obeyed. If only with $\mathbf{w}(P'_1 + P'_2) \leqslant 2\mathbf{L}$ DIMCRA does not always ensure both paths within constraints, i.e. $\mathbf{w}(P'_1) \leqslant \mathbf{L}$ and $\mathbf{w}(P'_2) \leqslant \mathbf{L}$. Hence Step 5 of DIMCRA checks both paths in the solution set returned at Step 4. If each of them obeys the constraints, DIMCRA will return the solution set and stop. On the other hand, if either of them does not obey the constraints, DIMCRA is redirected to Step 3 to continue the search for a feasible set. In Step 3, if no P_2 exists, DIMCRA will stop with no solution. We will illustrate the operation of DIMCRA with the following examples.

Example 1

Consider the example graph in Figure 8(a). We are required to find a set of two link-disjoint paths between a and b, each within the constraints $\mathbf{L} = (20, 20)$ and preferably with the minimum total length. In *Step 1*, the shortest path $P_1 = acdb$ is found. In *Step* 2, each P_1 link is reversed and is assigned with zero link weights. In *Step 3*, the shortest path in the modified graph G' is found as $P_2 = adcb$, with path vector $(3, 3) + (0, 0) + (4, 5) = (7, 8)$, as shown with bold lines in Figure 8(c). In *Step 4*, only for the P_1 link $c \to d$, its reversed link $d \to c$ appears on P_2 and *vice versa*. Thus these two links are removed from the union of P_1 and P_2, and the remaining links are grouped into a set of two paths $\{P'_1, P'_2\} = \{acb, adb\}$, shown with bold lines. In *Step* 5,

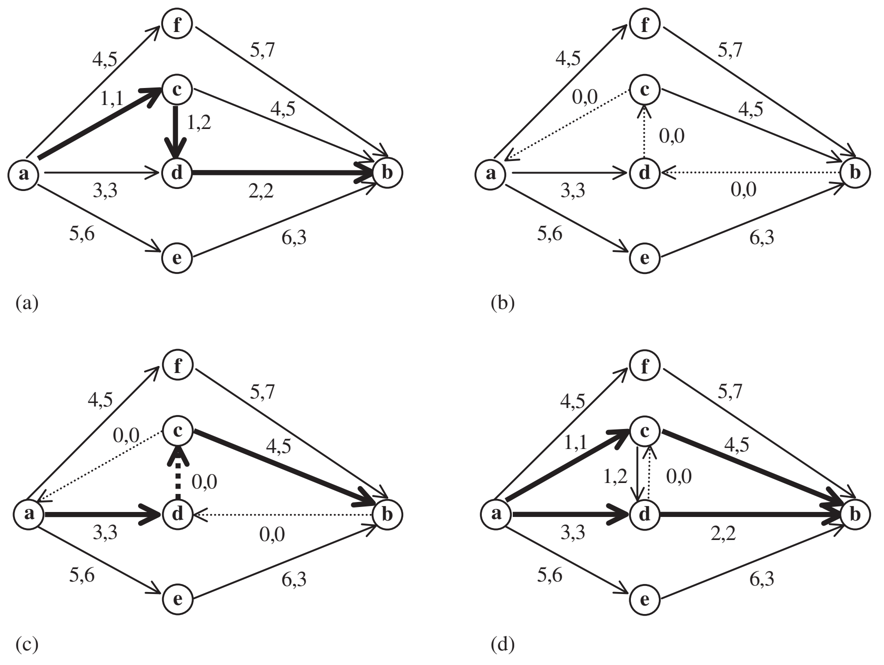

Figure 8. Example 1 of the operation of DIMCRA; (a) Step 1; (b) Step 2 (c) Step 3; (d) Step 4.

the constraints check is executed on both paths. As each of them obeys the constraints, DIMCRA stops. In this case, the optimal solution set of $\{acb, adb\}$ is returned by DIMCRA. The solution set that would have been returned by RF, is not optimal.

Example 2

Consider the graph in Figure 9(a), which is the same as in the previous example except that the link $e \rightarrow b$ is assigned a different vector (2,1). The constraints remain the same. In this example the optimal set of two link-disjoint multiple-constrained paths is still the set $\{adb, acb\}$ with path vectors (5,5) and (5,6) respectively, and the minimum total length $5/20 + 6/20 = 0.55$. In Step 3, the shortest path in the modified graph is found as $P_2 = aeb$ with path vector (7,7), shown in Figure 9(c). In Step 4, as for each P_1 link, its reversed link does not appear on P_2, or *vice versa*, the solution set $\{P_1', P_2'\}$ is constructed as $\{acdb, aeb\}$, exactly P_1 and P_2 themselves. The total length of $\{acdb, aeb\}$ is $5/20 + 7/20 = 0.6$. In this example, DIMCRA failed to return the optimal set, but DIMCRA's solution set is close to the optimal one and both paths obey the constraints. RF would have returned the same solution.

Example 3

We again consider Example 2 except with different constraints (6,6). Running DIMCRA, we obtain the same results as in Example 2 (for Steps 1–4). But in *Step 5*, when the constraints check is made on each path in the solution set $\{P_1', P_2'\} = \{acdb, aeb\}$, the longer path $P_2' = aeb$ with path vector (7,7) does not obey the constraints. This means that the currently built solution set is not feasible. The links that only appear on $P_2' = aeb$, i.e. link $a \rightarrow e$ and $e \rightarrow b$, are removed from the modified graph shown in Figure 9(b). The updated modified graph is shown

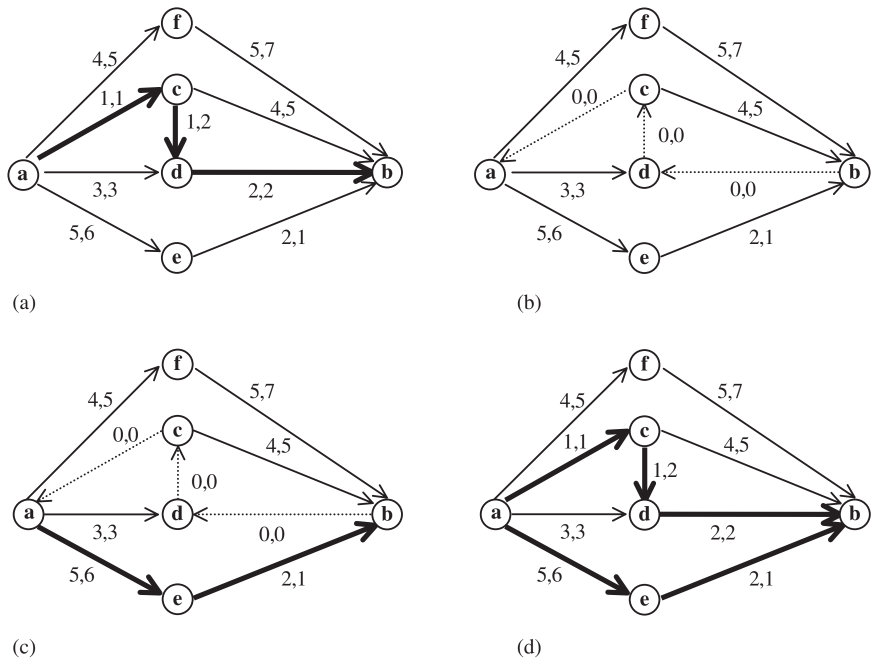

Figure 9. Example 2 of the operation of DIMCRA; (a) Step 1; (b) Step 2; (c) Step 3; (d) Step 4.

in Figure 10(a) and DIMCRA is redirected to Step 3. In Step 3, a shortest path in the updated modified graph is found as $P_2 = adcb$, depicted in Figure 10(b). In Step 4, the solution set is $\{acb, adb\}$, as shown in Figure 10(c). At last, in Step 5, each path in the current solution set obeys the constraints. The optimal set $\{acb, adb\}$ is returned and DIMCRA stops. RF would have failed to return a solution.

In Step 3 the constraints are set to 2**L**. With these modified constraints, if the shortest path P_2 in the modified graph violates the constraints **L** but obeys 2**L**, it can be returned by SAMCRA in Step 3, hence a feasible set related to such kind of P_2 will not be ignored, as illustrated in Example 3. Moreover, if a path P_2 does not exist in the updated modified graph, DIMCRA will stop. Thus DIMCRA will not bounce back and forth between Steps 3 and 5.

With the constraints check on each path in the solution set, Step 5 guarantees that DIMCRA returns a feasible set of two link-disjoint multiple-constrained paths, as illustrated in the above examples.

However it may occur, as illustrated in Figure 11, that DIMCRA cannot return a feasible set even if there exists one. The RF method also fails to return the feasible set in this case.

5.2. Properties of DIMCRA

As proved in Section 3, the way to construct a solution set by reversing the shortest path P_1, finding a shortest path P_2 in the modified graph and constructing the solution set based on these two shortest paths P_1 and P_2 guarantees the disjointness of the two paths in the solution set. Setting the direction-reversed P_1 links with zero link weights guarantees the loop-freeness of

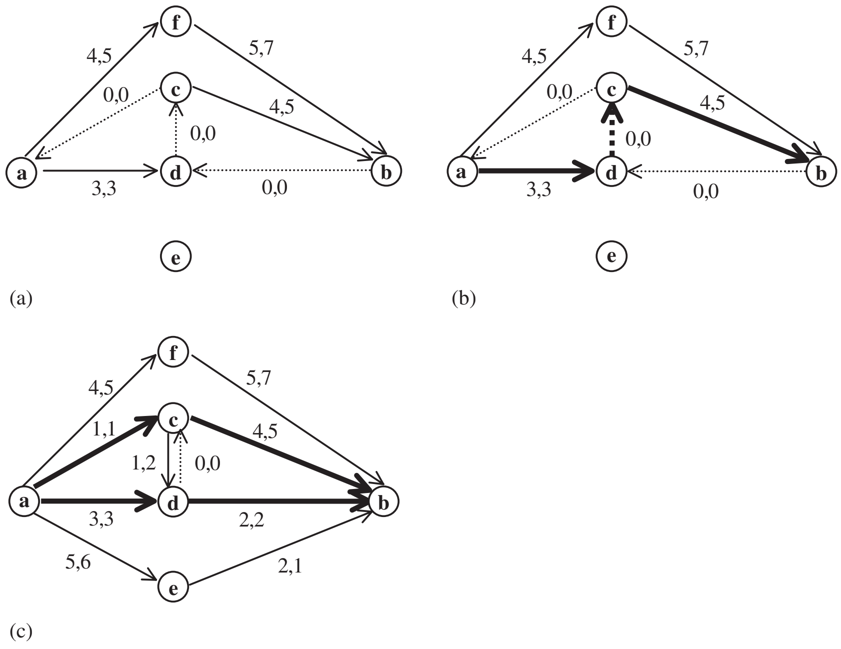

Figure 10. Example 3 of the operation of DIMCRA; (a) Step 5; (b) Step 3; (c) Step 4.

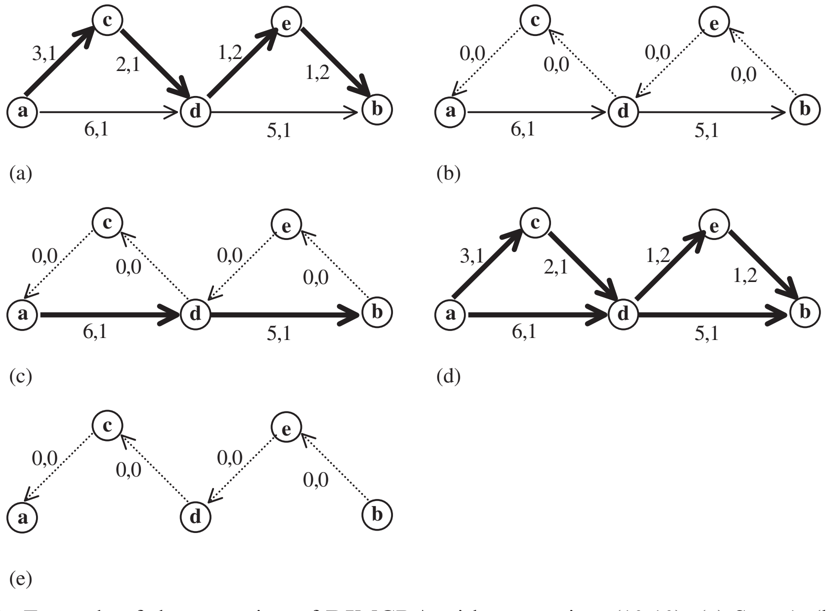

Figure 11. Example of the operation of DIMCRA with constraints (10,10); (a) Step 1; (b) Step 2; (c) Step 3; (d) Step 4; (e) Step 5.

DIMCRA. For, if no negative link weights are used in a graph, a loop can be avoided by the non-dominance check in SAMCRA. Comparing with the operation of setting direction-reversed P_1 links negative, the operation of setting such reversed P_1 links with zero link weights still encourages the choice of such reversed P_1 links on a path but with less intensity.

Unfortunately DIMCRA does not always find the set of feasible link-disjoint paths. Hence, it may be possible to further optimize DIMCRA, such that it can guarantee to always find a set of feasible link-disjoint paths, if they exist. However, DIMCRA in its current state is better than the RF method (as was indicated in the examples). Both methods return the same solution when $\{P_1', P_2'\} = \{P_1, P_2\}$ and $P_1 \cap P_2 = \emptyset$. In all other cases DIMCRA either returns a more optimal solution than RF or RF does not find a solution where DIMCRA does. Since, to our knowledge, no other algorithms for solving MCLPP exist, the performance of DIMCRA is difficult to assess.

6. CONCLUSIONS

The link-disjoint path problem occurs in network design where aspects as survivability, load balancing and network resource utilization are strived for. This problem has barely been investigated in the QoS routing context where a path is characterized by multiple metrics. A simple algorithm for solving the LPP problem for is presented in this paper. The problems surrounding the extension of this simple algorithm to multiple dimensions are discussed. A heuristic algorithm DIMCRA is proposed to find link-disjoint multiple-constrained paths between a pair of source and destination nodes. If DIMCRA returns a link-disjoint pair of paths they always obeys the constraints. However, DIMCRA's solution is not necessarily optimal in terms of minimizing the total length of the returned paths or guaranteeing to always find the feasible set. Its performance however is better than the simple remove-find method.

Some open issues remain, namely: making DIMCRA exact whilst still efficient, allowing maximally disjoint paths or bridges and simulating the performance.

ACKNOWLEDGEMENTS

The research for this paper was conducted when Y. Guo was a visiting scientist at Delft University of Technology sponsored by NUFFIC (Netherlands organization for cooperation in higher education) and CSC (Chinese scholarship council).

REFERENCES

1. Suurballe JW, Tarjan RE. A quick method for finding shortest pairs of disjoint paths. *Networks* 1984; **14**:325–333.
2. Van Mieghem P, De Neve H, Kuipers FA. Hop-by-hop quality of service routing. *Computer Networks* 2001; **37**(3–4):407–423.
3. De Neve H, Van Mieghem P. TAMCRA: a tunable accuracy multiple constraints routing algorithm. *Computer Communications* 2000; **23**(7):667–679.
4. Bhandari R. Optimal diverse routing in telecommunication fiber networks. *Proceedings of IEEE INFOCOM '94*. Toronto, Ontario, Canada, vol. 3, June, 1994; 1498–1508.
5. Suurballe JW. Disjoint paths in a network. *Networks* 1974; **4**:125–145.
6. Garey MR, Johnson DS. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. Freeman: San Francisco, 1979.

7. Wang Z, Crowcroft J. QoS routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications* 1996; **14**(7):1228–1234.
8. Castanon DA. Efficient algorithms for finding the K best paths through a trellis. *IEEE Transactions on Aerospace and Electronic Systems* 1990; **26**(2):405–410.
9. Cheng C, Kumar SPR, Garcia-Luna-Aceves JJ. A distributed algorithm for finding K disjoint paths of minimal total length. *Proceedings of 28th Annual Allerton Conference on Communication, Control, and Computing*. Urbana, Illinois, October, 1990.
10. Ogier RG, Rutenburg V, Shacham N. Distributed algorithms for computing shortest pairs of disjoint paths. *IEEE Transactions on Information Theory* 1993; **39**(2):443–455.
11. Sidhu D, Nair R, Abdallah S. Finding disjoint paths in networks. *ACM SIGCOMM Computer Communication Review. Proceedings of the Conference on Communications Architecture & Protocols*, vol. 21, No. 4. August, 1991.
12. Wolf JK, Viterbi AM, Dixon GS. Finding the best set of K paths through a trellis with application to multitarget tracking. *IEEE Transactions on Aerospace and Electronic Systems* 1989; **25**(2):287–296.
13. Diestel R. Graph Theory. *Graduate Texts in Mathematics*. Springer: New York, 1997.
14. Ford LR, Fulkerson DR. *Flows in Networks*. Princeton University Press: Princeton, NJ, 1962.
15. Papadimitriou CH, Steiglitz K. *Combinatorial Optimization–Algorithms and Complexity*. Prentice–Hall, Inc.: Englewood Cliffs, NJ, 1982.
16. Kar K, Kodialam M, Lakshman TV. Routing restorable bandwidth guaranteed connections using maximum 2-route flows. *Proceedings of IEEE INFOCOM'02*, 2002.
17. Kodialam M, Lakshman TV. Dynamic routing of bandwidth guaranteed tunnels with restoration. *Proceedings of IEEE INFOCOM'00*, 2000.
18. Kodialam M, Lakshman TV. Restorable dynamic quality of service routing. *IEEE Communications Magazine* 2002; 72–81.
19. Liang W. Robust routing in wide-area WDM networks. *Proceedings of 15th Int'l Parallel and Distributed Processing Symposium*, San Francisco, April 2001.
20. Shaikh SZ. Span-disjoint paths for physical diversity in networks. *Proceedings of IEEE Symposium on Computers and Communications* 1995; 127–133.
21. Li C-L, McCormick ST, Simchi-Levi D. The complexity of finding two disjoint paths with min–max objective function. *Discrete Applied Mathematics* 1990; **26**(1):105–115.
22. Sen A, Shen BH, Bandyopadhyay S, Capone JM. Survivability of lightwave networks—path lengths in WDM protection scheme. *Journal of High Speed Networks* 2001; **10**(4):303–315.
23. Ho P-H, Mouftah HT. Issues on diverse routing for WDM mesh networks with survivability. *Proceedings of 10th International Conference on Computer Communications and Networks* 1997; 61–66.
24. Tanaka Y, Rue-Xue F, Akiyama M. Design method of highly reliable communication network by the use of matrix calculation. *IEICE Transactions* 1987; **J70-B**(5):551–556.
25. Oki E, Yamanaka N. A recursive matrix-calculation method for disjoint path search with hop link number constraint. *IEICE Transactions on Communications* 1995; **E78-B**(5):769–774.
26. Lee SW, Wu CS. A k-best paths algorithm for highly reliable communication networks. *IEICE Transactions on Communications* 1999; **E82-B**(4):586–580.
27. Bejerano Y, Breitbart Y, Orda A, Rastogi R, Sprintson A. Algorithms for computing QoS paths with restoration. *Proceedings of IEEE INFOCOM'03*. April, 2003.
28. Italiana GF, Rastogi R, Yener B. Restoration algorithms for virtual private networks in the hose model. *Proceedings of IEEE INFOCOM'02*, 2002.
29. Gummadi KP, Pradeep MJ, Murthy CSR. An efficient primary-segmented backup scheme for dependable real-time communication in multihop networks. *ACM/IEEE Transactions on Networking* 2003; **11**(1):81–94.
30. Lo CC, Chuang BW. A novel approach of backup path reservation for survivable high-speed networks. *IEEE Communications Magazine*, March 2003.
31. Taft-Plotkin N, Bellur B, Ogier R. Quality-of-Service using maximally disjoint paths. *Proceedings of IWQoS (International Workshop on Quality-of-Service)*, June 1999.

AUTHORS' BIOGRAPHIES

Yuchun Guo received her BS and MS degrees from the Department of Electronics, Beijing Normal University, Beijing China in 1990 and 1993. After that she has been with the School of Electrical and Information Engineering, Northern Jiaotong University, Beijing China. She spent an academic year at Delft University of Technology, the Netherlands, as a visiting scientist, sponsored by NUFFIC (Netherlands organization for cooperation in higher education) and CSC (Chinese scholarship council). She started her PhD study in October 2002 in Northern Jiaotong University with a focus on network measurement and modelling.

Fernando A. Kuipers received the MSc degree in Electrical Engineering at the Delft University of Technology in June 2000. Currently he is working towards his PhD degree in the NAS group at the same faculty. He is also a member of the DIOC (interdisciplinary research center) on the Design and Management of Infrastructures, where he is taking part in the Telecommunications project. His PhD work mainly focuses on the algorithmic aspects and complexity of Quality of Service (QoS) routing. His research interests also involve the dynamics of QoS routing and Next Generation Infrastructures (NGI).

Piet F. A. Van Mieghem is professor at the Delft University of Technology with a chair in telecommunication networks and chairman of the basic unit Network Architectures and Services (NAS). His main research interests lie in new Internet-like architectures for future, broadband and QoS-aware networks and in the modelling and performance analysis of network behavior. Professor Van Mieghem received a Master's and PhD in Electrical Engineering from the K.U. Leuven (Belgium) in 1987 and 1991, respectively. Before joining Delft, he worked at the Interuniversity Micro Electronic Center (IMEC) from 1987 to 1991. From 1992–1993, he was a visiting scientist at MIT in the department of Electrical Engineering. During 1993–1998, he was a member of the Alcatel Corporate Research Center in Antwerp where he was engaged in performance analysis of ATM systems and in network architectural concepts of both ATM networks (PNNI) and the Internet.