A Heuristic for Widest Edge-disjoint Path Pair Lexicographic Optimization

Pedro Cruz*, Teresa Gomes*†
*Department of Electrical and Computer Engineering
University of Coimbra
†INESC Coimbra, Rua Antero de Quental 199,
3000-033 Coimbra, Portugal
Email: pedrocruz.main@gmail.com, teresa@deec.uc.pt

Deep Medhi‡
‡Computer Science & Electrical Engineering Department
School of Computing and Engineering
University of Missouri–Kansas City
Kansas City, Missouri 64110, USA
Email: DMedhi@umkc.edu

***Abstract*—Telecommunication services are pervasive in today's human activity and are required to offer reliable and quality-of-service(QoS)-aware guaranteed services. In global path protection, the working path between a source and a destination can be protected by a backup path, which ensures data transfer in the event of a failure that makes the working path to be unavailable. Multipath and disjoint routing may require the calculation of disjoint paths maximizing the total bandwidth of the path pair (or set of paths) or the calculation of maximum-bandwidth disjoint paths.**

In this paper, a lexicographic optimization problem for obtaining maximum-bandwidth disjoint paths, and then maximizing the bandwidth of the widest path in the pair, is formalized. An effective heuristic for addressing this problem is presented.

***Index Terms*—widest path; disjoint routing; lexicographic optimization**

I. Introduction

Telecommunication services are pervasive in today's human activities and are required to offer reliable and quality-of-service-aware guaranteed services. In global path protection, the working path between a source and a destination can be protected by a backup path, which ensures data transfer in the event of a failure that makes the working path to be unavailable. This can be accomplished by finding a pair of edge-disjoint paths if the nodes are considered not to fail or a pair of node-disjoint paths if node failures are to be taken into account. Our work here focuses on finding a pair of widest edge-disjoint paths lexicographically.

The literature on shortest path and related problems such as disjoint paths is extensive. The calculation of a set k ($k \geq 2$) of edge-disjoint paths with a minimum total additive cost (min-sum cost) was proposed by Suurballe [13]. The calculation of edge-disjoint path pairs from a source node to every other node such that each pair has a minimum additive cost was proposed in [14]. Bhandari [4] discussed possible implementations of Suurballe's algorithm for calculating an edge-disjoint path pair of min-sum cost. The underlying algorithm in this approach was based on Dijkstra's shortest paths algorithm [5] or a Breadth First Search (BFS). Moreover, the author proposed an alternative algorithm (Bhandari's algorithm), which can be more efficient than Suurballe's algorithm for calculating a min-sum edge-disjoint path pair. These problems can be solved in polynomial time; however, if additional restrictions are added, the problem may become NP-Complete. In [3], the authors examined the complexity of different variants of the min-sum edge-disjoint paths problem and proposed heuristics to address them.

For certain telecommunication services, the path with the most available capacity, known as the *widest path*, is of importance to ensure quality of service (QoS). The bandwidth (or capacity) of a path is defined by the bandwidth of the bottleneck arc of the path, i.e., the arc with the minimum bandwidth in the path. Pollack [11] first made the astute observation that several shortest path algorithms can be adapted for the calculation of the widest path. An algorithm for the calculation of paths with maximum capacity for all node pairs was proposed in [6].

Wang and Crowcroft [16] proposed an algorithm for the calculation of the shortest path among all paths of maximum bandwidth; this was designated as the *shortest-widest* path in [15]. An algorithm for finding the shortest path with bandwidth guarantee can be found in [8, Algorithm 17.1, page 592]; this is sometimes referred to as the *constrained shortest path* problem.

For network protection, multi-path and disjoint routing may require the determination of a set of edge-disjoint paths that maximize the sum of the bandwidth of a set of edge-disjoint paths. Shen et. al. [12] tackled two inter-related problems: 1) the calculation of a disjoint path pair such that the sum of the bandwidth in the path pair is larger than a bandwidth guarantee b_0, and 2) the calculation of two disjoint path pairs such that each of the paths in the pair satisfy (different) specific bandwidth guarantees. The authors proved that the two problems are NP-Complete and proposed an integer linear programming (ILP) formulation for solving them; additionally they also proposed two heuristics. An interesting related problem is considered in [7]: given a network, a source-destination pair s and t, and a bandwidth guaranteed value b_0, how to find a pair of disjoint paths such that the sum of the bandwidth in the path pair is larger than b_0 while minimizing the total additive cost of the path pair. They designated this problem as finding the shortest pair of disjoint paths with bandwidth guarantee (SPDP-BG). This problem was shown to be NP-

Complete and a heuristic for solving it was also proposed [7].

A maximum-bandwidth disjoint path pair is obtained that maximizes the minimum bandwidth of the paths that define the pair. In [9] and [15], the authors propose algorithm MADSWIP, which solves exactly and lexicographically the problem of calculating maximum-bandwidth maximally disjoint paths, and minimizes delay (or cost) as a secondary objective, from a source node to every other node.

The contribution of our work is different from the previous works; specifically, we present the formulation of a lexicographic optimization problem for obtaining maximum-bandwidth edge-disjoint paths and then maximize the bandwidth of the widest path in the pair. We also present an effective heuristic, with complexity identical to Dijkstra's algorithm, for solving this problem.

This paper is organized as follows. In Section II, we present the preliminaries for the model formulation presented in Section III. In Section IV, the resolution method is presented, followed by the algorithm description in Section IV-A and by an example illustrating the algorithm in Section IV-B. The performance of the heuristic is evaluated in Section V. The paper ends in Section VI.

II. Definitions and notation

Let $(\mathcal{N}, \mathcal{A})$ denote the graph G defining a network topology, where $\mathcal{N} = \{v_1, v_2, \ldots, v_n\}$ is the finite set of nodes and $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$ is the finite set of arcs, such that each arc is a pair of different nodes. No parallel arcs are considered. If all the pairs are ordered (unordered), the graph is said to be directed (undirected). Hereafter, and unless something explicit is said, the graph is supposed to be directed. Nevertheless, we will seek to obtain edge-disjoint paths, where edge refers to a pair of symmetrical pair of arcs in G.

A path p from s to t ($s, t \in \mathcal{N}$) is defined by an alternating sequence of nodes and arcs, $\langle s = v'_1, a'_1, v'_2, \ldots, a'_{r-1}, v'_r = t \rangle$, where: $a'_k \in \mathcal{A}$ for any $k = 1, \ldots, r-1$ and $v'_k \in \mathcal{N}$ for any $k = 1, \ldots, r$; $a'_k = (v'_k, v'_{k+1})$ for any $k = 1, \ldots, r-1$. The sub-path of p from node i to node j will be designated by p_{ij}.

A path from s to t in $(\mathcal{N}, \mathcal{A})$ is a *loopless path* if and only if all its nodes are different. A *cycle* is a path from a node to itself such that all nodes are different except the first which is identical to the last one. Let $\mathcal{P}_{st}$ designate the set of (loopless) paths from s to t. The set of edges of path p ($p \in \mathcal{P}_{st}$) is represented by $\mathcal{A}_p$. The set of edge-disjoint path pairs from s to t is designated by $\bar{\mathcal{P}}_{st} = \{(p, q) : \mathcal{A}_p \cap \mathcal{A}_q = \emptyset, p \neq q, p, q \in \mathcal{P}_{st}\}$.

Let p_{ij} be a path from node i to j; the *concatenation* of paths p_{ij} and p_{jl} is the path, $p_{ij} \diamond p_{jl}$, from i to l, which coincides with p_{ij} from i to j and with p_{jl} from j to l.

The bandwidth of path p is:

$$b(p) = \min_{(i,j) \in p} b_{ij} \tag{1}$$

where b_{ij} is the bandwidth of of arc (i, j).

The widest path from s to t is:

$$p^* = \arg \max_{p \in \mathcal{P}_{st}} b(p) \tag{2}$$

The bandwidth of an edge-disjoint path pair (p, q), for a specific node pair s-t, can be defined as:

$$b_m(p, q) = \min[b(p), b(q)] \tag{3}$$

Defining

$$(p^*, q^*) = \arg \max_{(p,q) \in \bar{\mathcal{P}}_{st}} b_m(p, q) \tag{4}$$

$b_m(p^*, q^*)$ corresponds to the maximum capacity of a path from s to t, for which a node disjoint protection path can be obtained. This path pair will be designated as maximum-bandwidth path pair.

The maximum bandwidth of the widest path of an edge-disjoint path pair (p, q), for a specific node pair s-t, can be defined as:

$$b_M(p, q) = \max[b(p), b(q)], \text{ with } b_m(p, q) > 0 \tag{5}$$

III. Problem formalization

Finding the path pair of maximum $b_m(p, q)$, while maximizing the bandwidth of the widest path in the pair (from s to t), can be achieved by solving the problem that will be formalized next. Let $f_1 = 1/b_m(p, q)$ and $f_2 = 1/b_M(p, q)$, with $(p, q) \in \bar{\mathcal{P}}_{st}$.

The problem addressed in this work is the lexicographic optimization of $f_i, i = 1, 2$:

$$(p^*, q^*) = \arg \min_{(p,q) \in \bar{\mathcal{P}}_{st}} f_i, i = 1, 2 \tag{6}$$

The minimization of f_1 and f_2 correspond to the maximization of $b_m(p, q)$ and $b_M(p, q)$, respectively.

Let us assume, without loss of generality, that $b_m(p, q) = b(p)$ and that $b(q) \geq b(p)$. If more than one edge-disjoint path with p exists or if more than one path with bandwidth equal to $b(p)$ exists, problem f_1 has multiple optimal solutions (multiple path pairs that maximize $b_m(p, q)$). Solving the problem in Equation (6) allows us to obtain the widest possible path q for the pair of widest bandwidth. This optimization problem—Widest Edge-disjoint Path Pair Lexicographic Optimization—will be denoted by WEDLO. This can be of practical interest in communication networks such as for MPLS-based virtual private network services

- when using path protection, for ensuring the selected *protection path* has spare bandwidth to take into account traffic fluctuations that take place in the event of failure;
- when using path protection, for ensuring the selected *active path* has spare bandwidth, to guarantee the desired QoS and that there is some extra available capacity to absorb traffic fluctuations.

IV. Proposed resolution approach

The main idea behind the resolution approach of the problem in Equation (6) is the use of dual labels at the nodes: each node will have primary and secondary node labels.

The algorithm first calculates the widest path $\bar{p}$ from node s to node t. Then it changes the network into G' in a way analogous to the procedure in [9] and [15], which is similar to the network transformation for edge-disjoint shortest paths

in [4]: the directed arcs in the path from s to t are removed and the reversed (symmetrical) arcs to the removed arcs are added with infinite bandwidth (in practical terms, a value larger than the bandwidth of the widest arc in the network).

To obtain the path pair that maximizes b_m, all that is required is to calculate $\bar{p}'$, the widest path in the transformed network G'. The interlacing arcs, that is, the reversed arcs of $\bar{p}$ that appear in $\bar{p}'$ are removed, and the remaining arcs define the widest edge-disjoint path pair (p, q).

When the calculation of the widest path in the transformed network G' begins, the source node has its labels equal to the bandwidth of the path $\bar{p}$. In the modified graph, the widest path is determined using the primary node labels. If interlacing arcs exist in $\bar{p}'$, they are removed and the resulting sub-paths between interlacing chains will belong alternately to the resulting path pair p and q.

During $\bar{p}'$ calculation, whenever the non-permanently labeled node presently with the largest bandwidth label (in G') is selected, let it be node v_k, then node v_k becomes a permanently labeled node. Let the predecessor of v_k be v_{k-1}. The secondary label of v_k takes the value of the secondary label of its predecessor, v_{k-1}.

Consider that the new permanently labeled node is v_{k_1}. Let us assume that the arc (v_{k_1-1}, v_{k_1}) belongs to the reversed $\bar{p}$ path and if the predecessor of v_{k_1-1}, let it be v_{k_1-2}, is such that (v_{k_1-2}, v_{k_1-1}) does not belong to the reversed $\bar{p}$ path, then the primary and secondary labels of node v_{k_1} swap, because arc (v_{k_1-1}, v_{k_1}) is the first reversed arc permanently added to the tree of widest paths, and is a candidate to be part of $\bar{p}'$.

Consider that this is the first time a reversed arc has appeared as a possible candidate to be on $\bar{p}'$. If this is the case then the sub-path of $\bar{p}'$ from s to v_{k_1-1}, $\bar{p}'_{sv_{k_1-1}}$, is the candidate sub-path of one of the paths of the path pair (say p) and the sub-path of $\bar{p}$ from s to the exit node (v_{x_1}) of the chain of reversed arcs (in $\bar{p}'$) starting in node v_{k_1-1}, $\bar{p}_{sv_{x_1}}$, will belong to the other path of the path pair (say q). Note that v_{x_1} may coincide with v_{k_1} if the chain of reversed arcs is made of a single arc.

The label swapping ensures that from this point onwards, the bandwidth of the sub-path starting in v_{k_1} will be calculated independently from the sub-path that ended in the predecessor of v_{k_1}. If v_{x_1} is the tail of the sub-path of q, its bandwidth will initially be the bandwidth of $\bar{p}$, because the first sub-path of q will be $\bar{p}_{sv_{x_1}}$. Hence, the first label swapping ensures v_{x_1} is now the primary label $b(\bar{p})$. The bandwidth of q will be at most $b(\bar{p})$, regardless of the bandwidth of the sub-path $\bar{p}_{sv_{x_1}}$. Hence, this is the correct label for calculating q bandwidth if the chain from v_{k_1-1} to v_{x_1} does in fact belong to $\bar{p}'$ – which implies $q_{sv_{x_1}} = \bar{p}_{sv_{x_1}}$. If the chain from v_{k_1-1} to v_{x_1} is not part of $\bar{p}'$, then the fact that we have modified the label of node v_{k_1} in that chain is irrelevant for the calculation of $\bar{p}'$. *As the tree of widest paths calculation progresses, the sub-paths with tail v_{x_1} and v_{k_1-1} will be calculated as widest as possible, taking into account the bandwidth of $\bar{p}_{sv_{x_1}}$ and $\bar{p}'_{sv_{k_1-1}}$.*

If, after exiting the first chain of reversed arcs (from v_{k_1-1} to v_{x_1}), a new reversed arc appears in $\bar{p}'$, that is, if the selected

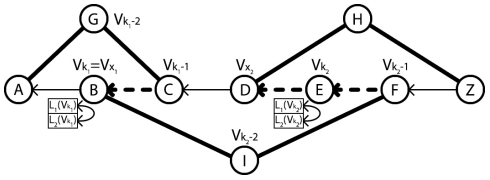


Fig. 1. Illustration of the label swapping procedure.

node v_{k_2} (and arc (v_{k_2-1}, v_{k_2})) is permanently added to the tree of widest paths, and it is the first reversed arc of the second chain of reversed arcs, the primary and secondary labels of v_{k_2} swap. Let v_{x_2} be the exit node of the chain of reversed arcs (in $\bar{p}'$) starting in node v_{k_2-1}. The candidate path p will be $\bar{p}'_{sv_{k_1-1}} \diamond \bar{p}_{v_{k_1-1}v_{x_2}}$, and the candidate sub-path of q path will be $\bar{p}_{sv_{x_1}} \diamond \bar{p}'_{v_{x_1}v_{k_2-1}}$. Due to the label swap, the calculation of the widest sub-paths with tails v_{x_2} and v_{k_2-1} will be made taking into account the bandwidth of $\bar{p}'_{sv_{k_1-1}} \diamond \bar{p}_{v_{k_1-1}v_{x_2}}$ and of $\bar{p}_{sv_{x_1}} \diamond \bar{p}'_{v_{x_1}v_{k_2-1}}$, respectively.

We can now generalize the procedure that has been described. Let the first reversed arc of the i-th ($i > 2$) chain of reversed arcs in $\bar{p}'$ be (v_{k_i-1}, v_k); this arc is permanently added to the tree of widest paths (in construction) when v_{k_i} is permanently labeled; because v_{k_i} is the head of the first reversed arc of the i-th chain of reversed arcs in $\bar{p}'$, the primary and secondary labels of v_{k_i} swap. Let v_{x_i} be the exit node of the i-th chain of reversed arcs in $\bar{p}'$ starting in node v_{k_i-1}. If i is even, the candidate sub-path of p will be:

$$\bar{p}'_{sv_{k_1-1}} \diamond \bar{p}_{v_{k_1-1}v_{x_2}} \diamond \cdots \diamond \bar{p}_{v_{k_{i-1}-1}v_{x_i}} \tag{7}$$

and the candidate sub-path of q will be:

$$\bar{p}_{sv_{x_1}} \diamond \bar{p}'_{v_{x_1}v_{k_2-1}} \diamond \cdots \diamond \bar{p}'_{v_{x_{i-1}}v_{k_i-1}} \tag{8}$$

If i is odd, the candidate sub-path of p will be:

$$\bar{p}'_{sv_{k_1-1}} \diamond \bar{p}_{v_{k_1-1}v_{x_2}} \diamond \cdots \diamond \bar{p}'_{v_{x_{i-1}}v_{k_i-1}} \tag{9}$$

and the candidate sub-path of q will be:

$$\bar{p}_{sv_{x_1}} \diamond \bar{p}'_{v_{x_1}v_{k_2-1}} \diamond \cdots \diamond \bar{p}_{v_{k_{i-1}-1}v_{x_i}} \tag{10}$$

Due to the label swap, the calculation of the widest sub-paths with tails v_{x_i} and v_{k_i-1} will be made taking into account the bandwidth of the candidate paths which end in v_{x_i} and v_{k_i-1} (according to Equations (7) to (10)), thus ensuring that when node t is reached, the primary and secondary labels of t will contain the bandwidth of p and q.

If the number of interlacing chains is even (odd), the *primary label* of t will be equal to $b(p)$ ($b(q)$) and the *secondary label* of t will be equal to $b(q)$ ($b(p)$). If no interlacing between $\bar{p}$ and $\bar{p}'$ takes place, the primary label will be the bandwidth of $\bar{p}'$ and the secondary label of t will be the bandwidth of $\bar{p}$. We designate this procedure as the Dual Path Label Dijkstra's algorithm. In this resolution approach, the paths p and q results from the union of the arcs in $\bar{p}$ and $\bar{p}'$, discarding every arc whose reversal appears on the other. However, the interlacing described above (and hence, the final values of the labels) is

only accurate if p and q use all those remaining arcs. To ensure this, and in the case of a tie, the reversed arcs in G', should be preferred to be part of $\bar{p}'$.

A. The Algorithm

To sum up, the steps required for solving problem Equation (6) are presented in 1.

Algorithm 1 Widest Edge-disjoint Path Pair Lexicographic Optimization (WEDLO): heuristic for problem given by Equation (6)

Require: $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, B matrix with arcs bandwidth $(b_{ij}, (i,j) \in \mathcal{A})$, nodes source s and target t.
Ensure: Returns a solution (possibly sub-optimal) to problem Equation (6), or $(\emptyset, \emptyset)$ if no solution was found.
1: $(p, q) \leftarrow (\emptyset, \emptyset)$ ▷ No solution
2: Application of the widest path Dijkstra's algorithm, calculating path $\bar{p}$, such that $\bar{p} = \arg\max_{p \in \mathcal{P}_{st}} b(p)$
3: **if** $\bar{p}$ exists **then**
4: Network is transformed in $\mathcal{G}'$ (the directed arcs in $\bar{p}$ from s to t are removed and the reversed arcs to the removed arcs are added with infinite bandwidth)
5: $\bar{p}' \leftarrow DPLD(G', B, s, t)$ ▷ Line 29
6: Restores $\mathcal{G}$ ▷ Undoes all changes required to obtain the transformed network $(\mathcal{G}')$.
7: **if** $\bar{p}' \neq \emptyset$ **then**
8: $(p, q) \leftarrow$ path pair resulting from the removal of the interlacing arcs of $\bar{p}$ and $\bar{p}'$.
9: **end if**
10: **end if**
11: **return** (p, q) ▷ Solution (sub-optimal) to Equation (6) or $(\emptyset, \emptyset)$

Line 5 in Algorithm 1 uses the Dual Path Label Dijkstra's algorithm, which is very similar to Dijkstra's algorithm, and requires the additional notation:

k	Present node
u	Candidate successor of the present node
S	Set of non-permanent nodes
$\psi(j)$	Node preceding node j
$L1(j)$	Label 1 of node j – primary label
$L2(j)$	Label 2 of node j – secondary label

The algorithm can be implemented using language C++. To determine k in Line 15 of Line 29, i.e., the node that leads to the path with the larger bandwidth among all non-permanent nodes, a multi-map from the C++ Standard Library can be used. A multi-map is a container whose elements are the association of keys and values. The key is used to order the elements according to an order relation. In these algorithms (widest Dijkstra's and Line 29), the elements are ordered solely by bandwidth in a decreasing fashion. Given two candidate nodes, a and b, if $L1(a) > L1(b)$, then a takes the top position in the multi-map. If the values are equal, the element that was first introduced remains on top.

Removal of the top element of the multi-map has a constant cost. Insertions, removals and searches on the multi-map have logarithmic complexity on the number of elements, so the complexity of this algorithm is $\mathcal{O}(|\mathcal{A}| \log_2 |\mathcal{N}|)$, identical to the complexity of Dijkstra's algorithm using a binary heap [2].

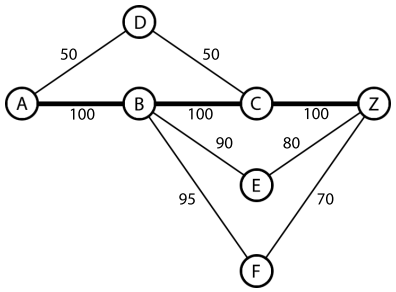
Algorithm 2 Dual Path Label Dijkstra's (DPLD)

Require: $\mathcal{G}' = (\mathcal{N}, \mathcal{A}')$, modified graph as described in step 4 of Algorithm 1, nodes source s and target t.
Ensure: Calculates $\bar{p}'$ (if it exits), the widest path in G', ensuring that $\bar{p}'$ and $\bar{p}$ contain the arcs solving Equation (6)
1: **for** all $i \in \mathcal{N}$ **do**
2: $\psi(i) \leftarrow s$
3: $L1(i) \leftarrow 0$, $L2(i) \leftarrow 0$ ▷ No path
4: **end for**
5: $L1(s) \leftarrow b(\bar{p})$, $L2(s) \leftarrow b(\bar{p})$ ▷ Widest path bandwidth
6: $k \leftarrow s$
7: $S \leftarrow \mathcal{N} - \{k\}$
8: **repeat**
9: **for** every arc $(k, u), u \in S$ **do**
10: **if** $L1(u) < \min[L1(k), b(k,u)]$ **then**
11: $\psi(u) \leftarrow k$ ▷ k becomes the predecessor of u
12: $L1(u) \leftarrow \min[L1(k), b(k,u)]$
13: **end if**
14: **end for**
15: $k \leftarrow \arg\max_{j \in S} L1(j)$
16: $S \leftarrow S - \{k\}$ ▷ Permanently labeled node
17: $L2(k) \leftarrow L2(\psi(k))$ ▷ Possible bandwidth of the other path in the pair
18: $l \leftarrow \psi(k)$ ▷ Arc $(\psi(l), l)$ is followed by $(\psi(k), k)$
19: **if** $b(\psi(k), k) = \infty \wedge b(\psi(l), l) \neq \infty$ **then**
20: ▷ $(\psi(k), k)$ is the first reversed arc of a chain
21: $L1(k)$ swaps value with $L2(k)$.
22: **end if**
23: **until** $k = t$
24: **if** $L1(t) = 0$ **then**
25: $\bar{p}' \leftarrow \emptyset$ ▷ No path from s to t
26: **else**
27: $\bar{p}'$ is described by the successive predecessors of t.
28: **end if**
29: **return** $\bar{p}'$.

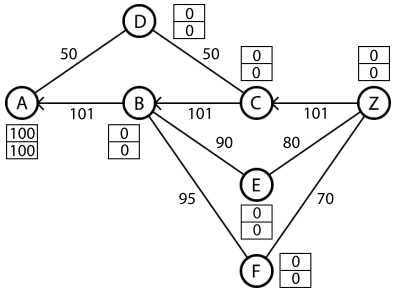
B. Illustrative Example

Consider the example network represented by the graph in Figure 2(a), where we considered an undirected graph to simplify the description. The label next to every edge is the corresponding edge bandwidth. The source node is A and the target node is Z. The first step is to calculate the widest path between A and Z, which is path A-B-C-Z (thicker lines in Figure 2(a)) with bandwidth 100.

The next step is to transform the network and assign labels to each node, as pictured in Figure 2(b). The nodes belonging to the widest path, A-B-C-Z, will require two labels in order to perform label swapping, while the rest of the nodes use the second label just to carry its value across the tree. In the figures, the top label represents the primary label, $L1$, and bottom label represents the secondary label, $L2$.

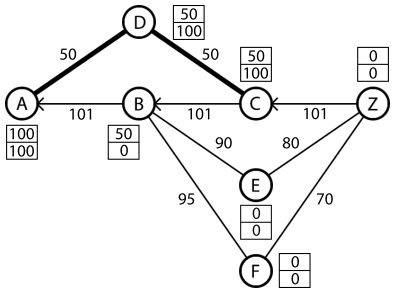


(a) Widest path.

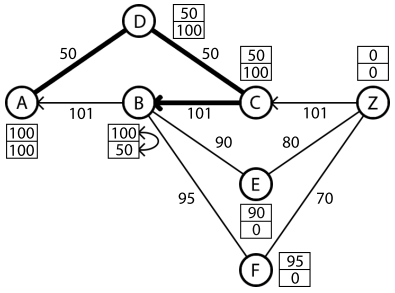


(b) Network transformation.

Fig. 2. Initial steps of the algorithm.

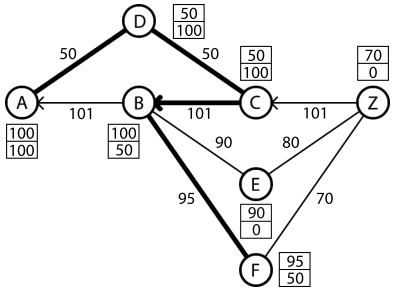


(a) After node C is permanently labeled

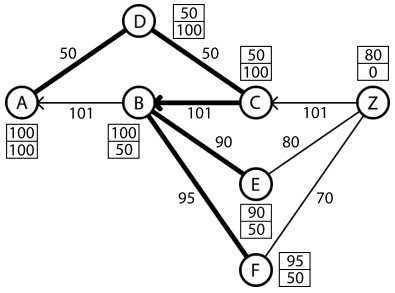


(b) Labels swap

Fig. 3. The Dual Path Label Dijkstra's algorithm first iterations.



(a) Node F is permanently labeled



(b) Node E is permanently labeled

Fig. 4. Impact of the dual labels.

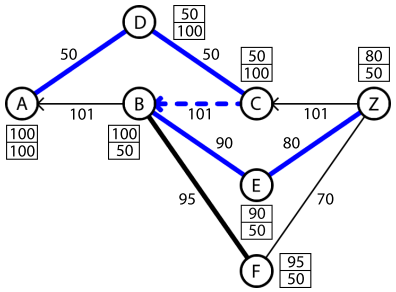
Also, since $L2$ will be passed on from node to node as the tree is calculated, the only labels that need to be initialized are the ones related to the source node, A. $L1(\mathrm{A})$ and $L2(\mathrm{A})$ receive the bandwidth value of the widest path, 100. The remaining nodes start with both their labels set as 0.

To compute the widest path tree in the transformed network, we run the Dual Path Label Dijkstra's algorithm. Similarly to the widest path Dijkstra's algorithm, the first step is to label A's neighbor nodes and pick the one with largest bandwidth to be part of the tree. Seeing as this is a small network and there are not many options, we easily reach node C (see Figure 3(a)). The value of $L2$ is passed on from A to D and then to C as these nodes became permanent. C's non-permanent neighbor, B, gets labeled with the current path bandwidth, $L1(\mathrm{B}) \leftarrow L1(\mathrm{C}) = 50$.

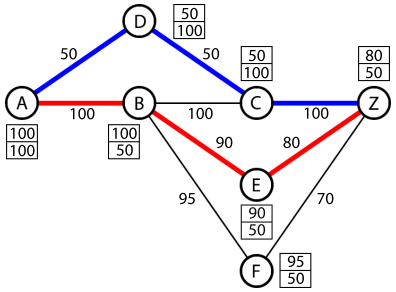
As B becomes part of the tree, $L2(\mathrm{B})$ takes the value of $L2(\psi(\mathrm{B}) = \mathrm{C})$. Since (C,B) is the first (in this case only) arc in a chain of arcs resulting from the previous graph transformation, the labels are swapped (see Figure 3(b)). If there was a node Y between C and B, then the primary and secondary labels of Y would swap once the node became permanent, but there would be no swapping when B was made permanent. Meaning that regardless of the number of reversed arcs that consecutively become part of the tree, the labels will swap only once per chain of reversed arcs. It is also worth noting that since the reversed arcs have bandwidth equal to the practical equivalent of infinity, the only difference between the labels at the start and at the end of the chain is that they were swapped.

Since the labels swapped, the current path bandwidth is now $L1(\mathrm{B}) = 100$, which means that the next choice between E and F is not irrelevant, as would happen in the case of the widest path Dijkstra's algorithm. Instead, $L1(\mathrm{E}) < L1(\mathrm{F})$, so F is the preferred node and, becoming part of the tree, labels the target node, Z, with $L1(\mathrm{Z})$ equal $\min[L1(\mathrm{F}), b(\mathrm{F,Z})] = 70$ (see Figure 4(a)). Also, $L2(\mathrm{F})$ takes the value of $L2(\mathrm{B}) = 50$.

However, now the best candidate is E with $L1(\mathrm{E}) > L1(\mathrm{Z})$. In Figure 4(b), E becomes part of the tree and labels the target node Z with a larger value, $L1(\mathrm{Z}) \leftarrow 80$, because $\min[L1(\mathrm{E}), b(\mathrm{E,Z})] > L1(\mathrm{Z})$. E also gets the secondary label from B, $L2(\mathrm{E}) \leftarrow L2(\mathrm{B}) = 50$.



(a) Path (and tree) completion.



(b) Path Pair computation.

Fig. 5. Final steps.

The target node is reached in Figure 5(a) and $L2(\mathrm{Z}) \leftarrow L2(\mathrm{E}) = 50$. In thick lines, we have the widest paths tree; in blue, the widest path on the modified network; and in a dashed line, the edge that will not be part of the widest path pair.

In Figure 5(b), the interlacing edge is removed and the path pair calculation is finished. Note that $L1(\mathrm{Z})$ has the bandwidth of the red path, which was the current path when the Dual Path Label Dijkstra's algorithm ended, and $L2(\mathrm{Z})$ has the bandwidth of the blue path. The C-Z edge does not affect the veracity of the last affirmation because it was part of the widest path in the original network and no path in the pair can have bandwidth larger than that.

V. Performance Evaluation

The performance of the heuristic is evaluated using fourteen networks from the SNDLib [10], where nodes of degree one were removed (and the corresponding networks have an '*' appended to their name in the figures). The bandwidth b_{ij} of each edge (i, j) was defined to be $1000/\log d(v_i, v_j)$, where $d(v_i, v_j)$ is the distance between nodes v_i and v_j, based on the GPS coordinates of v_i and v_j. For comparison, we use an integer linear programming formulation Equation (11) of the WEDLO. For this, we first introduce the following additional notation:

Parameters:

s	source node; $s \in \mathcal{N}$
t	target node; $t \in \mathcal{N}$
B_m	is $\max_{(p,q)\in\bar{\mathcal{P}}_{st}} b_m(p, q)$, with $B_m > 0$
M	a sufficiently large constant

Variables:

x_{ij}^k	1 if path k is using arc $(i, j) \in \mathcal{A}$, 0 otherwise, with $k = 1, 2$

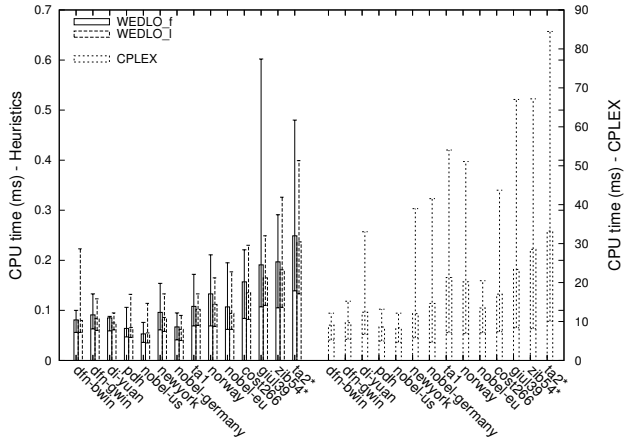


Fig. 6. CPU time in milliseconds per node pair

y_k bandwidth of path k.

The ILP formulation is as follows:

$$\max \quad y_1$$

Subject to:

$$y_k \geq B_m,\ k \in \{1, 2\} \quad \text{(11a)}$$

$$\sum_{(i,j)\in\mathcal{A}} x_{ij}^k - \sum_{(j,i)\in\mathcal{A}} x_{ji}^k = \begin{cases} 1 & \text{if } s = i \\ -1 & \text{if } t = i \\ 0 & \text{otherwise} \end{cases}$$

$$\forall i \in \mathcal{N},\ k \in \{1, 2\} \quad \text{(11b)}$$

$$x_{ij}^1 + x_{ji}^1 + x_{ij}^2 + x_{ji}^2 \leq 1,\ \forall (i, j) \in \mathcal{A} \quad \text{(11c)}$$

$$y_k \leq b_{ij} x_{ij}^k + M(1 - x_{ij}^k),\ \forall (i, j) \in \mathcal{A},\ k \in \{1, 2\} \quad \text{(11d)}$$

$$x_{ij}^k \in \{0, 1\},\ \forall (i, j) \in \mathcal{A} \quad \text{(11e)}$$

This ILP formulation is modified from [12, Fig. 5] to suit our need for WEDLO. Note that this formulation does not prevent the resulting paths from having cycles. In Equation (11) we maximize the bandwidth of path $k = 1$, while inequality (11a) ensures both paths must have at least a bandwidth equal to B_m. Equation (11b) expresses the standard flow conservation law for variables x_{ij}^k. Variables x_{ij}^k are additionally bounded by inequality (11c) that ensure no edge (represented by a pair of symmetrical arcs) can be shared by the paths. Inequality (11d) identifies the capacity of a bottleneck arc for each path, thus defining the corresponding bandwidth. Equation (11e) defines the bounds for decision variables.

Recall, that in the case of a tie, the reversed arcs in G', should be preferred by the Dual Path Label Dijkstra's algorithm, in order to ensure the correct final labels. If that is not the case, the algorithm may overestimate the bandwidth of the calculated paths, and one must recalculate them, after the paths are obtained. Two versions of Algorithm 1 were considered: WEDLO_f and WEDLO_l where, in the case of a tie, the unlabeled nodes adding a reversed arc (of the widest path) to the tree of shortest paths under construction, are ranked first and last, respectively.

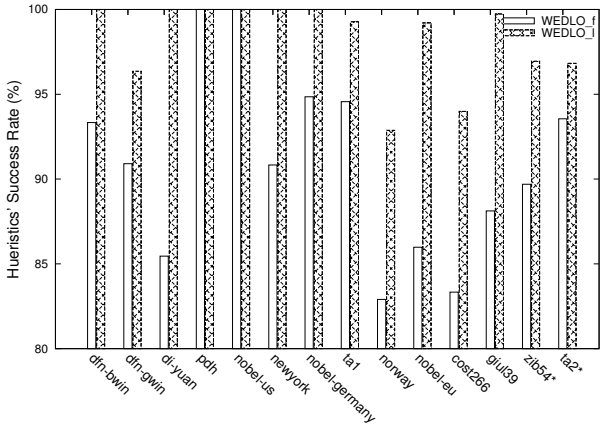


Fig. 7. Percentage of optimal solutions found by WEDLO_f and WEDLO_l

In Figure 6, we present the average CPU time per node pair for solving WEDLO considering all node pairs for each network, using Algorithm 1 and CPLEX 12.6 [1] for solving the ILP formulation of WEDLO, on a Desktop with an Intel(R) Core(TM) i7 CPU 950 @ 3.07GHz processor and 6GB of RAM. The error bars represent the minimum and maximum value observed for each network. Note the different vertical axis for the heuristics and the solver. The CPU time in all cases grows with the size of the networks (ordered in the figures by increasing number of nodes, from 10 to 64), but the CPU values for the heuristics are about 1% of the corresponding values obtained by CPLEX. Although the average value of WEDLO_l is slightly less than the corresponding value for the WEDLO_f, the maximum values of the former are not always less than the maximum values of the latter. Therefore, regarding the CPU time, the two heuristics may be considered to have a similar performance.

The percentage of node pairs where Algorithm 1 is able to find an optimal solution (verified using the solution obtained by CPLEX) is presented in Figure 7. It can be seen that WELDO_l performs much better than WELDO_f regarding the number of optimal solutions found. WELDO_l for six of the fourteen tested networks found 100% of the optimal solutions in the remaining eight networks WELDO_f obtained over 90% of the optimal solutions. WELDO_f only managed to obtain 100% of the optimal solutions in two networks, has six networks with 90%-95%, and the remaining six with 85%-90% sub-optimal solutions.

Regarding the sub-optimal solutions of Algorithm 1, the average relative error of $B_M(p,q)$ using the y_1 value as reference, was also calculated. In both cases, the relative error is on average less than 4% and the worst case is less than 7% and 14%, for WELDO_l and WELDO_f, respectively.

VI. Conclusion

Telecommunication networks need to offer reliable and QoS-aware guaranteed services. Path protection is a simple and effective approach to increase network resiliency.

In this work, we formally present a lexicographic optimization problem for obtaining maximum-bandwidth disjoint paths and then maximizing the bandwidth of the widest path in the pair. We then present an effective heuristic, with complexity identical to Dijkstra's algorithm for solving this problem. The heuristics proved to be effective and return solutions in a time significantly lower than the time needed by a commercial MIP solver. The resolution to the formalized optimization problem is of practical interest in QoS routing with protection such as for MPLS-based virtual private network services. Having maximized the extra bandwidth available in the widest path of the pair, it is possible to ensure that if used as a protection path, it has spare bandwidth to take into account traffic fluctuations that take place in the event of failure.

Acknowledgements

Teresa Gomes acknowledges financial support through project QREN 23301 PANORAMA II, co-financed by European Unions FEDER through Programa Operacional Factores de Competitividade (POFC) of QREN (FCOMP-01-0202-FEDER-023301), and by the Portuguese Foundation for Science and Technology (FCT) under project grant PEst-OE/EEI/UI308/2014. Deep Medhi has been partially supported by the National Science Foundation Grant No. CNS-1217736 and by a research visit to the University of Coimbra, supported by FCT Project PTDC/EEA-TEL/101884/2008.

References

[1] *IBM ILOG CPLEX Optimization Studio V12.6*. IBM, 2013.
[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms and applications*. Prentice Hall, 1993.
[3] Anteneh Beshir and Fernando Kuipers. Variants of the min-sum link-disjoint paths problem. In *Proceedings of the 16th Annual IEEE Symposium on Communications and Vehicular Technology (IEEE SCVT'09)*, Louvain-la-Neuve, Belgium, November 2009.
[4] R. Bhandari. *Survivable Networks, Algorithms for Diverse Routing*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1999.
[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269 – 271, 1959.
[6] TC Hu. The maximum capacity route problem. *Operations Research*, 9(6):898–900, 1961.
[7] H. Leng, J. Song, M. Liang, Z. Xie, and J. Zhang. Routing on shortest pair of disjoint paths with bandwidth guaranteed. In *Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 557–561, 2009.
[8] D. Medhi and K. Ramasamy. *Network Routing: Algorithms, Protocols, and Architectures*. Morgan Kaufmann Publishers (an imprint of Elsevier), 2007.
[9] R. Ogier, B. Bellur, and N. Taft-Plotkin. An efficient algorithm for computing shortest and widest maximally disjoint paths. Technical Report ITAD-1616-TR-170, SRI International, November 1998.
[10] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design library. *Networks*, 55(3):276–286, 2010.
[11] Maurice Pollack. The maximum capacity through a network. *Operations Research*, 8(5):733–736, 1960.
[12] B. H. Shen, B. Hao, and A. Sen. On multipath routing using widest pair of disjoint paths. In *2004 Workshop on High Performance Switching and Routing*, pages 134 – 140, 2004.
[13] J. W. Suurballe. Disjoint paths in networks. *Networks*, 4:125–145, 1974.
[14] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984.
[15] N. Taft-Plotkin, B. Bellur, and R. Ogier. Quality-of-service routing using maximally disjoint paths. In *Seventh International Workshop on Quality of Service 1999 (IWQoS '99)*, pages 119 – 128, 1999.
[16] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14:1228–1234, 1996.