Protected shortest path visiting specified nodes

Teresa Gomes[*†], Sofia Marques[*], Lúcia Martins [*†], Marta Pascoal [†‡] and David Tipper [§]
[*]Department of Electrical and Computer Engineering, University of Coimbra, 3030-290 Coimbra, Portugal
Email: teresa@deec.uc.pt, a2009118947@alunos.deec.uc.pt, lucia@deec.uc.pt
[†]INESC Coimbra, 3030-033 Coimbra, Portugal
[‡]Department of Mathematics, 3001 – 501 Coimbra, Portugal
E-mail: marta@mat.uc.pt
[§]Graduate Telecommunications and Networking Program
University of Pittsburgh, Pittsburgh, PA, USA
E-mail: dtipper@pitt.edu

***Abstract*—In this paper heuristics are proposed for finding the shortest loopless path, from a source node to a target node, that visits a given set of nodes in a directed graph, such that it can be protected using a node-disjoint path. This type of problem may arise due to network management constraints.**

The problem of calculating the shortest path that visits a given set of nodes is at least as difficult as the traveling salesman problem, and it has not received much attention. Nevertheless an efficient integer linear programming (ILP) formulation has been recently proposed for this problem. Here, the ILP formulation is adapted to include the constraint that the obtained path will be able to be protected by a node-disjoint path. Computational experiments show that this approach, namely in large networks, may fail to obtain a solution in a reasonable amount of time. Therefore three versions of a heuristic are proposed, for which extensive computational results show that they are able to find a solution in most cases, and that the calculated solutions present an acceptable relative error regarding the cost of the optimal active path. Further the CPU time required by the heuristics is significantly smaller than the required by the used ILP solver.

***Index Terms*—Resilient routing, visiting a given set of nodes.**

I. Introduction

The resilience and survivability of communication networks are very important, since they are critical infrastructures of society. See [1] for an architectural framework for resilience and survivability in communication networks. Services may require different levels of resiliency, which led to the introduction of the concept of Quality of Resiliency [2]–[4]. Network recovery can be ensured using protection (pre-designed restoration) or rerouting (restorations after fault detection).

Routing with path protection seeks to obtain a pair of node (or arc) disjoint paths, depending on the level of desired protection. In certain cases the active path must visit a specified set of nodes. These nodes may be selected due to their characteristics (for example, high reliability), to operators preferences resulting from inter-operators agreements, or to other network management constraints. There are very few works addressing the problem of the calculation of the shortest path from a source node to a target node that visits a specified set of nodes (it is assumed that the source and target nodes are different). The first known work is from Saksena and Kumar [5], where the authors attempted to develop an exact algorithm, using the optimality principle, for calculating the shortest path (possibly with cycles) that visits a specified set of nodes. Unfortunately their algorithm (designated hereafter SK66) is not correct. Dreyfus in [6] challenged their approach and stated it did not necessarily obtain the correct solution.

Ibaraki in [7] considered separately the problem of calculating the shortest loopless path that visits a specified set of nodes and the shortest path (possibly with cycles) that visits a specified set of nodes. Ibaraki proposed two approaches for the calculation of the shortest loopless path that visits a specified set of nodes, one based on dynamic programming and the other based on the branch and bound principle. Computational results presented in Ibaraki in [7] seem to indicate that the algorithm based on branch and bound principle is more efficient than the algorithm based on dynamic programming.

New formulations, for addressing the determination of a shortest loopless path, without cycles, from a source node s to a destination node t, that visits only once all nodes of a specified set, are given in [8]. The first new formulation (designated Q2 in [8]) is based on an adapted version of the cycle elimination constraints of the spanning tree polytope and the second one (designated Q3 in [8]) is a new primal-dual based mixed integer formulation. The numerical results presented in [8], with problem instances up to 80 nodes, and a set of specified nodes equal to 25%, 50%, 75% and 100% of the graph nodes (excluding the source and the target) show the significant efficiency improvement of Q3 with respect to Q2.

Regarding complexity, Dreyfus states in [6] that the calculation of the shortest path (possibly with cycles), from a source node to a target node that visits a specified set of nodes, can not be easier than the traveling salesman problem of dimension k, where $k-2$ is the number of specified nodes to be visited. Also Andrade [8] points out that if the set of specified nodes to be visited is made of all nodes in the graph, excluding the source and target nodes, this corresponds to finding an Hamiltonian path of minimum cost, which is NP-hard.

The rest of the text is structured as follows. In section II notation is introduced and the problem is formalized. The heuristics for calculating a shortest loopless path visiting specified nodes, are described in section III, starting in subsection III-A with a revision of SK66, followed by the description of the proposed modifications to that algorithm in subsection III-B.

In section IV three variants of the proposed heuristics for calculating a protected shortest loopless path visiting specified nodes are described. Computational results are presented in section V. Section VI concludes the paper.

II. Notation and problem formulation

In this work one seeks the shortest loopless path visiting a specified set of nodes, with the constraint that it must be protected using a node-disjoint path. A loopless path must visit each node only once. Hence, unless explicitly stated otherwise, all paths are considered to be loopless.

A. Notation

The heuristics proposed in sections III and IV use the following notation. Let the graph $G = (V, A)$ be defined by a set of nodes $V = \{v_1, \ldots, v_n\}$, and a set of (directed) arcs $A = \{a_1, \ldots, a_m\}$. An arc connects two vertices in a given order, and is an ordered pair of elements belonging to V. If $v_i, v_j \in V$, with $v_i \neq v_j$ and $a_k = (v_i, v_j) \in A$, it is said that the v_i is the tail (or source) of the arc and v_j is its head (or destination). Arc (v_i, v_j) is said to be emergent from node v_i and incident on node v_j. Arcs (v_i, v_j) and (v_j, v_i) are symmetrical arcs.

A path is a continuous sequence of distinct nodes from a source node, s, to a destination node t, $(s, t \in V)$, and is represented by $p = \langle s \equiv v_1, v_2, \ldots, v_k \equiv t \rangle$, where $(v_i, v_{i+1}) \in A, \forall i \in \{1, \ldots, k-1\}$, k being the number of nodes in the path. Let V_p be the set of nodes in the path p, and A_p be the set of arcs that form the path, $A_p = \cup_{\forall i \in \{1,\ldots,k-1\}} (v_i, v_{i+1})$. A segment is a continuous sequence of arcs that are part of a path. The cost of using an arc $(v_i, v_j) \in A$ in a path is given by $w(v_i, v_j)$, which is assumed to be strictly positive. The additive cost of a path p is the sum of the costs of the arcs constituting the path, $D_p = \sum_{(v_i,v_j) \in A_p} w(v_i, v_j)$. If a path between a given pair of nodes does not exist, it is represented by the empty set ($\emptyset$), and its cost is infinite.

Given a node pair (s, t), a pair of paths from s to t is represented by (p, q). The paths are node-disjoint if and only if $V_p \cap V_q = \{s, t\}$.

Let $\mathcal{P}_{st}$ represent the set of all paths from s to t in the network. Let V_S designate the set of specified nodes that must be visited by the active path. A path from a node v_i to a node v_j is represented by $p_{v_i v_j}$. The *concatenation* of paths $p_{v_i v_j}$ and $p_{v_j v_l}$ is the path, $p_{v_i v_j} \diamond p_{v_j v_l}$, from v_i to v_l, which coincides with $p_{v_i v_j}$ from v_i to v_j and with $p_{v_j v_l}$ from v_j to v_l. The constant $\tau = \lceil |A|/|V| \rceil^2 |V_S|$ will be used to limit shortest path enumeration in the proposed heuristics.

B. Problem formulation

The problem is to find a shortest loopless path visiting a specified set of nodes, with the constraint that it must be protected using a node-disjoint path:

$$(p_1^*, p_2^*) = \arg \min_{(p_1,p_2) \in \mathcal{P}_{st}} D_{p_1} \quad (1)$$

$$\text{subject to:} \quad V_{p_1} \cap V_S = V_S \quad (2)$$

A path from s to t which visits the nodes in V_S will also be represented by an s–V_S–t path, as in [8]. Once a path p_1^* as been found, the backup path p_2^* can be calculated as the min-cost path in the network where the intermediate nodes of p_1^* have been removed. This simple approach ensures that the backup path is node-disjoint.

The Integer Linear Programming formulation for obtaining (p_1^*, p_2^*) is given here, because the exact results obtained using this formulation will be used to evaluate the performance of the heuristics. The formulation requires some additional notation: $\delta(i)^+$, the set of arcs in A emergent from node v_i; $\delta(i)^-$, the set of arcs in A incident on node v_i; and $x_{(i,j),u}$ is the binary decision variable of arc $(v_i, v_j) \in A$ associated with path p_u ($u = 1, 2$), where,

$$x_{(i,j),u} = \begin{cases} 1 & \text{if arc } (v_i, v_j) \in A_{p_u} , \\ 0 & \text{otherwise;} \end{cases} \quad (3)$$

Let $p = \langle s = v_1, v_2, \ldots, v_k = t \rangle$ be a shortest s–V_S–t path, and π_{v_i}, $v_i \in V_p$, is the cost of going from node s to node v_i in this path. The ILP formulation which follows is an adaptation of the formulation Q3 in [8] with the additional constraint that the obtained path can be protected by a node-disjoint path.

$$\min \quad \sum_{(v_i,v_j) \in A} w(v_i, v_j) x_{(i,j),1} \quad (4)$$

$$\text{s.t.:} \quad \sum_{(v_i,v_j) \in \delta(i)^+} x_{(i,j),u} - \sum_{(v_j,v_i) \in \delta(i)^-} x_{(j,i),u} = \begin{cases} 1 & : \ v_i = s, \\ -1 & : \ v_i = t, \\ 0 & : \ v_i \in V \backslash \{s, t\} \end{cases} \quad \forall v_i \in V, \ u = 1, 2 \quad (5)$$

$$\sum_{(v_i,v_j) \in \delta(i)^+} x_{(i,j),1} = 1, \quad \forall v_i \in V_S, \quad (6)$$

$$v_i \in V \backslash \{s\},$$

$$\pi_{v_j} - \pi_{v_i} \leq w(v_i, v_j) + M(1 - x_{(i,j),1}), \quad \forall (v_i, v_j) \in A \quad (7)$$

$$\pi_{v_j} - \pi_{v_i} \geq w(v_i, v_j) - M(1 - x_{(i,j),1}), \quad \forall (v_i, v_j) \in A \quad (8)$$

$$\pi_s = 0 \quad (9)$$

$$\pi_{v_i} \geq 0, \forall v_i \in V \setminus \{s\} \quad (10)$$

$$\sum_{(v_i,v_j) \in \delta(i)^+} x_{(i,j),1} + x_{(i,j),2} \leq 1, \quad \forall v_i \in V \setminus \{s\} \quad (11)$$

x are the binary decision variables.

In the optimization problem M is a sufficiently large number and:

- constraint (5) ensures (unrestricted) paths p_u ($u = 1, 2$) from s to t exist;
- constraint (6) ensures that nodes in V_S are visited by the active path p_1 ($u = 1$) from s to t;
- constraints (7) and (8) impose that if an arc (v_i, v_j) is in the solution then $\pi_{v_j} - \pi_{v_i} = w(v_i, v_j)$ – see [8] for a

proof. Additionally, as all arc costs are strictly positive, then feasible solutions for the active path do not contain cycles;

- constraint (9) defines the cost (distance) of the source node s (in the active path); constraint (10) ensures all costs from s to $v \in V$ are positive;
- constraint (11) ensures a pair of node-disjoint paths exists.

Note that the protection path defined by $x_{(i,j),2}$ may contain cycles, and its cost is not minimized. Hence, having calculated p_1^*, the min-cost path node-disjoint with p_1^* can be calculated in a network where all intermediate nodes of p_1^* have been removed, or, if $p_1^* = \langle s, t\rangle$, where the arc (s,t) has been removed.

III. Shortest path visiting specified nodes

The algorithm by Ibaraki [7] based on dynamic programming easily requires a huge amount of memory and CPU time, because it corresponds to an almost exhaustive breadth first search of all the paths from source to target. The algorithm SK66, although it does not ensure an optimal solution, has a number of iterations proportional to the size of the set of specified nodes to visit ($|V_S|$); in the initial step it calculates $(|V_S|+2)|V_S|$ shortest paths; in each step it demands $|V_S|^2$ operations, of which the most time consuming is node counting, with worst case complexity $|V|\log_2|V|$; hence the worst case complexity of SK66 is $\mathcal{O}(|V_S+2|^2|A|\log_2|V| + |V_S|^2|V|\log_2|V|)$ if Dijkstra's algorithm (with a binary heap) is used in the calculation of the shortest sub-paths.

A. Revision of the algorithm by Saksena and Kumar (SK66)

The algorithm SK66, tries to obtain a path from a source node s to a target node t, visiting a set of specified nodes, by selecting the path (possibly with cycles) with minimum cost among the possible concatenation of sub-paths of minimum cost.

The initial steps of the algorithm are:

1) the calculation of the shortest path (without restrictions) between every node pair belonging to V_S, and also between the source node s and every node in V_S;
2) the calculation of the shortest path from every node in V_S to the target node.

Let $D(v_i, v_l)$ designate the cost of the shortest path from v_i to v_l, with $v_i \in V_S \cup \{s\}$ and $v_l \in V_S$. Let $f_{v_i}^0$ designate the cost of the shortest path from node $v_i \in V_S$ to t.

The algorithm can now iteratively calculate:

$$f_{v_i}^{\eta} = \min_{v_l \in V_S}[D(v_i, v_l) + f_{v_l}^{\eta-1}], \quad v_l \neq v_i \tag{12}$$

for $\eta = 1, 2, \ldots, |V_S|-1$, seeking to add at least one additional specified node in each iteration to the obtained sub-paths. However a path (possibly with cycles) from v_i, via an additional specified node v_l, to t, of cost $D(v_i, v_l) + f_{v_l}^{\eta-1}$ is only admissible if it contains at least η specified nodes, not counting node v_i, nor its repetitions on the path. This must be verified at every step of the algorithm.

The final step is the calculation of

$$f_0^{|V_S|} = \min_{v_l \in V_S}[D(s, v_l) + f_{v_l}^{|V_S|-1}] \tag{13}$$

and of the corresponding path (which may contain cycles).

Note that the sub-paths considered in the construction of the paths according to equation (12) are restricted to the shortest paths calculated in the initial steps of the algorithm.

B. Modification of the algorithm by Saksena and Kumar

In this subsection are introduced the proposed modifications of SK66. This new version of the algorithm, ensures the obtained path is loopless and also improves the performance of the original algorithm, because it keeps more intermediate sub-paths, as will be described next. This modified and improved version of SK66 will be designated SK.

In order to obtain loopless paths, the path selected by equation (13) and the sub-paths selected by equation (12) must satisfy the restriction that they must not contain a cycle.

In each iteration of SK66, according to equation (12), for every $v_i \in V_S$ a new via node v_l ($v_l \in V_S$) is selected for obtaining a path from v_i to t. This procedure may prematurely eliminate sub-paths (of the final path from s to t) of higher cost that would result in admissible solutions, due to selecting lower cost sub-paths that will latter on be discarded because they lead to a non admissible solution. Let

$$\begin{aligned} f_{v_i,v_l'}^1 &= D(v_i, v_l') + f_{v_l'}^0, \quad v_l' \neq v_i && (14)\\ f_{v_i,v_l}^{\eta} &= D(v_i, v_l) + \min_{v_j} f_{v_l,v_j}^{\eta-1}, && (15)\\ & \qquad v_l \neq v_i, v_j \wedge v_i \neq v_j \end{aligned}$$

with $\eta = 2, \ldots, |V_S|$. Hence the *main modification* is, in each iteration η, to keep the paths from node v_i to t, via all the possible new intermediate nodes v_l, instead of selecting the one of minimal cost as in (12). In the original version of the algorithm, at the end of each iteration $\eta = 1, \ldots, |V_S|-1$ only one path was selected (unless a tie in the minimum cost occurred) for each source node v_i – see (12). In this modified version up to $|V_S|-1$ paths are selected for each source node v_i.

Additionally the following modifications were introduced.

1) Trying to avoid cycles, the initial steps were modified as follows: the shortest path from s to $v_l \in V_S$ of cost $D(s, v_l)$ is calculated in a network where node t was removed; similarly the shortest path from v_i to v_j, with $v_i, v_j \in V_S$, is calculated in a network where nodes s and t were removed; finally the shortest path from node $v_i \in V_S$ to t of cost $f_{v_i}^0$ is calculated in a network where node s was removed.
2) If in iteration η, none of paths from v_i to t, using via node v_l, of cost $D(v_i, v_l) + f_{v_l,v_j}^{\eta-1}$, were considered admissible, and in the previous iteration, a path from node v_i to t, with $|V_S|-1$ specified nodes exists, which had resulted from adding the new specified via node v_l, then the path from the previous iteration is copied, and saved as if it had been obtained in the η-th iteration.

3) If in iteration η, the new path (from v_i to t) of cost $D(v_i, v_l) + f^{\eta-1}_{v_l,v_j}$, with r^* ($r^* \geq \eta$) specified nodes was considered admissible, and in the previous iteration, a path from v_i to t, adding via node v_l also exists with at least r^* specified nodes, and has lower cost than the path obtained in the present iteration, the path of the $\eta - 1$ iteration replaces the one obtained in the η-th iteration (and is saved as if it had been obtained in the η-th iteration).

IV. Protected shortest path visiting specified nodes

Next two different approaches for finding a shortest path visiting a specified set of nodes, with the constraint that it must be protected using a node-disjoint path are proposed.

A. Active path first

The algorithm described in this sub-section, designated ASK, tries to generate an active path, such that a node-disjoint path exists. It is based on SK, but it introduces additional restrictions in the generation of the sub-paths that are the building blocks of the active path. These restrictions ensure that each sub-path allows a node-disjoint path to be obtained from s to t. However this procedure does not ensure that a protection path can be found for the shortest path resulting from the concatenation of those sub-paths.

The initial steps of ASK are:

1) A k-shortest path enumeration algorithm [9] and [10] is used to generate up to τ (recall that $\tau = \lceil |A|/|V| \rceil^2 |V_S|$) paths from (v_i, v_j), with $v_i \in V_S \cup \{s\}$ and $v_j \in V_S$; as soon as a k-th path from v_i to v_j, p^k_{ij} ($k = 1, \ldots, \tau$), is found such that removing the set of nodes $(V_{p^k_{ij}} \cup V_S) \backslash \{s\}$ a path from node s to t can be found, the path generation procedure stops; the path p^k_{ij} is stored and $D(n_i, n_j)$ takes the cost of that path. If none of the generated paths can be protected $D(n_i, n_j) = \infty$.
2) A k-shortest path enumeration algorithm [9] and [10] is used to generate up to τ paths from (v_i, t), with $v_i \in V_S$; as soon as a k-th path from v_i to t, p^k_{it} ($k = 1, \ldots, \tau$), is found such that removing the set of nodes $(V_{p^k_{it}} \cup V_S) \backslash \{t\}$ a path from node s to t can be found, the path generation procedure stops; the path p^k_{it} is stored and $f^0_{v_i}$ takes the cost of that path. If none of the generated paths can be protected $f^0_{v_i} = \infty$.

Then SK is used with the additional restriction that at each iteration η, a path from node v_i to node t via node v_l is only admissible if in the network without the nodes in $V_S \cup V_{p_{v_i t}} \setminus \{t\}$ a path from s to t can be found. Similarly, in the final step, a path from s to t is only admissible if a node-disjoint path exists with each loopless candidate path of cost $D(s, v_l) + f^{|V_S|-1}_{v_l, vj}$.

The final active path selection in the algorithm is done initially using (13), with the restrictions that the path must be loopless and that it can be protected by a node-disjoint path. If a candidate path p, resulting from the concatenation of the sub-path from s to v_l (with cost $D(s, v_l)$) and of the sub-path v_l to t (with cost $f^{|V_S|-1}_{v_l}$) does not contain a cycle, one must then verify if the resulting path does have a node-disjoint backup path from s to t.

Note that although it is ensured that each of the sub-paths p_{sv_l} and $p_{v_l t}$, can be protected by a node-disjoint path from s to t, this does not ensure that a node-disjoint path exists for path p. Hence, the final active path selection sequentially adopts the following approaches, proceeding into the next one only if no feasible solution could be obtained in the previous one:

(a) *Final selection based on equation* (13).
Build path p_1 with cost $D(s, v_l) + f^{|V_S|-1}_{v_l,v_i}$ resulting from the concatenation of the shortest path p_{sv_l}, of cost $D(s, v_l)$, obtained in the initial steps, with $p_{v_l t}$ the path with cost $f^{|V_S|-1}_{v_l,v_i}$ ($v_l, v_i \in V_S$) obtained in the $\eta = |V_S| - 1$ iteration of the algorithm.
If $p_1 = p_{sv_l} \diamond p_{v_l t}$ is loopless, then verify if a node-disjoint path can be found with the path p_1.

(b) *Possibly replacing the sub-path of cost $D(s, v_l)$, obtained in the initial steps of the algorithm.*
For every sub-path $p_{v_l t}$, with cost $f^{|V_S|-1}_{v_l,v_i}$ ($v_l, v_i \in V_S$), a shortest path p_{sv_l} in the network where the nodes $V_{p_{v_l t}} \setminus \{t\}$ have been removed is calculated; then if a node-disjoint path can be found with the path $p_{sv_l} \diamond p_{v_l t}$, a feasible solution has been found.

(c) *Seeking to find a node-disjoint backup with the sub-path $p_{v_l t}$ with cost $f^{|V_S|-1}_{v_l,v_i}$.*
For every sub-path $p_{v_l t}$ with cost $f^{|V_S|-1}_{v_l,v_i}$ ($v_l, v_i \in V_S$), a backup path p_2 is calculated from s to t in a network where the nodes $V_{p_{v_l t}} \setminus \{t\}$ have been removed. If p_2 exists, a sub-path p_{sv_l} is then calculated in the network where the nodes in set $V_{p_2} \cup V_{p_{v_l t}} \setminus \{s, v_l\}$ have been removed. If this sub-path exists then $p_1 = p_{sv_l} \diamond p_{v_l t}$ is a feasible solution.

(d) *Seeking to find a node-disjoint backup with the sub-path p_{sv_l}, where v_l is the most downstream among the specified nodes of the path of cost $D(s, v_l) + f^{|V_S|-1}_{v_l,v_j}$.*
For every sub-path $p_{v_l t}$ with cost $f^{|V_S|-1}_{v_l,v_i}$ ($v_l, v_i \in V_S$), let p_{sv_l} be the shortest path from s to v_l of cost $D(s, v_l)$, and $p'_1 = p_{sv_l} \diamond p_{v_l t}$ a loopless path. Let v_i, with $v_i \in V_S$, be the node closest to t in p'_1; a backup path p_2 is calculated from s to t in a network where the nodes in set $V_{p_{sv_l}} \backslash \{s\}$ have been removed. If p_2 exists, a sub-path from v_l to t, $p_{v_l t}$, is then calculated in the network where the nodes in set $V_{p_2} \cup V_{p_{sv_l}} \setminus \{v_l, t\}$ have been removed.

Note approaches (b)-(d) are sequentially used, only if the previously used approach – (a), the first one to be used – resulted in no feasible solution. The solution with minimal cost among the obtained feasible solutions is the final selected active path (with the corresponding node-disjoint protection path).

B. Backup path first

In some networks ASK may fail to find a solution, because no protection path could be found for the calculated shortest paths. Hence another algorithm, which first calculates a backup path and then seeks the corresponding active path, was proposed.

This second algorithm, designated BSK, tries to obtain a possible backup path in a network where the nodes in V_S have been removed. First it generates the largest set of node-disjoint paths of min-sum cost, according to Bhandari's algorithm [11]. Then for each backup path, q in that set, the corresponding active path is calculated using algorithm SK, in the network $(V \setminus (V_q \setminus \{s,t\}), A)$, that is where the intermediate nodes of the calculated backup path have been previously removed.

If the previous procedure was unable to elicit a solution, then a k-shortest path enumeration algorithm [9] and [10] is used to calculate up to τ candidate backup paths in a network where the nodes in V_S have been removed. Then for each backup path, q_i $(i = 1, \ldots, \tau)$, the corresponding active path is calculated using algorithm SK, in the network $(V \setminus (V_{q_i} \setminus \{s,t\}), A)$. The procedure stops as soon as an admissible solution is found.

C. Considered heuristics

Three versions of the heuristic were implemented: the algorithm ASK in sub-section IV-A; the algorithm BSK in sub-section IV-B; heuristic ABSK corresponds to using algorithm ASK, and if no feasible solution was found then algorithm BSK is invoked. The discussion of the results in the next section will justify the proposal of the ABSK version.

V. Results

Experimental tests were made considering networks from two different sources. The first set of networks are from the SNDlib [12], which correspond to real-world reference networks. The used networks were newyork, norway, india35, pioro40 and germany50; the cost of each edge was considered to be the first module cost. The second set of networks was generated with the Doar-Leslie model [13] using Georgia Tech Internetwork Topology Models software (GT-ITM) (http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html). Five networks with 500 nodes with an average degree around 7 were generated. The number of specified nodes was considered to be equal to 2, 4 or 6[1]. The elements in each set V_S where randomly generated, considering 20 different seeds. For each set V_S of specified nodes, 100 node pairs were randomly generated for each network. This way twenty samples were obtained for each network, and 95% confidence intervals around the estimated mean were calculated, appearing in the graphs as error bars.

Algorithms SK66 and SK were not compared since the advantage of SK over SK66 is clear and the focus of this work is on routing with protection, not just on routing.

Results are presented regarding the percentage of problems solved (optimally or sub-optimally) with respect to the (optimal or sub-optimal) solutions obtained by the CPLEX solver. The relative error of the cost of the active path obtained by the heuristics, for which CPLEX also found an optimal solution, was calculated, in order to evaluate the quality of the solutions obtained by the heuristics.

The computational platform was a Desktop with 16 GB of RAM and an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz processor, with Kubuntu 12.4; the CPLEX solver, version 12.5 [14], was used for evaluating the performance of the heuristics.

Regarding the results obtained using networks from the SNDlib library, although the heuristics where in general significantly faster than CPLEX, the CPU time per node pair was in the latter case adequate for solving most of the problems. Nevertheless in several problem instances, namely for node pairs with no solution, CPLEX could take a very long time. For example, for some node pairs of the germany50 it took some hours (in fact 7 hours was the worst observed case) to conclude that the problem was infeasible. Hence, it was decided, in order obtain solutions in a reasonable time, to establish a limit of 5 minutes per node pair for the CPLEX solver.

Results obtained using networks from the SNDlib library are illustrated in Figures 1-3. Regarding the SNDlib tested networks it was observed that the number of feasible solutions increased with the average node degree of the network nodes (the best results being obtained for the newyork network). In average ABSK finds (as expected) the largest number of feasible solutions, followed by BSK and then by ASK. Also the number of feasible solutions found by the heuristics tends to decrease with the increase of the number of specified nodes. In general, results are quite good for three of the five networks, with values close to 99% and over 90% for newyork and india35 respectively (for $|V_S| = 2, 4, 6$), over 90% for pioro40 for $|V_S| = 2, 4$ and close to 90% for $V_S = 6$; the performance degrades for norway, and for the germany50 network the result regarding the number of feasible solution is rather low, namely for $V_S = 6$. The relative error of the cost of the active path also increases with the number of specified nodes to visit, from less than 6% in average to close to 13% in average for ABSK. Regarding the CPU time of the heuristics, the average was below 20 milliseconds, which is significantly faster than the average CPLEX result. CPLEX execution time varies significantly, from 20 milliseconds to 5 minutes (the latter usually for problems without solution) resulting in an average CPU time below 3 seconds per node pair. Some optimal solutions can take close to 3 minutes to be obtained (as was observed in the case of pioro40) and larger average CPU time would be observed if the CPU time limit of 5 minutes per node pair was not in place.

Results regarding the networks with 500 nodes are shown in Figures 4-6 for the five tested networks (with key 500_0 to 500_4, in the figures). These five networks were randomly generated by GT-ITM using the same parameter set. One can observe a high ratio of feasible solutions, always larger than 95% for ABSK, with two exceptions when $|V_S| = 4$ (where it is 93% and 94%); in these networks there is not a clear decrease of the ratio of feasible solutions with the increase of

[1]These numbers were suggested by PT Inovação (promoter of PANORAMA-II) based on their field experience.

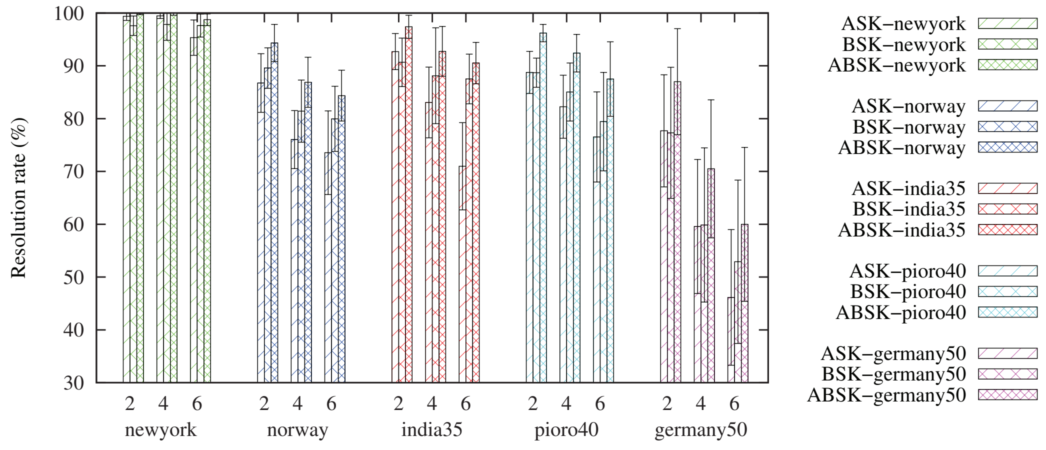

Fig. 1: Feasible solutions ratio with respect to CPLEX, for five SNDlib [12] networks

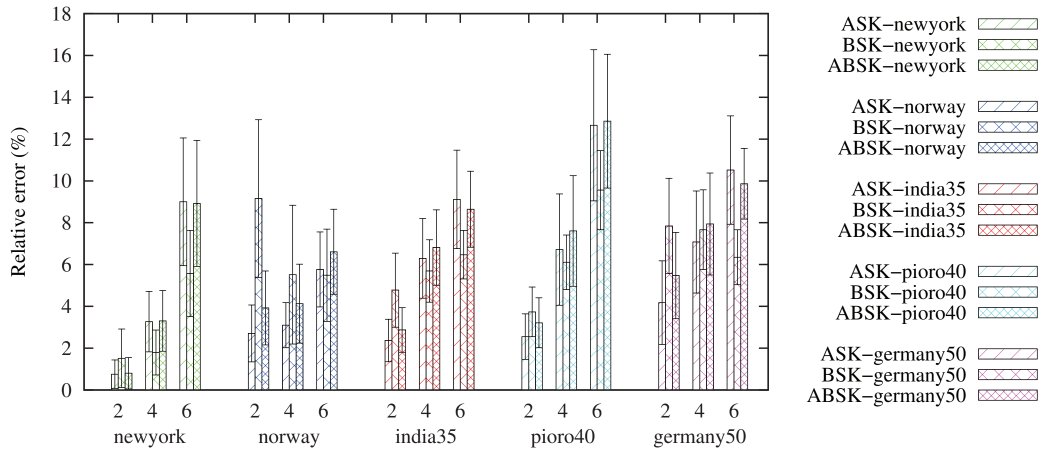

Fig. 2: Relative error of the feasible solutions found by the heuristics for five SNDlib [12] networks

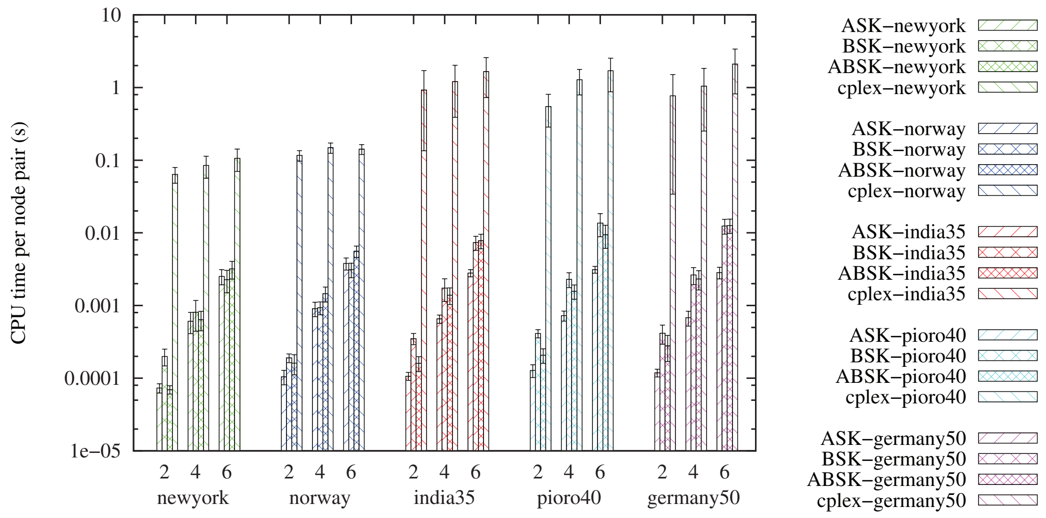

Fig. 3: CPU time per node pair for five SNDlib [12] networks

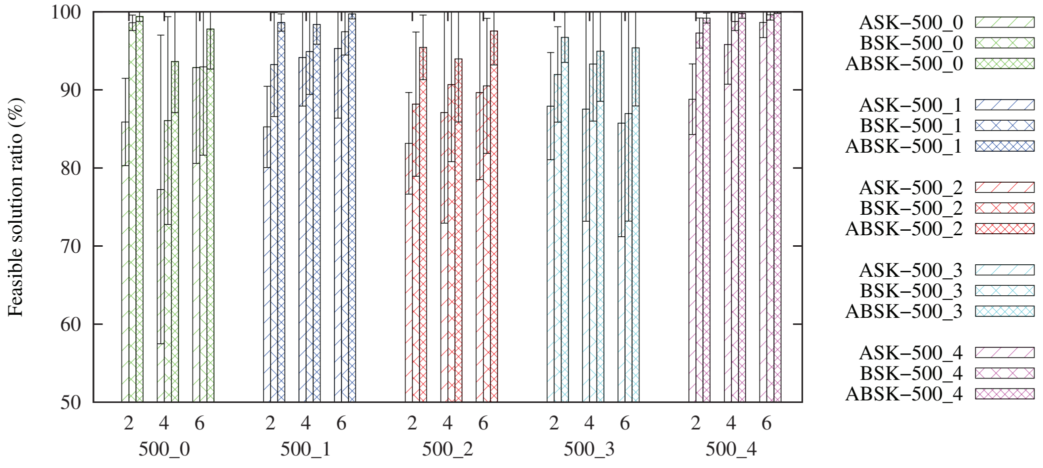

Fig. 4: Feasible solutions ratio with respect to CPLEX, in the 500 nodes networks

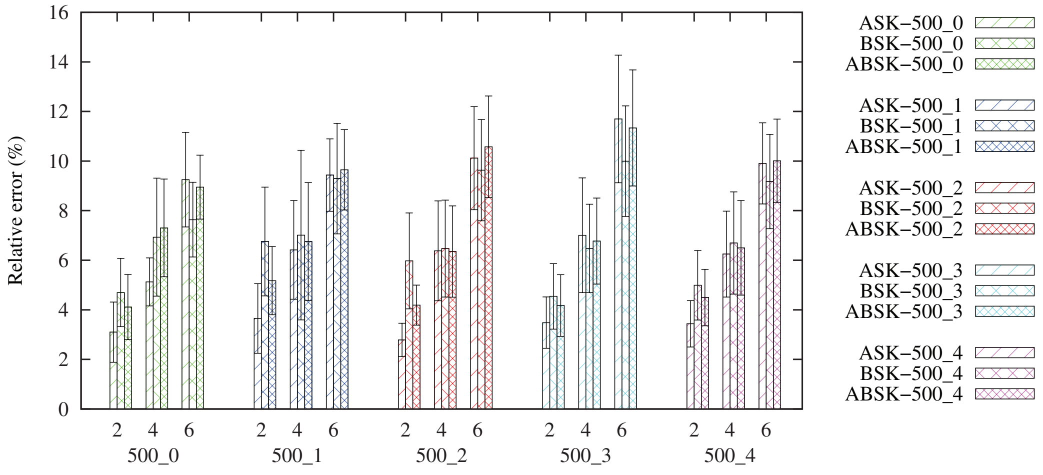

Fig. 5: Relative error of the feasible solutions found by the heuristics, in the 500 nodes networks

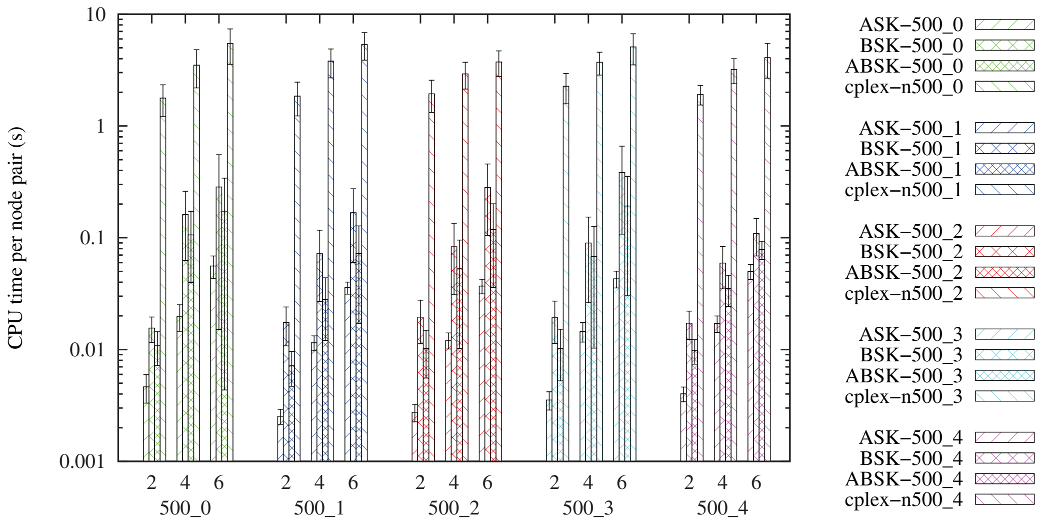

Fig. 6: CPU time per node pair, in the 500 nodes networks

the number of specified nodes to visit – see Figure 4. However the relative error of the feasible solutions does increase with the number of specified nodes to visit, going from up to 5% to up to 12% in average, as can be seen in Figure 5.

It is interesting to observe that the CPU time per node pair required by CPLEX in these larger networks (in average below 10 seconds) is not significantly larger than the average CPU time observed for the SNDlib networks. The CPU time of ABSK, although is larger than in the case of the SNDlib networks, is between 10 to 100 times smaller than the average CPU time required by CPLEX. The CPU times of CPLEX in Figure 6 present relatively large confidence intervals (notice the logarithmic scale). This results from the fact that for some problems CPLEX requires tens of seconds, and in some other (few cases) the allowed 5 minutes per node pair were exhausted and only a sub-optimal solution was found. Additionally it is worth noting that for one of the 500 nodes networks the heuristics found a solution for a node pair (when $|V_S| = 6$) while CPLEX after 5 minutes was not able to find a solution (optimal or sub-optimal).

The heuristic ABSK finds (in general) the largest number of feasible solutions, followed closely by BSK; moreover the former uses less CPU time than the later. Regarding accuracy, ASK presents in general a smaller relative error, but at the cost of a significantly less feasible solutions. The relative errors of BSK and ABSK differ less than 2%. Hence, one may conclude that ABSK is a good compromise between accuracy and CPU time.

VI. Conclusions

The need for the calculation of a path, from a source node to a target node, which must visit a given set of nodes may arise due to network management constraints. An ILP formulation [8] for solving this problem was adapted to include the restriction that the obtained solution must have a node-disjoint red backup path. This type of fixed route can be implemented in a number of network technologies and protocol layers such as, Internet Protocol (IP) networks with MPLS MultiProtocol Label Switching (MPLS) [15] (using explicit routing) and lightpaths in WDM optical networks.

SK, a modified and improved version of the algorithm proposed by Saksena and Kumar in [5], for calculating a shortest loopless path visiting specified nodes was developed. Three heuristics, based on SK, are put forward for calculating a protected shortest loopless path visiting specified nodes.

The performance of the proposed heuristics were evaluated using CPLEX to solve the ILP formulation given by equations (4)-(11). It was verified that although CPLEX may solve some problems in a short time it may also take many hours solving some problems in a 50 node network, namely in germany50 [12]. Therefore, in the results presented here, CPLEX was used considering a maximum of 5 minutes CPU time per node pair.

Results show that the heuristics were able to find solution in a significant number of cases and that the calculated solutions presented an acceptable relative error regarding the cost of the active path. The CPU time used by the heuristics is significantly smaller than the required by the ILP used solver.

The heuristics are capable of finding feasible solutions in a very short time, and were even able to find a solution for a node pair where after 5 minutes the ILP solver was not able to find a solution (optimal or sub-optimal). It was verified that ABSK presents a CPU time that does not exceed significantly the CPU time of ASK or BSK, while presenting higher ratio of feasible solutions and similar relative error. Therefore one may conclude that ABSK is a good compromise between accuracy and used CPU time. A study comparing the backup path costs is left for future work.

Acknowledgment

The authors acknowledge financial support through project QREN 23301 PANORAMA II, co-financed by European Union's FEDER through "Programa Operacional Factores de Competitividade" (POFC) of QREN (FCOMP-01-0202-FEDER-023301) and by Fundação para a Ciência e a Tecnologia (FCT) under project grant UID/MULTI/00308/2013.

References

[1] J. P. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks*, vol. 54, no. 8, pp. 1245 – 1265, 2010. Resilient and Survivable networks.

[2] P. Cholda, A. Mykkeltveit, B. Helvik, O. Wittner, and A. Jajszczyk, "A survey of resilience differentiation frameworks in communication networks," *Communications Surveys Tutorials, IEEE*, vol. 9, pp. 32–55, Fourth 2007.

[3] P. Cholda, J. Tapolcai, T. Cinkler, K. Wajda, and A. Jajszczyk, "Quality of resilience as a network reliability characterization tool," *Network, IEEE*, vol. 23, pp. 11–19, March 2009.

[4] R. Stankiewicz, P. Cholda, and A. Jajszczyk, "Qox: What is it really?," *Communications Magazine, IEEE*, vol. 49, pp. 148–158, April 2011.

[5] J. P. Saksena and S. Kumar, "The routing problem with 'k' specified nodes," *Operations Research*, vol. 14, no. 5, pp. 909–913, 1966.

[6] S. E. Dreyfus, "An appraisal of some shortest-path algorithms," *Operations Research*, vol. 17, no. 3, pp. 395–412, 1969.

[7] T. Ibaraki, "Algorithms for obtaining shortest paths visiting specified nodes," *SIAM Review*, vol. 15, no. 2, pp. 309–317, 1973.

[8] R. C. de Andrade, "Elementary shortest-paths visiting a given set of nodes," in *Simpósio Brasileiro de Pesquisa Operacional*, pp. 16–19, Setember 2013.

[9] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Management Science*, vol. 17, pp. 712–716, July 1971.

[10] E. Martins and M. Pascoal, "A new implementation of Yen's ranking loopless paths algorithm," *4OR – Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, no. 2, pp. 121–134, 2003.

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

[12] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design library," *Networks*, vol. 55, no. 3, pp. 276–286, 2010.

[13] M. Doar and I. Leslie, "How bad is naive multicast routing?," in *INFOCOM '93. Proceedings.Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future, IEEE*, pp. 82–89 vol.1, 1993.

[14] *IBM ILOG CPLEX Optimization Studio V12.5*. IBM, 2012.

[15] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP tunnels." IETF RFC 3209, December 2001.