Algorithms for Determining a Node-Disjoint Path Pair Visiting Specified Nodes

Teresa Gomes[a,b,*], Lúcia Martins[a,b], Sofia Ferreira[a], Marta Pascoal[c,b], David Tipper[d]

[a] *Department of Electrical and Computer Engineering, University of Coimbra, 3030-290 Coimbra, Portugal*
[b] *INESC Coimbra, Rua Antero de Quental 199, 3000-033 Coimbra, Portugal*
[c] *Department of Mathematics, University of Coimbra, 3001 – 501 Coimbra, Portugal*
[d] *Graduate Telecommunications and Networking Program, University of Pittsburgh, Pittsburgh, PA, USA*

Abstract

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. That ILP formulation is firstly adapted to include the constraint that the obtained path can be protected by a node-disjoint path, and secondly to obtain a pair of node disjoint paths, of minimal total additive cost, each having to visit a given set of specified nodes. Computational experiments show that these approaches, namely in large networks, may fail to obtain solutions in a reasonable amount of time. Therefore heuristics are proposed for solving those problems, that may arise due to network management constraints. Extensive computational results show that they are capable of finding a solution in most cases, and that the calculated solutions present an acceptable relative error regarding the cost of the obtained path or pair of paths. Further the CPU time required by the heuristics is significantly smaller than the required by the used ILP solver.

Keywords: Resilient routing, node-disjoint, visiting a given set of nodes.

1. Introduction

Communication networks are critical infrastructures of nowadays society. One of its most important features is their ability of providing a "reliable" service, in the sense that they contain mechanisms which allow to maintain or to restore the service in the case some failure occurs. This is related with the availability of the service and with the concepts of

[☆] Part of this work was originally presented at RNDM 2015 - 7^{th} International Workshop on Reliable Networks Design and Modeling, pages 120-127, Munich, Germany, 5-7 October 2015, with the title "Protected shortest path visiting specified nodes"

[*] Corresponding author

Email addresses: teresa@deec.uc.pt (Teresa Gomes), lucia@deec.uc.pt (Lúcia Martins), a2009118947@alunos.deec.uc.pt (Sofia Ferreira), marta@mat.uc.pt (Marta Pascoal), dtipper@pitt.edu (David Tipper)

resilience and survivability of communication networks. A recent systematic architectural framework that surveys and unifies these concepts was presented in [1]. In [2] the authors discuss deployment issues of fault recovery mechanisms in commercial optical networks, namely problem arising from hardware constraints. The existent resilience mechanisms can lead to several levels of services availability. These levels have been characterized by the notion of service Quality of Resilience (QoR), which was formally introduced in [3] and later explored in [4, 5].

In case of service interruption, different schemes of network recovery can be used: routing with protection or routing with restoration. In the first case the recovery procedure to implement is defined *a priori*, whereas in the second the recovery is sought in real time, after the fault has been detected. In the following we will focus on the first case, routing with protection.

Very often routing with protection is handled by computing in advance a pair of paths formed by an active, or primary, path (AP), and by a backup path (BP). As the names suggest, the backup path is to be used as an alternative to the active path, in case of failure in the latter. In order to ensure the backup path suits its purposes, usually routing with path protection is preceded by computing pairs of paths that are node (or arc, depending on the specific application) disjoint, such that a cost function is minimized. Polynomial time methods to find the minimum cost pair of disjoint paths, adapted from Dijkstra's algorithm, were proposed in [6, 7]. Several variants of these methods have been proposed by Bhandari [8].

Communication networks are intrinsically multi-layered. Because protection can be made at multiple layers, some links at an upper layer may be *resilient* because they are protected at a lower layer. In this case, when path diversity is used at an upper layer, it may not be necessary for the AP and BP to be fully disjoint. Paths that may share resilient arcs are said to be failure disjoint. The calculation of shortest pair of failure-disjoint paths is solved in [9], by a polynomial time algorithm. If a resilient arc is used by both paths, its cost is counted only once. In [10] the problem of calculating a pair of paths (without loops), from node s to node t, such that they are node-disjoint, except possibly at the end nodes of shared resilient arcs, is formalized and two novel algorithms are proposed for solving it.

For certain types of applications the active path must visit a set of nodes which is specified in advance. The nodes to be visited can be determined for different reasons. For instance, due to their characteristics or high reliability, depending on operator preferences resulting from agreements, or for other network management constraints. Fixed routes can be implemented in several network technologies, namely as lightpaths in WDM optical networks.

In general the problem of finding the shortest path that visits a given set of nodes is more difficult to solve than the unconstrained version, where no nodes are mandatory besides the source and the target. The problem was first mentioned by Kalaba [11] and a method for solving it was introduced by Saksena and Kumar [12]. This method, hereafter designated as SK66, is a dynamic programming algorithm, based on Bellman's optimality principle. The found solution is allowed to have repeated nodes (and, thus, cycles). However, later Dreyfus [13] stated that the solution this algorithm outputs is not always optimal. Dreyfus

also stated that the unconstrained version of the problem cannot be easier than the traveling salesman problem of dimension k, where $k-2$ nodes have to be visited.

Different variants of the problem were also studied in [14, 15, 16, 17, 18]. In particular, Ibaraki [16] addressed separately two versions of the problem of calculating the shortest path that visits a specified set of nodes, one where the solutions are loopless and another where solutions are paths (possibly with cycles). For the loopless paths case, two methods were presented: one based on dynamic programming, as well as a branch and bound algorithm. For the reported empirical results, the branch and bound method outperformed by the dynamic programming method.

More recently, the problem of finding a shortest loopless path that visits exactly once the nodes of a specified set was addressed by Andrade [14]. An initial linear integer formulation for the problem, with an exponential number of sub-tour elimination constraints, was considered. Two further formulations of the problem were also proposed. The first (Q2) is based on an adapted version of the cycle elimination constraints of the spanning tree polytope, whereas the second (Q3) is a primal-dual based mixed integer formulation. The presented computational results, for instances with up to 80 nodes and a set of 25%, 50%, 75% or 100% nodes to be visited, revealed that formulation Q3 was significantly easier to solve than formulation Q2. Andrade also pointed out that if all nodes in the graph, except the source and the target, are to be visited, then this corresponds to the NP-hard problem of finding a minimum cost Hamiltonian path.

Our contribution focuses on two problems: finding a minimum cost disjoint pair of loopless paths, the active path of which visits a given set of nodes, and finding a node-disjoint path pair of min-sum cost such that each of the paths visits a different given set of nodes. The integer linear model proposed in [14] is adapted for both these problems. Empirical tests show that an ILP solver has limitations regarding the size of the problems it can solve and regarding the demanded running times. Therefore, the purpose of this work is to present heuristics capable of providing a feasible solution to each of these problems that is close to the optimum, within a reasonable running time. To achieve this goal, heuristics for calculating the shortest loopless path visiting specified nodes had first to be developed.

The rest of the text is organized as follows. The next section is devoted to introducing notation, formally defining the two problems mentioned above and by adapting the model in [14] for solving them. In Section 3 the algorithm SK66 is reviewed, a modified version of SK66, designated SK [19], and a new heuristic, algorithm VSN, for calculating a shortest loopless path visiting specified nodes, are presented. In Section 4 heuristics for computing a protected shortest loopless path visiting a given set of nodes are proposed; two different approaches using SK as underlying algorithm are presented in Sub-sections 4.1 and 4.2; a new heuristic, designated VTA based on VSN, is described in Sub-section 4.3. The selection of the heuristics considered in the result analysis is given in Sub-section 4.4. Two heuristics for calculating a pair of node-disjoint loopless paths, each visiting a specified set of nodes, such that the sum of the cost of the paths is minimum, are presented in Section 5. Extensive computational results are presented and analyzed in Section 6. Finally, conclusions are drawn in Section 7.

2. Notation and problems formulation

In this work one seeks to solve the following problems:

Problem $\mathcal{P}_1$: obtaining the shortest loopless path visiting a specified set of nodes, with the constraint that it must be protected using a node-disjoint path.

Problem $\mathcal{P}_2$: obtaining a pair of node-disjoint loopless paths, each visiting a specified set of nodes, such that the sum of the cost of the paths is minimum.

Unless explicitly stated otherwise, all paths are considered to be loopless.

2.1. Notation

The heuristics proposed in sections 3 and 4 use the following notation. Let the graph $G = (V, A)$ be defined by a set of n nodes $V = \{v_1, \ldots, v_n\}$, and a set of (directed) m arcs $A = \{a_1, \ldots, a_m\}$. An arc connects two vertices in a given order, and is represented as 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_1} 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 $\tau = \lceil m/n \rceil^2 |V_{S_1}|$ will be used to limit shortest path enumeration in some of the proposed heuristics.

2.2. Formulation of problem $\mathcal{P}_1$

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:

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

$$\text{subject to:} \quad V_{p_1} \cap V_{S_1} = V_{S_1} \wedge V_{p_1} \cap V_{p_2} = \{s, t\} \quad (2)$$

A path from s to t which visits the nodes in V_{S_1} will also be represented by an s–V_{S_1}–t path, as in [14]. Once a path p_1^* has 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 deleted. This simple approach ensures that the paths p_1^* and p_2^* are 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,

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

Let $p = \langle s = v_1, v_2, \ldots, v_k = t \rangle$ be a shortest s–V_{S_1}–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 [14] with the additional constraint that the obtained path can be protected by a node-disjoint path.

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

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

$$\sum_{(v_i,v_j)\in\delta(i)^+} x_{(i,j),1} = 1, \quad \forall v_i \in V_{S_1}, \tag{6}$$

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

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

$$\pi_s = 0 \tag{9}$$

$$\pi_{v_i} \geq 0, \forall v_i \in V \setminus \{s\} \tag{10}$$

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

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_1} 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 [14] for a proof. Otherwise, if $x_{ij} = 0$, then the constraints (7) and (8) are redundant, for a satisfactory M. 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 deleted (recall that $p_1^* \neq \langle s, t \rangle$, assuming $V_{S_1} \neq \emptyset$).

2.3. Formulation of problem $\mathcal{P}_2$

The problem is to find a pair of node-disjoint loopless paths, each visiting a specified set of nodes, such that the sum of the cost of the paths is minimum. This a min-sum problem with constraints. Let V_{S_2} designate the set of specified nodes that must be visited by the backup path.

$$
\begin{aligned}
(p_1^*, p_2^*) &= \arg \min_{p_1, p_2 \in \mathcal{P}_{st}} \{D_{p_1} + D_{p_2}\} && (12)\\
\text{subject to:} \quad & V_{p_1} \cap V_{S_1} = V_{S_1} && (13)\\
& V_{p_2} \cap V_{S_2} = V_{S_2} && (14)\\
& V_{p_1} \cap V_{p_2} = \{s, t\} && (15)\\
& && (16)
\end{aligned}
$$

Whenever possible the notation introduced in the previous subsection will be used. Let $p_u = \langle s = v_1, v_2, \ldots, v_k = t \rangle$, with $u = 1, 2$, be a s–V_{S_u}–t path, and $\pi_{v_i,u}$, $v_i \in V_p$, be the cost of going from node s to node v_i in the path p_u ($u = 1, 2$). The ILP formulation which follows is an extension of the formulation in the previous section.

$$\min \quad \sum_{(v_i,v_j)\in A} w(v_i,v_j)[x_{(i,j),1} + x_{(i,j),2}] \tag{17}$$

$$\text{s.t.:} \quad \sum_{(v_i,v_j)\in\delta(i)^+} x_{(i,j),u} - \sum_{(v_j,v_i)\in\delta(i)^-} x_{(j,i),u} = \begin{cases} 1 & : \quad v_i = s, \\ -1 & : \quad v_i = t, \\ 0 & : \quad v_i \in V\backslash\{s,t\} \end{cases} \quad \forall v_i \in V,\ u = 1,2 \tag{18}$$

$$\sum_{(v_i,v_j)\in\delta(i)^+} x_{(i,j),u} = 1, \quad \forall v_i \in V_{S_u}, u = 1,2 \tag{19}$$

$$\pi_{v_j,u} - \pi_{v_i,u} \leq w(v_i,v_j) + M(1 - x_{(i,j),u}), \quad \forall (v_i,v_j) \in A, u = 1,2 \tag{20}$$

$$\pi_{v_j,u} - \pi_{v_i,u} \geq w(v_i,v_j) - M(1 - x_{(i,j),u}), \quad \forall (v_i,v_j) \in A, u = 1,2 \tag{21}$$

$$\pi_{s,u} = 0, \quad u = 1,2 \tag{22}$$

$$\pi_{v_i,u} \geq 0, \forall v_i \in V \setminus \{s\}, u = 1,2 \tag{23}$$

$$\sum_{(v_i,v_j)\in\delta(i)^+} x_{(i,j),1} + x_{(i,j),2} \leq 1, \quad \forall v_i \in V \setminus \{s\} \tag{24}$$

x are the binary decision variables.

In the optimization problem M is, as in the previous problem, a sufficiently large number and:

- constraints (18) and (24) are similar to (5) and (11), respectively;
- constraints (19)-(23) extend constraints (6)-(10), for considering node-disjoint paths p_u $(u = 1, 2)$, from node s to node t, where p_u must visit the set of specified nodes V_{S_u}.

3. Shortest path visiting specified nodes

The algorithm by Ibaraki [16] 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 ($|V_{S_1}|$); in the initial step it calculates $(|V_{S_1}| + 2)|V_{S_1}|$ shortest paths; in each step it demands $|V_{S_1}|^2$ operations, of which the most time consuming is node counting, with worst case complexity $\mathcal{O}(n \log n)$; hence the worst case complexity of SK66 is $\mathcal{O}(|V_{S_1}|^2(m+n)\log n)$ if Dijkstra's algorithm (with a binary heap) is used in the calculation of the shortest sub-paths.

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

The algorithm SK66, seeks to determine 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_1}, and also between the source node s and every node in V_{S_1};
2. the calculation of the shortest path from every node in V_{S_1} to the target node.

Let $D(v_i, v_l)$ designate the cost of the shortest path from v_i to v_l, with $v_i \in V_{S_1} \cup \{s\}$ and $v_l \in V_{S_1}$. Let $f_{v_i}^0$ designate the cost of the shortest path from node $v_i \in V_{S_1}$ to t.

The algorithm can now iteratively calculate:

$$f_{v_i}^{\eta} = \min_{v_l \in V_{S_1}} \{D(v_i, v_l) + f_{v_l}^{\eta-1}], \quad v_l \neq v_i\} \tag{25}$$

for $\eta = 1, 2, \ldots, |V_{S_1}| - 1$, intending 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

$$f_s^{|V_{S_1}|} = \min_{v_l \in V_{S_1}} \{D(s, v_l) + f_{v_l}^{|V_{S_1}|-1}\} \tag{26}$$

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

The shortest paths calculated in the initial steps of the algorithm, will be the building blocks of the path computed according to equations (25) and (26).

3.2. 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 (26) and the sub-paths selected by equation (25) must satisfy the restriction that they must not contain a cycle.

In each iteration of SK66, according to equation (25), for every $v_i \in V_{S_1}$ a new via node v_l ($v_l \in V_{S_1}$) 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 further on be abandoned because they lead to an infeasible solution. Let

$$f_{v_i,v_l'}^1 = D(v_i, v_l') + f_{v_l'}^0, \quad v_l' \neq v_i \tag{27}$$

$$f_{v_i,v_l}^{\eta} = D(v_i, v_l) + \min_{v_j} f_{v_l,v_j}^{\eta-1}, \quad v_l \neq v_i, v_j \wedge v_i \neq v_j \tag{28}$$

with $\eta = 2, \ldots, |V_{S_1}|$. 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 (25). In the original version of the algorithm, at the end of each iteration $\eta = 1, \ldots, |V_{S_1}| - 1$ only one path was selected (unless a tie in the minimum cost occurred) for each source node v_i – see (25). In this modified version up to $|V_{S_1}| - 1$ paths are selected for each source node v_i.

Additionally the following modifications were introduced.

1. Seeking to avoid cycles, the initial steps were modified as follows: the shortest path from s to $v_l \in V_{S_1}$ of cost $D(s, v_l)$ is calculated in a network where node t was deleted; similarly the shortest path from v_i to v_j, with $v_i, v_j \in V_{S_1}$, is calculated in a network where nodes s and t were deleted; finally the shortest path from node $v_i \in V_{S_1}$ to t of cost $f^0_{v_i}$ is calculated in a network where node s was deleted.
2. If in iteration η, all the paths from v_i to t, using via node v_l, of cost $D(v_i, v_l) + f^{\eta-1}_{v_l,v_j}$, were considered non admissible, and in the previous iteration, a path from node v_i to t, with $|V_{S_1}| - 1$ 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 stored as if it had been computed 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 computed in the present iteration, the path of the $\eta-1$ iteration replaces the one computed in the η-th iteration (and is stored as if it had been computed in the η-th iteration).

The complexity of SK is in the worst case $|V_{S_1}| - 1$ times larger then that of SK66, because it stores, for each new added specified node, up to $|V_{S_1}| - 1$ solutions, instead of only the min-cost solutions as SK66.

3.3. A new heuristic for calculating a path visiting specified nodes

In most practical cases within the context of our application, the number of specified nodes that a path must visit is relatively small (at most 6). Assuming that the number of elements of set V_{S_1} is small, the following approach is efficient, as the empirical results will show.

The main idea is to build an auxiliary graph G_{S_1}, with node set $\{s, t\} \cup V_{S_1}$, where the shortest paths visiting all nodes in the graph can be obtained using a k-shortest path enumeration algorithm like Yen's [20] or MPS [21].

The computation of the auxiliary graph is explained next. First the shortest path from s to each node in V_{S_1}, from t to each node in V_{S_1} and between every node pair of nodes in V_{S_1} is calculated, as described next. The shortest path from s to each node $v_i \in V_{S_1}$ is calculated in the network resulting from removing the nodes in set $\{t\} \cup V_{S_1} \setminus \{v_i\}$ – this ensures these paths do not contain t or any other specified node. Similarly, the shortest path from v_i ($v_i \in V_{S_1}$) to t is calculated in the network resulting from removing the nodes in set $\{s\} \cup V_{S_1} \setminus \{v_i\}$ – this ensures these paths do not contain s or any other specified node. Finally for every node pair $v_i, v_j \in V_{S_1}$, the shortest path in the network where the nodes in set $\{s, t\} \cup V_{S_1} \setminus \{v_i, v_j\}$ have been deleted are calculated – this ensures these paths

do not contain s, t or any other specified node. These paths are calculated using a breadth first search (BFS) [22, 23] algorithm to ensure the shortest path with the minimum number of arcs is obtained.

Every calculated path (if it exists) will be represented in the auxiliary graph by an arc, emergent from the source node of the path and incident in the destination node of the path, with cost equal to the cost of the path represented by that arc. Now one can calculate shortest paths from s to t until a shortest path visiting all nodes in this auxiliary graph is obtained. Every arc of this path is expanded into the path it corresponds to in the original network. If the resulting expanded path does not contain any cycle it is a feasible solution to the min-cost loopless path visiting specified nodes, and the algorithm ends. Otherwise the path (with cycles) is stored in a first in first out queue (say Q) for possible later processing, and the k-shortest paths method keeps generating paths as long as they have the same minimum cost; this last condition has two purposes: avoiding accepting a path that is more expensive than necessary and preventing (if possible) the generation of all paths in G_{S_1}. If these actions do not result in finding a loopless path visiting all nodes in V_{S_1}, then the strategy is to successively delete one arc (selected using the list Q) from the original network, and repeat the above procedure (which starts by recomputing the auxiliary graph) until a solution is found, or n arcs have been deleted, or no paths from s to t can be obtained in the auxiliary graph.

The algorithm VSN contains a detailed description of the this new approach, where the function *auxGraph* corresponds to building the auxiliary graph G_{S_1}, as described above. The function *arcInCycle* is described next.

Given the sequence of generated paths stored in Q, the function *arcInCycle* will identify among the arcs in cycles the one present (in cycles) in more paths, and in case of a tie it selects the one with the largest cost. If only nodes are repeated, then function *arcInCycle* will determine the largest number of paths (f) where the same repeated node appears. Afterwards it will search for the first path inserted in Q containing a node which appears in f paths, and deletes from the original network the first arc, in the path, incident in that node.

The problem above boils down to the Hamiltonian path problem, which is a particular case of the traveling salesman problem, and both are NP-complete. The applicability of the strategy of algorithm VSN is limited in practice by $|V_{S_1}|$, as was already mentioned. Other exact methods for the Hamiltonian path problem, or for the more general traveling salesman problem, could be used, for instance [24, 25]. When $|V_{S_1}|$ is big an approximate method will have to be used instead, for instance [26, 27].

The complexity of VSN is given by the cost of calculating $(|V_{S_1}| + 2)|V_{S_1}|$ shortest paths using BFS and by calculating at most n times the required number of paths in the auxiliary graph. Let $n_g = |V_{S_1}| + 2$ and $m_g = |V_{S_1}|^2 + |V_{S_1}|$ be the number of nodes and maximum number of arcs of the auxiliary sub-graph. Therefore VSN complexity is $\mathcal{O}\left(|V_{S_1}|^2(m+n) + nKn_g(m_g + n_g \log n_g)\right)$, where K is the number of paths calculated in the auxiliary graph using Yen's algorithm.

Algorithm VSN: New heuristic for calculating a min-cost path visiting specified nodes

Input: $G = (V, A)$, s, t, V_{S_1}
Output: p

```
1  begin
2      p ← ∅; pathFound ← false; ;                          // No solution yet
3      i ← 0;                                  // When equal to n will stop the cycle
4      A_r ← ∅;                                          // set of arcs deleted from G
5      repeat // Generates auxiliary graph in a successively pruned network
6          G_{S_1} ←auxGraph(G, V_{S_1}, s, t);          // G_{S_1} = (V_{S_1} ∪ {s,t}, A_{S_1})
7          Q ← ∅;                                // resets queue of paths with cycles
8          noPath ← true ;   // until finding the 1st p_{S_1} ∈ G_{S_1} : |V_{p_{S_1}}| = 2 + |V_{S_1}|
9          firstPathCost ← ∞ ;                                // cost of the 1st path
10         Starts a k-shortest path enumeration algorithm from s to t in G_{S_1};
11         repeat                          // Until a path is found or cost is too high
12             p_{S_1} ←k-shortest(G_{S_1}, s, t);       // next s-t shortest path in G_{S_1}
13             if |V_{p_{S_1}}| = 2 + |V_{S_1}| then          // p_{S_1} visits all nodes of G_{S_1}
14                 if noPath then       // first p_{S_1} with 2 + |V_{S_1}| nodes is found
15                     noPath ← false;
16                     firstPathCost ← D_{p_{S_1}} ;     // saves cost of the first path
17                 end
18                 p_G ←expandPath(G, p_{S_1}) ;     // expands p_{S_1} into a path in G
19                 if p_G has a cycle then push p into Q;          // No solution yet
20                 else
21                     pathFound ← true;
22                     p ← p_G;                                // a feasible solution
23                 end
24             end
25         until pathFound ∨ p_{S_1} = ∅ ∨ D_{p_{S_1}} > firstPathCost;
26         if ¬pathFound ∧ Q ≠ ∅ then
27             i ← i + 1;                          // arc removal counter is updated
28             a ← arcInCycle(Q) ;                    // selects arc to be deleted
29             A ← A \ {a} ;                              // deletes arc a from G
30             A_r ← A_r ∪ {a} ;                     // updates set of deleted arcs
31         end
32     until pathFound ∨ i > n ∨ Q = ∅;
33     A ← A ∪ A_r ;                          // restores the input graph G = (V, A)
34     return p;          // if p is not empty it ends with a feasible solution
35 end
```

4. Protected shortest path visiting specified nodes

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 in Sub-sections 4.1 and 4.2, the first modifying algorithm SK for obtaining segments of the AP that can be protected, and the second using algorithm SK after having calculated a candidate BP path. These two approaches are then combined into a third heuristic as explained in Sub-section 4.4 for improving the number of obtained feasible solutions as was shown in [19].

In Sub-section 4.3 algorithm VTA is developed, which incorporates an approach inspired by the Trap Avoidance algorithm proposed in [28], for solving the same problem.

4.1. 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.

The initial steps of ASK are:

1. A k-shortest path enumeration algorithm [20, 21] is used to generate up to τ (recall that $\tau = \lceil m/n \rceil^2 |V_{S_1}|$) paths from (v_i, v_j), with $v_i \in V_{S_1} \cup \{s\}$ and $v_j \in V_{S_1}$; as soon as a k-th path from v_i to v_j, p_{ij}^k $(k = 1, \ldots, \tau)$, is found such that removing the set of nodes $(V_{p_{ij}^k} \cup V_{S_1}) \setminus \{s\}$ a path from node s to node t can be found, the path generation procedure stops; the path p_{ij}^k is stored and $D(v_i, v_j)$ takes the cost of that path. If none of the generated paths can be protected $D(v_i, v_j) = \infty$.
2. A k-shortest path enumeration algorithm [20, 21] is used to generate up to τ paths from (v_i, t), with $v_i \in V_{S_1}$; as soon as a k-th path from v_i to t, p_{it}^k $(k = 1, \ldots, \tau)$, is found such that removing the set of nodes $(V_{p_{it}^k} \cup V_{S_1}) \setminus \{t\}$ a path from node s to t can be found, the path generation procedure stops; the path p_{it}^k is stored and $f_{v_i}^0$ takes the cost of that path. If none of the generated paths can be protected $f_{v_i}^0 = \infty$.

Using these sub-paths as building blocks, 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 $V_{S_1} \cup V_{p_{v_i t}} \setminus \{t\}$ 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 $D(s, v_l) + f_{v_l, vj}^{|V_{S_1}|-1}$.

The final active path selection in the algorithm is done initially using equation (26), 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 $f_{v_l}^{|V_{S_1}|-1}$) does not contain a cycle, one must then verify if the resulting path has 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* (26).

Build path p_1 with cost $D(s, v_l) + f_{v_l,v_i}^{|V_{S_1}|-1}$, 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_lt} the path with cost $f_{v_l,v_i}^{|V_{S_1}|-1}$ $(v_l, v_i \in V_{S_1})$ obtained in the $\eta = |V_{S_1}| - 1$ iteration of the algorithm.

If $p_1 = p_{sv_l} \diamond p_{v_lt}$ 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_lt}, with cost $f_{v_l,v_i}^{|V_{S_1}|-1}$ $(v_l, v_i \in V_{S_1})$, a shortest path p_{sv_l} in the network where the nodes $V_{p_{v_lt}} \setminus \{v_l\}$ have been deleted is calculated; then if a node-disjoint path can be found with the path $p_{sv_l} \diamond p_{v_lt}$, a feasible solution has been found.

(c) *Seeking to find a node-disjoint backup with the sub-path p_{v_lt} with cost $f_{v_l,v_i}^{|V_{S_1}|-1}$.*

For every sub-path p_{v_lt} with cost $f_{v_l,v_i}^{|V_{S_1}|-1}$ $(v_l, v_i \in V_{S_1})$, a backup path p_2 is calculated from s to t in a network where the nodes $V_{p_{v_lt}} \setminus \{t\}$ have been deleted. If p_2 exists, a sub-path p_{sv_l} is then calculated in the network where the nodes in set $V_{p_2} \cup V_{p_{v_lt}} \setminus \{s, v_l\}$ have been deleted. If this sub-path exists then $p_1 = p_{sv_l} \diamond p_{v_lt}$ is a feasible solution.

(d) *Seeking to find a node-disjoint backup with the sub-path p_{sv_k}, where v_k is the most downstream among the specified nodes of the path of cost $D(s, v_l) + f_{v_l,v_j}^{|V_{S_1}|-1}$.*

For every sub-path p_{v_lt} with cost $f_{v_l,v_i}^{|V_{S_1}|-1}$ $(v_l, v_i \in V_{S_1})$, let p_{sv_l} be the shortest path from s to v_l of cost $D(s, v_l)$, and $p'_1 = p_{sv_l} \diamond p_{v_lt}$ a loopless path. Let v_k be the most downstream of the nodes of p'_1, such that $v_k \in V_{S_1}$; a backup path p_2 is calculated from s to t in a network where the nodes in set $V_{p_{sv_k}} \setminus \{s\}$ have been deleted. If p_2 exists, a sub-path from v_k to t, p_{v_kt}, is then calculated in the network where the nodes in set $V_{p_2} \cup V_{p_{sv_k}} \setminus \{v_k, t\}$ have been deleted.

Note that 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).

4.2. 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_1} have been deleted. First it generates the largest set of node-disjoint paths of min-sum cost, according to Bhandari's algorithm [8]. 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 deleted. If more than one solution was found, it selects the one with smaller cost for the AP.

If the previous procedure was unable to elicit a solution, then a k-shortest path enumeration algorithm [20] and [21] is used to calculate up to τ candidate backup paths in a network where the nodes in V_{S_1} have been deleted. 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.

4.3. A trap avoidance approach

Algorithm VSN can be used to calculate a path p, from s to t, visiting a set of specified nodes; then, in a network where the intermediate nodes of p were deleted it calculates a shortest path. If this path can be obtained a feasible solution for problem $\mathcal{P}_1$ has been found. However, some times it is not possible to derive this second path, and it can be said that this procedure has fallen into a trap. The trap can be a real trap if the problem has no solution or an avoidable trap if a different first path would make it possible to obtain a disjoint path [28].

The approach proposed here is based on the idea underlying the Trap Avoidance algorithm proposed in [28] for Shared Risk Link Group protection. Whenever the path obtained by algorithm VSN, the candidate AP, can not be protected by a node-disjoint BP, there is at least one node in the AP that prevents a solution from being found. In order to be capable of identifying this conflicting node, the arcs adjacent to the intermediate nodes of the AP, which do not belong to V_{S_1}, are not deleted but have their cost increased by a sufficiently large value (M). Hence, when the backup path is calculated, if its cost is larger than M, then no solution was found for problem $\mathcal{P}_1$ but a conflicting node can now be identified. In the algorithm proposed here, the first common node in the candidate AP is identified as being the cause of the algorithm falling into a trap. However, instead of removing the node, which could drastically limit the possible solutions to the problem, the arc (of the candidate AP) incident in the conflicting node is selected for removal. This procedure is repeated, successively removing arcs until a solution is found or a candidate AP can no longer be obtained. If the candidate AP can not be obtained, the algorithm is allowed to backtrack and look for the last deleted incident arc and replace it with the corresponding emergent arc. The maximum number of allowed backtracking procedures is equal to n.

The corresponding algorithm is VTA. Notice that before attempting to calculate the candidate AP the arcs in stack *Risky* (the ones considered, that if used by the AP will not allow it to have a node-disjoint BP) are deleted in line 7 of VTA. Then, after calculating the candidate AP (p_c in line 8), it must restore the network – see line 9.

If a candidate AP was found, the unconstrained backup path is calculated in the network where nodes V_{S_1} have been deleted and the arcs incident in the remaining intermediate nodes

of the candidate AP have their cost increased of a sufficiently large value (see lines 19-21 of algorithm VTA). If the backup path (q_c in the algorithm) exists and has cost less than M, a solution has been found (and the algorithm ends). Otherwise a conflicting node is identified (see lines 27 and 28). The arc of p_c incident in that node is stored both in *inRisky* and *Risky*, and will be deleted before calculating the next candidate AP; the emergent arc of that node is stored in *outRisky* (for allowing some backtracking) – see lines 29-31.

If no candidate AP was found and the maximum allowed number of backtracking has not been reached, arcs from *Risky* (and from *inRisky* and *outRisky*) are deleted until an incident arc is found and replaced by the corresponding emergent arc in *outRisky* (see lines 12-15). If *Risky* becomes empty the algorithm will end (due to *stop* becoming true in line 16) because no more backtracking is to be considered. Note that, at the start of each iteration, these three stacks have always the same size, and any element at the top of *Risky* will always be equal to the element at the top of *inRisky* or (exclusive) *outRisky* (assuming the stacks are not empty).

4.4. Considered heuristics

Four versions of the heuristic were implemented: the algorithm ASK, described in Sub-section 4.1; the algorithm BSK, presented in Sub-section 4.2; the heuristic ABSK, which consist in invoking algorithm ASK, and then, if no feasible solution was found, invoking also algorithm BSK; and finally VTA, the algorithm proposed in Sub-section 4.3

The discussion of the results in [19] justified the proposal of the ABSK version, and in [19] it was concluded that ABSK, with respect to ASK and BSK, was a good compromise between accuracy and CPU time. Therefore in the present work only ABSK will be compared with VTA, the new proposed heuristic.

5. Min-sum node-disjoint path pair visiting specified nodes

Two different approaches for solving problem $\mathcal{P}_2$ are proposed next. Note that, because V_{S_2} is the set of specified nodes of the BP, the AP or the sub-paths that will be the building blocks of the AP, are calculated (by the next two algorithms) in a network where the nodes in V_{S_2} have been previously deleted.

5.1. Min-sum node-disjoint path pair visiting specified nodes based on ASK

The heuristic based on algorithm ASK, designated MSASK, follows the same strategy as ASK when trying to find an AP visiting nodes V_{S_1}: every time a new specified node is added to a sub-path, it verifies if this sub-path has a node-disjoint path from s to t. However, because in the present problem the BP must visit nodes V_{S_2}, MSASK must replace the use of Dijkstra or BFS algorithm (used by ASK) by the SK algorithm to check if a BP visiting nodes V_{S_2} can be obtained. The network where SK tries to obtain the BP, corresponds to the original network, where the nodes V_{S_1} and the intermediate nodes of the AP (under construction) have been deleted.

When MSASK has to select among possible sub-paths of the AP it will select the one with the smallest cost, just like ASK. However, when MSASK obtains more than one solution

Algorithm VTA: Tap avoidance and Protected path visiting specified nodes

Input: $G = (V, A)$, s, t, V_{S_1}
Output: (p, q)

begin
  $(p, q) \leftarrow (\emptyset, \emptyset)$; `// No solution yet`
  $A_{S_1} \leftarrow$ set of arcs adjacent to the nodes in V_{S_1};
  inRisky, *outRisky* and *Risky* are empty stacks of arcs.;
  $stop \leftarrow false$; $k \leftarrow 0$; `// for stopping;` k `counts backtracking`
  repeat
    Remove from A the set of arcs in *Risky* ;
    $p_c \leftarrow$ VSN$(G = (V, A), s, t, V_{S_1})$; `// seeks a candidate AP`
    Restore into A the set of arcs in *Risky* ;
    if $p_c = \emptyset \wedge k < n$ **then** `// no candidate AP was found`
      $k \leftarrow k + 1$;
      while *(Risky is not empty)* $\wedge$ `top(`*Risky*`)` $\neq$ `top(`*inRisky*`)` **do**
        `pop(`*Risky*`); pop(`*inRisky*`); pop(`*outRisky*`)` ;
      end
      if *Risky is not empty* **then** `pop(`*Risky*`); push(`*Risky*`,` `top(`*outRisky*`))` ;
      else *stop* $\leftarrow$ *true* ; `// no more backtracking is considered`
    end
    else `// a candidate AP was found`
      $A \leftarrow A \setminus A_{S_1}$; `// deletes arcs adjacent to nodes in` V_{S_1}
      $A_c \leftarrow$ arcs adjacent to the nodes in $V_{p_c} \setminus (V_{S_1} \cup \{s, t\})$;
      forall the $a \in A_c$ **do** $c(a) \leftarrow c(a) + M$;`// add` M `to cost of` A_c `arcs`
      $q_c \leftarrow$shortest path from s to t in G; `// calculates BP`
      $A \leftarrow A \cup A_{S_1}$; `// restores network topology`
      forall the $a \in A_c$ **do** $c(a) \leftarrow c(a) - M$; `// restores costs`
      if $q_c \neq \emptyset$ **then**
        if $D_{q_c} > M$ **then** `// AP and BP are not node-disjoint`
          $v_i \leftarrow$ first node, from s to t, of p_c in V_{q_c};
          $v_{i-1} \leftarrow$ the node before v_i in p_c; $v_{i+1} \leftarrow$ the node after v_i in p_c;
          `push(`*Risky*`,` (v_{i-1}, v_i)`)` ; `// incident arc in` v
          `push(`*inRisky*`,` (v_{i-1}, v_i)`)`; `// incident arc in` v
          `push(`*outRisky*`,` (v_i, v_{i+1})`)` ; `// emergent arc from` v
        end
        else $(p, q) \leftarrow (p_c, q_c)$; $stop \leftarrow true$; `// AP and BP are node-disjoint`
      end
      else *stop* $\leftarrow$ *true* ; `// No solution was found`
    end
  until *stop*;
  return (p, q); `// if` $(p, q) \neq (\emptyset, \emptyset)$ `there is a feasible solution`
end

(full paths from s to t visiting nodes V_{S_1}, for which a BP visiting nodes V_{S_2} was found), it will select the path pair of min-sum total cost.

5.2. Min-sum node-disjoint path pair visiting specified nodes based on VTA

The heuristic based on algorithm VTA, designated MSVTA, follows the same strategy as VTA when trying to find an AP visiting nodes V_{S_1}, and will be described here pointing out the differences with respect to that algorithm.

Firstly, in line 7, besides removing the arcs in *Risky* it must also delete the arcs adjacent to the nodes in V_{S_2}. Then, after calculating the candidate AP (p_c in line 8), it must restore the network. Hence, in line 9, it must add back not only the arcs in *Risky* but also the arcs adjacent to the nodes in V_{S_2}.

In line 22, instead of calculating an unconstrained shortest path (in the previously modified network, see lines 19-21), it must invoke algorithm VSN to calculate a path from s to t visiting the nodes in V_{S_2}. If that path (q_c in the algorithm) exists and has cost less than M, a feasible solution has been found (and the algorithm ends). Otherwise, a conflicting node (common to both q_c and p_c) is identified. The arc of p_c incident in that node is stored both in *inRisky* and *Risky*, and will be deleted before calculating the next candidate AP; the emergent arc of that node is stored in *outRisky* (for allowing some backtracking).

This approach will minimize the cost of the AP and then will try to obtain a BP of minimum cost. As the objective is to minimize the total cost, MSVTA after having tried (successfully or not) to obtain an AP and a BP visiting node sets V_{S_1} and V_{S_2}, respectively, it will swap the contents of V_{S_1} and V_{S_2} and try again. Finally it will select the path pair of min-sum cost, assuming both runs resulted in a solution (having previously swapped the AP and BP obtained in the second run).

6. Results

Experimental tests were made considering networks from two different sources. The first set of networks are from the SNDlib [29], 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 [30] 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 to be visited[1] was considered to be equal to 2, 4 or 6 for the shortest path with or without protection; for the path pair of min-sum cost the number of specified nodes in each path was considered to be 2, 3 and 4, for the networks from the SNDlib and 2, 3, 4 and 6 for the 500 nodes networks (and for simplicity $|V_{S_1}| = |V_{S_2}|$, in problem $\mathcal{P}_2$). For the considered SNDlib instances, problem $\mathcal{P}_2$, with $|V_{S_1}| = |V_{S_2}| = 6$, did not have a solution for most origin-destination node pairs, and therefore this value was

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

not used. The elements in each set V_{S_1} (and V_{S_2}) were randomly generated, considering 20 different seeds. For each set V_{S_1} (or pair of sets V_{S_1} and V_{S_2}) of specified nodes, 100 origin-destination node pairs were randomly generated for each network. Twenty samples (average values resulting from each set of 100 node-pairs) were obtained for each network (and number of specified nodes), and 95% confidence intervals around the estimated mean were calculated, appearing in the graphs as error bars. The results presented here were obtained using Yen's algorithm [20, 21] for the k-shortest path enumeration.

Algorithms SK66 and SK are not compared since the first generates paths with cycles, and the objective is to calculate loopless paths.

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 or of the path pair (depending on the problem being solved) for which CPLEX also found a solution, was calculated, in order to evaluate the quality of the solutions obtained by the heuristics.

In our previous work [19] it was observed that in several problem instances, namely for node pairs with no solution, CPLEX could take a very long time to conclude that the problem was infeasible. In the present work, and to confirm the need for a CPU limit also for problem $\mathcal{P}_2$, a CPU time limit of 8 hours was used in the pioro40 network, with $|V_{S_1}| = |V_{S_2}| = 2$; it was observed that a few node pairs took several minutes to obtain an optimal solution, that some other few node pairs took several hours to obtain an optimal solution, and that for at least five node pairs the ILP solver ended without any solution after using the allowed 8 hours of CPU time. Hence, it was decided, in order to obtain solutions in a reasonable time, to use (as in [19]) the limit of 5 minutes per node pair for the CPLEX solver.

The computational platform, for collecting the data presented in Figs. 1-18, was a Desktop with 16 GB of RAM and an Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz processor, with Kubuntu 12.4. The CPLEX solver, version 12.6.1 [31], was used for evaluating the performance of the heuristics.

6.1. Analysis of the performance of SK and VSN

The relative performance of algorithms SK and VSN, which are used as subroutines in the heuristics proposed for solving problems $\mathcal{P}_1$ and $\mathcal{P}_2$, is analyzed here.

In the figures and close to the xx axis is the number of elements in the set V_{S_1} for each network. The ratio of feasible solutions with respect to CPLEX, presented in Fig. 1, shows that SK has good performance for newyork network, but for norway, india35 and pioro40 it only presents around 70% of feasible solutions, and for germany50 its performance is even poorer. On the other hand VSN finds in average over 97.5% of all feasible solutions for all networks, except germany50, where in average it finds over 92.5% solutions. One can verify in Fig. 2 that the average error of the feasible solutions of VSN is below 3%, and that the error of the feasible solutions of SK grows with the size of the set of nodes to visit, and is around 9% for $|V_{S_1}| = 6$ on newyork network. Regarding CPU time, VSN uses slightly more CPU time than SK, and 10-100 times less than the ILP solver, as can be seen in Fig. 3. It should be noted that if a CPU limit of 5 minutes was not in place for the ILP solver the

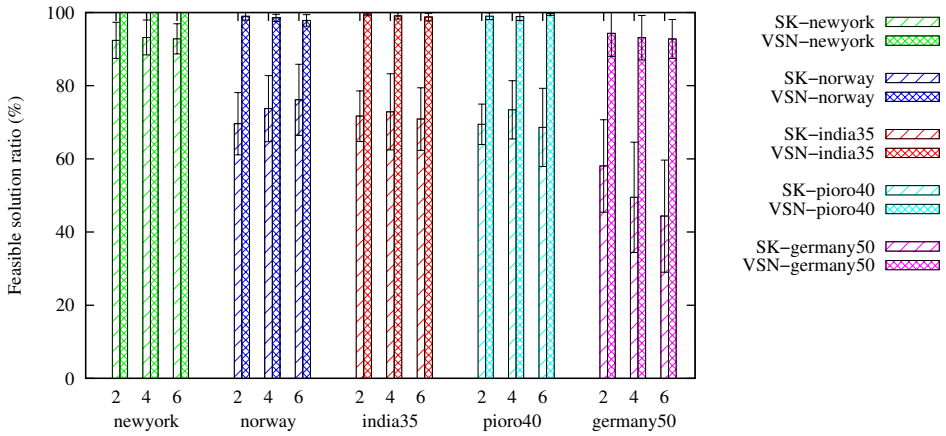

Figure 1: Feasible solutions ratio with respect to CPLEX, for five SNDlib [29] networks (shortest path visiting specified nodes)

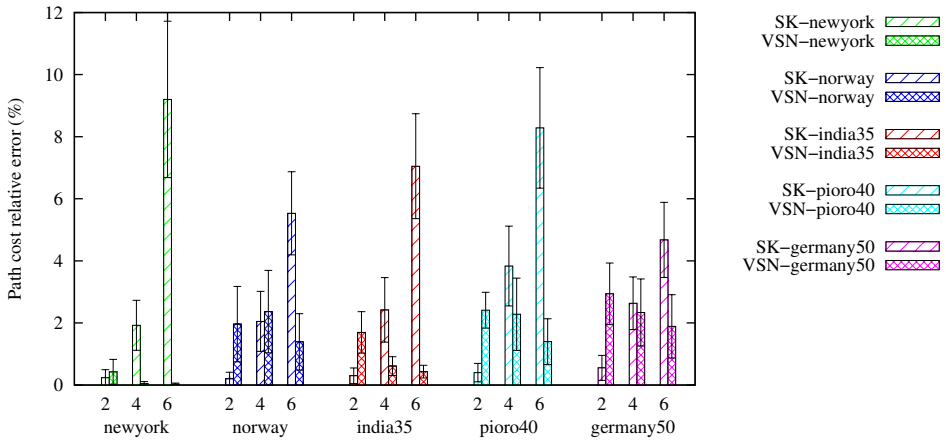

Figure 2: Relative error of the feasible solutions found by the heuristics for five SNDlib [29] networks (shortest path visiting specified nodes)

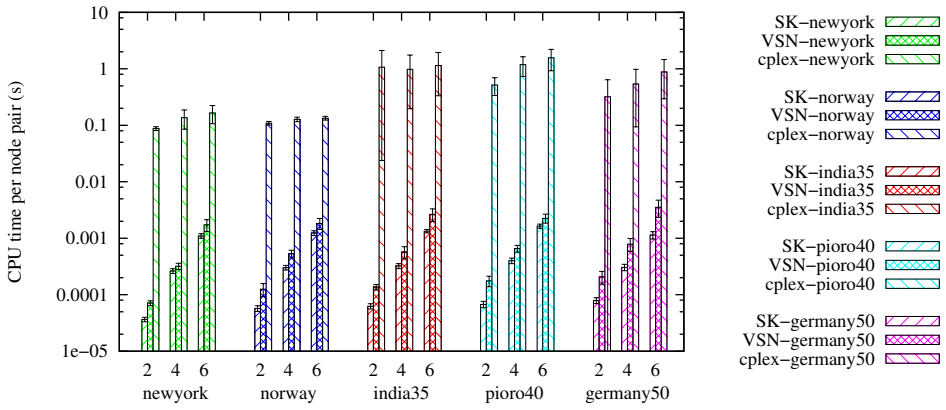

Figure 3: CPU time per node pair for five SNDlib [29] networks (shortest path visiting specified nodes)

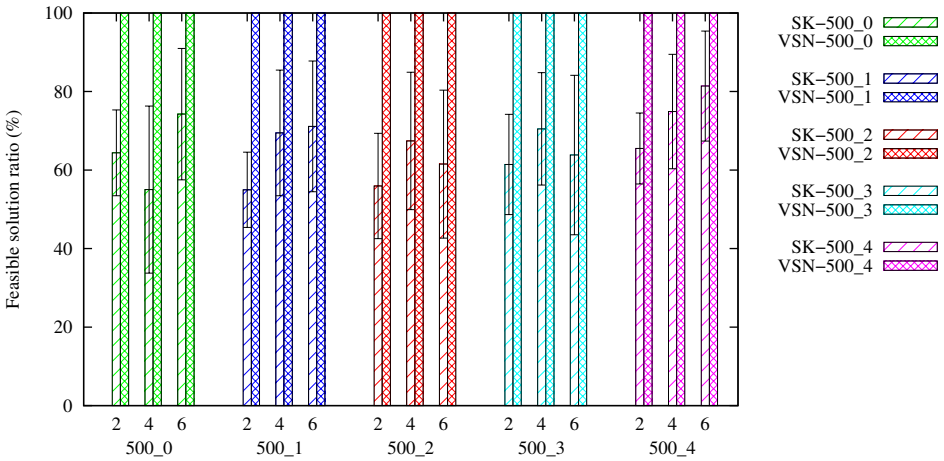

Figure 4: Feasible solutions ratio with respect to CPLEX, in the 500 nodes networks (shortest path visiting specified nodes)

CPU could be much larger. In fact for the india35 network and germany50 there were a total of 17 and 27 node pairs, respectively, not solved by CPLEX.

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.

For these networks the feasible solution ratio of SK is between 50% and 80% (depending on $|V_{S_1}|$ and of the network, and for VSN is very close to 100% (over 99.9%) for all the networks – see Fig. 4. Due to the CPU time limitation CPLEX found 46 sub-optimal solutions (in 30000 problems); in two cases the ILP solver was unable to give any result while both heuristics found a solution (the one found by VSN had the smaller cost). Regarding the cost of the paths, SK for $|V_{S_1}| = 6$ presents up to 10% relative error, but for $|V_{S_1}| = 2, 4$, the error is less than 2%, and 6%, respectively, as can be seen in Fig. 5. VSN presents an error always below 4%, and in general it has a smaller relative error than SK , with the exception of $|V_{S_1}| = 2$. Note that when the path obtained in the sub-graph is expanded and has a cycle, arcs are successively deleted until a solution can be found. This tends to change the sub-path close to the source, and keep in the solution the shortest paths between the two specified nodes, possibly resulting in paths longer than necessary. It should nevertheless be noted that, albeit this slightly larger error in the cost of the paths for $|V_{S_1}| = 2$, VSN does have a much larger resolution ratio than SK, and smaller error than SK for $|V_{S_1}| = 4, 6$.

Regarding the CPU time, which was around 1 second per node pair for CPLEX in the SNDlib networks, it is now larger than 2 seconds and for larger $|V_{S_1}| = 6$ around 10 seconds per node pair (see Fig. 6). The CPU time of the heuristics remains smaller by two orders of magnitude. It should be noted that SK requires less CPU time than VSN, and that in average VSN requires less than 100 milliseconds per node pair.

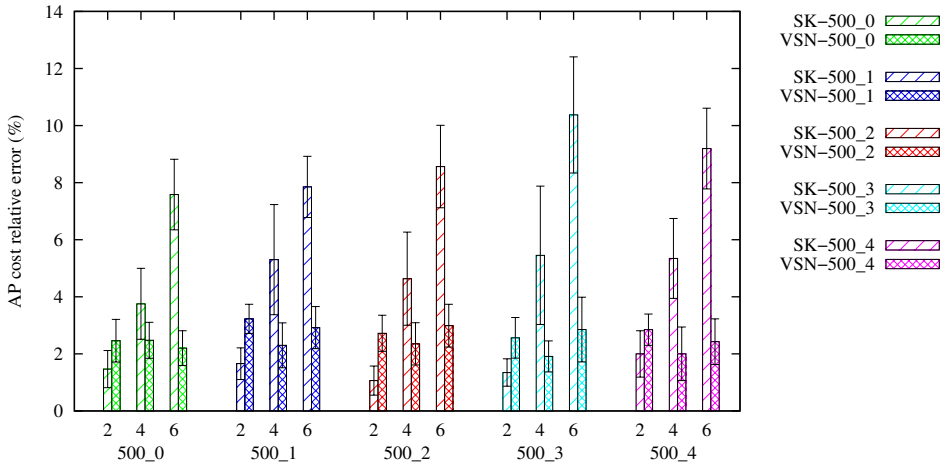

Figure 5: Relative error of the feasible solutions found by the heuristics, in the 500 nodes networks (shortest path visiting specified nodes)

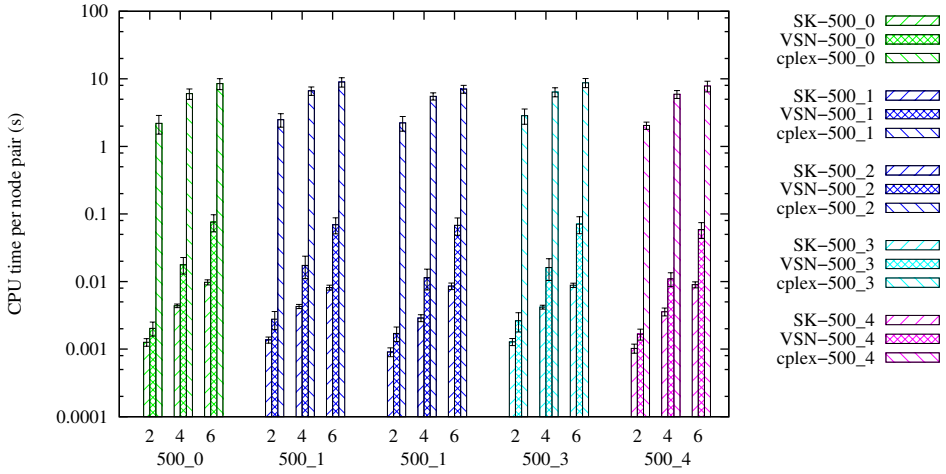

Figure 6: CPU time per node pair, in the 500 nodes networks (shortest path visiting specified nodes)

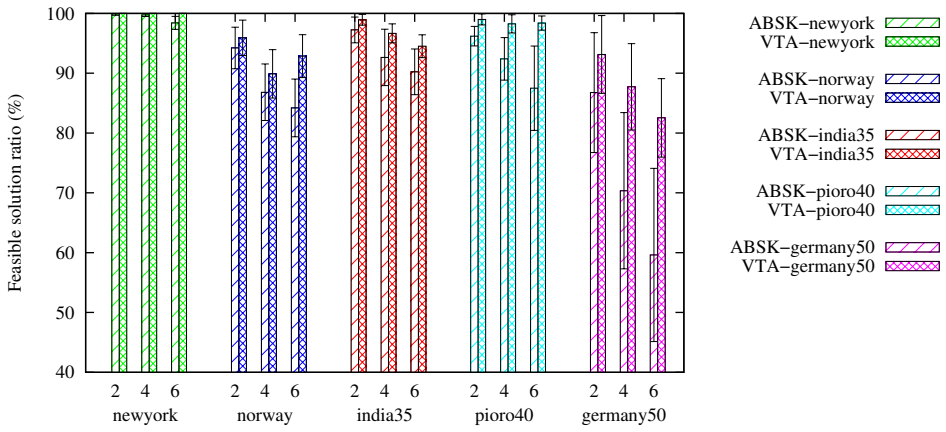

Figure 7: Feasible solutions ratio with respect to CPLEX, for five SNDlib [29] networks (problem $\mathcal{P}_1$)

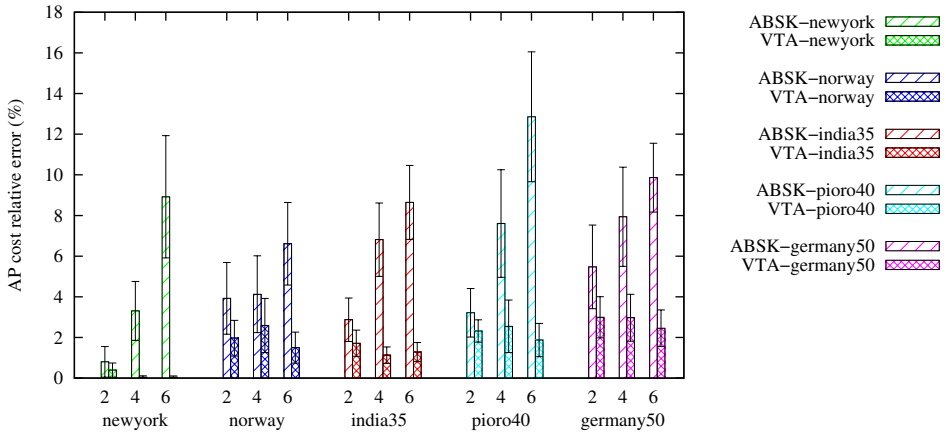

Figure 8: Relative error of the feasible solutions found by the heuristics for five SNDlib [29] networks (problem $\mathcal{P}_1$)

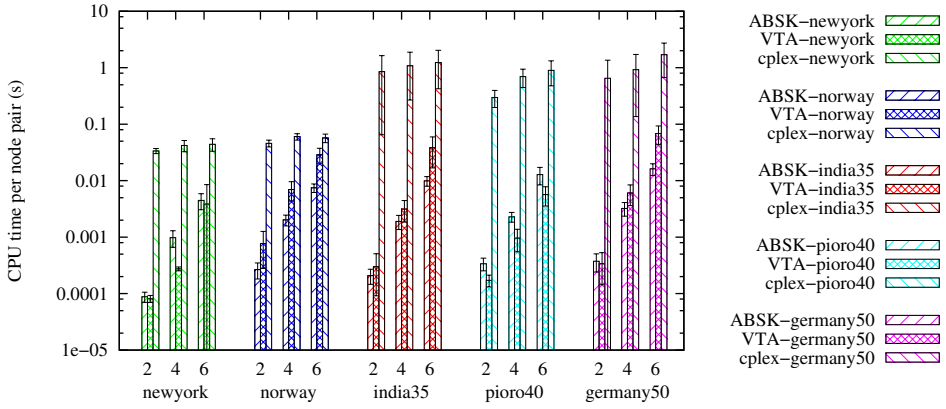

Figure 9: CPU time per node pair for five SNDlib [29] networks (problem $\mathcal{P}_1$)

6.2. Analysis of the performance of ABSK and VTA

Results obtained using networks from the SNDlib library are illustrated in Figures 7-9. 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 VTA finds the largest number of feasible solutions. Also the number of feasible solutions found by the heuristics tends to decrease with the increase of the number of specified nodes. In general, for ABSK results are quite good for three of the five networks, with values close to 99% and over 90% for newyork and india35 respectively (for $|V_{S_1}| = 2, 4, 6$), over 90% for pioro40 for $|V_{S_1}| = 2, 4$ and close to 90% for $|V_{S_1}| = 6$; the performance degrades for norway, and for the germany50 network the result regarding the number of feasible solution is rather low, namely for $|V_{S_1}| = 6$. The relative error of the cost of the active path also increases with the number of specified nodes to visit, from less than 1% in average (in the case of newyork network) to close to 13% in average (in the case of pioro40 network) for ABSK. Regarding the CPU time of the heuristics, the average was below 40 milliseconds, which is significantly faster than the average CPLEX result for the larger networks (india35, pioro40 and germany50). For the other networks the CPU of the heuristics is only significantly smaller than the CPU time of the ILP solver for the smaller values of $|V_{S_1}|$. CPLEX execution time varies significantly, from 20 milliseconds to 5 minutes (for problems without solution) resulting in an average CPU time below 2 seconds per node pair, and larger average CPU time would be observed if the CPU time limit of 5 minutes per node pair was not in place. Some optimal solutions can take close to 1.5 minutes to be obtained (as was observed in the case of germany50).

It is worth to point out that while VTA resolution ratio will tend to be smaller than the resolution ratio of VSN, the same is not observed in the case of ABSK and SK. In fact the resolution ratio of ABSK is much larger than the resolution ratio of SK. This result can be explained as follows. SK is quite greedy, seeking to obtain the shortest path possible, sometimes discarding sub-paths that would possibly allow it to find a solution. Algorithm ASK (first heuristic used by ABSK) guides SK calculation of the AP, due to the constraint that a node disjoint path must exist with each candidate sub-path of the AP, and this limits the greedy nature of SK, specially for $|V_{S_1}| = 2$. Also note that ASK is followed by BSK whenever ASK fails to find a solution thus further improving the resolution ratio of ABSK with respect to SK.

Results regarding the networks with 500 nodes are shown in Figures 10-12. Algorithm VTA presents a resolution ratio very close to 100% (over 99.9%). One can observe a high ratio of feasible solutions, always larger than 95% for ABSK, with two exceptions when $|V_{S_1}| = 4$ (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 the number of specified nodes to visit – see Fig. 10.

The relative error of the solutions obtained by VTA is relatively stable and in average less than 3% for VTA. However, for ABSK, 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 Fig. 11. It is interesting to observe that the CPU time per node pair required by CPLEX in these larger networks (in average between 2 and 5 seconds) is

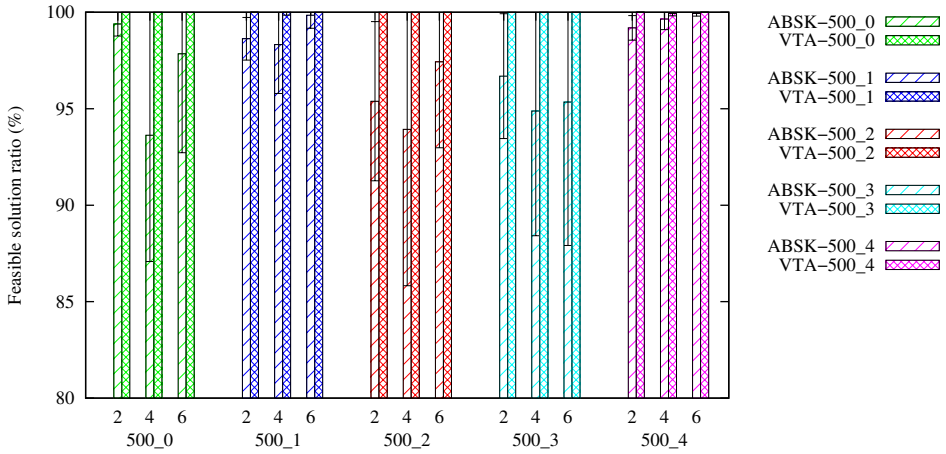

Figure 10: Feasible solutions ratio with respect to CPLEX, in the 500 nodes networks (problem $\mathcal{P}_1$)

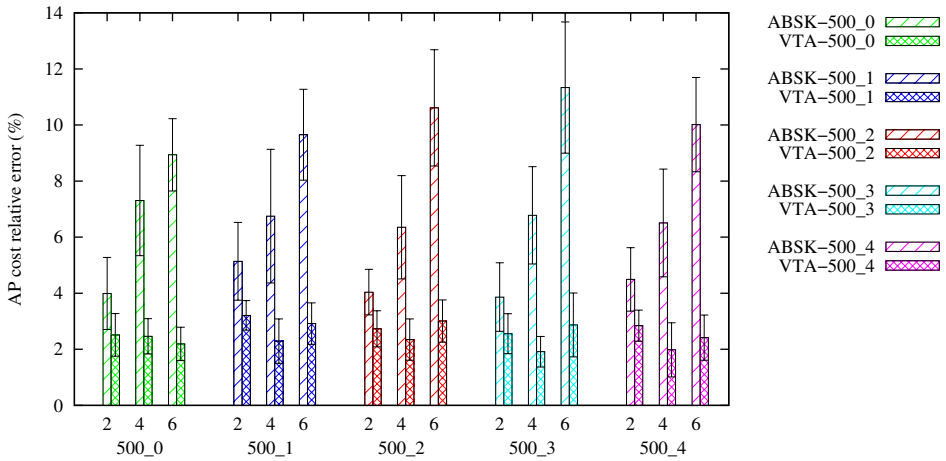

Figure 11: Relative error of the feasible solutions found by the heuristics, in the 500 nodes networks (problem $\mathcal{P}_1$)

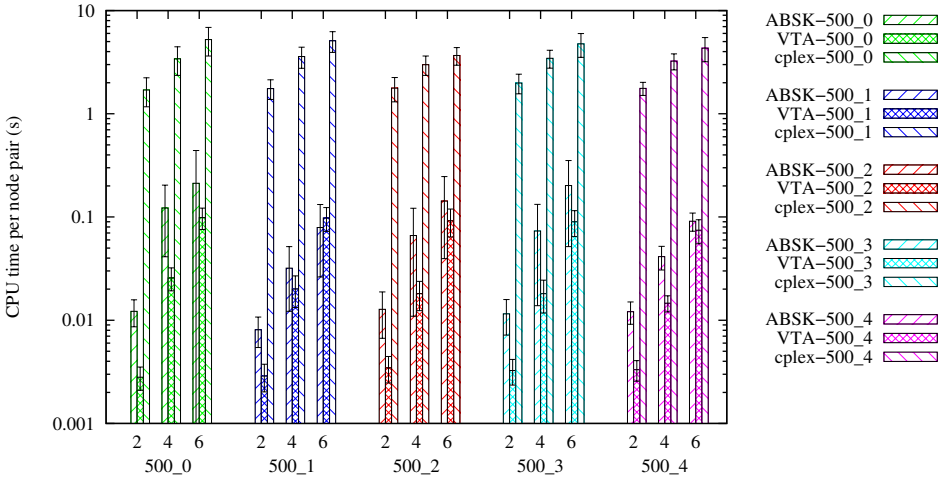

Figure 12: CPU time per node pair, in the 500 nodes networks (problem $\mathcal{P}_1$)

not significantly larger than the average CPU time observed for the SNDlib networks. The CPU time of ABSK and VTA, although larger than in the case of the SNDlib networks, is between 10 to 100 times smaller than the average CPU time required by CPLEX. For some problems CPLEX requires tens of seconds, and in some other cases (less than 1%) the allowed 5 minutes per node pair were exhausted and only a sub-optimal solution was found.

The heuristic VTA finds the largest number of feasible solutions, followed closely by ABSK; moreover the former uses less CPU time than the later in most cases (see Fig. 12). Regarding accuracy, VTA presents a smaller relative error. Hence, one may conclude that VTA outperforms ABSK.

Regarding the cost of the resulting BP obtained by the heuristics, with respect to the cost of the BP associated with the AP calculated by CPLEX, the error is less than 5% and 15% in the case of the SNDlib networks for VSN and ABSK, respectively; for the 500 node networks the error is smaller, less than 2% and 10% for VSN and ABSK, respectively.

6.3. Analysis of the performance of MSASK and MSVTA

As already mentioned, for problem $\mathcal{P}_2$, it was considered that $|V_{S_1}| = |V_{S_2}|$, and in this Sub-section V_{S_i} $(i = 1, 2)$ will be used omitting the value of i.

Obtaining a pair of node-disjoint loopless paths, each visiting a specified set of nodes, such that the sum of the cost of the paths is minimum, is a more challenging problem than the ones previously considered. The CPU time required by CPLEX for solving problem $\mathcal{P}_2$ is around 10 times larger than the one required for solving problem $\mathcal{P}_1$, for $|V_{S_i}| = 4$, as can be seen comparing Figs. 9 and 12 with Figs. 15 and 18, respectively.

In the case of the SNDlib networks, it can be observed in Fig. 13 that the resolution ratio of MSVTA (based on VSN and using the same strategy as VTA) is above 95% for newyork network above 80% for norway and india35, over 70% for pioro40, and close to 70% for germany50. The second proposed heuristic, MSASK (based on SK and using the same approach as ASK) presents a much lower resolution ratio: less than 50% for germany50 and

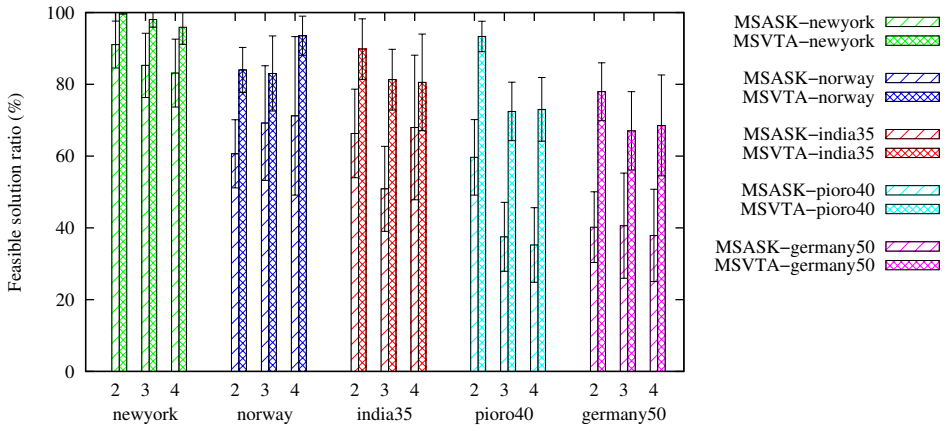

Figure 13: Feasible solutions ratio with respect to CPLEX, for five SNDlib [29] networks (problem $\mathcal{P}_2$)

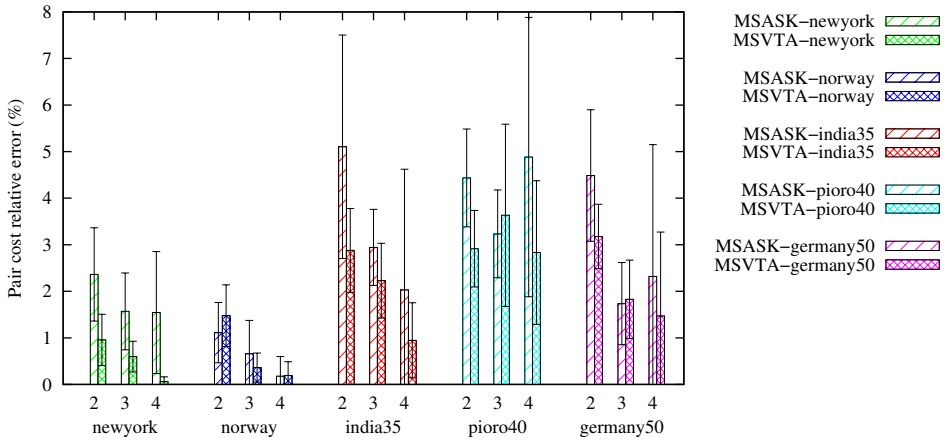

Figure 14: Relative error of the feasible solutions found by the heuristics for five SNDlib [29] networks (problem $\mathcal{P}_2$)

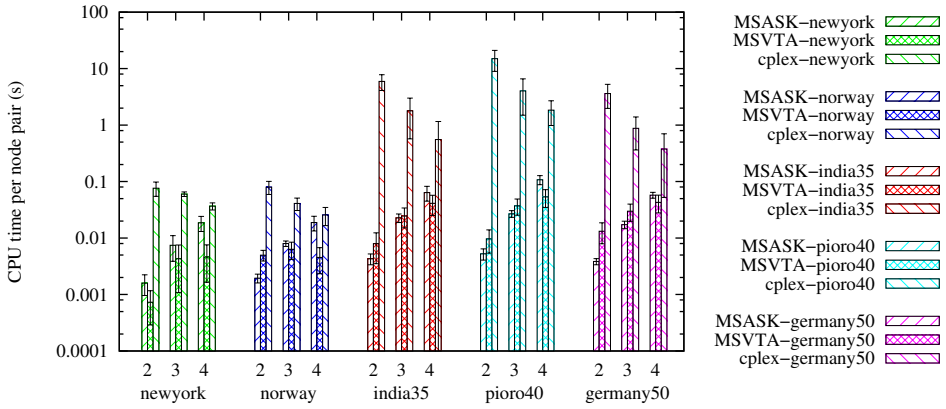

Figure 15: CPU time per node pair for five SNDlib [29] networks (problem $\mathcal{P}_2$)

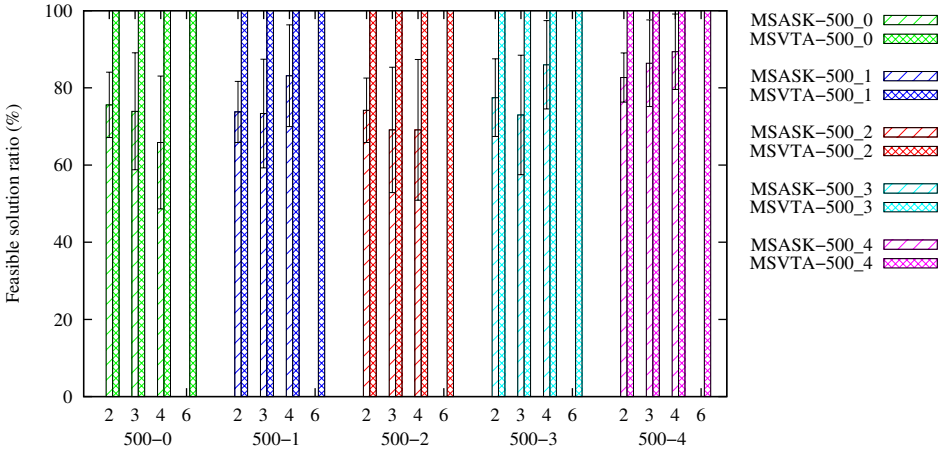

Figure 16: Feasible solutions ratio with respect to CPLEX, in the 500 nodes networks (problem $\mathcal{P}_2$)

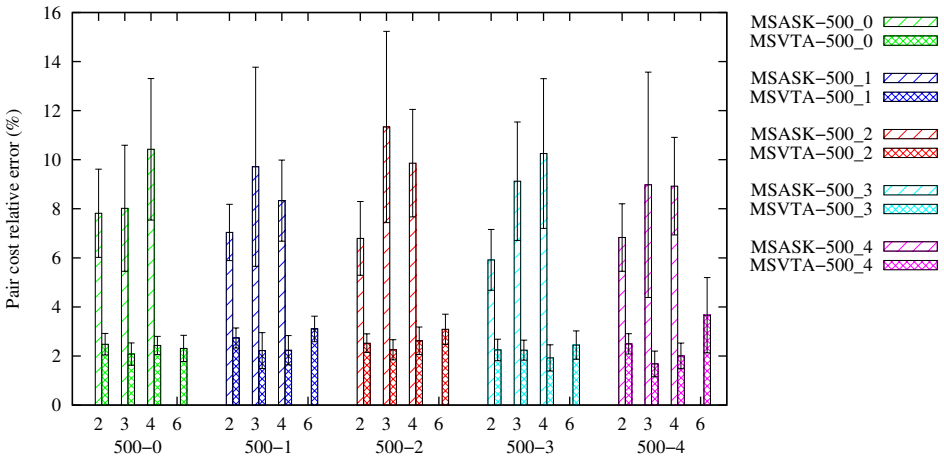

Figure 17: Relative error of the feasible solutions found by the heuristics, in the 500 nodes networks (problem $\mathcal{P}_2$)

also for pioro40 with $|V_{S_i}| = 3, 4$. The best result of MSASK is for newyork, with 90% and 80% of solutions for $|V_{S_i}| = 2$ and $|V_{S_i}| = 3, 4$, respectively. The error of the path pair cost is in general less than 5% for MSASK, but it presents large confidence intervals. For MSVTA the error is less than 4% and presents rather small confidence intervals – see Fig. 14 For these networks the CPU of both heuristics is rather similar, as can be seen in Fig. 15. However the CPU time of CPLEX for the smaller networks and $|V_{S_i}| = 3, 4$ is not significantly larger than the used by the heuristics. Nevertheless for india35, pioro40 and germany50, CPLEX requires one, two or more orders of magnitude with respect to both heuristics. Note that there are 13, 29 and 5 node pairs for which a solution was not found by the ILP solver, in india35, pioro40 and germany50, respectively, and most of those cases occur when $|V_{S_i}| = 2$, due to the CPU time limitation. Moreover for pioro40 sub-optimal solutions were obtained for 31 node pairs, due to the same CPU time limitation.

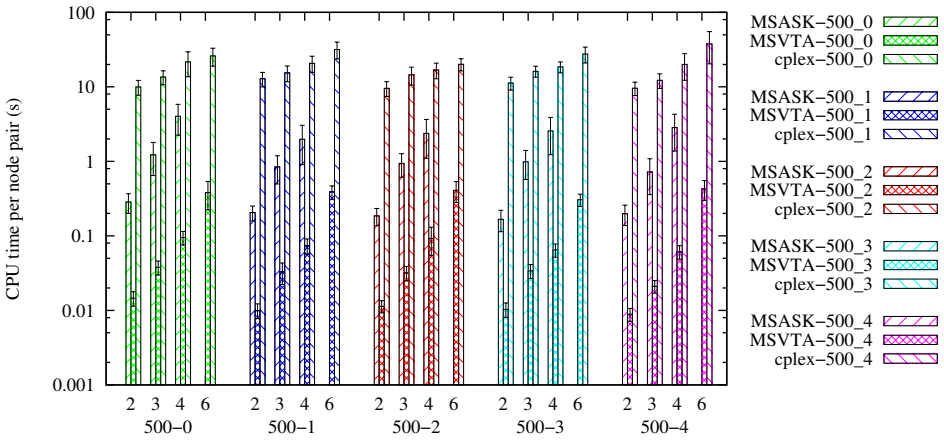

Figure 18: CPU time per node pair, in the 500 nodes networks (problem $\mathcal{P}_2$)

In contrast with what was observed in the SNDlib networks, the resolution ratio of MSVTA was 100% for the 500 nodes networks (see Fig. 16) – in fact it was slightly over 100% because the reference is the number of solutions found by CPLEX. Algorithm MSASK presents a resolution ratio between 65% and 90% (which is also a significant improvement when compared with the results obtained in the SNDlib networks). The relative error of the path pair cost was less than 3% for MSVTA, but MSASK presents a relative error between 6% and 12% and with some variability, as indicated by the relatively large confidence intervals around the average values, as can be seen in Fig.17.

CPLEX requires in average between 10 and 20 seconds of CPU per node pair, while MSTVA requires 0.02 to 0.2 seconds in average – see Fig. 18, for $|V_{S_1}| = |V_{S_2}| = 2, 3, 4$. In these networks (and for $|V_{S_1}| = |V_{S_2}| = 2, 3, 4$) MSASK requires significantly more CPU time than MSVTA, but still much less than CPLEX. Note that if the CPU time limitation was not active, the CPU time of the ILP solver would be much higher because, in this problem and for these networks, a total of 322 pairs had sub-optimal solutions. Additionally, in a total of 35 node pairs CPLEX was unable to give any answer within the time limit of 5 minutes, and the heuristics managed to find an admissible solution to most of those problems.

It can be seen in Figures 16-18, that for $|V_{S_1}| = |V_{S_2}| = 6$, MSVTA is still an effective heuristic, solving almost all problems, with an average relative error below 4% and using a CPU time which is almost two orders of magnitude smaller than the CPU time required by CPLEX. For $|V_{S_1}| = |V_{S_2}| = 6$ it was verified that MSASK used too much CPU time (similar to CPLEX) and therefore those results were not included in the figures, because the heuristic becomes useless. For this number of specified nodes to visit, CPLEX obtained a total of 200 sub-optimal solutions.

It can be concluded that MSVTA presents overall a better performance than MSASK. MSVTA can successfully be used in large networks, as long as $|V_{S_1}|$ is small, it requires significantly less CPU time than CPLEX, and obtains solutions with small relative error.

7. 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 [14] for solving this problem was adapted to include the restriction that the obtained solution must have a node-disjoint backup path (problem $\mathcal{P}_1$), and then extended to solve the min-sum problem with constraints (problem $\mathcal{P}_2$). 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) [32] (using explicit routing) and lightpaths in WDM optical networks.

SK, a modified and improved version of the algorithm proposed by Saksena and Kumar in [12], for calculating a shortest loopless path visiting specified nodes was developed. Three heuristics, based on SK, are put forward for obtaining feasible solutions for problem $\mathcal{P}_1$: ASK, BSK and ABSK [19]. Then heuristic MSASK, which uses SK and an approach based on ASK, was proposed for solving problem $\mathcal{P}_1$.

Additionally, an alternative approach was proposed for solving these problems. Firstly, heuristic VSN was proposed which uses an auxiliary graph to solve the shortest loopless path visiting specified nodes; secondly, heuristics VTA and MSVTA were proposed for problems $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. Both VTA and MSVTA use VSN together with an approach inspired by the Trap Avoidance algorithm [28].

The performance of the proposed heuristics was evaluated using CPLEX to solve the ILP formulations (given in [14] and extended by equations (4)-(11) and (17)-(24)) of the corresponding problems. 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 40 node network, namely in pioro40 [29]. 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 based on VSN present the best relative performance. They were capable of finding a solution in a significant number of cases, and such that the calculated solutions presented an acceptable relative error regarding the relevant cost in each problem. The CPU time used by the heuristics is significantly smaller than the required by the used ILP solver, namely for networks with 500 nodes.

The heuristics VTA and MSVTA were capable of finding feasible solutions in a very short time, and in the case of Problem $\mathcal{P}_2$ they were even capable of finding feasible solutions for several node pairs where after 5 minutes the ILP solver was not capable of finding a solution (optimal or sub-optimal).

The applicability of the proposed heuristics is limited by the number of specified nodes to visit. Other approaches may be considered in future work, namely regarding the methods for calculating Hamiltonian paths. When few nodes must be visited, alternative exact method can be used [24, 25]. If there are many of those nodes, then an approximate method will be more convenient [26, 27]. It should be noticed, however, that a loopless path is sought and the solution output by these methods, in the present context of application, does not necessarily meet this constraint. Therefore, their utilization cannot be straightforward.

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, P. Smith, Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines, Computer Networks 54 (8) (2010) 1245 – 1265, resilient and Survivable networks. doi:10.1016/j.comnet.2010.03.005.

[2] G. Ellinas, D. Papadimitriou, J. Rak, D. Staessens, J. P. Sterbenz, K. Walkowiak, Practical issues for the implementation of survivability and recovery techniques in optical networks, Optical Switching and Networking 14, Part 2 (0) (2014) 179 – 193, special Issue on RNDM 2013. doi:10.1016/j.osn.2014.05.002.

[3] P. Cholda, A. Mykkeltveit, B. Helvik, O. Wittner, A. Jajszczyk, A survey of resilience differentiation frameworks in communication networks, Communications Surveys Tutorials, IEEE 9 (4) (2007) 32–55. doi:10.1109/COMST.2007.4444749.

[4] P. Cholda, J. Tapolcai, T. Cinkler, K. Wajda, A. Jajszczyk, Quality of resilience as a network reliability characterization tool, Network, IEEE 23 (2) (2009) 11–19. doi:10.1109/MNET.2009.4804331.

[5] R. Stankiewicz, P. Cholda, A. Jajszczyk, Qox: What is it really?, Communications Magazine, IEEE 49 (4) (2011) 148–158. doi:10.1109/MCOM.2011.5741159.

[6] J. W. Suurballe, Disjoint paths in networks, Networks 4 (1974) 125–145.

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

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

[9] M. Żotkiewicz, W. Ben-Ameur, M. Pióro, Finding failure-disjoint paths for path diversity protection in communication networks, IEEE Communications Letters 14 (8) (2010) 776 –778. doi:10.1109/LCOMM.2010.08.100653.

[10] T. Gomes, M. Żotkiewicz, Resilient arcs and node disjointness in diverse routing, Optical Switching and Networking 14, Part 2 (0) (2014) 149 – 163, special Issue on RNDM 2013. doi:10.1016/j.osn.2014.05.022.

[11] K. Kalaba, On some communication network problems, Tech. rep., Engineering Division, The RAND Corporation (June 1959).

[12] J. P. Saksena, S. Kumar, The routing problem with 'k' specified nodes, Operations Research 14 (5) (1966) 909–913. doi:10.1287/opre.14.5.909.

[13] S. E. Dreyfus, An appraisal of some shortest-path algorithms, Operations Research 17 (3) (1969) 395–412. doi:10.1287/opre.17.3.395.

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

[15] C. P. Bajaj, Some constrained shortest-route problems, Mathematical Methods of Operations Research 15 (1) (1971) 287–301.

[16] T. Ibaraki, Algorithms for obtaining shortest paths visiting specified nodes, SIAM Review 15 (2) (1973) 309–317.

[17] G. Laporte, H. Mercure, , Y. Nobert, Optimal tour planning with specified nodes, RAIRO, Recherche Opérationnelle 3 (18) (1984) 203–210.

[18] T. Volgenant, R. Jonker, On some generalizations of the traveling-salesman problem, Journal of Operational Research Society 11 (38) (1987) 1073–1079.

[19] T. Gomes, S. Marques, L. Martins, M. Pascoal, D. Tipper, Protected shortest path visiting specified nodes, in: 7th International Workshop on Reliable Networks Design and Modeling (RNDM 2015), 2015, pp. 120–127. doi:10.1109/RNDM.2015.7325218.

[20] J. Y. Yen, Finding the k shortest loopless paths in a network, Management Science 17 (11) (1971) 712–716.

[21] E. Martins, M. Pascoal, A new implementation of Yen's ranking loopless paths algorithm, 4OR – Quarterly Journal of the Belgian, French and Italian Operations Research Societies 1 (2) (2003) 121–134.

[22] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, Network Flows : Theory, Algorithms and Applications, Prentice Hall, Englewood Cliffs, NJ, 1993.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, The MIT Press, Cambridge, MA, 2001.

[24] F. Rubin, A search procedure for hamilton paths and circuits, Journal of the ACM 21 (4) (1974) 576–580.

[25] R. Bellman, Dynamic programming treatment of the travelling salesman problem, Journal of the ACM 9 (1) (1962) 61–63.

[26] S. Lin, B. W. Kernighan, An effective heuristic algorithm for the traveling-salesman problem, Operations research 21 (2) (1973) 498–516.

[27] G. Gutin, A. P. Punnen (Eds.), The traveling salesman problem and its variations, Combinatorial optimization, Kluwer Academic, Dordrecht, London, 2002.

[28] D. Xu, Y. Xiong, C. Qiao, G. Li, Trap avoidance and protection schemes in networks with shared risk link groups, Journal of Lightwave Technology 21 (11) (2003) 2683–2693.

[29] S. Orlowski, R. Wessäly, M. Pióro, A. Tomaszewski, SNDlib 1.0–Survivable Network Design library, Networks 55 (3) (2010) 276–286, http://sndlib.zib.de. doi:10.1002/net.20371.

[30] M. Doar, 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, 1993, pp. 82–89 vol.1. doi:10.1109/INFCOM.1993.253246.

[31] IBM ILOG CPLEX Optimization Studio V12.6.1, IBM, 2014.

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