The Constrained Shortest Path Tour Problem

Daniele Ferone * Paola Festa * Francesca Guerriero †‡ Demetrio Laganà†

April 4, 2016

Abstract

In this paper, we study the constrained shortest path tour problem. Given a directed graph with non-negative arc lengths, the aim is to find a single-origin single-destination shortest path, which needs to cross a sequence of node subsets that are given in a fixed order. The subsets are disjoint and may be of different size. In addition, it is required that the path does not include repeated arcs.

Theoretical properties of the problem are studied, proving that it belongs to the complexity class **NP**-complete. To exactly solve it, a Branch & Bound method is proposed. Given the problem hardness, a Greedy Randomized Adaptive Search Procedure is also developed to find near-optimal solutions for medium to large scale instances.

Extensive computational experiments, on a significant set of test problems, are carried out in order to empirically evaluate the performance of the proposed approaches. The computational results show that the Greedy Randomized Adaptive Search Procedure is effective in finding optimal or near optimal solutions in very limited computational time.

Keywords*: Shortest path problems, Network flow problems, Combinatorial optimization, Branch & Bound, GRASP.*

1 Introduction

The constrained shortest path tour problem ($\mathcal{CSPTP}$) consists in finding a single-origin single-destination shortest path in a directed graph such that a given set of constraints is satisfied. In particular, in addition to the restrictions imposed in the original version of the problem (see [4], [10], and [11]), requiring that the path needs to cross a sequence of node subsets that are given in a fixed order, in the $\mathcal{CSPTP}$ it is imposed that the path does not include repeated arcs.

The $\mathcal{CSPTP}$ can be viewed as a special case of the network interdiction problem on a flow network ([23]), in which an attacker disables all the arcs of a network, whenever they

*Department of Mathematics and Applications, University of Napoli FEDERICO II, Italy. E-mail: daniele.ferone@unina.it, paola.festa@unina.it

†Department of Mechanical, Energetic and Management Engineering, University of Calabria, Italy. E-mail: francesca.guerriero@unical.it, demetrio.lagana@unical.it

‡Corresponding author. Phone: 39-0984-494620, francesca.guerriero@unical.it

are used to ship flow, with the aim of minimizing the net profit that can be obtained from shipping a commodity across the network. Observe that a profit can be associated with nodes or arcs of the directed network. The net profit is given by the difference between the total profit minus the total flow cost traveling over the directed network. Whenever the profit of an arc is collected, it is forbidden to traverse it again since it is disrupted after traveling on it for the first time. In this context, a special case of the network interdiction modeled by the CSPTP occurs, when a profit is associated with all the subsets of a given sequence that must be visited in a fixed order. Since the profit of each subset is collected the first time that it is reached, the problem reduces to a single-origin single-destination shortest path crossing each subset at least once in the fixed order, in which no repeated arcs are included in order to reduce the impact of disabling the arcs that have been already used to ship flow.

Another practical application that can be addressed as a $\mathcal{CSPTP}$ arises in the network design field ([20]). Once a network is built it is expected to perform reliably; that is, the operation should be continuous, effective, and above all should provide excellent value. However, the effective continuous operation of large scale networks depends on the initial design strategy. The performance may drop to an unacceptable level due to the failure of links, that are vital to the network. More specifically, to reduce the possibility of virus infection, the architecture of the networks can be changed by deleting some nodes and arcs, or reducing arcs' capacity. This implies addressing the selection of the one way traveling arcs, whose usage does not compromise the survivability of the flow that must be sent from a given origin to a given destination. In this context, networks of workstations are vulnerable to attack and failure, and it is generally recognized that the survivability of our computing systems is a critical issue. We are interested in a particular type of survivability: Which is the best way to connect two endpoints in a communication network by crossing not more than once some sets of vital nodes in a given order? The $\mathcal{CSPTP}$ represents an effective way to model such real problems.

To the best of our knowledge, the $\mathcal{CSPTP}$ has not been previously addressed in the scientific literature. In this paper, we firstly theoretically investigate its computational complexity. In particular, we show that the Hamiltonian Path Problem ([1]) ($\mathcal{HAM}-\mathcal{PATH}$) can be polynomially reduced to the $\mathcal{CSPTP}$ and that the $\mathcal{CSPTP}$ belongs to the **NP**-complete class.

In order to solve the problem, we have designed and implemented both an exact and a metaheuristic approach. To find optimal solutions, a Branch & Bound (B&B) method is proposed, which takes advantage of the specific structure of the problem. Nevertheless, given the problem hardness, the exact approach can be used to solve in a reasonable amount of time only small-size instances. Thus, to address larger problem instances, a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic is proposed.

An extensive computational phase is carried out to assess the performance of the proposed approaches in terms of both computational effort and solution quality.

The main contributions of this work are threefold:

- introducing and modeling the $\mathcal{CSPTP}$;
- analyzing the theoretical properties of the $\mathcal{CSPTP}$;

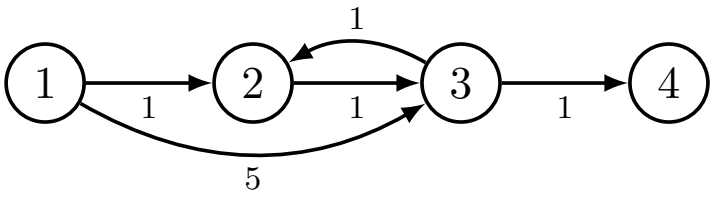

Figure 1: A $\mathcal{CSPTP}$ instance on a small graph G.

- designing and testing exact and heuristic solution approaches.

The remainder of this paper is organized as follows. In Section 2, we provide a mathematical formulation of the $\mathcal{CSPTP}$ and a deep analysis of its complexity. In Section 3, we present the solution approaches designed for solving the problem under investigation. Computational results and the analysis of the performance of the proposed algorithms are presented in Section 4. Concluding remarks are given in Section 5. The paper ends with Appendix A that contains the formal proofs of the problem hardness.

2 Problem description

Let $G = (V, A, C)$ be a directed graph, where

- $V = \{1, \ldots, n\}$ is a set of nodes;
- $A = \{(i, j) \in V \times V \mid i, j \in V \wedge i \neq j\}$ is a set of m arcs;
- $C : A \to \mathbb{R}^+ \cup \{0\}$ is a function that assigns a nonnegative length c_{ij} to each arc $(i, j) \in A$;
- for each node $i \in V$, let $FS(i) = \{j \in V \mid (i, j) \in A\}$ and $BS(i) = \{j \in V \mid (j, i) \in A\}$ be the *forward star* and *backward star* of node i, respectively;
- a path $P = \{i_1, \ldots, i_k\}$ is a sequence of k nodes $i_j \in V$, such that $(i_j, i_{j+1}) \in A$, for all $j = 1, \ldots, k-1$;
- in this context, a path P is said to be simple whenever no repeated arcs occur in it;
- the length $L(P)$ of any path P is defined as the sum of the lengths of the arcs on the path, i.e. $L(P) = \sum_{j=1}^{k-1} c_{j,j+1}$.

The classical shortest path tour problem ($\mathcal{SPTP}$) consists in finding a shortest path from a source node s to a destination node d, $s, d \in V$, $s \neq d$, by ensuring that at least one node of each node subset $T_1, \ldots, T_N$, where $T_h \cap T_l = \emptyset$, $\forall\ h, l = 1, \ldots, N$, $h \neq l$, is crossed according to the sequence wherewith the subsets are ordered. Any intermediate nodes between visits to the subsets T_h, $h = 1, \ldots, N$ are allowed. Note that it not necessarily results that $\cup_{l=1}^{N} T_l = V$. In [10], it was proven that the $\mathcal{SPTP}$ belongs to the class **P**.

Let us consider the small graph $G = (V, A, C)$ depicted in Figure 1, where $V = \{s = 1, 2, 3, 4 = d\}$. It is easy to see that $P = \{1, 2, 3, 4\}$ is the shortest path from node 1 to node 4 and its length is equal to 3. Let us now define on G the $\mathcal{SPTP}$ instance by considering $N = 4$ node subsets $T_1 = \{s = 1\},\ T_2 = \{3\},\ T_3 = \{2\},\ T_4 = \{d = 4\}$. The shortest path tour from node 1 to node 4 is the path $P_T = \{1, 2, 3, 2, 3, 4\}$ of length 5; P_T is not simple, since it crosses twice arc $(2, 3)$.

In this paper, we introduce a variant of the $\mathcal{SPTP}$, in the following referred to as the $\mathcal{CSPTP}$, where an integer capacity $u_{ij} \geq 1$ is associated with each arc $(i, j) \in A$. It denotes the maximum number of times that arc (i, j) can be traversed in any $\mathcal{CSPTP}$ solution.

Given a $\mathcal{CSPTP}$ instance $\left\langle G = (V, A, C), s, d, N, \{T_h\}_{h=1,\ldots,N}, \{u_{ij}\}_{(i,j)\in A}\right\rangle$, any feasible solution P_T is a chain of paths such that subsets T_h, $h = 1, \ldots, N$, are visited in the order in which they are defined and the total number of times that each arc $(i, j) \in A$ is traversed is at most u_{ij}. Each path P_h in the chain P_T starts from a node in T_h and ends to a node in T_{h+1}, $h = 1, \ldots, N - 1$.

Let $\oplus$ denote the concatenation of two paths, P_T can be formally represented as follows:

$$P_T = P_1 \oplus \cdots \oplus P_{N-1};$$

$$P_h = \{i_h, \ldots, i_{h+1}\};\ L(P_h) < +\infty,\ \forall\ 1 \leq h < N;$$

$$i_h \in T_h,\ \forall\ 1 \leq h \leq N;$$

$$i_1 = s;\ i_N = d.$$

The objective function value corresponding to P_T is the sum of the length of the paths in the chain and can be evaluated as follows:

$$C(P_T) = \sum_{h=1}^{N-1} L(P_h).$$

Clearly, if $u_{ij} = +\infty$, for all $(i, j) \in A$, the $\mathcal{CSPTP}$ reduces to the $\mathcal{SPTP}$. Conversely, even if $u_{ij} = 1$ for all $(i, j) \in A$, the resulting $\mathcal{CSPTP}$ is **NP**-complete, as proved in the following theoretical results, by demonstrating first that the $\mathcal{CSPTP}$ belongs to **NP**, and then that there exists at least one **NP**-complete problem that is polynomially Karp-reducible to $\mathcal{CSPTP}$*. The formal proof of these theoretical results are reported in the Appendix A.

Theorem 1 *$\mathcal{CSPTP}$ belongs to the class **NP**.*

Proof: A formal proof is given in the Appendix A.

*A problem R is polynomially Karp-reducible to a problem $\bar{R}$ ($R <_m^p \bar{R}$) if there exists a polynomial-time computable function f such that

$$x \text{ is a positive instance of } R \Longleftrightarrow f(x) \text{ is a positive instance of } \bar{R}.$$

f is called *Karp-reduction function* and a polynomial-time algorithm $\mathcal{A}$ that computes f is called a *Karp-reduction algorithm*.

To prove the hardness of the $\mathcal{CSPTP}$, we will prove that the *Hamiltonian Path problem* ($\mathcal{HAM}-\mathcal{PATH}$) is polynomially Karp-reducible to the $\mathcal{CSPTP}$.

Let $G=(V,A,C)$ be a directed graph, where $V=\{1,\ldots,n\}$, $A=\{(i,j)\in V\times V \mid i,j\in V \wedge i\neq j\}$, and $C: A\rightarrow \mathbb{R}^+\cup\{0\}$ is a function that assigns a nonnegative length c_{ij} to each arc $(i,j)\in A$. Let s and d, $s,d\in V$, $s\neq d$, be the origin and the destination node, respectively. Then, the $\mathcal{HAM}-\mathcal{PATH}$ consists in finding in G a minimum cost Hamiltonian path from s to d. In [21], it was proven that the $\mathcal{HAM}-\mathcal{PATH}$ is **NP**-complete.

To prove that $\mathcal{HAM}-\mathcal{PATH}$ is polynomially Karp-reducible to the $\mathcal{CSPTP}$ ($\mathcal{HAM}-\mathcal{PATH}<^p_m\mathcal{CSPTP}$), we need to define a polynomially computable function $f(\mathcal{I}_{\mathcal{HAM}-\mathcal{PATH}})$ that transforms any instance $\mathcal{I}_{\mathcal{HAM}-\mathcal{PATH}}$ of the $\mathcal{HAM}-\mathcal{PATH}$ in an instance $\mathcal{I}_{\mathcal{CSPTP}}$ of the $\mathcal{CSPTP}$. In other words, we need to design a polynomial-time Karp-reduction algorithm that takes as input an instance $\mathcal{I}_{\mathcal{HAM}-\mathcal{PATH}}$ of the $\mathcal{HAM}-\mathcal{PATH}$ and builds a directed graph containing a feasible path tour P_T if and only if there exists a feasible Hamiltonian path in $\mathcal{I}_{\mathcal{HAM}-\mathcal{PATH}}$.

Given any $\mathcal{HAM}-\mathcal{PATH}$ instance $\mathcal{I}_{\mathcal{HAM}-\mathcal{PATH}}$

$$\langle G=(V,A,C),s,d\rangle\,,$$

Figure 2 is the pseudo-code of the reduction algorithm that outputs a $\mathcal{CSPTP}$ instance

$$\left\langle G'=(V',A',C'),s^-,d^+,\{T_h\}_{h=1,\ldots,n+1}\right\rangle$$

by performing the following operations:

- for each node $i\in V$,
  - insert in V' nodes i^- and i^+;
  - insert in A' arc (i^-,i^+) with cost 0;
- for each arc $(i,j)\in A$ and for each $k=2,\ldots,n$,
  - insert in V' node ij^k;
  - insert in T_k node ij^k;
  - insert in A' arc (i^+,ij^k) with cost c_{ij} and arc (ij^k,j^-) with cost 0;
- set $T_1=\{s^-\}$ and $T_{n+1}=\{d^+\}$.

Summarizing, the set of nodes V' can be defined as follows:

$$V'=\bigcup_{i\in V}\left\{\, i^-,i^+\,\right\}\cup\bigcup_{(i,j)\in A}^{k=2,\ldots,n}\left\{\, ij^k\,\right\},$$

while the set A' can be defined as:

$$A'=\bigcup_{i\in V}\left\{\,(i^-,i^+)\,\right\}\cup\bigcup_{(i,j)\in A}^{k=2,\ldots,n}\left\{\,(i^+,ij^k),(ij^k,j^-)\,\right\}.$$

```
1: function HAM − PATH-TO-CSPTP(V, A, C, s, d)
2:     V' ← A' ← ∅;
3:     n ← |V|;
4:     for i ← 2 to n do
5:         T_i ← ∅;
6:     end for
7:     for all i ∈ V do
8:         V' ← V' ∪ {i⁻, i⁺};
9:         A' ← A' ∪ {(i⁻, i⁺)};
10:        c'(i⁻, i⁺) ← 0;
11:    end for
12:    for all (i, j) ∈ A do
13:        for k ← 2 to n do
14:            V' ← V' ∪ {ij^k};
15:            A' ← A' ∪ {(i⁺, ij^k), (ij^k, j⁻)};
16:            c'(i⁺, ij^k) ← c(i, j);
17:            c'(ij^k, j⁻) ← 0;
18:            T_k ← T_k ∪ {ij^k};
19:        end for
20:    end for
21:    T_1 ← {s⁻}; T_{n+1} ← {d⁺};
22:    return (G = (V', A', C'), s⁻, d⁺, {T_h}_{h=1,...,n+1});
23: end function
```

Figure 2: Polynomial reduction algorithm from the $\mathcal{HAM} - \mathcal{PATH}$ to the $\mathcal{CSPTP}$.

A small instance $\mathcal{I}_{\mathcal{HAM}-\mathcal{PATH}}$ is depicted in Figure 3. Figure 4 shows the corresponding instance $\mathcal{I}_{\mathcal{CSPTP}}$ as returned by algorithm in Figure 2, whose computational complexity is $O(n \cdot m)$.

Lemma 2 *There exists a feasible path* $P = i_1, i_2, \ldots, i_k$, $k \leq n$, *in*

$$\langle G = (V, A, C), s, d \rangle ,$$

if and only if in

$$\left\langle G' = (V', A', C'), s^-, d^+, \{T_h\}_{h=1,\ldots,n+1} \right\rangle$$

there exists a feasible path tour P' *from* i_l^- *to* i_k^+, *such that*

$$P' = \left\{ \bigoplus_{l=1}^{k-1} \left(i_l^-, i_l^+, i_l {i_{l+1}}^{l+1} \right), i_k^-, i_k^+ \right\}.$$

Proof: A formal proof is given in the Appendix A.

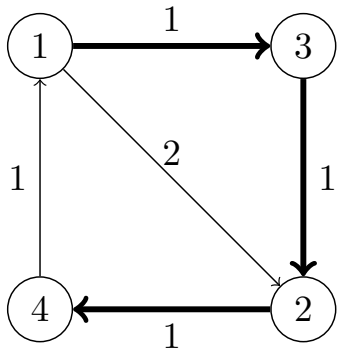

Figure 3: A small $\mathcal{HAM}-\mathcal{PATH}$ instance: bold line style indicates a Hamiltonian Path from node 1 to node 4.

Theorem 3 *Function $f(\mathcal{I}_{\mathcal{HAM}-\mathcal{PATH}})$ computed by algorithm in Figure 2 is a polynomially computable function that transforms any instance $\mathcal{I}_{\mathcal{HAM}-\mathcal{PATH}}$ of $\mathcal{HAM}-\mathcal{PATH}$ in an instance $\mathcal{I}_{\mathcal{CSPTP}}$ of the $\mathcal{CSPTP}$.*

Proof: A formal proof is given in the Appendix A.

Corollary 4 *The $\mathcal{CSPTP}$ is* **NP***-complete.*

Proof: A formal proof is given in the Appendix A.

3 Solution approaches

3.1 B&B Algorithm

In this section, we present an exact solution approach based on the B&B technique. To design the proposed B&B algorithm, the $\mathcal{CSPTP}$ has been reduced to the *Path Avoiding Forbidden Pairs Problem ($\mathcal{PAFPP}$)* (see Appendix A for the formal proof), which was proved to be **NP**-complete in [18].

Given a digraph $G = (V, A)$ and a set of pairs of nodes

$$F = \{\, (a_1, b_1), \ldots, (a_k, b_k) \,\},$$

where for all $i = 1, \ldots, k$, $a_i \neq b_i$ and $(a_i, b_i) \in (V \times V)$, the $\mathcal{PAFPP}$ consists in finding the shortest path P from a given node s to a given node d, with the constraint that P must contain at most one node of each pair in F.

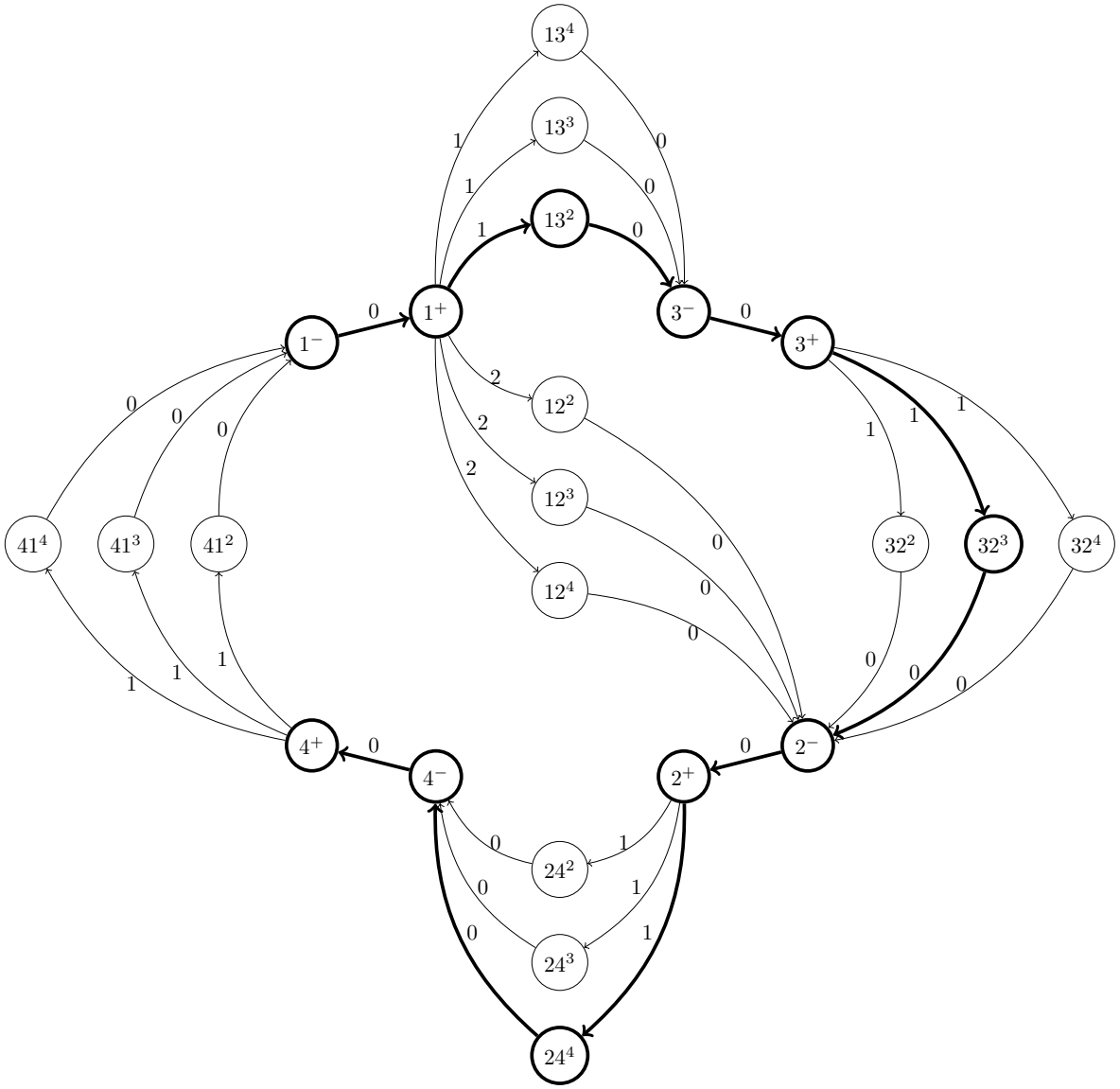

Figure 4: The $\mathcal{CSPTP}$ instance corresponding to the small $\mathcal{HAM}-\mathcal{PATH}$ instance depicted in Figure 3.

The $\mathcal{PAFPP}$ admits the following 0-1 integer programming formulation:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

s.t.

$$\sum_{j\in FS(i)} x_{ij} - \sum_{j\in BS(i)} x_{ji} = \begin{cases} 1, & i = s; \\ -1, & i = d; \\ 0, & \text{otherwise}; \end{cases} \tag{1a}$$

$$\sum_{j\in BS(a)} x_{ja} + \sum_{j\in BS(b)} x_{jb} \leq 1 \qquad \forall\ (a,b) \in F \tag{1b}$$

$$x_{ij} \in \{\ 0,1\ \} \qquad \forall\ (i,j) \in A. \tag{1c}$$

The objective function minimizes the total length of a solution. Constraints (1a) represent the flow balance constraint at each node, while constraints (1b) guarantee that no forbidden pair is violated.

As formally proved in Appendix A, any $\mathcal{CSPTP}$ instance

$$\left\langle G = (V, A, C), s, d, N, \{\ T_h\ \}_{h=1,\ldots,N}, \{u_{ij}\}_{(i,j)\in A} \right\rangle$$

can be polynomially transformed into a $\mathcal{PAFPP}$ instance

$$\left\langle G' = (V', A', C'), s, d' = d + (N-1)\cdot m, F = \{\ (a_1,b_1), \ldots, (a_p,b_p)\ \} \right\rangle,$$

where $p = \frac{m(N-2)(N-1)}{2}$ and G' is a multi-stage graph with N stages, each of them replicating G.

The nodes in V' can be classified in *actual* nodes (from 1 to $n \cdot N$) and *dummy* nodes (from $N \cdot n + 1$ to $N \cdot (n + m)$). Each arc of A' connects an actual node to a dummy node (or vice-versa). The actual nodes are used to replicate the paths, dummy nodes are used to preserve feasibility.

For each arc $(v, w) \in A$ and for each $h = 1, \ldots, N$, we insert an arc in A' from the actual node $v + n \cdot (h-1)$ to the dummy node i. In addition, if $w \in T_{h+1}$ arc $(i, w + h \cdot n)$ is inserted in A', while arc $(i, h + n \cdot (k-1))$ is added in A' if $w \notin T_{h+1}$. In practice, for each of the N stages, each arc $(v, w) \in A$ is represented in A' by a pair of arcs: one connects v to a dummy node, and the other one connects the dummy node to w. If $v \in T_h$ and $w \in T_{h+1}$, then those arcs originate in stage h and end in stage $h + 1$; otherwise their tail node and head node are in the same stages.

The multi-stage graph G' associated to the original graph G showed in Figure 1 is depicted in Figure 5. The dotted nodes indicate the dummy nodes, whereas pair of dummy nodes, belonging to the same level, represent forbidden pairs. The optimal path P' in G' is $P' = \{\ 1, 21, 7, 30, 10, 27, 11, 35, 16\ \}$, whose length is 8. The corresponding optimal solution in G is $P = \{\ 1, 3, 2, 3, 4\ \}$ with the same length.

To obtain a lower bound, a relaxation of the $\mathcal{PAFPP}$ (referred to as $\mathcal{PAFPP}_R$) is built by deleting constraints (1b). The resulting problem is a classical $\mathcal{SPP}$ that can be solved

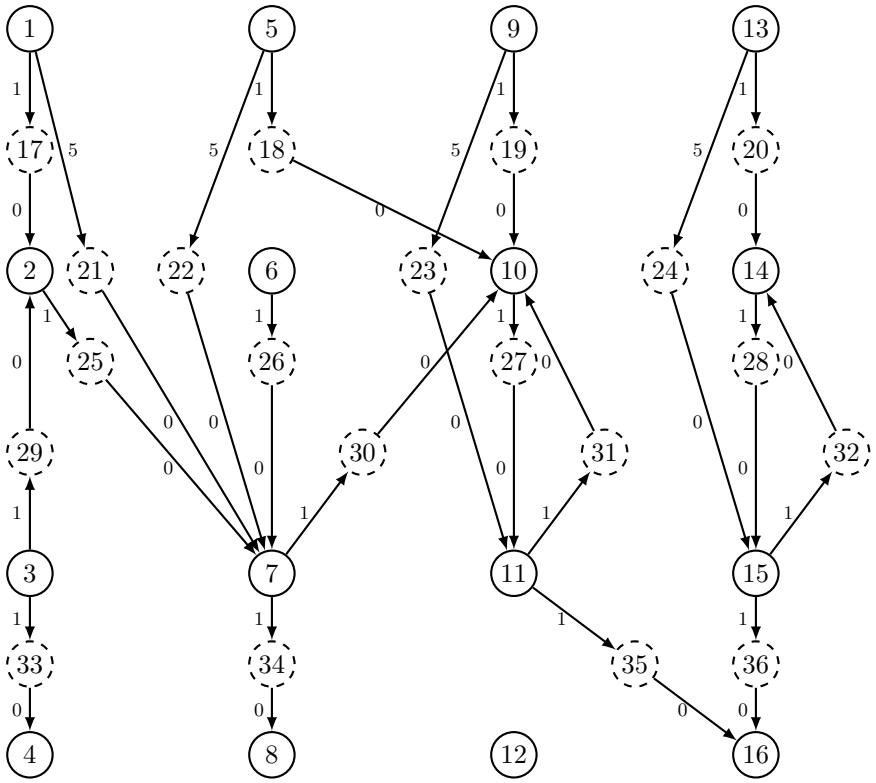

Figure 5: Multistage graph G' associated to the graph G represented in Figure 1.

Figure 6: Branching rule: for each node of the B&B tree we report in bold the forbidden pair chosen among those that make the related relaxed problem infeasible.

by using any of the well-known methods to address this kind of problem (for example, the Dijkstra's algorithm [7]).

At a generic iteration t of the B&B, if the optimal solution P^t of the relaxed problem $\mathcal{PAFPP}^t_R$ does not contain any forbidden pair, then it is also feasible for the $\mathcal{CSPTP}$. On the contrary, if $\mathcal{PAFPP}^t_R$ contains at least one forbidden pair, two subproblems are generated starting from $\mathcal{PAFPP}^t_R$ and adding the violated constraints.

In particular, let G'^t denote the multi-stage graph associated with the relaxed problem $\mathcal{PAFPP}^t_R$ and let V'^t and A'^t be the set of nodes and the set of arcs of G'^t, respectively. If the optimal path in G'^t is not feasible for the $\mathcal{CSPTP}$, because it contains at least one forbidden pair, two problems (i.e, the multistage graphs G'^{t^1} and G'^{t^2}) are determined starting from $\mathcal{PAFPP}^t_R$ (i.e., G'^t). Let F^t be the set of forbidden pairs included in P^t. A forbidden pair (a_k, b_k) is selected from F^t and two problems are defined on multistage graphs, each of them obtained by deleting from G'^t, one at a time, the nodes belonging to the chosen forbidden pair. In other words, the sub-problems G'^{t^1} and G'^{t^2} are built by deleting from V'^t the node a_k and the node b_k, respectively.

A graphical representation of the branching rule is depicted in Figure 6.

In order to get an idea about the way in which the outlined branching strategy works, it is useful to consider the expanded graph G' of Figure 5. The optimal path from node 1 to node 16 in G' is $P = \{ 1, 17, 2, 25, 7, 30, 10, 27, 11, 35, 16 \}$. This solution is not feasible for the $\mathcal{CSPTP}$ since it contains the nodes 25 and 27 that belong to a forbidden pair. Two subproblems are determined, by deleting the nodes 27 and 25, respectively. The first subproblem is infeasible, since node 16 cannot be reached starting from node 1. On the other hand, starting from the solution of the relaxed of the second subproblem, the optimal solution of the original problem is determined.

The strategy for selecting the next sub-problem to investigate determines how the B&B algorithm should proceed through the search tree and can have a significant effect on the behavior of the algorithm. To choose the next node for exploration, the proposed approach adopts both a depth first search (DF, for short) strategy, where the node with the largest level in the search tree is chosen, and a best bound first strategy (BF, for short), that chooses a node with the best relaxed objective function value.

The last issue, to be addressed when a B&B approach is defined, is related to the fathoming rule. In the proposed B&B, a subproblem is fathomed if the value of the optimal solution of the relaxed problem is worse than the current incumbent.

3.2 GRASP

Owing the computational intractability of the $\mathcal{CSPTP}$, we have designed a GRASP to find good sub-optimal feasible solutions.

The GRASP is an iterative multistart metaheuristic for difficult combinatorial optimization problems [8, 9]. Since 1989, it has been applied to a large set of problems, ranging from

```
algorithm GRASP(MaxIterations)
    for i = 1, ..., MaxIterations do
        Build a greedy randomized solution x
        x ← LocalSearch(x)
        if i = 1 then
            x* ← x
        else if x is better than x* then
            x* ← x
        end if
    end for
    return x*
end algorithm
```

Figure 7: Pseudo-code of a generic GRASP.

scheduling and routing to drawing and turbine balancing. The reader can refer to [13, 14, 15] for a study of a generic GRASP metaheuristic framework and its applications.

It is characterized by the execution, for a certain number of iterations (in the following, denoted as `MaxIterations`), of two main phases: a construction phase and a local search phase.

The construction phase iteratively adds one element at a time to a set, that ends up with a representation of a feasible solution. At each iteration, an element is randomly selected from a *restricted candidate list* (RCL), whose elements are among the best ordered, according to some greedy function that measures the (myopic) benefit of selecting each element.

Once a feasible solution is obtained, the local search procedure attempts to improve it by producing a locally optimal solution with respect to some suitably defined neighborhood structure. Construction and local search phases are repeatedly applied. The best solution found is returned as approximation of the optimal one. Figure 7 depicts the pseudo-code of a generic GRASP heuristic for a minimization problem.

3.2.1 Construction phase

The GRASP construction phase relies on an adaptive greedy function, a construction mechanism for the RCL, and a probabilistic selection criterion. The greedy function takes into account the contribution to the objective function achieved by selecting a particular element. In the case of the $\mathcal{CSPTP}$, the construction phase is iterative and is described in Figure8. It starts with an empty chain of paths and ends with a complete solution given by a chain of paths from s to d.

At a generic iteration, the choice of the next path to be added is determined by ordering all candidate paths (i.e. those that can be added to the solution) in a candidate list CL, with respect to a greedy function related to the length of the candidate paths, computed by DijkstraVariant function, that applies Dijkstra's algorithm taking into account the arc capacities.

```
1: function SimpleConstruction(V, A, s, d, N, {T_i}_{i=1,...,N}, C, K)
2:     if N = 2 then
3:         return DijkstraVariant(V, A, s, d, C, K)
4:     else
5:         P_T ← Nil
6:         min ← +∞
7:         for all v ∈ T_2 do
8:             (l_1, P_1) ← DijkstraVariant(V, A, s, v, C, K)
9:             if l_1 < +∞ then
10:                increase(P_1, v, K)
11:                (l_2, P_2) ← DijkstraVariant(V, A, v, d, C, K)
12:                if l_1 + l_2 < min then
13:                    P_T ← P_1 ⊕ P_2
14:                    min ← l_1 + l_2
15:                end if
16:                Decrease(P_1, v, K)
17:            end if
18:        end for
19:        return (min, P_T)
20:    end if
21: end function
```

Figure 8: Construction of a Greedy Randomized Solution when $N \in \{2, 3\}$

The greedy function is adaptive because the benefits associated with every candidate path are updated at each iteration of the construction phase to reflect the changes involved by the selection of the previous element in terms of number of times each arc is used. The probabilistic component is characterized by randomly choosing one of the best candidates in the RCL, but not necessarily the top candidate.

If the number N of subsets of the problem instance to be solved is strictly less than 4, a feasible solution P_T is a chain made of either a shortest simple path from s to d ($N = 2$) or of two simple paths ($N = 3$). In both cases, for building P_T there is no gain in using the RCL mechanism, since it can be constructed by simply invoking the DijkstraVariant function, as done by algorithm in Figure 8.

If $N \geq 4$, index i is selected at random in $[2, N-1]$ (line 5 of algorithm in Figure 9). Path $P_i = \{s_i, \ldots, d_i\}$ is computed from $s_i \in T_i$ to $d_i \in T_{i+1}$, path P_T is initialized with a chain made of only P_i. Then, the partial chain P_T is iteratively augmented both toward the origin s and the destination d. In fact, starting from $j = i + 2$ and until a complete feasible solution is obtained, at each iteration of the loop **while** two more paths are added in a greedy randomized adaptive fashion: a path from a node in T_{i-1} to s_i and a path from d_{j-1} to a node in T_j.

Figure 9 uses a $n \times n$ matrix K such that u_{ij} is the number of times arc (i, j) has been involved in the partial solution P_T. Procedures Increase and Decrease update K to reflect

```
1: function CONSTRUCTION(V, A, s, d, N, {T_i}_{i=1,...,N}, C, K, α)
2:     if N < 4 then
3:         return SIMPLECONSTRUCTION(V, A, s, d, N, {T_i}_{i=1,...,N}, C, K)
4:     end if
5:     i ← RAND(2, N − 2);   P_T ← NIL;   CL ← ∅
6:     for all v ∈ T_i do
7:         for all w ∈ T_{i+1} do
8:             (l, P) ← DIJKSTRAVARIANT(V, A, v, w, C, K)
9:             CL ← CL ∪ { (l, P, v, w) }
10:        end for
11:    end for
12:    RCL ← MAKERCL(α);   (l, P, v, w) ← SELECT(RCL)
13:    P_i ← P; P_T ← P_T ⊕ P_i
14:    s_i ← v; d_i ← w
15:    INCREASE(P_i, w, K)
16:    j ← i + 2; i ← i − 1
17:    while (i > 0) ∨ (j < N) do
18:                                ▷ Look for a candidate path from a node in T_i to node s_{i+1}
19:        CL ← ∅
20:        if i > 1 then
21:            for all v ∈ T_i do
22:                (l, P) ← DIJKSTRAVARIANT(V, A, v, s_{i+1}, C, K)
23:                CL ← CL ∪ { (l, P, v, s_{i+1}) }
24:            end for
25:        else if i = 1 then
26:            (l, P) ← DIJKSTRAVARIANT(V, A, s, s_2, C, K)
27:            CL ← CL ∪ { (l, P, s, s_2) }
28:        end if
29:        RCL ← MAKERCL(α);   (l, P, v, w) ← SELECT(RCL)
30:        P_i ← P; P_T ← P_i ⊕ P_T
31:        s_i ← v; d_i ← w
32:        INCREASE(P_i, w, K)
33:                                ▷ Look for a candidate path from node d_{j−1} to a node in T_j
34:        CL ← ∅
35:        if j < N then
36:            for all v ∈ T_j do
37:                (l, P) ← DIJKSTRAVARIANT(V, A, d_{j−1}, v, C, K)
38:                CL ← CL ∪ { (l, p, d_{j−1}, v) }
39:            end for
40:        else if j = N then
41:            (l, P) ← DIJKSTRAVARIANT(V, A, d_{j−1}, d, C, K)
42:            CL ← CL ∪ { (l, P, d_{j−1}, d) }
43:        end if
44:        RCL ← MAKERCL(α);   (l, P, v, w) ← SELECT(RCL)
45:        P_j ← P; P_T ← P_T ⊕ P_j
46:        s_j ← v; d_j ← w                    14
47:        INCREASE(P_j, w, K)
48:        j ← j + 1; i ← i − 1
49:    end while
50:    return P_T
51: end function
```

Figure 9: Construction of a Greedy Randomized Solution

the choices made by the construction algorithm.

```
1: function LocalSearch(pt, V, A, s, d, N, {T_i}_{i=1,...,N}, C, K)
2:     flag ← True
3:     while flag = True do
4:         flag ← False
5:         for i ← 2 to N − 1 do
6:             Decrease(P_{i−1}, d_{i−1}, K)
7:             Decrease(P_i, d_i, K)
8:             min ← L(P_{i−1}) + L(P_i)
9:             for all v ∈ T_i do
10:                (l′, P′) ← DijkstraVariant(V, A, s_{i−1}, v, C, K)
11:                (l″, P″) ← DijkstraVariant(V, A, v, d_i, C, K)
12:                if l′ + l″ < min then
13:                    min ← l′ + l″
14:                    P_{i−1} ← P′
15:                    P_i ← P″
16:                    flag ← True
17:                end if
18:            end for
19:            Increase(P_{i−1}, d_{i−1}, K)
20:            Increase(P_i, d_i, K)
21:        end for
22:    end while
23:    return P_T
24: end function
```

Figure 10: Local Search

3.2.2 Local search

At each GRASP iteration, once obtained a greedy randomized adaptive path tour P_T, a local search procedure is applied starting from P_T in the attempt of improving it by producing a locally optimal solution with respect to a suitably defined neighborhood structure.

For the $\mathcal{CSPTP}$, given two solutions $P_T = P_1 \oplus \cdots \oplus P_{N-1}$ and $R_T = R_1 \oplus \cdots \oplus R_{N-1}$ their symmetric difference set is defined as follows:

$$\Delta(P_T, R_T) = \{\, i = 1, \ldots, N-1 \mid P_i \neq R_i \,\}.$$

Coherently, the distance between P_T and R_T can be defined as:

$$d(P_T, R_T) = |\Delta(P_T, R_T)|.$$

The pseudocode of the local search procedure we have designed for the $\mathcal{CSPTP}$ is reported in Figure 10. It takes as input the GRASP path tour P_T and outputs a local optimal feasible

tour with the respect to the neighborhood $\mathcal{N}(P_T)$, defined as follows:

$$\mathcal{N}(P_T) = \{ R_T \mid d(P_T, R_T) \leq 2 \}.$$

The loop **while** (lines 3–22) stops as soon as any improving solution in the neighborhood of the current solution can be found. At each iteration, given the current solution $P_T = \bigoplus_{i=1}^{N-1} P_i$, iteratively for $i = 2, \ldots, N-1$ it checks whether $P_{i-1} \oplus P_i$ can be substituted by any shorter $P' \oplus P''$, where P' originates in s_{i-1} and ends in some node v in T_i and P'' originates in v and ends in d_i (lines 9–18).

The neighborhood is explored with a best improvement strategy.

4 Computational results

In this section, we report on some computational experiments designed to analyze the performance of the proposed algorithms, that have been coded in *C++ language* and run on a *Intel Core i7 Quad core @ 2.67 GHz* processor, under the *Linux* (*Ubuntu 11.10*) operating system.

The objective of the computational study has been to compare the running times achieved by the algorithms as a function of the parameter N, when applied on several different networks, with different densities and number of nodes. All test problems have been pseudo-randomly generated by using a generator proposed by Festa and Pallottino in 2003 ([12]). For a detailed description of how such instances are generated, the reader is referred to [10].

B&B has been implemented using two different strategies to build the branching tree: *depth first* (`BBdf`) and *best bound first* (`BBbf`). The criterion used to stop `GRASP` has been `MaxIterations` = 100.

For each problem family, ten different instances have been generated and the mean running time (in seconds) has been computed and stored. The quality of the solutions determined by `GRASP` has been evaluated by computing the mean relative error $\epsilon = \frac{z'-z^*}{z^*}$, where z' is the objective function value corresponding to the suboptimal solution and z^* is the optimal value.

4.1 Complete graphs

A first set of experiments involves complete graphs with the scope of analyzing how running times of B&B and `GRASP` vary depending on the number n of nodes. The number of sets T_i is $N = .25n$ and $.40n$ nodes belong to some T_i.

Looking at the results reported in Table 1, it is evident that `GRASP` is a robust heuristic, able to find an optimal solution in very limited running times compared with those required by B&B. Furthermore, about the two different B&B strategies, we notice that for most instances `BBdf` slightly outperforms `BBbf`.

On complete graphs of size higher than 260 nodes we could test only `GRASP`, because both B&B implementations failed to solve the problem. Figure 13 plots the running times required by `GRASP` to solve instances on complete graphs with $n \in \{300, 350, 400, 450, 500\}$ and $N = .25n$. Looking at the results, it emerges that in a reasonable running time, medium-large size problem instances can be managed.

	BBbf	BBdf	GRASP	
n	Time	Time	Time	ϵ
100	1.09	1	1.27	0
150	4.2	4.2	4.72	0
200	14.8	15	11.57	0
250	50.36	32	21.59	0
252	54	65.6	23.55	0
254	213.9	122.5	24.45	0
256	255.9	225.6	24.04	0
258	197.1	186.7	24.20	0
260	371.6	370.5	25.82	0

Table 1: Complete graphs: $N = .25n$ and $\sum_{i=1}^{N} |T_i| = .40n$.

In order to assess the behavior of the algorithms depending on the number N of node subsets, we have collected computational results on complete graphs with $n \in \{100, 150\}$ and $N \in \{ .2n, .4n, .6n, .7n, .75n, .8n \}$. Table 2 shows the mean running time for both B&B implementations to find an optimal solution. For `GRASP`, we report the mean relative error and the mean running time to perform 100 iterations.

	BBbf	BBdf	GRASP	
n, N	Time	Time	Time	ϵ
100, 20	0.6	0.8	3	0
100, 40	2.1	2.3	2.6	0
100, 60	5.4	6.3	2.7	0
100, 80	22.21	43.35	1.57	0.001
150, 20	3.1	3.1	10	0.005
150, 40	9.8	10	9.7	0.0005
150, 60	55.4	63.9	8.4	0.00004
150, 70	96.4	110.8	8.5	0.0007
150, 75	282.7	315.1	7.6	0.001

Table 2: Complete graphs, $\sum_{i=1}^{N} |T_i| = .80n$.

From the results, it is evident that the running time of `BBbf` and `BBdf` increases with N. This behavior can be explained by observing that the size of the expanded graph G' depends on N, and therefore to build it a higher computational cost is required. About `GRASP`, the higher the number N the lower the mean running time.

To deeper analyze this behavior, Figure 11 plots the mean running times required by `GRASP` to find the best local optimal solution over `MaxIterations` = 100 iterations for complete graphs with $n = 200$ and $N \in \{ .1n, .2n, .3n, .4n, .5n, .6n, .7n, .8n, .9n, n \}$. Looking at the results, for $N \sim n$ the mean running time is very low, less than about 30 seconds. This behavior can be explained by observing that the higher N, the lower the cardinality of each

T_i, $i = 1, \ldots, N$, and this impacts in both construction and local search phases. In the construction phase, a smaller number of iterations is performed each time a new path must be found to be added to the partial chain. In the local search, a smaller effort must be spent to explore the lower cardinality neighborhood of the current solution.

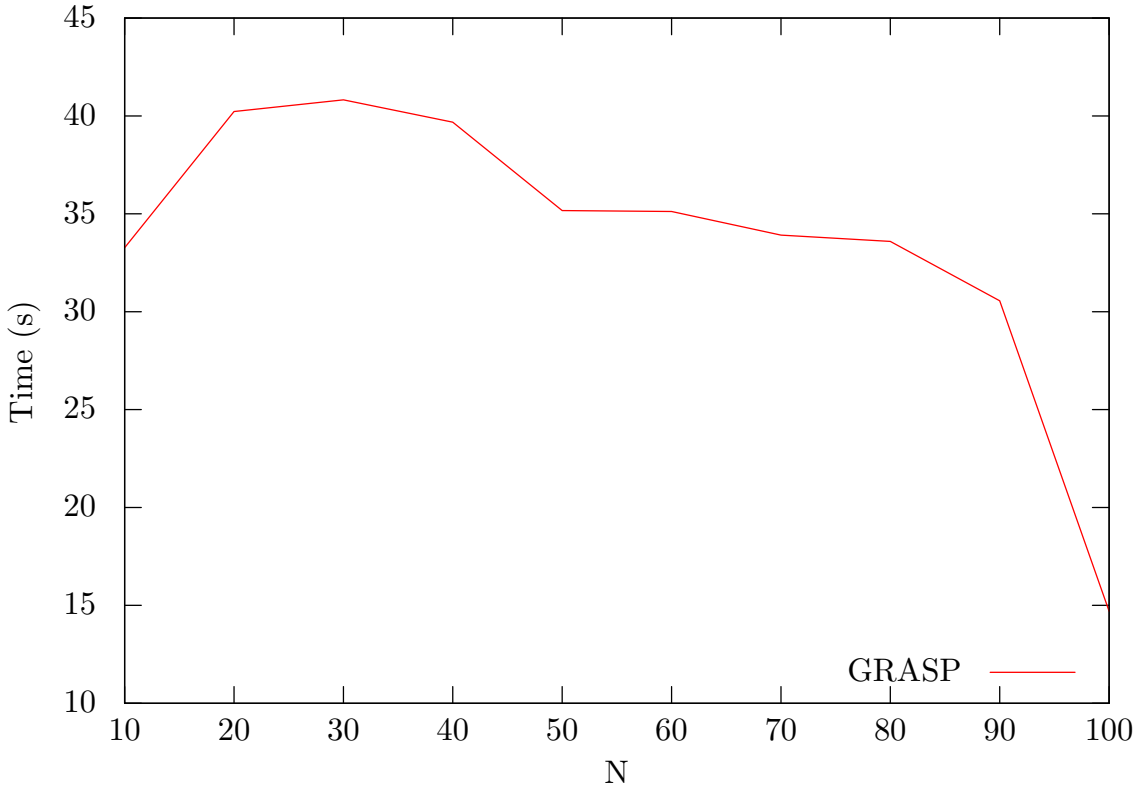

Figure 11: Mean running times (over ten trials) required by `GRASP` to find a suboptimal solution for complete graphs with $n = 200$ and $N \in \{.1n, .2n, .3n, .4n, .5n, .6n, .7n, .8n, .9n, n\}$.

On complete graphs with $n = 200$, $N = 10$, and

$$\sum_{i=1}^{N} |T_i| \in \{\ .1n, .2n, .3n, .4n, .5n, .6n, .7n, .8n, .9n, n\ \},$$

we have performed a further experiment, whose results are showed in Figure 12. The figure plots the mean running times required by B&B implementations to find an optimal solution and the mean running time required by `GRASP` to find the best local optimal solution. Contrary to the previous case, in these experiments the number N of node subsets is equal to 15 and varies the cardinality of each subset.

Looking at the figure, it is evident that the running time of `GRASP` varies proportionally to the percentage of nodes belonging to some T_i, $i = 1, \ldots, N$. This result confirms the conclusion drawn above about the relationship between the cardinality of node subsets and running time to perform each local search and construction iterations.

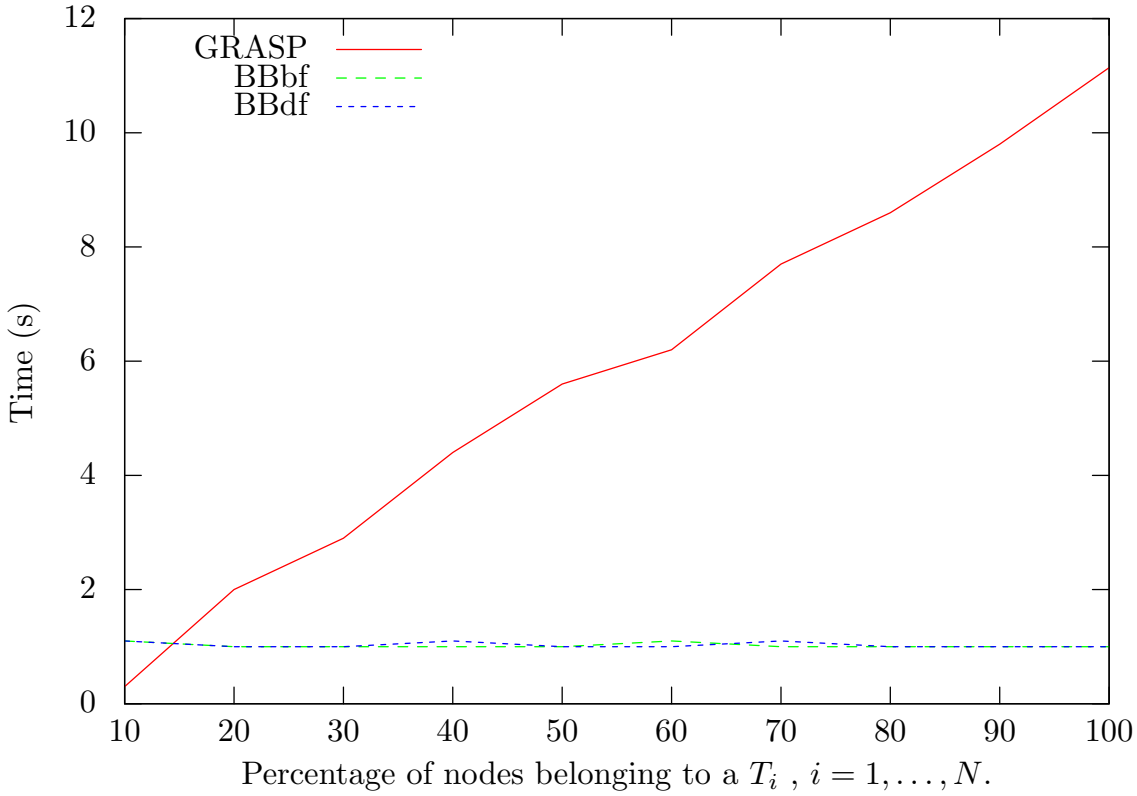

Figure 12: Mean running times (over ten trials) for complete graphs with $n = 150$, $N = 15$ and $\sum_{i=1}^{N} |T_i| \in \{.1n, .2n, .3n, .4n, .5n, .6n, .7n, .8n, .9n, n\}$.

4.2 Random graphs

Table 3 reports running times on random graphs with $n = 100$ and $m \in \{4n, 8n, 16n, 32n\}$. For these instances, `GRASP` found high quality solutions with a mean relative error ϵ less than or equal to 0.004. Running times of `GRASP` and B&B implementations are competitive, with the exception of sparse graphs with 400 arcs. This is not surprising because in case of sparse graphs B&B techniques require to perform a higher number of branching operations. Contrary to the complete graphs case, the `BBdf` is outperformed by `BBbf`. This behavior is explained by observing that in the sparse case there are less feasible solutions. Therefore, it is not worth applying a depth strategy that tends to find rapidly complete solutions that in this case can be infeasible with high probability.

Summarizing, the higher m the lower the computational effort, since the higher the density of the graph, the higher the number of paths connecting two nodes. Consequently, it is less likely that any arc constraint is violated. In other words, the more dense is the graph, the lower the number of the generated subproblems and coherently the depth of the corresponding branching tree.

To better analyze the performances of the two B&B implementations compared to the

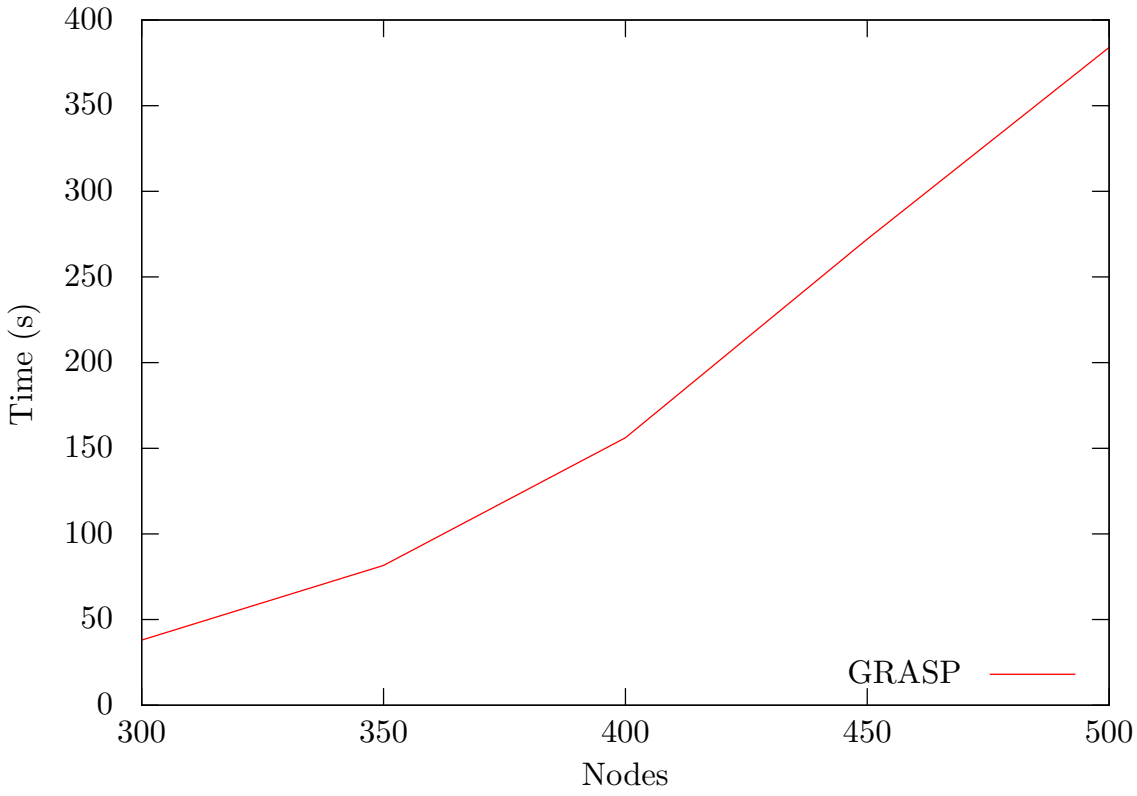

Figure 13: Mean running times (over ten trials) required by `GRASP` to find a suboptimal solution for complete graphs with $N = .25n$ and $n \in \{300, 350, 400, 450, 500\}$.

`GRASP`, additional experiments have been carried out on difficult instances solved within a very small computational time, that is on sparse random graphs and letting all the algorithms run for a fixed amount of time equal to 3 seconds.

For both B&B implementations, Table 4 reports in the second and third columns the mean running times to find an optimal solution (columns `BBbf` Time and `BBdf` Time). In the fourth, the fifth, and the sixth columns (`BBbf_time` ϵ, `BBdf_time` ϵ, and `GRASP_time` ϵ) it reports the mean relative errors of twoB&B implementations and of `GRASP` after 3 seconds of running time, respectively. The symbol "—" means that the related approach is not able to find a feasible solution within the time limit of 3 seconds, for each of the considered instances.

It is worth noting that when a feasible solution is obtained, it represents the optimal one. For the considered networks, for the case $n = 120$, a feasible solution is obtained only for 6 instances , for $n = 140$ the number of instances solved is equal to 4, while only 2 instances are solved for $n = 160$ and only 1 for $n = 180$.

It is evident that with the same amount of time **BBdf** has a higher probability to find a feasible solution respect to **BBbf**. Except for the case $n = 100, m = 500$, the version of the `GRASP` that is executed for only 3 seconds (i.e., `GRASP_time`) has always a mean relative error lower than `BBdf_time`. This confirms the robustness of `GRASP` for sparse graphs.

	BBbf	BBdf	GRASP	
m	Time	Time	Time	ϵ
400	24.9	85.9	0.2	0.004
800	0.7	2	0.22	0.001
1600	0.2	0.5	0.32	0
3200	0.3	0.4	0.53	0.0004

Table 3: Random graphs: $n = 100$, $N = .24n$, and $\sum_{i=1}^{N} |T_i| = .40n$.

	BBbf	BBdf	BBbf_time	BBdf_time	GRASP_time
Size (n, m)	Time	Time	ϵ	ϵ	ϵ
100, 500	0.1	0.8	0	0	0.001
120, 600	3	7.4	—	0.009	0.0005
140, 700	34.3	166.8	—	0.058	0.005
160, 800	86.2	271	—	0.068	0.003
180, 900	25.4	386.6	—	0.046	0.006

Table 4: Random graphs, $N = .20n$ and $\sum_{i=1}^{N} |T_i| = .40n$.

4.3 Grid graphs

A final set of experiments involves two types of grid graphs: square grids with side length $l \in \{9, 10, 12, 14, 20, 25\}$ (i.e., $n \in \{81, 100, 144, 196, 400, 625\}$) and elongated grids with shortest side length $l_1 \in \{4, 5, 7, 10, 12\}$ and highest side length $l_2 \in \{15, 20, 25, 40, 50\}$. The number of sets T_i is $N = .p \cdot n$ with $p \in \{15, 16, 17, 18, 19\}$ and $.35n$ nodes belong to some T_i. For each grid type and for each size, we have computed the average running times employed by BBbf and GRASP over 100 different randomly generated instances. The criterion adopted to stop GRASP has been either the achievement of an optimal solution or, in the worst case, the achievement of the time limit equal to the one required by BBbf.

For small sized squared and elongated grids, the results obtained are summarized in Tables 5 and 6, where for each problem type, the instance size is reported in the first column. The remaining columns report BBbf mean running times over all instances optimally solved by GRASP, GRASP mean running times over all instances optimally solved by GRASP, the number of instances optimally solved by GRASP out of 100 versus that solved by BBbf, the mean relative error ϵ computed over all instances solved by BBbf, BBbf mean running times over all instances solved by BBbf, and GRASP mean running times over all instances solved by BBbf.

The results obtained confirm that the B&B method can only be used to solve small sized instances of the problem, since only for three square grid type graphs (i.e., $09 \times 09N15$, $09 \times 09N16$, and $09 \times 09N17$) it found in reasonable running times an optimal solution for all the 100 instances.

For the same three square grid type graphs, GRASP found an optimal solution for the 95%, 96%, and 91% out the 100 instances, respectively. For 20 square grid type graphs, GRASP found high quality solutions with a mean relative error ϵ less than or equal to 0.002.

Both mean running times of GRASP (one computed over all instances optimally solved

Instance $l \times lNp$	BBbf time (# GRASP optima)	GRASP time (# GRASP optima)	# GRASP optima	ϵ	BBbf time (# BBbf optima)	GRASP time (# BBbf optima)
$09 \times 09N15$	0.284211	0.0210526	95 / 100	0.000462315	0.3	0.12
$09 \times 09N16$	0.3125	0.0625	96 / 100	0.000441504	0.33	0.14
$09 \times 09N17$	0.351648	0.010989	91 / 100	0.000829703	1.23	0.19
$09 \times 09N18$	0.897436	0.141026	78 / 99	0.0026438	4.34343	1.17172
$09 \times 09N19$	4.73239	0.197183	71 / 98	0.00227323	6.03061	2.07143
$10 \times 10N15$	3.66667	0.390805	87 / 99	0.000512269	4.57576	0.949495
$10 \times 10N16$	10.8442	0.532468	77 / 92	0.00126105	18.0652	3.54348
$10 \times 10N17$	15.8429	0.971429	70 / 85	0.00119165	19.3529	4.32941
$10 \times 10N18$	18.4839	1.06452	62 / 84	0.00209905	36.4881	10.4762
$10 \times 10N19$	31.2143	2.85714	42 / 58	0.00165879	42.3966	13.931

Table 5: Small sized square grids: $l \in \{9, 10\}$ (i.e., $n \in \{81, 100\}$), $N = .p \cdot n$ with $p \in \{15, 16, 17, 18, 19\}$, and $\sum_{i=1}^{N} |T_i| = .35n$.

Instance $l_1 \times l_2 N p$	BBbf time (# GRASP optima)	GRASP time (# GRASP optima)	# GRASP optima	ϵ	BBbf time (# BBbf optima)	GRASP time (# BBbf optima)
$05 \times 20N15$	7.77143	1.14286	70 / 88	0.00102714	16.2614	4.38636
$05 \times 20N16$	10.45	2.15	60 / 73	0.00104625	28.2877	6.93151
$05 \times 20N17$	23.6098	2.07317	41 / 45	0.000448966	24.4222	4.11111
$05 \times 20N18$	71.3077	5.5	26 / 34	0.00156575	68.3824	20.4412
$05 \times 20N19$	22.8824	3.52941	17 / 23	0.00215951	62.0435	19.0435
$07 \times 15N15$	7.16279	0.360465	86 / 97	0.000663294	8.16495	1.34021
$07 \times 15N16$	15.3506	3.22078	77 / 90	0.000982485	111.644	18.9333
$07 \times 15N17$	15.5	1.79032	62 / 79	0.0014181	21.8861	6.13924
$07 \times 15N18$	17.2321	2.17857	56 / 63	0.000384317	32.9524	5.60317
$07 \times 15N19$	21.9444	3.63889	36 / 48	0.00182312	39.4792	12.7292

Table 6: Small sized elongated grids: $l_1 \in \{5, 7\}$ and $l_2 \in \{15, 20\}$ (i.e., $n \in \{100, 105\}$), $N = .p \cdot n$ with $p \in \{15, 16, 17, 18, 19\}$, and $\sum_{i=1}^{N} |T_i| = .35n$.

by GRASP and one computed over all instances solved by BBbf) are lower of several order of magnitude compared to BBbf. This is not surprising because grid graphs are sparse and, as already shown in other experiments, in case of sparse graphs B&B techniques require to perform a higher number of branching operations.

Instance	GRASP time
12x12N19	0.61
14x14N19	1.4
25x04N19	0.22
20x20N19	3.35
25x25N19	6.31
10x40N19	2.43
12x50N19	4.54

Table 7: Higher sized square and elongated grids: $l \in \{12, 14, 20, 25\}$ (i.e., $n \in \{144, 196, 400, 625\}$), $l_1 \in \{4, 10, 12\}$ and $l_2 \in \{25, 40, 50\}$, $N = .19 \cdot n$, and $\sum_{i=1}^{N} |T_i| = .35n$.

As in other experiments, on grid graphs of higher size we could test only GRASP, because BBbf failed to solve the problem.

Table 7 reports the mean running times required by GRASP to solve instances on square grids with side length $l \in \{12, 14, 20, 25\}$ (i.e., with $n \in \{144, 196, 400, 625\}$) and elongated grids with shortest side length $l_1 \in \{4, 10, 12\}$ and highest side length $l_2 \in \{25, 40, 50\}$ (i.e., with $n \in \{100, 400, 600\}$). Looking at the results, it emerges that in a very reasonable running time, medium-large sized problem instances can be managed.

5 Conclusions and future work

In this paper, we have studied a constrained version of the shortest path tour problem ($\mathcal{CSPTP}$), in which a feasible solution must not include any repeated arc. The theoretical properties of the problem at hand have been studied, including the membership of the problem to the NP-Hard class. Furthermore, two solution approaches have been designed and tested: an exact Branch & Bound algorithm and a GRASP metaheuristic.

An extensive computational study has been carried out on a variety of network instances with the goal of assessing the behavior of the proposed solution procedures. The computational results obtained have shown that, given the intrinsic complexity of the $\mathcal{CSPTP}$, the Branch & Bound method allows to compute the optimal solution only when small sized instances are considered, (i.e., complete networks with a number of nodes less than 260, random networks with a number of nodes less than 200, and grid networks with a number of nodes less than or equal to 105), whereas GRASP is able to solve complete graph instances of up 400 nodes in less than 200 seconds and grid graph instances of up 625 nodes in less than 7 seconds. Future research comprises the development of a tailored dynamic programming based algorithm. This represents the subject of current investigation.

Appendix A

Theorem 1 $\mathcal{CSPTP}$ belongs to the class **NP**.

Proof: To show that a given problem belongs to the class **NP**, one needs to show that there exists an algorithm that takes as input a *certificate* of a solution and in polynomial time in the input size verifies whether this certificate is correct or not.

For the $\mathcal{CSPTP}$, to argue that it belongs to the class **NP**, we need to show that a certificate $Q = \{i_1, \ldots, i_l\}$ consisting of a sequence of l nodes can be verified in polynomial time. Algorithm in Figure 14 is the verifying algorithm. It checks that the sequence Q starts in s and ends in d in lines 2–4. The loop **for** in lines 10–18 checks if Q involves an arc at most once (line 11) and if Q visits successively and sequentially all the given subsets $T_1, T_2, \ldots, T_N$ (lines 15–17 and 19–21).

It is easy to see that the computational complexity of the verification algorithm in Figure 14 is $O(m + l)$, since the loop **for** starting at line 5 requires $O(m)$ and the loop **for** starting at line 10 requires $O(l)$.

If the input sequence Q is a feasible solution, then it must result that $l \leq m + 1$. On the contrary, if Q is not feasible because it involves a number of arcs greater than m, then the condition tested in line 11 would result TRUE at iteration $m + 1$.

Therefore, in both cases the loop **for** starting at line 10 requires $O(m)$, resulting in $O(m)$ the computational complexity of the whole algorithm. ■

Lemma 2 There exists a feasible path $P = i_1, i_2, \ldots, i_k$, $k \leq n$, in

$$\langle G = (V, A, C), s, d \rangle ,$$

if and only if in

$$\left\langle G' = (V', A', C'), s^-, d^+, \{T_h\}_{h=1,\ldots,n+1} \right\rangle$$

there exists a feasible path tour P' from i_l^- to i_k^+, such that

$$P' = \left\{ \bigoplus_{l=1}^{k-1} \left(i_l^-, i_l^+, i_l i_{l+1}{}^{l+1} \right), i_k^-, i_k^+ \right\}.$$

Proof: Suppose that there exists in G a feasible path $P = \{i_1, i_2, \ldots, i_k\}$, $k \leq n$. Then, by construction there exists in A' an arc (i_l^-, i_l^+), for each $l = 1, \ldots, k$.
Moreover, for each arc (i_l, i_{l+1}) in P, there exist arcs $(i_l^+, i_l i_{l+1}{}^q)$ and $(i_l i_{l+1}{}^q, i_{l+1}^-)$ for each $q = 2, \ldots, n$.
Therefore, there must exist also arcs $(i_l^+, i_l i_{l+1}{}^{l+1})$ and $(i_l i_{l+1}{}^{l+1}, i_{l+1}^-)$.
Conversely, suppose that there exists in G' the path P', whereas path P is not present in G. This last situation occurs if either at least one node $i_l \notin V$ or at least one arc $(i_l, i_{l+1}) \notin A$. If a node $i_l \notin V$, then nodes i_l^- and i_l^+ would not be in V', which is not true. Similarly, if an arc $(i_l, i_{l+1}) \notin A$, then arcs $(i_l^+, i_l i_{l+1}{}^{l+1})$ and $(i_l i_{l+1}{}^{l+1}, i_{l+1}^-)$ would not be in A' and this contradicts the hypothesis of existence of path P'. ■

```
1: algorithm VERIFIER(V, A, C, s, d, N, {T_h}_{h=1,...,N}, Q = {i_1, i_2, ..., i_l})
2:     if (i_1 ≠ s) ∨ (i_l ≠ d) then
3:         return FALSE
4:     end if
5:     for all (i, j) ∈ A do
6:         u_ij ← 0
7:         exists_ij ← TRUE
8:     end for
9:     q ← 1
10:    for h ← 1 to l − 1 do
11:        if ¬exists_ij ∨ (u_{i_h i_{h+1}} > 0) then
12:            return FALSE
13:        end if
14:        u_{i_h i_{h+1}} ← 1
15:        if (i_h ∈ T_q) ∧ (q ≤ N) then
16:            q ← q + 1
17:        end if
18:    end for
19:    if q < N then
20:        return FALSE
21:    end if
22:    return TRUE
23: end algorithm
```

Figure 14: Verifier for the $\mathcal{CSPTP}$.

Theorem 3 Function $f(\mathcal{I}_{\mathcal{HAM}-\mathcal{PATH}})$ computed by algorithm in Figure 2 is a polynomially computable function that transforms any instance $\mathcal{I}_{\mathcal{HAM}-\mathcal{PATH}}$ of $\mathcal{HAM}-\mathcal{PATH}$ in an instance $\mathcal{I}_{\mathcal{CSPTP}}$ of the $\mathcal{CSPTP}$.

Proof: $f(\mathcal{I}_{\mathcal{HAM}-\mathcal{PATH}})$ is polynomially computable given the polynomial worst case computational complexity of algorithm in Figure 2.
To show that $f(\mathcal{I}_{\mathcal{HAM}-\mathcal{PATH}})$ is a reduction function, we must prove that there exists in G a Hamiltonian path P from s to d with length $L(P)$ if and only if there exists in G' a constrained path tour P' from s^- to d^+ with length $L(P') = L(P)$.

$\Rightarrow$ By hypothesis, there exists in G a Hamiltonian path $P = \{i_1, i_2, \cdots, i_n\}$, where $i_1 = s$ and $i_n = d$. We have already shown in Lemma 2 that there exists in G' a path

$$P' = \left\{ \bigoplus_{l=1}^{n-1} \left(i_l^-, i_l^+, i_l {i_{l+1}}^{l+1} \right) i_n^-, i_n^+ \right\},$$

where $i_1^- = s^-$ and $i_n^+ = d^+$.

P' is a feasible constrained path tour from s^- to d^+. In fact, let us suppose that P' is not feasible. This can happen if at least one of the following cases occurs: 1) P' crosses some arcs more than once; 2) P' does not involve any node in some node subsets T_i, $i = 1, \ldots, n+1$; 3) P' involves at least a node for each T_i, , $i = 1, \ldots, n+1$, but not successively and sequentially.

Suppose that P' crosses some arcs twice. Since only nodes of type i^- are such that $|FS(i^-)| > 1$, if some arc is involved at least twice, it must be some arc of type (i^-, i^+). Nevertheless, if this is the case, then necessarily node i must be involved by P at least twice and this contradicts the hypothesis of P as Hamiltonian path.

Finally, cases 2) and 3) can not ever occur by construction. In fact, path P' starts at $s^- \in T_1$ and ends in $d^+ \in T_{n+1}$. Then, it involves successively and sequentially all nodes $i_l {i_{l+1}}^{l+1}$, for each $l = 1, \ldots, n-1$, and each node $i_l {i_{l+1}}^{l+1}$ belongs to T_{l+1}.

$\Leftarrow$ By hypothesis, there exists in G' a feasible constrained path tour from s^- to d^+.

Remember that by construction, it holds that

- for each node $i^- \in V'$, $FS(i^-) = \{i^+\}$;
- for each node $i^+ \in V'$, $FS(i^+) = \{ij^k \mid k = 2, \ldots, n\}$;
- for each node $ij^k \in V'$, $FS(ij^k) = \{j^-\}$.

Therefore, path P' must be necessarily as follows

$$P' = \left\{ \bigoplus_{l=1}^{n-1} \left(i_l^-, i_l^+, i_l {i_{l+1}}^{l+1} \right) i_n^-, i_n^+ \right\},$$

where $i_1^- = s^-$ and $i_n^+ = d^+$.

In fact, if for some $k = 2, \ldots, n$, $k \neq l+1$, P' contains a subpath

$$i_l, i_{l+1}, i_l {i_{l+1}}^k,$$

then P' would not be feasible, because it would violate the constraint of successively and sequentially passing through at least one node of the node subsets T_i. Finally, if P' involves a smaller number of nodes, then for some subset T_i, no node in T_i would be crossed. Similarly, if P' involves a higher number of nodes, then P' would cross at least one arc more than once.

From Lemma 2, it follows that there exists in G a path $P = \{i_1, i_2, \ldots, i_n\}$, such that $i_1 = s$ and $i_n = d$.

P must be Hamiltonian. In fact, let us suppose that P is not Hamiltonian. Since P visits exactly n nodes, there must be i_j and i_k, $j, k \in \{1, \ldots, n \mid j \neq k\}$, such that $i_j = i_k$. But this implies that P' crosses arcs (i_j^-, i_j^+) and (i_k^-, i_k^+) such that $(i_j^-, i_j^+) \equiv (i_k^-, i_k^+)$ and this contradicts the hypothesis of feasibility of the constrained path tour P'.

The Hamiltonian path P in G and the constrained path tour P' in G' have the same length by construction and by the definition of cost functions C and C', respectively. ■

The $\mathcal{CSPTP}$ is polynomially Karp-reducible to the $\mathcal{PAFPP}$, another known **NP**-complete problem. Figure 15 is the reduction algorithm and is a slightly variant of the reduction algorithm from the $\mathcal{SPTP}$ to a single source–single destination shortest path problem ($\mathcal{SPP}$) presented in [10]. It takes as input a $\mathcal{CSPTP}$ instance

$$\left\langle G = (V, A, C), s, d, N, \{\ T_h\ \}_{h=1,\ldots,N}, \{u_{ij}\}_{(i,j)\in A} \right\rangle$$

and outputs a corresponding $\mathcal{PAFPP}$ instance

$$\left\langle G' = (V', A', C'), s, d' = d + (N-1)\cdot m, F = \{\ (a_1, b_1), \ldots, (a_p, b_p)\ \} \right\rangle,$$

where $p = \frac{m(N-2)(N-1)}{2}$ and G' is a multi-stage graph with N stages, each of them replicating G, as detailed in Section 3.

Theorem 5 *Figure 15 is a polynomial reduction algorithm from the $\mathcal{CSPTP}$ to the $\mathcal{PAFPP}$.*

Proof: To prove this result, we need to show the validity of the following statements:

i. there is a path P' from s to d' of length K in G' if and only if there is a path tour P_T from s to d of length K in G;

ii. each path P' from s to d' in G' does not cross any forbidden pair if and only if the corresponding path tour P_T in G crosses each arc at most once.

To show i. suppose that there is a path P' of length K from s to d' in G'. Since P' connects $s \in T_1$ to $d' \in T_N$, it follows that, for each $k = 1, \ldots, N-1$, at least two adjacent arcs connecting two nodes in consecutive stages k and $k+1$ (in between there is a dummy

```
1: function SPTP-PAFPP(V, A, C, N, (T_h)_{h=1,2,...,N}), {u_ij}_{(i,j)∈A})
2:     V' ← { 1, 2, ..., N · (n + m) }; A' ← ∅; F ← ∅
3:     i ← N * n + 1
4:     for all (v, w) ∈ A do
5:         for h ← 1 to N − 1 do
6:             ADD(A', (v + (h − 1) * n), i, c_vw)
7:             if w ∈ T_{h+1} then
8:                 ADD(A', i, w + h * n, 0)
9:             else
10:                ADD(A', i, w + (h − 1) * n, 0)
11:            end if
12:            i ← i + 1
13:        end for
14:        ADD(A', v + (N − 1) * n, i)
15:        ADD(A', i, w + (N − 1) * n)
16:        for j ← i − N to i − 1 do
17:            for l ← j + 1 to i do
18:                F ← F ∪ { (j, l) }
19:            end for
20:        end for
21:        i ← i + 1
22:    end for
23:    return (V', A', C', F)
24: end function
```

Figure 15: Polynomial reduction algorithm from the $\mathcal{CSPTP}$ to the $\mathcal{PAFPP}$.

node) must necessarily belong to P' . Therefore, P' contains at least one node of each of the N stages and, thus, it corresponds to a path tour P_T of length K in G, that successively passes through at least one node from the given node subsets $T_1, T_2, \ldots, T_N$.
Conversely, suppose that in G there is a path tour P_T of length K that successively passes through at least one node from the given node subsets $T_1, T_2, \ldots, T_N$. Then, by construction, for each arc in P_T connecting two nodes in G belonging to consecutive subsets, there exist two adjacent arcs in the simple path P' connecting two nodes in G' belonging to consecutive stages, until the node d' in the last stage N is reached.

To prove ii., we will proceed by contradiction. Suppose that in G' there is a path P' from s to d' that does not involve any forbidden pair, while the corresponding path tour P_T in G is not simple. This means that there exist two different paths P_i and P_j, $i \neq j$, in P_T that cross the same arc (v, w). By construction, P_i and P_j generate in G' two sub-paths of P' whose origin nodes belong to different stages and the repetition of arc (v, w) generates a forbidden pair involved by P'.
Conversely, suppose that in G there is a simple path tour P_T. The corresponding path P' does not involve any forbidden pair, because dummy nodes belong to a forbidden pair if and only if they represent the same arc in different stages. Since there are no arc repetitions, no forbidden pairs are involved by P'.

The reduction algorithm introduces $N(n+m)$ nodes, $2mN$ arcs and $\frac{m(N-2)(N-1)}{2}$ forbidden pairs. The complexity is

$$O(nN + mN + 2mN + \frac{m(N-2)(N-1)}{2}) = O(mN^2) = O(n^4)$$

and is polynomial. ■

References

[1] A.A. Bertossi. The edge hamiltonian path problem is np-complete. *Information Processing Letters*, 13(4):157–159, 1981.

[2] Dimitri P. Bertsekas. *Linear Network Optimization: Algorithms and Codes*. The MIT Press, October 1991.

[3] D.P. Bertsekas. A simple and fast label correcting algorithm for shortest paths. *Networks*, 23(8):703–709, 1993.

[4] D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, 3rd edition edition, 2005.

[5] B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest paths algorithms: theory and experimental evaluation. *Math. Program.*, 73(2):129–174, May 1996.

[6] T.H. Cormen, C. Stein, R.L. Rivest, and C.E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[7] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[8] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

[9] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[10] P. Festa. Complexity analysis and optimization of the shortest path tour problem. *Optimization Letters*, 6:163–175, 2012.

[11] P. Festa, F. Guerriero, D. Laganà, and R. Musmanno. Solving the shortest path tour problem. *European Journal of Operational Research*, 230:464–474, 2013.

[12] P. Festa and S. Pallottino. A pseudo-random networks generator. Technical report, Department of Mathematics and Applications "R. Caccioppoli", University of Napoli FEDERICO II, Italy, 2003.

[13] P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In P. Hansen and C. C. Rebeiro (eds), editors, *Essays and Surveys on Metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

[14] P. Festa and M.G.C. Resende. An annotated bibliography of GRASP - Part I: Algorithms. *International Transactions in Operational Research*, 16(1):1–24, 2009.

[15] P. Festa and M.G.C. Resende. An annotated bibliography of GRASP - Part II: Applications. *International Transactions in Operational Research*, 16(2):131–172, 2009.

[16] R.W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345+, June 1962.

[17] G. Fuellerer, K.F. Doerner, R.F. Hartl, and M. Iori. Metaheuristics for vehicle routing problems with three-dimensional loading constraints. *European Journal of Operational Research*, 201(3):751–759, March 2010.

[18] H.N. Gabow, S.N. Maheswari, and L.J. Osterweil. On Two Problems in the Generation of Program Test Paths. *IEEE Transactions on Software Engineering*, 2:227–231, 1976.

[19] C.H. Galen, M.M. Maged, P. Srinivasan, and M.L. Scott. An efficient algorithm for concurrent priority queue heaps. *Inf. Proc. Letters*, 60:151–157, 1996.

[20] Poul E. Heegaard and Kishor S. Trivedi. Network survivability modeling. *Computer Networks*, 53(8):1215 – 1234, 2009. Performance Modeling of Computer Networks: Special Issue in Memory of Dr. Gunter Bolch.

[21] R.M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[22] D. Klingman, A. Napier, and J. Stutz. *NETGEN: A Program for Generating Large Scale (UN)Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems.* Defense Technical Information Center, 1974.

[23] C. Lim and J.C. Smith. Algorithms for discrete and continuous multicommodity flow network interdiction problems. *IIE Transactions*, 39(1):15–26, 2007.

[24] Frits C.R. Spieksma. *Multi index assignment problems: complexity, approximation, applications*, chapter 1, pages 1–11. Kluwer Academic Publishers, 2000.