The Constrained Forward Shortest Path Tour Problem: mathematical modeling and GRASP approximate solutions

F. Carrabs[*], C. D'Ambrosio[*], D. Ferone[†], P. Festa[‡], F. Laureana[*]

Abstract

This paper deals with the *Constrained Forward Shortest Path Tour Problem*, an NP-complete variant of the Forward Shortest Path Tour Problem. Given a directed weighted graph $G = (V, A)$, where the set of nodes V is partitioned into clusters $T_1, ..., T_N$, the aim is determining a shortest path between two given nodes, s and d, with the properties that clusters must be visited according to a given order, and each arc can be crossed at most once. We introduce a mathematical formulation of the problem, and a reduction procedure to reduce the number of variables involved in the model. Furthermore, we propose a Greedy Randomized Adaptive Search Procedure (GRASP) algorithm to solve large instances of the problem. Computational tests show that the reduction procedure is very effective and its application significantly speeds up the resolution of the model. Moreover, the computational results certify the effectiveness of GRASP that often finds the optimal solution and, in general, provides quickly high-quality sub-optimal solutions.

Keywords: Shortest path problems; Network optimization; GRASP

[*]Department of Mathematics, University of Salerno, Italy. fcarrabs@unisa.it, cdambrosio@unisa.it, flaureana@unisa.it

[†]Department of Mechanical, Energy and Management Engineering, University of Calabria, Rende, Italy. daniele.ferone@unical.it

[‡]Department of Mathematics and Applications, University of Napoli Federico II, Napoli, Italy. paola.festa@unina.it

1 Introduction

The Constrained Forward Shortest Path Tour Problem (CFSPTP) consists in finding a single-origin single-destination shortest path in a directed weighted graph such that a given set of constraints must be satisfied. In particular, in the CFSPTP it is required that a feasible path i) crosses a sequence of node subsets (clusters) that are given in a fixed order, ii) can involve a node in a given cluster if and only if at least one node for each preceding cluster has been already visited, and iii) does not include arcs crossed more than ones. Since the triangular inequality does not hold in this problem, the optimal solution could require to visit the same cluster more than once.

The CFSPTP has many applications. It arises, for example, in the context of freight transportation of hazardous materials carried out by vehicles with one door, on the rear, where loading and unloading operations are executed. Avoiding unnecessary handling is very important when delivering hazardous materials, and then the loading and unloading operations have to be carried out in LIFO order (LAST-IN-FIRST-OUT). This means that when loading, the goods are always placed at the rear of the vehicle. Similarly, unloading at a delivery location is allowed only if the goods of the current delivery are at the rear (Carrabs et al. (2013)). Each cluster contains several unloading points belonging to the same company, and the vehicle must visit a cluster at least once in order to deliver the required materials. The vehicle starts from the depot completely loaded, unloads the goods in a location for each cluster, and finally completes the tour reaching the destination. The visiting sequence of the clusters is stated by the loading order of the goods into the vehicle. Finally, to reduce the traffic of hazardous materials on a single arc, and the associated risks, double use of the same arc is explicitly forbidden (see Bianco et al. (2013), Wolfler Calvo and Cordone (2003) and Michallet et al. (2014)). The routing problems associated with this type of constraints are usually named "peripatetic" routing models.

Another application of the CFSPTP arises in designing a tourist trip itinerary for visiting several kinds of points of interest (POIs), that meets the preferences of the tourists. The POIs are grouped according to their typology, for instance, museums, historical monuments, national parks, lakes, and these groups are sorted by the preferences of the tourists. The aim is to visit at least a POI for each group, according to the given order, by minimizing the total travelling time. Finally, to avoid that tourists see more than once the same landscape, it is forbidden repeating the same journey

between two POIs. This is particularly relevant when journeys have scenic backgrounds, such as mountains areas, cliffs, and so on.

The Shortest Path Tour problem was firstly introduced in Bajaj (1971), and proposed as a dynamic programming exercise in Bertsekas (2005). Nevertheless, Festa (2010) is the first study focused on the problem, that was successively tackled by Festa et al. (2013). Later, Ferone et al. (2016) proposed a variant of SPTP called Constrained Shortest Path Tour Problem, where an arc cannot be traversed more than once. The authors showed that the Hamiltonian Path Problem (Bertossi, 1981) can be polynomially reduced to the CSPTP and that the CSPTP belongs to the **NP**-hard class. The CSPTP was further studied in de Andrade and Saraiva (2018) and Ferone et al. (2019). These two works independently proposed two similar mathematical models for the problem. The CSPTP is a variant of the all color shortest path problem Carrabs et al. (2018, 2020), in which the vertices are grouped into clusters according to their color and a shortest path visiting at least one vertex for each cluster should be visited.

Finally, Di Puglia Pugliese et al. (2020) studied a further variant of the SPTP, where time windows constraints are introduced.

In 2017, Carrabs et al. defined the Forward Shortest Path Tour Problem, a simpler version of the CFSPTP, where a feasible path can cross repeated arcs. The word *forward* in the name of the problem means that a node of a given cluster can be visited if and only if at least one node for each preceding cluster has been already visited. This is a crucial difference with respect to the SPTP problem, where this precedence constraint is not applied. To the best of our knowledge, the CFSPTP has not been previously addressed in the scientific literature. Since it can be viewed as a variant of the CSPTP (Ferone et al., 2016), it remains an **NP**-hard problem.

The contribution of this paper can be summarized as follows. We propose a mathematical formulation of the problem and a procedure that significantly reduces the number of variables of the model speeding up its convergence. To find optimal solutions, we solve its mathematical model by using CPLEX. To address large problem instances, we design and implement a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic. 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 remainder of this paper is organized as follows. In Section 2, we define the problem and introduce some notations. In Section 3, we mathematically formulate the CFSPTP, while the GRASP algorithm is described

in Section 4. Computational results and the analysis of the performance of the proposed algorithms are presented in Section 5. Concluding remarks are given in Section 6.

2 Problem description and notation

Let $G = (V, A)$ be a directed graph, where $|V| = n$ and $|A| = m$. Given a source node $s \in V$ and a destination node $d \in V$, the set of nodes V is partitioned into N clusters, $T_1, ..., T_N$, such that $T_1 = \{s\}$ and $T_N = \{d\}$. Let us introduce a function $c : A \to \mathbb{R}^+$, that associates a non-negative cost c_{ij} to each arc $(i, j) \in A$. Given two nodes $u, v \in V$, we denote with $SP_{u,v}$ the shortest path from u to v, computed on graph G. The cost of such path, $c(SP_{u,v})$, is the sum of the cost of the arcs belonging to the path. Let us consider the small graph shown in Figure 1. The shortest path from node s to node 4 is the path $SP_{s,4} =< (s, 1), (1, 4) >$, and $c(SP_{s,4}) = 3$.
Given a node $u \in T_k$, with $k = 2, .., N$, a Constrained Forward Path Tour (CFPT) from s to u, denoted with $P_{s,u}$, is a path in G from s to u, such that:

(i) at least one node of each cluster T_p, with $p \leq k$, is visited;

(ii) for every $2 \leq p \leq k$, a node in T_p can be visited if and only if at least a node for each cluster $T_1, ..., T_{p-1}$ has been already visited;

(iii) each arc can be crossed at most once.

We denote with $c(P_{s,u})$ the cost of the path $P_{s,u}$. If we consider again the graph shown in Figure 1, a CFPT from s to 4, is for example $P_{s,4} =< (s, 1), (1, 3), (3, 4) >$, and its cost is $c(P_{s,4}) = 8$, which is obviously greater than $c(SP_{s,4})$ on the same graph. In more detail, it results that $c(P_{s,j}) \geq c(SP_{s,j})$, for any $j \in V$. Let $l(P_{s,u})$ be the *length* of $P_{s,u}$, that is the number of arcs belonging to it. The Constrained Forward Shortest Path Tour Problem (CFSPTP) consists of finding the cheapest CFPT from s to d in G. In the graph shown in Figure 1, the optimal solution of CFSPTP is the path $P_{s,d} =< (s, 1), (1, 3), (3, 4), (4, 1), (1, 2), (2, d) >$, its cost is $c(P_{s,d}) = 13$ and its length is $l(P_{s,d}) = 6$.

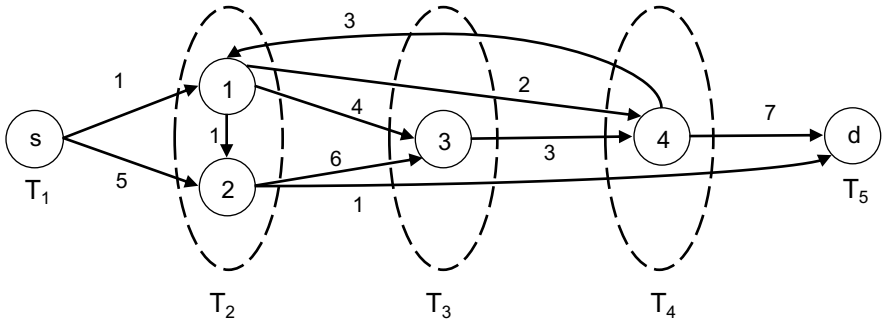

Figure 1: Toy example.

3 Mathematical formulation

CFSPTP can be formulated as an integer linear program (ILP) as follows. We first introduce some notations. Given $S, T \subseteq V$, we define $E(S : T) = \{(i, j) \in A : i \in S, j \in T\}$. Let $\delta^+(S) = E(S : V \setminus S)$ and $\delta^-(S) = E(V \setminus S : S)$ be the sets of outgoing and ingoing arcs of S, respectively, and $E(S) = E(S : S)$ is the set of the arcs having both extremes in S. When $S = \{i\}$, we write $\delta^+(i)$ and $\delta^-(i)$. Let x_{ij}^t be a binary variable equal to 1 if and only if arc $(i, j) \in A$ occurs in the t-th position in $P_{s,d}$. Let us observe that since a feasible solution for CFSPTP cannot contain repeated arcs, it can contain at most all the arcs of the graph once, and then its length is at most m. Therefore, index t belongs to the set $\{1, ..., m\}$. Finally, we define $x^t(A') = \sum_{(i,j)\in A'} x_{ij}^t$, where $A' \subseteq A$ and $t \in \{1, ..., m\}$.
With the above notation, the ILP formulation of CFSPTP is the following:

$$\text{Minimize } z = \sum_{(i,j)\in A} \sum_{t=1}^{m} c_{ij} x_{ij}^t \tag{1}$$

subject to

$$\sum_{(s,j)\in\delta^+(s),j\in T_2} x^1_{sj} = 1 \tag{2}$$

$$\sum_{t=N-1}^{m} x^t(\delta^-(d)) = 1 \tag{3}$$

$$\sum_{i\in T_k}\sum_{t=k-1}^{m} x^t(\delta^-(i)) \geq 1 \qquad k = 2,...,N-1 \tag{4}$$

$$\sum_{i\in T_k} x^t(\delta^-(i)) \leq \sum_{t'=k-2}^{t-1}\sum_{j\in T_{k-1}} x^{t'}(\delta^-(j)) \quad k = 3,...,N, t = k-1,...,m \tag{5}$$

$$x^t(\delta^-(i)) = x^{t+1}(\delta^+(i)) \qquad i \in V\setminus\{s,d\}, t = 1,...,m-1 \tag{6}$$

$$x^t(A) \leq 1 \qquad t = 1,...,m \tag{7}$$

$$\sum_{t=1}^{m} x^t_{ij} \leq 1 \qquad (i,j)\in A \tag{8}$$

$$x^t_{ij} \in \{0,1\} \qquad (i,j)\in A, t = 1,...,m \tag{9}$$

The objective function (1) to be minimized is the sum of the costs of the selected arcs. Constraint (2) ensures that the first selected arc must be an arc from the source node to a node belonging to the second cluster. Constraint (3) forces the selection of exactly an arc ending to the destination node. Constraints (4) ensure that at least a node from every cluster is visited. Constraints (5) make sure that the sequence of clusters is respected. Constraints (6) ensure the connectivity of the path. Finally, constraints (7) and (8), guarantee that the number of arcs selected in each position is at most one, and that each arc is selected at most once, respectively.
Let us observe that constraints (4) are redundant, since they can be obtained through the combination of constraints (3) and (5). Nevertheless, we introduce them in the model since they are used by CPLEX to derive cuts so as to improve the value of the LP relaxation. Therefore, this leads to a significant reduction in terms of the computational time.

3.1 Reduction of the binary variables

The number of binary variables of the model is m^2, and this slows down the convergence of the model even for small instances. Indeed, by considering a complete graph with only 50 nodes, the number of variables in the model is about six million. To overcome this problem, we design a procedure that, through the computation of an upper bound of the number of arcs involved in the optimal solution, allows us to reduce the number of binary variables of the model. As we pointed out before, the length of a feasible solution for CFSPTP can be at most m, but this bound is rarely reached. Indeed, even if in the worst case a feasible solution contains all the arcs of the graph, usually its length is much lower than m and it is strictly affected by the characteristics of the instance. Hence, we aim to determine a better bound for the index t, in other words, we look for an upper bound on the length of any optimal solution. Let us denote with $G_k = (V_k, A_k)$ the subgraph of G induced by clusters T_i, with $i = 1, ..., k$. More in detail, $V_k = \bigcup_{i=1}^{k} T_i$ and $A_k = \{(i, j) \in A : i, j \in V_k\}$. Firstly, we compute a feasible solution of CFSPTP through a greedy algorithm (see Algorithm 1). At the first iteration of Algorithm 1 we compute $P_{s,v}$, the CFPT from s to v, for every $v \in T_2$, applying Dijkstra's algorithm (Dijkstra, 1959) on the subgraph G_2 (line 1-4). Then, at each iteration $k \in \{2, ..., N-1\}$, we compute the CFPT from s to every node $v \in T_{k+1}$, $P_{s,v}$. To this aim, we consider the shortest path $SP_{u,v}$ from every node $u \in T_k$ to every node $v \in T_{k+1}$, on the subgraph $G^u_{k+1} = (V_{k+1}, A^u_{k+1})$, where $A^u_{k+1} = \{(i, j) \in A_{k+1} : (i, j) \notin P_{s,u}\}$ (line 7). The construction of $SP_{u,v}$ is carried out ensuring that no vertices belonging to following clusters are visited, and that it does not contain arcs already used in $P_{s,u}$. In such a way, the path obtained through the concatenation of $P_{s,u}$ and $SP_{u,v}$ is a CFPT from s to v in G. Once we have $SP_{u,v}$, if $c(P_{s,u}) + c(SP_{u,v}) < c(P_{s,v})$, we update $P_{s,v}$ doing the concatenation between $P_{s,u}$ and $SP_{u,v}$ (line 8-10 of Algorithm 1). The algorithm stops returning as output $P_{s,d}$, that is a CFPT from s to d.

For example, let us apply Algorithm 1 to the graph $G = (V, A)$ depicted in Figure 2. At the first iteration, we compute $P_{s,1}$ and $P_{s,2}$, which are simply shortest paths in the subgraph G_2 from s to node 1 and from s to node 2, respectively. We have that $P_{s,1} =< (s, 1) >$, while $P_{s,2} =< (s, 1), (1, 2) >$. Then, we look for $P_{s,3}$ and $P_{s,4}$, and we compute $SP_{1,3}$ and $SP_{1,4}$ in the subgraph G^1_3 (see Figure 3). It is easy to see that $SP_{1,3} =< (1, 2), (2, 3) >$ and $SP_{1,4} =< (1, 2), (2, 3), (3, 4) >$. Thus, $P_{s,3} = P_{s,1} \oplus$

Algorithm 1 Greedy

$c(P_{s,i}) \leftarrow \infty$
$P_{s,i} \leftarrow NULL$
for all $v \in T_2$ **do**
    $(P_{s,v}, c(P_{s,v})) \leftarrow Dijkstra(G_2, s, v, c)$
for $k \leftarrow 2$ **to** $N-1$ **do**
    for all $u \in T_k, v \in T_{k+1}$ **do**
        $(SP_{u,v}, c(SP_{u,v})) \leftarrow Dijkstra(G^u_{k+1}, u, v, c)$
        if $c(P_{s,u}) + c(SP_{u,v}) < c(P_{s,v})$ **then**
            $P_{s,v} \leftarrow P_{s,u} \oplus SP_{u,v}$
            $c(P_{s,v}) \leftarrow c(P_{s,u}) + c(SP_{u,v})$
return $P_{s,d}$

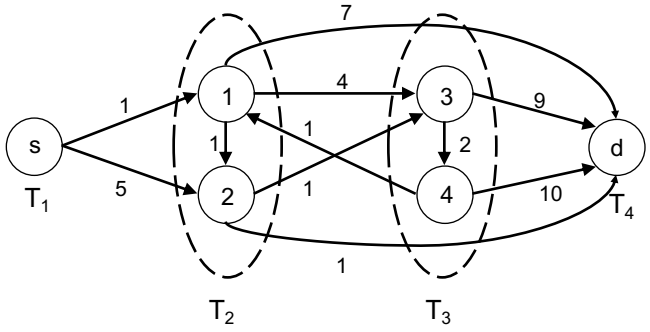

Figure 2: A graph $G = (V, A)$, where $|V| = 6$, $T_1 = \{s\}$, $T_2 = \{1, 2\}$, $T_3 = \{3, 4\}$, and $T_4 = \{d\}$.

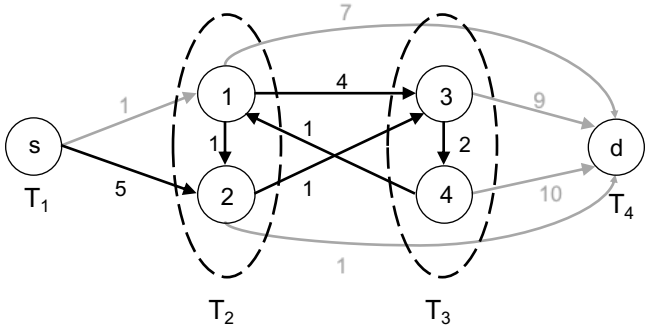

Figure 3: The subgraph G^1_3.

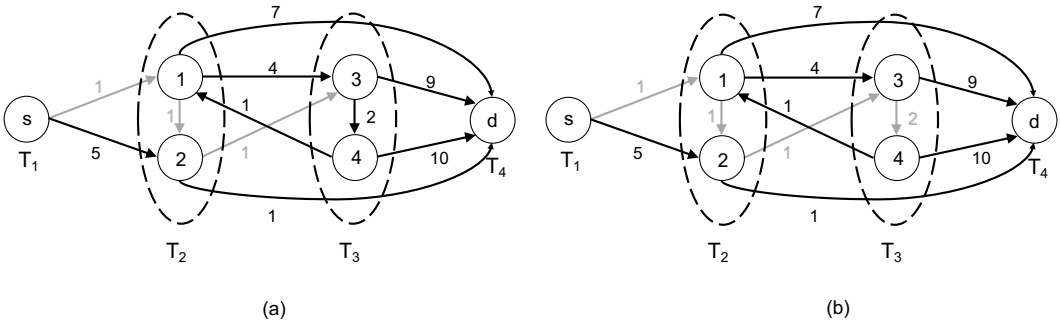

Figure 4: (a) The subgraph G_4^3. (b) The subgraph G_4^4.

$SP_{1,3} =< (s,1),(1,2),(2,3) >$, and $P_{s,4} = P_{s,1} \oplus SP_{1,4} =< (s,1),(1,2),(2,3),(3,4) >$. The next step consists in computing $SP_{2,3}$ and $SP_{2,4}$ in the subgraph G_3^2: if $c(P_{s,2}) + c(SP_{2,3}) < c(P_{s,3})$ then $P_{s,3} = P_{s,2} \oplus SP_{2,3}$, and if $c(P_{s,2}) + c(SP_{2,4}) < c(P_{s,4})$ then $P_{s,4} = P_{s,2} \oplus P_{s,4}$. It is easy to see that, for the given graph, we do not have to update neither $P_{s,3}$ nor $P_{s,4}$. Finally, to obtain $P_{s,d}$, we compute $SP_{3,d}$ in the subgraph G_4^3 (see Figure 4(a), and $SP_{4,d}$ in G_4^4 (see Figure 4(b)). We have that $SP_{3,d} =< (3,d) >$, and $SP_{4,d} =< (4,1),(1,d) >$. Therefore, since $12 = c(P_{s,3}) + c(SP_{3,d}) < c(P_{s,4}) + c(SP_{4,d}) = 13$, we have that Algorithm 1 returns the path $P_{s,d} =< (s,1),(1,2),(2,3),(3,d) >$, where $c(P_{s,d}) = 12$ and $l(P_{s,d}) = 4$. Let us note that this is not the optimal solution of CFSPTP, since the path $P_{s,d}^* =< (s,2),(2,3),(3,4),(4,1),(1,2),(2,d) >$ is a CFPT from s to d, and $c(P_{s,d}^*) = 11$. Given the feasible solution $P_{s,d}$ found by Algorithm 1, by definition the cost of an optimal solution is lower than or equal to $c(P_{s,d})$. As a consequence, the maximum number of arcs of the graph, whose sum of the costs does not exceed $c(P_{s,d})$, is the maximum length of any optimal solution. Hence, to obtain a tighter upper bound t_{max} for t, we sort the arcs in a non-decreasing order of their cost, and we count the number of arcs that can be selected from this ordered list, until the sum of their costs does not exceed $c(P_{s,d})$. Using this procedure for the graph of the previous example, we have that $t_{max} = 7$, and then the number of variables passes from 121 to 77. Let us note that t_{max} is the optimal solution of a Knapsack problem, where the number of items is equal to the number of arcs of the graph, every item has value 1, and the weight of an item is the cost of the corresponding arc. Since this is a particular case of the Knapsack problem, where each item has the same value, the procedure we used to determine t_{max} returns the optimal solution.

4 GRASP

GRASP is an iterative multistart meta-heuristic for difficult combinatorial optimization problems and has been introduced in Feo and Resende (1989). It has been applied to a large set of problems, like graph planarization (Resende and Ribeiro, 1997), antibandwidth problem (Duarte et al., 2010), and scheduling (González-Neira et al., 2017). For an extensive survey the reader is referred to Festa and Resende (2002, 2009a,b).

Each GRASP iteration is characterized by two main phases: a construction phase and a local search phase. The construction phase iteratively adds one element at a time until it obtains a complete feasible solution. Each 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. The pseudocode of the GRASP heuristic is shown in Algorithm 2.

Algorithm 2 Pseudocode of a generic GRASP.

```
1: procedure GRASP(MaxIterations)
2:     for i = 1, ..., MaxIterations do
3:         Build a greedy randomized solution x
4:         x ← LocalSearch(x)
5:         if i = 1 then
6:             x* ← x
7:         else if x is better than x* then
8:             x* ← x
9:     return x*
```

4.1 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 CFSPTP, the

construction phase is iterative and is described in Algorithm 3. 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 each iteration of the construction phase, all the candidate paths (i.e. those that can be added to the solution) are ordered in a candidate list CL, with respect to the length of the candidate paths, the better ones are inserted in the RCL. The paths' length is computed by DijkstraVariant function, that applies Dijkstra's algorithm taking into account both that arcs are traversed more than once and the clusters that have been already visited.

The probabilistic component is characterized by randomly choosing one of the candidates in the RCL, but not necessarily the top candidate.

The pseudocode of the construction phase is reported in Algorithm 3. Firstly, an index i is selected at random in $[2, N-1]$. 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.

Two key components of the algorithm are $n \times n$ matrix $K = (k_{ij})$, and the DijkstraVariant procedure. k_{ij} represents the number of times arc (i, j) has been involved in the partial solution P_T. The procedure DijkstraVariant takes as input both K and an integer t that represents that the last visited set is T_t. Differently from the standard Dijkstra algorithm, when an arc (i, j) must be relaxed, its cost is considered as $+\infty$ if either $k_{ij} > 0$ or $j \in T_q$, with $q > t+1$. In this way, it is not possible wither to traverse arcs already traversed, or visit forbidden sets. Procedures Increase and Decrease update K to reflect the choices made by the construction algorithm.

4.2 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.

The pseudocode of the local search procedure we have designed for the CFSPTP is reported in Algorithm 4. It takes as input the GRASP path

Algorithm 3 Construction of a Greedy Randomized Solution

1: **function** CONSTRUCTION($V, A, s, d, N, \{T_i\}_{i=1,\ldots,N}, C, K, \alpha$)
2: $i \leftarrow$ RAND$(2, N-2)$; $P_T \leftarrow$ NIL; $CL \leftarrow \emptyset$
3: **for all** $v \in T_i$ **do**
4: **for all** $w \in T_{i+1}$ **do**
5: $(l, P) \leftarrow$ DIJKSTRAVARIANT(V, A, v, w, C, K, i)
6: $CL \leftarrow CL \cup \{ (l, P, v, w) \}$
7: $RCL \leftarrow$ MAKERCL(α); $(l, P, v, w) \leftarrow$ SELECT(RCL)
8: $P_i \leftarrow P$; $P_T \leftarrow P_T \oplus P_i$
9: $s_i \leftarrow v$; $d_i \leftarrow w$
10: INCREASE(P_i, w, K)
11: $j \leftarrow i+2$; $i \leftarrow i-1$
12: **while** $(i > 0) \vee (j < N)$ **do**
13: ▷ Look for a candidate path from a node in T_i to node s_{i+1}
14: $CL \leftarrow \emptyset$
15: **if** $i > 1$ **then**
16: **for all** $v \in T_i$ **do**
17: $(l, P) \leftarrow$ DIJKSTRAVARIANT$(V, A, v, s_{i+1}, C, K, i)$
18: $CL \leftarrow CL \cup \{ (l, P, v, s_{i+1}) \}$
19: **else if** $i = 1$ **then**
20: $(l, P) \leftarrow$ DIJKSTRAVARIANT(V, A, s, s_2, C, K, i)
21: $CL \leftarrow CL \cup \{ (l, P, s, s_2) \}$
22: $RCL \leftarrow$ MAKERCL(α); $(l, P, v, w) \leftarrow$ SELECT(RCL)
23: $P_i \leftarrow P$; $P_T \leftarrow P_i \oplus P_T$
24: $s_i \leftarrow v$; $d_i \leftarrow w$
25: INCREASE(P_i, w, K)
26: ▷ Look for a candidate path from node d_{j-1} to a node in T_j
27: $CL \leftarrow \emptyset$
28: **if** $j < N$ **then**
29: **for all** $v \in T_j$ **do**
30: $(l, P) \leftarrow$ DIJKSTRAVARIANT$(V, A, d_{j-1}, v, C, K, j-1)$
31: $CL \leftarrow CL \cup \{ (l, p, d_{j-1}, v) \}$
32: **else if** $j = N$ **then**
33: $(l, P) \leftarrow$ DIJKSTRAVARIANT$(V, A, d_{j-1}, d, C, K, j-1)$
34: $CL \leftarrow CL \cup \{ (l, P, d_{j-1}, d) \}$
35: $RCL \leftarrow$ MAKERCL(α); $(l, P, v, w) \leftarrow$ SELECT(RCL)
36: $P_j \leftarrow P$; $P_T \leftarrow P_T \oplus P_j$
37: $s_j \leftarrow v$; $d_j \leftarrow w$
38: INCREASE(P_j, w, K)
39: $j \leftarrow j+1$; $i \leftarrow i-1$
40: **return** P_T

Algorithm 4 Local Search

```
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, i − 1)
11:                (l″, P″) ← DijkstraVariant(V, A, v, d_i, C, K, i)
12:                if l′ + l″ < min then
13:                    min ← l′ + l″
14:                    P_{i−1} ← P′
15:                    P_i ← P″
16:                    flag ← True
17:            Increase(P_{i−1}, d_{i−1}, K)
18:            Increase(P_i, d_i, K)
19:    return P_T
```

tour P_T and outputs a local optimal feasible tour. The loop **while** (lines 3–18) 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 9–16).

The neighborhood is explored with a best improvement strategy.

5 Computational Results

In this section, we describe the results of the mathematical model and the GRASP algorithm obtained during our computational test phase. The computational tests phase has three main purposes: the validation of the effectiveness of the reduction procedure, the validation of the effectiveness of the GRASP heuristic by comparing its solutions with the optimal ones, where available, and, finally, the evaluation of the GRASP performance on the large instances. GRASP was coded in C++ and ran on a machine with an Intel Core i5-6400 2.70GHz x 4 processor, 8GB RAM, under the Linux (Ubuntu 18.04) operating system. The mathematical formulation was coded in C++ using the Concert library of IBM ILOG CPLEX 12.8. A time limit of 3600 seconds has been fixed.

5.1 Generation of Instances

As no benchmark instances are available in the literature for the CFSPTP, we have generated two sets of instances: *small* and *large*. Small instances have been generated according to the following parameters: $n \in \{50, 60, 70, 80, 90, 100\}$, $m = d \times n \times (n-1)$, where the density $d \in \{0.2, 0.4, 0.6, 0.8\}$, and $N = \lfloor \beta\, n \rfloor$, where $\beta \in \{0.10, 0.15, 0.20, 0.25\}$, for a total of 96 instances. For the large instances, $n \in \{200, 350, 500\}$, while the density d and the number of clusters N have been chosen as in the generation of the small instances.

The generation of the instances is carried out according to the following steps:

1. The source and the destination nodes are assigned to T_1 and T_N, respectively, while the remaining nodes are randomly assigned to the clusters $T_2, \ldots, T_{N-1}$, assuring that each cluster contains at least one node.

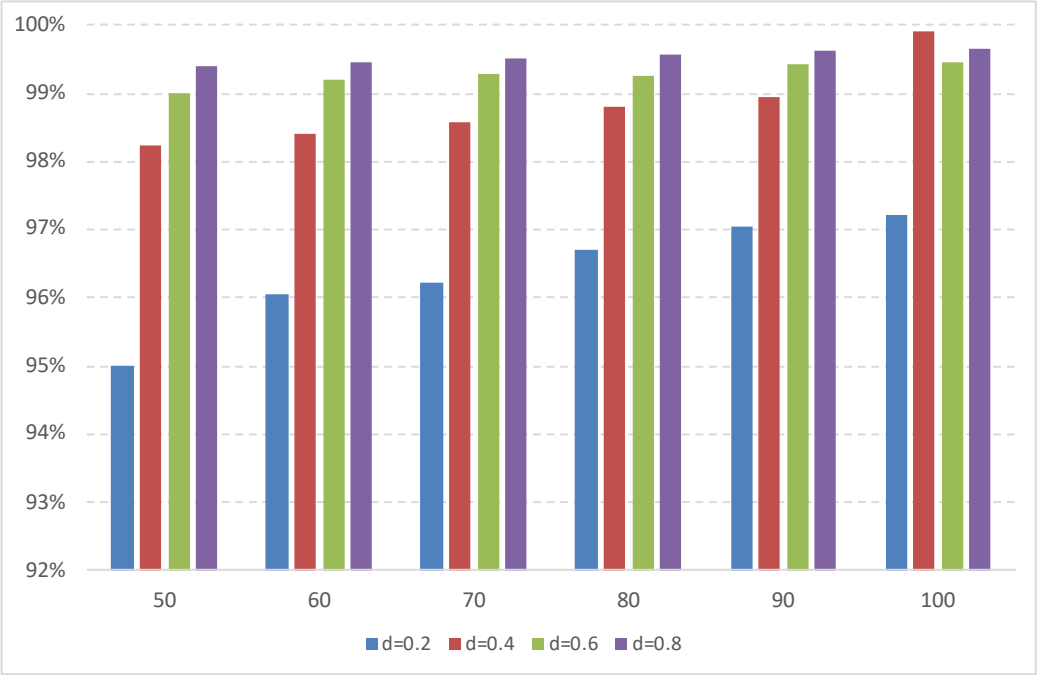

Figure 5: Percentage of the reduction of the number of variables for Small instances, in relation to the number of nodes and the density.

2. The generation of the m arcs is carried out randomly but ensuring that a feasible solution always exists.

3. The cost of each arc is randomly chosen in the interval $[10, 50]$.

5.2 Effectiveness of the reduction procedure

In this section, we verify the effectiveness of the reduction procedure introduced in Section 3.1, by checking the number of variables that it removes from the ILP model. We remind here that the number of variables x_{ij}^t in the model is equal to m^2 while with the application of the reduction procedure this value decreases to $m \times t_{max}$. The results of this procedure are shown in the bar charts in Figure 5 and Figure 6.

Fixed a size n and a density d, a bar in Figure 5 shows the percentage reduction of the number of variables. The chart concerns only small instances,

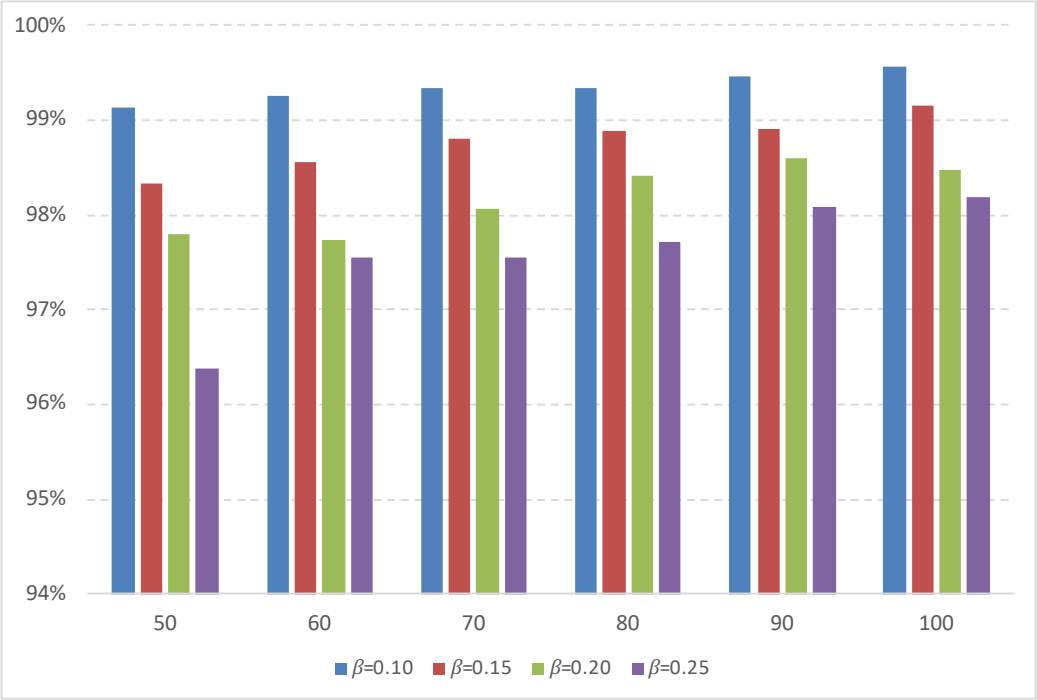

Figure 6: Percentage of the reduction of the number of variables for Small instances, in relation to the number of nodes and the parameter β.

where the model runs. The percentage of reduction obtained is impressive because it is always higher than 94%. More in detail, we observe that, on instances with $d = 0.2$, as the size n increases, the percentage increases from 95% to 97.2%. For the other densities, the percentage increases very slowly, but it is always higher than 98%.

A bar in Figure 6 represents the percentage reduction of the number of variables with respect the number of nodes and β. As β increases, namely as the number of clusters increases, the percentage reduction decreases. The reduction of the variables heavily impacts on the performance of the model. Indeed, without the reduction procedure, even the smallest instances with 50 nodes and density equal to 0.8, were not optimally solved within the time limit. For higher size, even the construction of the matrix of variables, is really expensive from a computational point of view. Indeed, for $n = 100$ and $d = 0.8$, this matrix contains more than 62 million of variables and

n	ImprovLP	RedTime
50	4.08%	18.51%
60	5.04%	33.64%
70	5.98%	30.03%
80	6.97%	28.91%
90	8.35%	16.24%
100	9.00%	23.67%

Table 1: Percentage improvement of the value of the LP relaxation, and improvement percentage of the computational time gained by the introduction of constraints (4) in the mathematical formulation.

CPLEX either reaches the time limit, before the declaration of the matrix is completed, or it runs of memory.

5.3 Small instances

In this section, we show the computational results over the set of Small instances. First of all, we want to highlight the impact of constraints (4) on the performances of the mathematical formulation. Let LP and $LP(4)$ be the values of the LP relaxation of the mathematical formulation, without constraints (4) and with constraints (4), respectively. Furthermore, we denote by $Time$ and $Time(4)$ the computational times of CPLEX, without constraints (4) and with constraints (4), respectively. For any instance, we computed the improvement obtained with the introduction of constraints (4) in terms of the value of the LP relaxation, according to the formula $(LP(4) - LP)/LP$, and the reduction of the computational time as $(Time - Time(4))/Time$. In Table 1 we reported the number of nodes (n), the average of the percentage of improvement of the value of the LP relaxation ($ImprovLP$), and the average of the percentage of reduction of the computational time ($RedTime$). We can see that, thanks to the introduction of constraints (4) in the model, we obtain lower bounds which are at least 4.08% better, and the computational time of CPLEX reduces by at least 16.24%. Finally, the introduction of constraints (4) allowed us to optimally solve within the time limit three additional instances: two instances having $n = 70$, and one with $n = 100$.
The computational tests carried out on the *Small instances* aim to verify the quality of the solutions found by GRASP, by comparing these solutions with

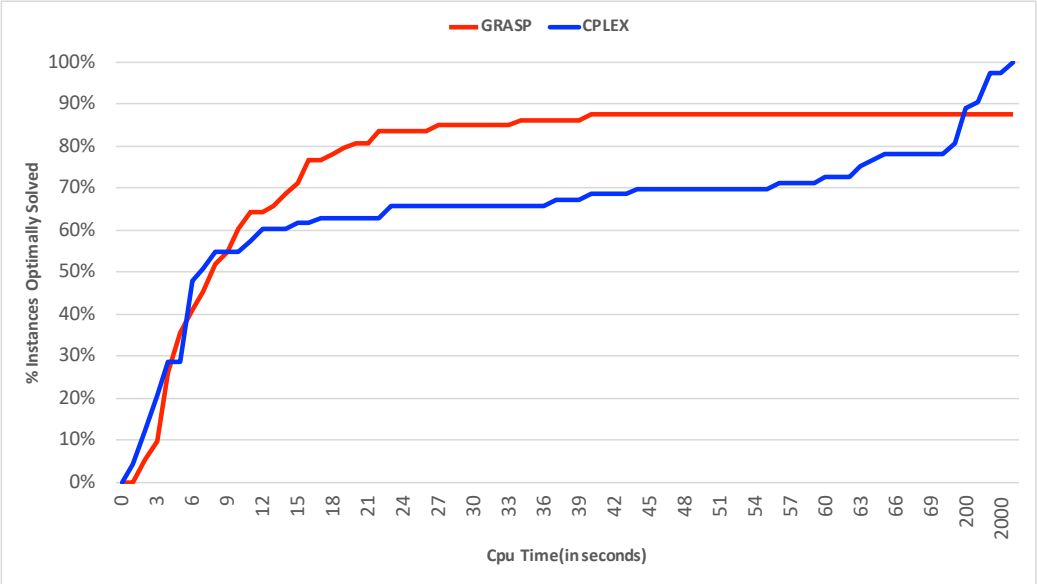

Figure 7: Percentage of optimal solutions found by GRASP (red) and CPLEX (blue), within the computational time reported on the x-axis.

the optimal ones, in case the latter are available. The mathematical formulation provides the optimal solution for 73 instances over 96, while for the remaining 21 instances an upper bound of the optimal solution value is given. For what concerns the GRASP metaheuristic, it finds the optimal solution on 64 out of 73 instances. The detailed results are reported in Tables 3 and 4 (Appendix A).

The results of these tables are summarized in Figure 7 The horizontal axis represents the CPU time, in seconds, and the vertical axis represents the percentage of instances optimally solved within a fixed CPU time. In other words, given a CPU time α, we compute the percentage of optimally solved instances within α seconds. For instance, in 10 seconds GRASP finds the optimal solution for the 55% of the instances for which the optimal solution is known. This means that the faster the growth, the better the performance of the algorithm. It is evident from this figure that the GRASP metaheuristic is very effective and fast, because it finds the optimal solution in less than 40 seconds for the 88% of the instances for which the optimal solution is known, while CPLEX takes 200 seconds to reach the same percentage.

5.4 Large instances

In this section, we focused on the performance of GRASP over the set of *Large instances*. For these instances CPLEX was not even able to find a feasible solution within the time limit. Therefore, the lower bound we used to evaluate the quality of the solutions of the GRASP metaheuristic, is the optimal solution of the FSPTP on the set of Large instances. To solve the FSPTP we used the polynomial time algorithm proposed by Carrabs et al. (2017).

Results are shown in Table 2. The first three columns show the characteristics of the instances: the number of nodes n, of arcs m and the number of clusters N, respectively. Column LB reports the optimal solution value for the FSPTP, that is a lower bound for the optimal solution of CFSPTP. The next two columns report the solution value (Obj) and the computational time ($Time$), in seconds. Finally, the last column (GAP) reports the percentage gap between the solution found by GRASP and the lower bound.
In all but six cases, Gap is lower than 5%, and it is equal to 8.78%, in the worst case. For two instances with $n = 200$, the optimal solution of the FSPTP is equal to the solution found by GRASP, therefore they are optimal solutions of the CFSPTP. GRASP provides a solution in less than 90 seconds for all the instances with 200 nodes. From the results of column Time, it is evident that the performance of the algorithm is affected by both the density of the graph and, in particular, by the number of clusters. For example, on the instances with 200 nodes and 31840 arcs, the computational time of GRASP ranges from 40 seconds, when $N = 20$, to 61 seconds, when $N = 50$.

The outlier results for the instance with $n = 200, m = 31840, N = 50$ 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. Indeed, small sets T_i implies that a small number of paths must be found during the construction, and there exist few neighbors for the local search. In particular, this instance presents several sets T_i for which $|T_i| < 3$.

On the instances with 350 nodes, GRASP requires up to 300 seconds. Even in these instances, the trend about the density and the number of clusters is confirmed. In particular, the gap time required by GRASP on the instances with the same number of nodes and arcs but a different number of clusters increases. For instance, on the instances with $n = 350$ and $m = 97720$, the time ranges from 142 seconds, when $N = 35$, to 299 seconds (the double), when $N = 87$. Finally, the results on the largest instances with 500

n	m	N	LB	GRASP		GAP
				Obj	Time	
200	7960	20	337	351	12.53	3.99%
	7960	30	574	591	16.23	2.88%
	7960	40	780	785	18.13	0.64%
	7960	50	1135	1142	21.15	0.61%
	15920	20	272	277	22.22	1.81%
	15920	30	516	529	28.87	2.46%
	15920	40	700	718	32.84	2.51%
	15920	50	915	915	39.46	0.00%
	23880	20	270	281	32.53	3.91%
	23880	30	437	449	42.74	2.67%
	23880	40	662	673	57.62	1.63%
	23880	50	846	860	85.49	1.63%
	31840	20	247	247	40.74	0.00%
	31840	30	377	391	50.84	3.58%
	31840	40	566	581	77.67	2.58%
	31840	50	779	782	61.18	0.38%
350	24430	35	592	649	39.82	8.78%
	24430	52	1027	1062	56.33	3.30%
	24430	70	1470	1483	71.91	0.88%
	24430	87	2000	2011	78.13	0.55%
	48860	35	482	514	71.95	6.23%
	48860	52	803	827	99.6	2.90%
	48860	70	1164	1214	122.35	4.12%
	48860	87	1626	1667	149.64	2.46%
	73290	35	440	467	97.37	5.78%
	73290	52	720	744	145.17	3.23%
	73290	70	1079	1095	176.26	1.46%
	73290	87	1459	1492	220.08	2.21%
	97720	35	425	457	142.84	7.00%
	97720	52	698	729	197.63	4.25%
	97720	70	1019	1050	254.97	2.95%
	97720	87	1372	1394	299.36	1.58%
500	49900	50	856	917	162.95	6.65%
	49900	75	1471	1518	213.25	3.10%
	49900	100	2024	2072	269.49	2.32%
	49900	125	2840	2916	299.77	2.61%
	99800	50	712	747	292.16	4.69%
	99800	75	1176	1226	431.68	4.08%
	99800	100	1741	1804	515.38	3.49%
	99800	125	2327	2366	618.4	1.65%
	149700	50	624	656	462.92	4.88%
	149700	75	1089	1148	640.52	5.14%
	149700	100	1527	1586	697.14	3.72%
	149700	125	2123	2164	877.23	1.89%
	199600	50	614	659	584.07	6.83%
	199600	75	979	1024	813.54	4.39%
	199600	100	1486	1545	1029.7	3.82%
	199600	125	1973	2011	1114.52	1.89%

Table 2: Computational results of GRASP on the large instances with up to 500 nodes.

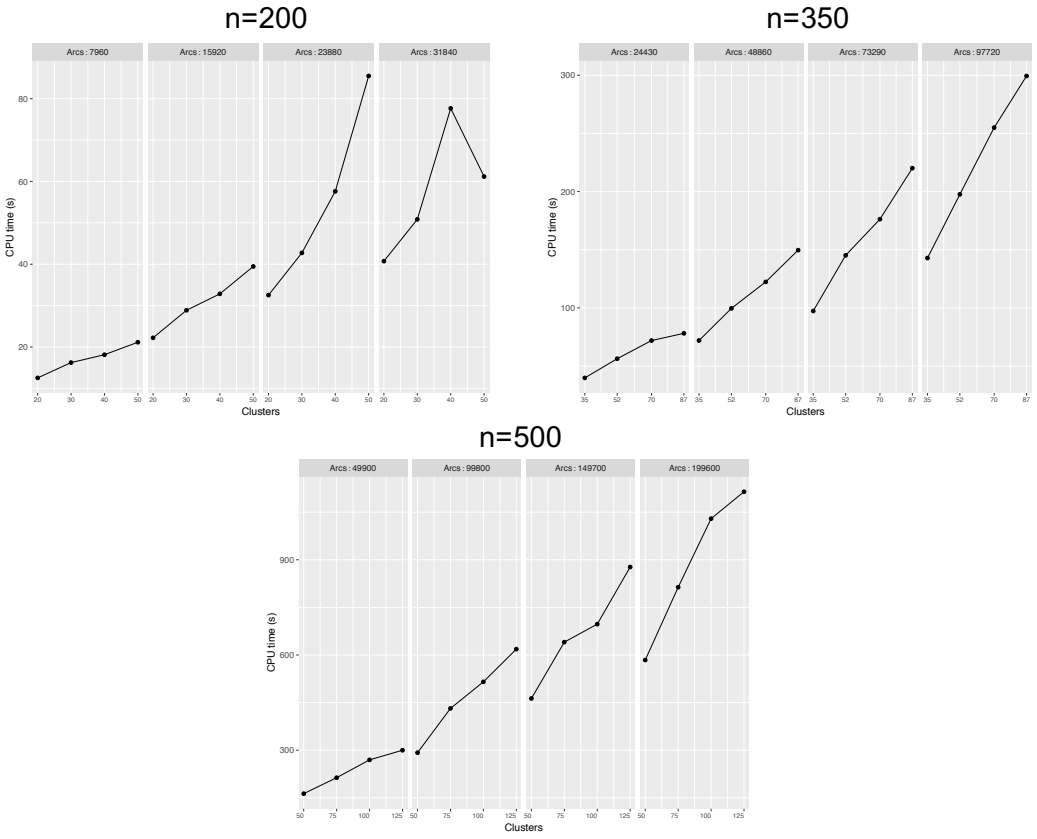

Figure 8: Trend of Cpu Time required by GRASP with respect the number of clusters, for instances with 200, 350 and 500 nodes.

nodes confirm the previous trend. Since, in the worst case, GRASP requires less than 1115 seconds, we can conclude that the algorithm remains fast even on the largest instances with maximum density, and with maximum number of clusters.

The results of Table 2 are graphically summarized in Figure 8. Here it is shown the trend of the computational time required by GRASP, with respect the number of clusters, of instances with 200, 350 and 500 clusters, for each value of density.

6 Conclusions

We introduced the Constrained Forward Shortest Path Tour Problem, a variant of the Forward Shortest Path Tour Problem. We proposed an integer linear programming formulation for the CFSPTP. Since the number of variables involved in the model is the square of m, we devised a reduction procedure, which removes at least 94% of the variables for the set of Small instances. Since the model is exploitable only for instances with up to 100 nodes, then we designed a GRASP metaheuristic to run on the large instances. The computational results showed that GRASP is very effective because it finds the optimal solution for the 88% of the instances which were optimally solved by CPLEX. On the Large instances, its computational time never exceeds 1115 seconds. Furthermore, we compared the solutions found by GRASP on the set of Large instances, with the optimal solutions of the Forward Shortest Path Tour Problem, which is a lower bound for the optimal solution of the CFSPTP. We found that the percentage gaps between the solutions returned by GRASP and these lower bounds is often less than 5%. Future directions will be focused on the development of exact approaches for the CFSPTP, such as a Branch and Cut algorithm.

References

C. P. Bajaj. Some constrained shortest-route problems. *Unternehmensforschung Operations Research - Recherche Opérationnelle*, 15 (1):287–301, 1971.

A. Bertossi. The edge Hamiltonian path problem is NP-complete. *Information Processing Letters*, 13(4-5):157–159, 1981.

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

L. Bianco, M. Caramia, S. Giordani, and V. Piccialli. Operations research models for global route planning in hazardous material transportation. *International Series in Operations Research and Management Science*, 193: 49–101, 2013.

F. Carrabs, R. Cerulli, and M. Speranza. A branch-and-bound algorithm for

the double travelling salesman problem with two stacks. *Networks*, 61(1): 58–75, 2013.

F. Carrabs, R. Cerulli, P. Festa, and L. F. On the Forward Shortest Path Tour Problem. In *Optimization and Decision Science: Methodologies and Applications*, volume 217, pages 529–537. Springer Proceedings in Mathematics & Statistics, 2017.

F. Carrabs, R. Cerulli, R. Pentangelo, and A. Raiconi. A two-level metaheuristic for the all colors shortest path problem. *Computational Optimization and Applications*, 71(2):525–551, 2018.

F. Carrabs, R. Cerulli, and A. Raiconi. Solving the all-colors shortest path problem as an equality generalized tsp problem. Technical report, Submitted to RAIRO - Operations Research (second revision), 2020.

R. C. de Andrade and R. D. Saraiva. An integer linear programming model for the constrained shortest path tour problem. *Electronic Notes in Discrete Mathematics*, 69:141–148, aug 2018.

L. Di Puglia Pugliese, D. Ferone, P. Festa, and F. Guerriero. Shortest path tour problem with time windows. *European Journal of Operational Research*, 282(1):334–344, 2020.

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

A. Duarte, R. Martí, M. G. C. Resende, and R. M. A. Silva. GRASP with path relinking heuristics for the antibandwidth problem. *Networks*, 58(3): 171–189, dec 2010.

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

D. Ferone, P. Festa, F. Guerriero, and D. Laganà. The constrained shortest path tour problem. *Computers and Operations Research*, 74:64–77, 2016.

D. Ferone, P. Festa, and F. Guerriero. An efficient exact approach for the constrained shortest path tour problem. *Optimization Methods and Software*, pages 1–20, jan 2019.

P. Festa. Complexity analysis and optimization of the shortest path tour problem. *Optimization Letters*, 6(1):163–175, nov 2010.

P. Festa and M. G. C. Resende. Grasp: An annotated bibliography. In *Essays and surveys in metaheuristics*, pages 325–367. Springer, 2002.

P. Festa and M. G. C. Resende. An annotated bibliography of grasp - part i: Algorithms. *International Transactions in Operational Research*, 16(1): 1–24, 2009a.

P. Festa and M. G. C. Resende. An annotated bibliography of grasp - part ii: Applications. *International Transactions in Operational Research*, 16 (2):131–172, 2009b.

P. Festa, F. Guerriero, D. Laganà, and R. Musmanno. Solving the shortest path tour problem. *European Journal of Operational Research*, 230(3): 464–474, nov 2013.

E. M. González-Neira, D. Ferone, S. Hatami, and A. A. Juan. A biased-randomized simheuristic for the distributed assembly permutation flow-shop problem with stochastic processing times. *Simulation Modelling Practice and Theory*, 79(Supplement C):23 – 36, 2017.

J. Michallet, C. Prins, L. Amodeo, F. Yalaoui, and G. Vitry. Multi-start iterated local search for the periodic vehicle routing problem with time windows and time spread constraints on services. *Computers and Operations Research*, 41(1):196–207, 2014.

M. G. C. Resende and C. C. Ribeiro. A GRASP for graph planarization. *Networks*, 29(3):173–189, may 1997.

R. Wolfler Calvo and R. Cordone. A heuristic approach to the overnight security service problem. *Computers and Operations Research*, 30(9):1269–1287, 2003.

Appendix A Detailed Computational Results

In Tables 3 and 4 the detailed results on the small instances of CPLEX and GRASP are reported. The first four columns show the characteristics of the instances: the number of nodes n, of arcs m, and the number of clusters N, and t_{max} value computed by the reduction procedure, respectively. The next three columns report the best objective function value returned by CPLEX (Opt), together with the corresponding computation time ($Time$) and the optimality gap (Gap), respectively. We use the symbol $-$, if no feasible solution has been found within the time limit. Finally, the last three columns report the solution value (Obj) and the computational time ($Time$) in seconds of GRASP, and the relative error in percentage (Er), computed as $Er = \frac{Obj-Opt}{Opt}$. We highlight in bold the Obj value of the 64 out of 96 instances which are optimally solved by GRASP. Furthermore, GRASP provides an upper bound which is better than the one returned by CPLEX for 21 out of the 32 remaining instances. Whenever this happens the solution value of the GRASP is marked with a $*$ symbol and the corresponding relative error is negative. In more detail, the smaller the relative error, the better the upper bound found by the GRASP.

n	m	N	t_{max}	CPLEX			GRASP		Er
				Opt	Time	Gap	Obj	Time	
50	490	5	10	90	0.57	0.00%	**90**	1.48	0.00%
	490	7	21	126	1.68	0.00%	**126**	1.65	0.00%
	490	10	24	243	3.36	0.00%	**243**	1.49	0.00%
	490	12	43	348	5.59	0.00%	**348**	1.71	0.00%
	980	5	9	58	0.91	0.00%	**58**	3.44	0.00%
	980	7	12	113	1.20	0.00%	**113**	2.70	0.00%
	980	10	18	151	2.60	0.00%	**151**	2.50	0.00%
	980	12	30	210	5.24	0.00%	**210**	3.19	0.00%
	1470	5	4	49	0.15	0.00%	**49**	3.23	0.00%
	1470	7	10	92	1.42	0.00%	**92**	3.22	0.00%
	1470	10	19	136	2.92	0.00%	**136**	3.87	0.00%
	1470	12	25	182	7.65	0.00%	**182**	3.83	0.00%
	1960	5	5	49	2.05	0.00%	**49**	3.25	0.00%
	1960	7	9	90	2.18	0.00%	**90**	3.82	0.00%
	1960	10	16	147	6.43	0.00%	**147**	4.61	0.00%
	1960	12	18	176	5.82	0.00%	**176**	4.29	0.00%
60	708	6	13	85	1.17	0.00%	**85**	3.06	0.00%
	708	9	26	211	5.01	0.00%	**211**	2.96	0.00%
	708	12	40	205	5.10	0.00%	**205**	3.27	0.00%
	708	15	33	362	55.64	0.00%	**362**	4.20	0.00%
	1416	6	9	76	1.39	0.00%	**76**	3.42	0.00%
	1416	9	16	145	3.27	0.00%	**145**	3.87	0.00%
	1416	12	24	233	76.51	0.00%	**233**	4.57	0.00%
	1416	15	42	278	436.38	0.00%	**278**	6.02	0.00%
	2124	6	7	70	2.10	0.00%	**70**	5.15	0.00%
	2124	9	12	109	2.82	0.00%	**109**	6.08	0.00%
	2124	12	22	208	64.03	0.00%	**208**	5.69	0.00%
	2124	15	28	255	188.37	0.00%	**255**	7.25	0.00%
	2832	6	6	63	5.40	0.00%	**63**	6.28	0.00%
	2832	9	11	108	3.15	0.00%	**108**	7.38	0.00%
	2832	12	19	175	11.66	0.00%	**175**	9.02	0.00%
	2832	15	25	226	71.68	0.00%	**226**	9.57	0.00%
70	966	7	14	105	1.62	0.00%	**105**	5.10	0.00%
	966	10	27	182	5.45	0.00%	**182**	4.79	0.00%
	966	14	42	304	156.24	0.00%	**304**	4.90	0.00%
	966	17	63	473	3610	14.40%	472*	4.96	-0.21%
	1932	7	14	90	3.43	0.00%	**90**	5.88	0.00%
	1932	10	21	161	43.05	0.00%	**161**	7.79	0.00%
	1932	14	34	257	36.46	0.00%	**257**	8.02	0.00%
	1932	17	42	349	3610	16.10%	340*	8.07	-2.58%
	2898	7	9	77	3.14	0.00%	**77**	9.40	0.00%
	2898	10	17	144	10.43	0.00%	**144**	8.85	0.00%
	2898	14	27	231	510.80	0.00%	236	11.26	2.16%
	2898	17	31	271	234.11	0.00%	**271**	10.36	0.00%
	3864	7	7	66	5.55	0.00%	**66**	10.44	0.00%
	3864	10	13	118	5.88	0.00%	119	10.80	0.85%
	3864	14	28	206	510.57	0.00%	**206**	15.23	0.00%
	3864	17	27	254	545.99	0.00%	**254**	13.96	0.00%

Table 3: Computational results of GRASP, on small instances with $n = 50, 60, 70$.

n	m	N	t_{max}	CPLEX			GRASP		Er
				Opt	Time	Gap	Obj	Time	
80	1264	8	21	148	5.23	0.00%	**148**	4.75	0.00%
	1264	12	32	223	10.53	0.00%	**223**	7.40	0.00%
	1264	16	45	341	472.24	0.00%	**341**	7.85	0.00%
	1264	20	68	565	3610	17.80%	541*	9.30	-4.25%
	2528	8	12	106	3.35	0.00%	108	7.76	1.89%
	2528	12	23	188	39.51	0.00%	**188**	12.42	0.00%
	2528	16	38	316	3610	8.61%	316*	16.08	0.00%
	2528	20	49	392	3610	6.70%	392*	13.82	0.00%
	3792	8	11	109	5.61	0.00%	**109**	14.00	0.00%
	3792	12	25	194	63.37	0.00%	**194**	15.76	0.00%
	3792	16	33	248	154.38	0.00%	**248**	18.50	0.00%
	3792	20	42	370	3610	19.76%	348*	24.31	-5.95%
	5056	8	10	92	5.73	0.00%	**92**	14.43	0.00%
	5056	12	17	140	11.59	0.00%	**140**	19.53	0.00%
	5056	16	22	213	62.33	0.00%	**213**	21.23	0.00%
	5056	20	38	329	3610	7.40%	329*	23.37	0.00%
90	1602	9	21	188	16.80	0.00%	191	9.07	1.60%
	1602	13	48	282	3610	4.24%	282*	9.31	0.00%
	1602	18	53	439	3610	16.15%	432*	9.73	-1.59%
	1602	22	67	654	3610	22.66%	607*	13.71	-7.19%
	3204	9	14	127	5.54	0.00%	**127**	13.37	0.00%
	3204	13	21	201	138.84	0.00%	**201**	15.77	0.00%
	3204	18	37	352	3610	9.40%	352*	17.46	0.00%
	3204	22	63	499	3610	24.03%	459*	21.49	-8.02%
	4806	9	12	118	7.55	0.00%	**118**	17.85	0.00%
	4806	13	22	190	62.86	0.00%	**190**	21.52	0.00%
	4806	18	31	282	3610	11.75%	285	26.08	1.06%
	4806	22	43	406	3610	21.37%	383*	29.42	-5.67%
	6408	9	11	101	7.97	0.00%	104	19.64	2.97%
	6408	13	17	165	22.27	0.00%	167	24.31	1.21%
	6408	18	30	251	2005.91	0.00%	**251**	33.52	0.00%
	6408	22	40	409	3610	20.46%	387*	32.61	-5.38%
100	1980	10	19	150	5.03	0.00%	**150**	9.15	0.00%
	1980	15	42	300	2779.07	0.00%	**300**	10.81	0.00%
	1980	20	77	507	3610	30.34%	429*	14.03	-15.38%
	1980	25	85		-		625	15.60	-
	3960	10	13	135	6.55	0.00%	**135**	15.67	0.00%
	3960	15	22	207	59.05	0.00%	212	27.14	2.42%
	3960	20	42	383	3610	23.79%	360*	24.07	-6.01%
	3960	25	63	567	3610	30.21%	492*	33.05	-13.23%
	5940	10	16	130	22.62	0.00%	**130**	26.60	0.00%
	5940	15	24	220	193.31	0.00%	226	28.48	2.73%
	5940	20	40	305	3610	2.92%	303*	32.80	-0.66%
	5940	25	50	519	3610	32.02%	458*	42.08	-11.75%
	7920	10	12	121	14.28	0.00%	126	30.08	4.13%
	7920	15	21	202	107.22	0.00%	**202**	39.86	0.00%
	7920	20	37	343	3610	11.74%	325*	46.49	-5.25%
	7920	25	43	451	3610	28.75%	385*	50.96	-14.63%

Table 4: Computational results of GRASP, on small instances with $n = 80, 90, 100$.