Noname manuscript No.
(will be inserted by the editor)

A two-level metaheuristic for the All Colors Shortest Path Problem

F. Carrabs · R. Cerulli · R. Pentangelo · A. Raiconi

Received: date / Accepted: date

Abstract Given an undirected weighted graph, in which each vertex is assigned to a color and one of them is identified as source, in the all-colors shortest path problem we look for a minimum cost shortest path that starts from the source and spans all different colors. The problem is known to be NP-Hard and hard to approximate. In this work we propose a variant of the problem in which the source is unspecified, and show the two problems to be computationally equivalent. Furthermore, we propose a compact representation for feasible solutions, as well as a mathematical formulation and a VNS metaheuristic that is based on it. Computational results show the effectiveness of the proposed approach.

Keywords Shortest Path · Colored Graph · Variable Neighboord Search

1 Introduction

The *All-Colors Shortest Path* (ACSP) is a combinatorial optimization problem, first introduced in [3]. The problem is defined on undirected graphs, in which a numerical attribute (weight) is associated to each edge, while a logical attribute (called *color*, or *label*) is given for each vertex. Therefore the different colors, appearing in the graph, partition the set of vertices into disjoint subsets. The aim of ACSP is to find the shortest, possibly non-simple path, that starts from a predefined source vertex and spans each color of the graph; that is, each path composing a feasible solution needs to visit at least a vertex belonging to each color. In the *All-Colors Shortest Path with Unconstrained Endpoints* (ACSP-UE) problem variant, the source vertex is not provided, and therefore the path can start from any vertex of the graph.

F. Carrabs, R. Cerulli, R. Pentangelo, A. Raiconi
Department of Mathematics, University of Salerno
E-mail: {fcarrabs,raffaele,rpentangelo,araiconi}@unisa.it

As mentioned, the optimal solution of these problems could correspond to a non-simple path, meaning that a vertex may be visited more than once. As an example, consider the graph illustrated in Figure 1. The value reported on each edge represents its cost (for instance, the weight of $\{v_1, v_2\}$ is 4), while the c_i label next to each vertex denotes its color. A (simple) path reaching every color is, for instance, $[v_1, v_4, v_2, v_3, v_6, v_5, v_7]$, whose cost is 14. Despite visiting twice vertices v_4 and v_5, and visiting once the additional vertex v_8, the optimal ACSP-UE solution is instead $[v_1, v_4, v_2, v_4, v_5, v_3, v_5, v_6, v_8, v_7]$. Indeed, in this case, the cost of the path is 10.

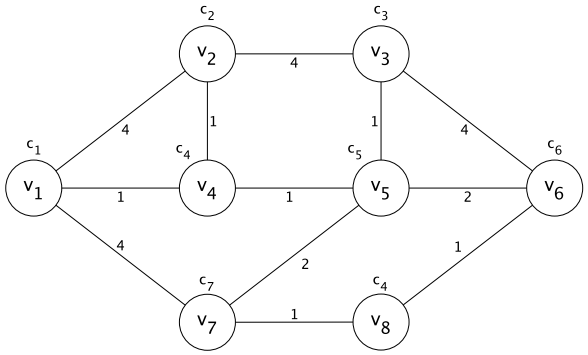


Fig. 1 Example graph with 8 vertices, 12 edges and 7 colors

The problems can find application in different contexts. For instance, in a road network for the distribution of goods, vertices associated with the same color can represent different locations (warehouses or stores) in which specific types of goods can be picked up or stocked. Applications related to mobile sensor roaming and path planning are cited in [3].

Among similar problems presented in the literature, we recall the Shortest Path Tour (SPTP), the Forward Shortest Path Tour (FSPTP), the Generalized Traveling Salesman (GTSP) and the Generalized Minimum Spanning Tree (GMST). The SPTP ([11]) is a polynomially solvable optimization problem in which, given a source vertex s and a destination vertex d, the aim is to find a shortest path from s to d that crosses in a given sequence at least a vertex for each different color. Any node of the graph can be crossed while going from a color of the sequence to the following one. As for ACSP, the optimal solution can be non-simple. In the FSPTP variant ([5]), instead, the nodes associated to a given color can be visited only if at least a node of each preceding color has already been visited. Despite the similarities in the solution structure, the lack of two predefined endpoints and of the predefined color visiting order make ACSP and ACSP-UE significantly harder to tackle than SPTP and FSPTP. In GTSP the aim is to find a minimum-cost hamiltonian tour that includes exactly a vertex for each different color. Therefore, in this

case each vertex can be visited at most once, and the solution needs to be a cycle. GTSP is NP-Hard, and solution approaches have been mainly focused on Integer Programming methods ([18], [12], [13]), heuristics ([23],[22],[4]) or transformations to reduce the problem to the classical TSP ([7]). Finally, in GMST, a tree spanning all different colors with minimum weight is sought. Two different variants, in which the tree is required to contain either exactly a vertex ([19],[9],[10],[14],[21],[20],[16]) or at least a vertex ([17],[8], [15]) for each different color have been proposed, both being NP-Hard and hard to approximate.

ACSP was studied in [3],[1],[2]. In [3], after introducing the problem, the authors show it to be NP-Hard and inapproximable to a constant factor. They then prove a property of optimal solutions and present an integer linear programming flow-based formulation, as well as three heuristic and three metaheuristic algorithms. The heuristic algorithms are iterative rounding methods based on LP relaxations of the mathematical formulation, while the metaheuristics include a simulated annealing (SA), an ant colony optimization method (ACO) and a genetic algorithm (GA). In [1], a reduction from ACSP is used to prove the NP-Hardness of the Mobile Assisted Trilateration Based Energy Optimum Localization (MATBOL) problem, aimed at minimizing the traveling distance of a mobile beacon used to aid trilateration in the context of wireless sensor networks. In [2], the authors present an ACSP variant defined on trees (ACSP-t). In this work, the authors prove that also this variant is NP-Hard and cannot be approximated to a constant factor. They also propose an ILP formulation, an iterative rounding heuristic and two metaheuristics, namely a GA and a Tabu Search.

In this work we mainly focus on the ACSP-UE problem variant, that to the best of our knowledge has not been introduced before. However, all the approaches that we present are easily adaptable to solve ACSP. Indeed, we present a comparison among our proposed methods and those proposed in [3] for ACSP in Section 5. We also describe how to adapt these approaches to solve a further variant that considers as set of candidate starting vertices those assigned to a given color, that we call *All-Colors Shortest Path with Starting Color* (ACSP-SC). Moreover, in Section 2 we show ACSP and ACSP-UE to be computationally equivalent.

Our proposed approaches are a mathematical formulation and a metaheuristic algorithm. The metaheuristic is a Variable Neighborhood Search (VNS), and is based on the concept of two-level solutions. That is, for each problem solution we consider a *high-level*, abstract representation of it, corresponding to the order in which colors are encountered, in addition to the *low-level* (actual) one. As will be shown, by jointly operating on the two levels we obtained a fast and effective algorithm.

The rest of the paper is organized as follows. In Section 2 we define the problems formally, present some properties and show reductions between ACSP and ACSP-UE. In Section 3 we present the mathematical formulation, while the VNS algorithm is discussed in Section 4. Section 5 presents our

computational results, while conclusions and other remarks are presented in Section 6.

2 Problems definition and properties

Let $G = (V, E, C)$ be an undirected, connected and vertex labeled graph, where $V = \{v_1, \ldots, v_n\}$ is the set of vertices, $E = \{e_1, \ldots, e_m\}$ is the set of edges and $C = \{c_1, \ldots, c_k\}$ is a set of labels (or colors), with $|C| \leq |V|$. Moreover, let $\omega : E \rightarrow \mathbb{R}^+$ be a function assigning a positive weight to each edge, and $\gamma : V \rightarrow C$ be a function assigning a color to each vertex. We denote by V_c the subset of vertices of V having the color c, that is $V_c = \{v \in V : \gamma(v) = c\}$. We use the notation $p = [v_1^p, v_2^p, \ldots, v_h^p]$ to denote a path p of G; that is, for each $i \in \{1, \ldots h-1\}$, $v_i^p \in V$, $v_{i+1}^p \in V$ and $\{v_i^p, v_{i+1}^p\} \in E$.

The aim of ACSP-UE is to find a path $p = [v_1^p, v_2^p, \ldots, v_h^p]$ such that **i)** all colors are reached at least once, that is, $\forall c_j \in C \ \exists v_i^p : \gamma(v_i^p) = c_j$; **ii)** the overall weight of the path $\omega(p) = \sum_{i=1}^{h-1} \omega(\{v_i^p, v_{i+1}^p\})$ is minimized.

In ACSP, the starting vertex of any given feasible solution must be a predefined source vertex $v_{src} \in V$. In ACSP-SC, the starting vertex must be chosen among those associated with a predefined color $c_{src} \in C$.

Clearly, the problems are correctly defined only if $|C| \geq 2$; indeed, if $|C| = 1$ selecting any vertex brings to a trivial ACSP-UE optimal solution with value 0. Similarly, v_{src} or any vertex with color c_{src} would be trivially optimal solutions for ACSP and ACSP-SC, respectively. It is also straightforward to observe that no feasible solution exists if a given color $c_i \in C$ is not assigned to any vertex in V.

We now discuss some ACSP-UE properties. In [3], the authors proved that the following results hold for ACSP:

Theorem 1 *ACSP is NP-Hard.*

Theorem 2 *ACSP is inapproximable to a constant factor.*

Proposition 1 *Let $[v_{src}, \ldots, v_h]$ be the optimal solution for an ACSP instance. The path does not traverse any edge $\{v_i, v_j\}$ more than once in the same direction.*

The proof provided in [3] for Proposition 1 is directly applicable to ACSP-UE as well, while the ones for Theorems 1 and 2 are adaptable with trivial modifications. Furthermore, in the following we show the existence of polynomial-time reductions from each of the two problems to the other.

We start with the reduction from ACSP-UE to ACSP. Let $G = (V, E, C)$ be an input graph for ACSP-UE. Considering a new vertex v' and a new color c', such that $\gamma(v') = c'$, we build a new graph $G' = (V', E', C')$, where $V' = V \cup \{v'\}$, $E' = E \cup \{\{v', v_i\} \ \forall v_i \in V\}$ and $C' = C \cup \{c'\}$. Each edge $\{v_i, v_j\} \in E$ has the same weight in both G and G', and each node $v_i \in V$ is assigned the same color in the two graphs. Furthermore, the weight of each edge

incident to v' in G' is equal to $2|E|\omega_{max}+1$, where ω_{max} is $\max(\{\omega(\{v_i,v_j\}) : \{v_i,v_j\} \in E\})$.

We first need to show a preliminary result.

Lemma 1 *Let $p' = [v', v_1^p \ldots, v_h^p]$ be an optimal solution for ACSP in the above described graph G', with source vertex v'. The path p' visits v' exactly once.*

Proof By contradiction, let us suppose that p' visits v' more than once, i.e. there is at least a $k = 1, \ldots, h$ such that $v_k^p = v'$. It follows by construction that $\omega(p') \geq 4|E|\omega_{max} + 2$. Now, let $q = [v_1^q \ldots, v_k^q]$ be an optimal ACSP-UE solution in G. From Proposition 1, $\omega(q) \leq 2|E|\omega_{max}$. But then $q' = [v', v_1^q \ldots, v_k^q]$ is a feasible ACSP solution with $\omega(q') \leq 4|E|\omega_{max} + 1 < \omega(p')$, contradicting the hypothesis. □

We are now ready to prove the reduction.

Proposition 2 *The path $p = [v_1^p \ldots, v_h^p]$ is an optimal solution for ACSP-UE in G if and only if $p' = [v', v_1^p \ldots, v_h^p]$ is an optimal solution for ACSP in G', with source vertex v'.*

Proof $\boxed{\Longrightarrow}$ Let us first assume that p is optimal for ACSP-UE in G and, by contradiction, that p' is not optimal for ACSP in G' with source v'. However, p' is surely a feasible solution for the latter problem; let $q' = [v', v_1^q \ldots, v_k^q]$ be the optimal one ($\omega(q') < \omega(p')$). By Lemma 1, we know that $q = [v_1^q \ldots, v_k^q]$ does not contain v' and is therefore a feasible ACSP-UE solution in G. Moreover, $\omega(q) < \omega(p)$, which contradicts the hypothesis on the optimality of p.

$\boxed{\Longleftarrow}$ Now, let us assume that p' is optimal and that p is not. Again from Lemma 1, we know that p is feasible for ACSP-UE in G. Let $q = [v_1^q \ldots, v_k^q]$ be an optimal ACSP-UE solution in G; therefore, $\omega(q) < \omega(p)$. We obtain that $q' = [v', v_1^q \ldots, v_k^q]$ is feasible for ACSP in G' with source v', and $\omega(q') < \omega(p')$, which is again a contradiction. □

Let us now illustrate the reduction from ACSP to ACSP-UE. Let $G = (V, E, C)$ be an input graph for ACSP, with source vertex $v_{src} \in V$. Considering a new vertex v' and a new color c', such that $\gamma(v') = c'$, we build a new graph $G' = (V', E', C')$, where $V' = V \cup \{v'\}$, $E' = E \cup \{\{v', v_{src}\}\}$ and $C' = C \cup \{c'\}$. Edges $\{v_i, v_j\} \in E$ have the same weight in both G and G', and nodes $v_i \in V$ have the same color in the two graphs. Moreover, the weight of $\{v', v_{src}\}$ is equal to $2|E|\omega_{max}+1$ (again, ω_{max} is $\max(\{\omega(\{v_i,v_j\}) : \{v_i,v_j\} \in E\})$).

We first introduce a preliminary result (Lemma 2), and then prove the reduction (Proposition 3). We omit the proof of the lemma, given that it follows easily from the construction of G' and by reasoning as for Lemma 1.

Lemma 2 *Any optimal solution for ACSP-UE in the above described graph G' visits v' exactly once, and v' is one of the two endpoints of the path.*

Proposition 3 *The path $p = [v_{src} \dots, v_h^p]$ is an optimal solution for ACSP in G (with source v_{src}) if and only if $p' = [v', v_{src} \dots, v_h^p]$ is an optimal solution for ACSP-UE in G'.*

Proof First of all we note that, given Lemma 2 and the fact that the problem is defined on undirected graphs, whenever considering an optimal ACSP-UE solution in G' we can consider v' to be its starting vertex. We also note that, by construction, its adjacent vertex will always be v_{src}.
$\boxed{\Longrightarrow}$ Let us first assume that p is optimal for ACSP in G with source v_{src}, and that p' is not optimal for ACSP-UE in G'. Obviously, p' is feasible for the latter problem. Let $q' = [v', v_{src} \dots, v_k^q]$ be the optimal one ($\omega(q') < \omega(p')$), and q be the subpath $[v_{src} \dots, v_k^q]$. From Lemma 2, q does not contain v', therefore it is a feasible ACSP solution in G with source v_{src}. Moreover, $\omega(q) < \omega(p)$, which contradicts the hypothesis on the optimality of p.
$\boxed{\Longleftarrow}$ Let $p' = [v', v_{src} \dots, v_h^p]$ be optimal for ACSP-UE in G'; again from Lemma 2, $p = [v_{src} \dots, v_h^p]$ is feasible for ACSP in G with source v_{src}. If we suppose that p is not optimal, then there must exist a feasible solution $q = [v_{src} \dots, v_k^q]$ such that $\omega(q) < \omega(p)$. However, in this case, $q' = [v', v_{src} \dots, v_k^q]$ is feasible for ACSP-UE in G' and its objective function is better than the one of p', which is again a contradiction. □

We now report some additional properties that we found for ACSP-UE, and that we used in both our mathematical model and VNS.

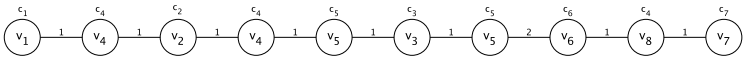
Proposition 4 *Let $p^* = [v_1^{p^*}, \dots, v_h^{p^*}]$ be an optimal solution for the ACSP-UE problem. Then **i)** $\gamma(v_h^{p^*})$ must be different from $\gamma(v_i^{p^*})$ $\forall i = \{1, .., h-1\}$, and **ii)** $\gamma(v_1^{p^*})$ must be different from $\gamma(v_i^{p^*})$ $\forall i = \{2, .., h\}$.*

Proof Let us first prove **i)**. It is easy to understand that if $\gamma(v_i^{p^*}) = \gamma(v_h^{p^*})$ for some $i = \{1, .., h-1\}$, then all colors are visited by the subpath $p' = [v_1^{p^*}, \dots, v_{h-1}^{p^*}]$. But then, p' is a feasible solution and it is cheaper than p^*, contradicting the hypothesis. Given that no assumption is made on $v_1^{p^*}$, the sub-property also holds for the ACSP and ACSP-SC problem variants. We can prove **ii)** analogously. However, this sub-property does not necessarily hold for ACSP and ACSP-SC, since the choice of the first endpoint is constrained. □

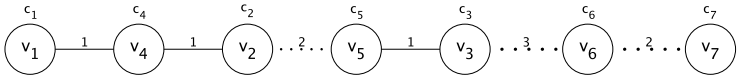
Before introducing the next property, we present an alternative representation for any feasible solution. Let $p = [v_1^p, v_2^p, \dots, v_h^p]$ be a feasible path for ACSP, that we also define the low-level representation of the path itself. The high-level representation of p is the path $p' =< v_1^{p'}, v_2^{p'}, \dots, v_{|C|}^{p'} >$ of $G' = (V, \{V \times V\}, C)$, containing the vertices corresponding to the first occurrence of each color in p. For each consecutive couple of vertices $v_i^{p'}$ and $v_{i+1}^{p'}$, the weight of the related edge is equal to $\omega(\{v_i^{p'}, v_{i+1}^{p'}\})$ if it also belongs to p, or to the sum of the weight of the edges between the two vertices in p otherwise. In the following, we will also use the term high-level (or low-level) solution to refer to a solution in the corresponding representation. Note

that we use square and angle brackets to distinguish low-level and high-level solutions, respectively.

In Figure 2(a) we show the optimal solution for the example of Figure 1, while in Figure 2(b) its high-level representation is shown. In this figure, we use dotted lines to highlight edges that substitute subpaths of p. That is, in Figure 2(b) edge $\{v_2, v_5\}$ replaces the subpath $[v_2, v_4, v_5]$, $\{v_3, v_6\}$ replaces $[v_3, v_5, v_6]$, and $\{v_6, v_7\}$ replaces $[v_6, v_8, v_7]$.



(a) Low-level (actual) solution



(b) High-level solution

Fig. 2 Low-level and high-level optimal solution for the instance of Figure 1

Clearly, if $p' =< v_1^{p'}, v_2^{p'}, \ldots, v_{|C|}^{p'} >$ is the high-level representation of a path p, they have the same cost and share the same starting vertex. Furthermore, from Proposition 4, we know that if $v_{|C|}^{p'}$ is not the final vertex of p, we can drop all subsequent vertices from the path and obtain a better feasible solution. In particular, the high-level and the low-level representation of an optimal solution will always share the same endpoints.

We can now present the following result, connected to the concept of high-level solution:

Proposition 5 *Let $p' =< v_1^{p'}, \ldots, v_i, v_j, \ldots, v_{|C|}^{p'} >$ be a high-level, optimal ACSP-UE solution. Then, the subpath between any couple of consecutive vertices v_i and v_j in the corresponding low-level solution is the shortest path between v_i and v_j in the input graph G.*

Proof Let v_i and v_j be any couple of consecutive vertices in p', and let $SP(i, j)$ denote their shortest path in G. Let us suppose by contradiction that the low-level representation p of p' contains a subpath with a cost strictly greater than $SP(i, j)$. By replacing this subpath with $SP(i, j)$, we obtain a feasible ACSP-UE solution that is better than p', contradicting the hypothesis. Given that no assumption is made on the first endpoint, the property also holds for ACSP and ACSP-SC. □

3 Mathematical model

In this section we introduce a mathematical model for ACSP-UE, whose solutions will be used to verify the effectiveness of our metaheuristic.

Starting from the graph G, we build a new directed graph $G^d = (V^d, E^d, C^d)$. V^d contains all vertices of V, as well as a dummy source s and a dummy sink t. E^d contains both arcs (v_i, v_j) and (v_j, v_i) for each edge $\{v_i, v_j\} \in E$, and both arcs have the same weight of the original edge. Additionally, E^d contains arcs $\{(s, v_i) : v_i \in V\} \cup \{(v_i, t) : v_i \in V\}$; these arcs have cost zero. We define $\delta^+(v_i, G^d) = \{v_j \in V^d : (v_i, v_j) \in E^d\}$ and $\Delta^+(v_i, G^d) = \{(v_i, v_j) \in E^d\}$. We similarly define $\delta^-(v_i, G^d) = \{v_j \in V^d : (v_j, v_i) \in E^d\}$ and $\Delta^-(v_i, G^d) = \{(v_j, v_i) \in E^d\}$. Finally, C^d contains all colors in C, as well as two additional ones, c_s, c_t; these colors are assigned to s and t, respectively, while all other nodes have the same color assigned in the two graphs. In the solution, we will look for a path from s to t crossing at least a vertex for each color in C.

The decision variables are the following:

- x_i: binary variable equal to 1 if vertex $v_i \in V$ belongs to the solution, and 0 otherwise.
- y_{ij}: binary variable equal to 1 if arc $(v_i, v_j) \in E^d$ belongs to the solution, and 0 otherwise. These variables are binary since no arc of E^d will be crossed more than once in the solution found by the model. By effect of Proposition 1, this does not compromise its optimality.

The mathematical model is the following:

$$\min \sum_{(v_i,v_j)\in E^d} \omega(v_i, v_j) y_{ij} \tag{1}$$

$$\sum_{v_i \in V_c} x_i \geq 1 \qquad c \in C \tag{2}$$

$$\sum_{v_k \in \delta^-(v_i, G^d)} y_{ki} \geq x_i \qquad v_i \in V \tag{3}$$

$$\sum_{v_k \in \delta^-(v_i, G^d)} y_{ki} = \sum_{v_j \in \delta^+(v_i, G^d)} y_{ij} \qquad v_i \in V \tag{4}$$

$$\sum_{v_i \in \delta^+(s, G^d)} y_{si} = 1 \tag{5}$$

$$\sum_{i \in \delta^-(t, G^d)} y_{it} = 1 \tag{6}$$

$$y_{ij} \leq x_i \qquad (v_i, v_j) \in E^d : v_i \in V \tag{7}$$

$$\sum_{(v_i,v_j)\in \Delta^-(v_j, G^d) | v_i \notin S, v_j \in S} y_{ij} \geq x_k \qquad S \subseteq V^d \setminus \{s\}, v_k \in S \tag{8}$$

$$x_i \in \{0, 1\} \qquad v_i \in V \tag{9}$$

$$y_{ij} \in \{0, 1\} \qquad (v_i, v_j) \in E^d \tag{10}$$

The objective function (1) minimizes the cost of the individuated path. Constraints (2) ensure that at least a vertex for each color is visited. Constraints (3) and (4) impose that there is at least an ingoing arc for each visited

vertex, and that the number of ingoing and outgoing arcs is the same for each of them, respectively. Constraints (5) and (6) state that there must be exactly one edge leaving s and one edge entering t, respectively. Constraints (7) state than an arc (v_i, v_j) with $i \in V$ can belong to the solution if v_i belongs to it as well. Constraints (8) ensure that all visited vertices are connected to s, and hence that the solution is connected. They are a modified version of the "directed connectivity constraints" and they state that, for each subset $S \subseteq V^d \setminus \{s\}$, if a visited vertex v_k belongs to S then there must be at least one arc entering in S. Finally, constraints (9)-(10) are variable definitions.

We now present some additional valid inequalities for our model, and briefly discuss how to adapt it to the ACSP and ACSP-SC variants.

3.1 Valid inequalities

In order to speed up the resolution of the model, in the following we introduce further constraints to break symmetry and take advantage of some of the properties introduced in the previous section.

- Symmetry often heavily affects the computational time required by integer programming models to find the optimal solution. Unfortunately, in the case of ACSP-UE there is symmetry that must be managed by using additional constraints. Indeed, since the input graph is undirected, if $[v_1^p, \ldots, v_h^p]$ is a feasible ACSP-UE solution, then $[v_h^p, \ldots \ldots, v_1^p]$ is feasible as well and has identical cost. These two paths would correspond to two distinct feasible solutions for our model ($[s, v_1^p, \ldots, v_h^p, t]$ and $[s, v_h^p, \ldots, v_1^p, t]$, respectively).
  We break this symmetry by introducing a new constraint, ensuring that the index of the first endpoint is always lower than the index of the last one. Formally,

$$\sum_{v_i \in V} i y_{si} \leq \sum_{v_j \in V} j y_{jt} \tag{11}$$

  This constraint, along with constraints (5)-(6), produces the desired effect.
- As a consequence of Proposition 4, we know that in the optimal solution the two endpoints have different colors. Hence, we introduced the following valid inequalities:

$$\sum_{v_i \in V_c} y_{si} + \sum_{v_j \in V_c} y_{jt} \leq 1 \qquad\qquad c \in C \tag{12}$$

  Our computational tests showed that these constraints improve the LP relaxation value and, in general, they reduce the time needed to find the optimal solution.

– Let v_i be a vertex contained in the optimal path; let c be its color. Again from Proposition 4, we know that no vertex $v_j \in V_c \setminus \{v_i\}$ can be the final endpoint. This is expressed by the following constraints:

$$\sum_{v_j \in V_c \setminus \{v_i\}} y_{jt} \leq 1 - x_i \qquad c \in C, v_i \in V_c \tag{13}$$

Analogous constraints are also valid with respect to the starting endpoint:

$$\sum_{v_j \in V_c \setminus \{v_i\}} y_{sj} \leq 1 - x_i \qquad c \in C, v_i \in V_c \tag{14}$$

In the following we refer to the ACSP-UE formulation (1)-(14) as ILP2-UE.

3.2 Adapting the model to ACSP and ACSP-SC

In order to adapt our model for ACSP and ACSP-SC, it is sufficient to solve the above presented mathematical formulation on differently defined directed graphs. Let us define $G^{d'} = (V^d, E^{d'}, C^d)$ the directed graph for ACSP-SC, and $G^{d''} = (V^d, E^{d''}, C^d)$ the one for ACSP. For each $v_i \in V^d \setminus \{s\}$ and $v_j \in V^d \setminus \{s\}$, both $E^{d'}$ and $E^{d''}$ contain (v_i, v_j) if and only if the arc is contained in E^d, and the arc has the same weight that it has in E^d. Furthermore, $E^{d'}$ contains arcs $\{(s, v_i) : v_i \in V_{c_{src}}\}$, while $E^{d''}$ contains only the arc (s, v_{src}), and these arcs have cost zero.

It is easy to understand that $G^{d'}$ and $G^{d''}$ model the requirements on the first endpoint of the path related to the ACSP-SC and ACSP problems, respectively. Furthermore, given the above mentioned requirements, Constraints (11) and (14) are not valid for these variants of the problem.

In Section 5, we refer to the ACSP formulation (that is, (1)-(10),(12),(13) on graph $G^{d''}$) as ILP2.

4 Variable Neighborhood Search

In Section 2 we introduced the concept of high-level solutions, whose main advantage is that they represent feasible solutions as fixed-length, simple paths, showing in which order (and with which vertex) each color is first reached. As already discussed, supposing to be able to determine the color visiting sequence of the optimal solution, as well as the correct vertex for each color, finding the optimal solution would be an easy task, since we would just need to connect such vertices by means of shortest paths. Our VNS algorithm is based on this fundamental idea, trying to iteratively improve a current candidate solution as follows:

1. Given a feasible, high-level solution, we look for new solutions by perturbing the color visiting sequence using two classical neighborhood strategies for TSP problems, that is, relocate and 2-opt;

Algorithm 1: VNS pseudocode

```
1  sol ← GreedyInit();
2  bestSol ← sol;
3  for i = 1 to MaxShakes do
4      improvement ← true;
5      while improvement = true do
6          while improvement = true do
7              sol′ ← Relocate(sol);
8              if objFunction(sol′) < objFunction(sol) then
9                  sol ← sol′;
10             else
11                 improvement ← false;
12         sol′ ← 2-opt(sol);
13         if objFunction(sol′) < objFunction(sol) then
14             sol ← sol′;
15             improvement ← true;
16     if objFunction(sol) < objFunction(bestSol) then
17         bestSol ← sol;
18     sol ← Shake(sol);
19 return bestSol;
```

2. Once a new color visiting sequence has been decided in the above step, we determine locally optimal choices for the vertices of the colors involved in the perturbation.

Hence, we determine first the color sequence, and then the actual vertices of new high-level solutions. Once these vertices are chosen for a new high-level solution, it is easy to reconstruct the corresponding low-level one, since these vertices will always be connected by means of shortest paths. In this sense, we say that our approach works on two levels.

We also developed a greedy heuristic based on this fundamental idea, that operates in $|C| - 1$ steps and is used to produce the first feasible solution.

Algorithm 1 presents the pseudocode of our metaheuristic approach. After individuating a starting feasible solution using the heuristic algorithm described in Section 4.1, we look for improvements by means of a local search that uses relocate neighborhoods. As soon as this local search step fails to find an improvement, we apply a new local search using a 2-opt neighborhood strategy. If we manage to improve the current solution, the algorithm goes back to the relocate local search, otherwise we attempt to escape from the current local optimum by means of a shake operator, and the algorithm iterates. The two local search operators are presented in Section 4.2, while the shake operator is discussed in Section 4.3. When the 2-opt local search fails and a pre-defined number of shakes has been reached, the algorithm ends and the best solution found is returned.

4.1 Initialization algorithm

In a preliminary step, we evaluate the shortest paths among each couple of vertices of G by using the Floyd-Warshall algorithm (see [6]). As well known, the algorithm operates in $O(|V|^3)$ time.

The initialization algorithm then performs $|C|$ steps to produce a feasible ACSP-UE solution. In the first step, a random vertex is chosen as first endpoint. In the i-th step ($i = 2, \ldots, |C|$) the algorithm chooses, among all vertices whose colors differ from those of the vertices chosen in the previous steps, the one whose shortest path from the vertex chosen in the $(i-1)$-th step has minimum weight.

It is clear that, after the $|C|$-th step, the sequence of chosen vertices is a high-level feasible solution. Since the Floyd-Warshall algorithm allows to reconstruct each shortest path by means of an auxiliary predecessor matrix, we are also able to reconstruct the corresponding low-level solution.

The initialization algorithm is easy to adapt to ACSP and ACSP-SC. Indeed, in the first case the starting vertex is always chosen to be v_{src}, while in the second case it will be a random one among those associated to c_{src}.

4.2 Relocate and 2-Opt Local Search

Let $p =< v_1^p, \ldots, v_{|C|}^p >$ be the current high-level solution. Consider the associated color sequence $< c_1^p, \ldots, c_{|C|}^p >$, where $c_i^p = \gamma(v_i^p)$. Our two local search operators work as follows.

- The relocate local search performs $|C|^2 - |C|$ iterations, each building a new neighbor, operating in two steps:
  1. For $i \in \{1, \ldots, |C|\}$, the vertex in the i-th position is removed from p, hence we obtain an incomplete high-level solution $< v_1^p, \ldots, v_{i-1}^p, v_{i+1}^p, v_{|C|}^p >$. If $i = 1$ or $|C|$, the solution is simply truncated to remove the corresponding endpoint. Otherwise, new locally optimal vertex choices are made for colors c_{i-1}^p and c_{i+1}^p, since v_{i-1}^p and v_{i+1}^p may no longer be favorable choices now that they are directly connected in the high-level solution. Consider for instance the case of Figure 3(b), in which new vertices must be chosen for c_3^p and c_5^p after removing v_4^p from the solution show in Figure 3(a).
  2. For each incomplete solution obtained from the previous step, $|C| - 1$ new complete high-level solutions are obtained by adding a newly chosen vertex with color c_i^p in each position $j \in \{1, \ldots, |C|\}, j \neq i$. For instance, in Figure 4(a), color $c_i^p = c_4^p$ is relocated in the second position ($j = 2$).
- In the 2-opt local search, each new neighbor is generated by removing two edges from the current high-level solution and replacing them with two different ones. Overall, $\frac{|C^2|-|C|}{2}$ neighbors are created, operating as follows:

– For each $i = 1, \ldots, |C|-1$ and $j = i+1, \ldots, |C|$, a neighbor p' whose sequence of colors in its high-level representation is $< c^p_1, \ldots, c^p_{i-1}, c^p_j, \ldots, c^p_i, c^p_{j+1}, \ldots, c^p_{|C|} >$ is generated, where the color visiting sequence between j and i in p' is the inverse of p. For instance, given the high-level solution p with $|C| = 6$ in Figure 5(a), by applying a 2-opt operation corresponding to $i = 3$ and $j = 5$ we obtain a neighbor whose color visiting sequence is shown in Figure 5(b), that is, $< c^p_1, c^p_2, c^p_5, c^p_4, c^p_3, c^p_6 >$. The choice for a given color c^p_k in p' will be v^p_k if $k \notin \{i-1, i, j, j+1\}$, while a new choice is required otherwise. That is, we determine new choices for the endpoints of the edges involved in the swap. If $i = 1$, c^p_j becomes the color of the new starting endpoint, and similarly, if $j = |C|$, c^p_i is the new final color.

We now describe how new vertices are chosen when needed for the two steps of each relocate iteration, as well as for each 2-opt iteration. In the following, we define *undecided* the colors for which a new vertex has to be chosen, according to the previously described steps.

The underlying idea is to generate for each of these three steps an auxiliary directed graph $G^a = (V^a, E^a)$, containing all candidate vertices belonging to each undecided color. In G^a, each vertex of an undecided color c_i has an ingoing arc that connects it to the vertex that would precede it in the high-level solution that we are building, as well as an outgoing one to the vertex that would follow it. If c_i is preceded or followed in our solution by another undecided color c_j, its vertices are connected to all vertices of c_j. The weight of each arc in G^a is equal to the cost of the shortest path between its endpoints in the original graph G. Dummy source or destination nodes are considered to handle special cases in which preceding or following nodes do not exist. By looking for shortest paths in G^a, we identify the new vertices for the undecided colors. In more detail:

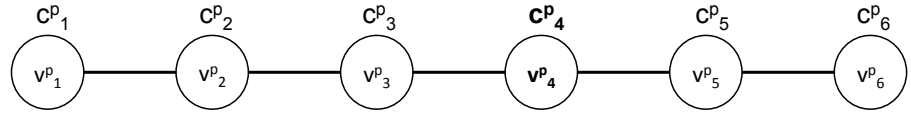
– **Relocate, Step 1**: Let us assume that v^p_{i-2} and v^p_{i+2} both exist. In this case, V^a is composed of $\{v^p_{i-2}, v^p_{i+2}\} \cup V_{c^p_{i-1}} \cup V_{c^p_{i+1}}$. E^a contains edges $(v^p_{i-2}, v'), \forall v' \in V_{c^p_{i-1}}$, arcs $(v', v''), \forall v' \in V_{c^p_{i-1}}, v'' \in V_{c^p_{i+1}}$ and arcs $(v'', v^p_{i+2}), \forall v'' \in V_{c^p_{i+1}}$. We then use the Dijkstra algorithm to find a shortest path in G^a from v^p_{i-2} and v^p_{i+2}; by construction, this path will cross exactly one vertex with color c^p_{i-1} and one vertex with color c^p_{i+1}. Figure 3(c)-(d) shows graph G^a and the individuated shortest path (edge weights are omitted).
If c^p_{i-1} is the starting color, v^p_{i-2} is substituted by a dummy source, which is connected in G^a to the vertices with color c^p_{i-1} though zero-weighted edges. With an analogous reasoning, we replace v^p_{i+2} with a dummy destination if c^p_{i+1} is the last color reached by the solution.
– **Relocate, Step 2**: Let us assume that c^p_i has to be relocated in the j-th position, with $1 < j < |C|$. Let v^{prv} and v^{nxt} be the vertices that will precede and follow the new vertex. In the auxiliary graph, V^a contains $\{v^{prv}, v^{nxt}\} \cup V_{c^p_i}$, while E^a contains arcs (v^{prv}, v') and (v', v^{nxt}) $\forall v' \in V_{c^p_i}$.

By looking for a shortest path between v^{prv} and v^{nxt}, we identify a vertex with color c_i^p. Figure 4(b)-(d) show these steps and the final high-level representation of the newly built neighbor.
If $j = 1$, v^{prv} does not exist; the procedure reduces to selecting the vertex in $V_{c_i^p}$ whose shortest path distance from $v^{nxt} = v_1^p$ is minimal. The same holds with respect to $v^{prv} = v_{|C|}^p$ if $j = |C|$.

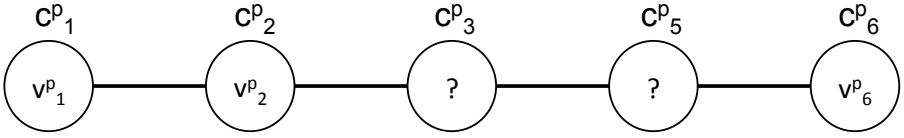
– **2-opt**: In G^a, for each color $c_k^p \in C$, V^a contains v_k^p if $k \notin \{i-1, i, j, j+1\}$, or $V_{c_k^p}$ otherwise. Given any couple of consecutive colors c_k^p and c_q^p in the new solution, E^a will contain an arc from each vertex with color c_k^p to each vertex with color c_q^p in V^a.
If the first and last endpoint of the path do not belong to undecided colors ($i - 1 > 1$ and $j + 1 < |C|$), we look for a shortest path between them. Otherwise, if $i-1 \leq 1$, the first endpoint belongs to either color $c_{i-1}^p = c_1^p$ or c_j^p; in both cases, this endpoint is not known. Hence, we consider a dummy source that is connected to each vertex with color c_1^p (or c_j^p, respectively) with zero-weighted arcs. Analogously, if $j + 1 \geq |C|$, the last endpoint of the path has color $c_{j+1}^p = c_{|C|}^p$ or c_i^p. In these cases, we add a dummy destination.
Figure 5(c) shows the G^a auxiliary graph for the considered example; note that since $i-1 = 2 > 1$ we do not need the dummy source (the first endpoint v_1^p is known), while the dummy destination is needed ($j = 5 = |C| - 1$). Figures 5(d)-(e) show the shortest path found and the related high-level neighbor solution, respectively.

Each relocate local search iteration stores the neighbor solution with minimum cost encountered. Once all neighbors have been generated, if the best one is an improvement with respect to the current solution, a new relocate local search iteration starts. Otherwise, the current solution does not change and the 2-opt local search step is invoked. Similarly to the relocate one, our 2-opt local search explores the whole neighborhood, storing the best solution found. If an improvement with respect to the current solution is found, we go back to the relocate local search; otherwise we attempt to escape from the current local minimum by performing a shake operation.

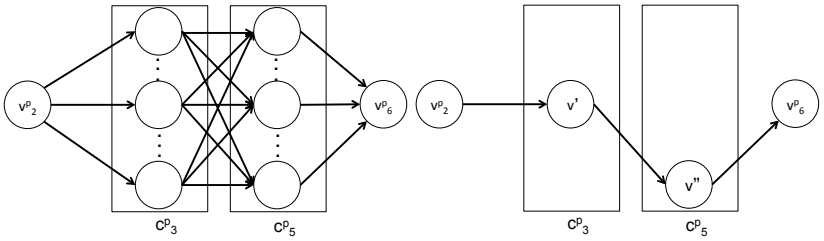
It is easy to adapt both operators to ACSP or ACSP-SC. For the relocate, the color corresponding to the first endpoint is not removed from the solution in step 1, and no color is relocated in the first position in step 2. Furthermore, with respect to ACSP, when the color in the the second position is removed from the path in step 1, we make sure that the first endpoint is not changed. In order to adapt the 2-opt local search, we avoid the case $i = 1$, since, as described, this case corresponds to a choice of a new color for the first endpoint. Moreover, for ACSP, when $i = 2$ we impose the shortest path in G^a to start from $v_1^p = v_{src}$.



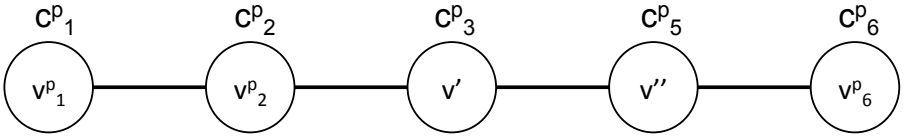
(a) Current high-level solution; vertex with color c_4^p chosen for removal



(b) Incomplete solution after the removal; choices needed for c_3^p and c_5^p



(c) Auxiliary graph G^a (d) Shortest path identification

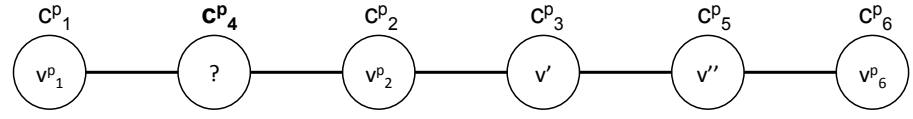


(e) Incomplete solution after the new choices for c_3^p and c_5^p

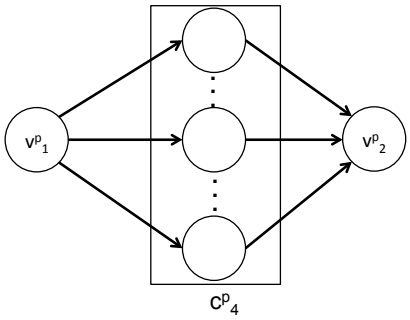
Fig. 3 Neighbor construction through Relocate Local Search (step 1, $i = 4$)

4.3 Shake operator

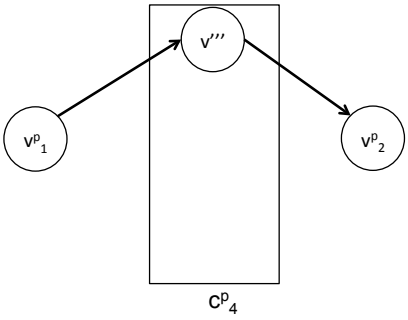
The shake operator performs a number of random perturbations to the current solution, in order to try to escape from the local optimum. In more detail, every time that the shake operator is invoked, it performs $\frac{|C|}{shake_1}$ relocate operations. For each of these operations, the i and j values used in the two steps of the relocate procedure are chosen randomly. Overall, a total of $shake_2$ shake operations are performed, and every $\frac{shake_2}{shake_3}$ invocations of the operator, the current solution is rebuilt from scratch by using the initialization algorithm described in Section 4.1. The values chosen for parameters $shake_1$, $shake_2$ and $shake_3$ are reported in Section 5.



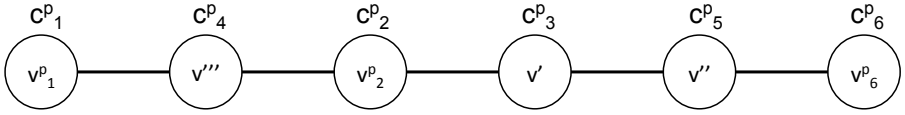
(a) Color c_4^p to be relocated in position 2



(b) Auxiliary graph G^a



(c) Shortest path identification



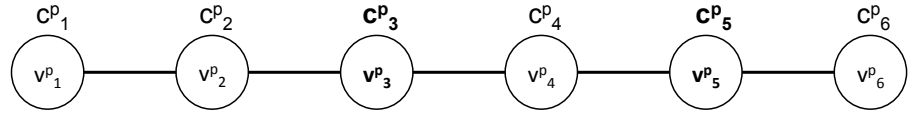
(d) Complete high-level solution after the new choice for c_4^p

Fig. 4 Neighbor construction through Relocate Local Search (step 2, $j = 2$)

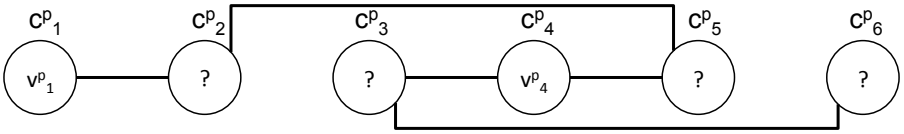
5 Computational results

This section presents the test scenarios and the results obtained during our computational test phase. Our VNS algorithm was coded using the C++ programming language, while the mathematical formulations were implemented and solved using the IBM ILOG CPLEX 12.6.1 solver. All tests were performed in single thread mode on a machine with an Intel Xeon E5-2650 v3 processor running at 2.3 GHz and 128 GB of RAM. With respect to the VNS parameters, after a preliminary tuning phase we chose the values $shake_1 = 3$, $shake_2 = |V|$ and $shake_3 = 5$. For CPLEX, we considered a time limit equal to 3600 seconds. Whenever the solver reaches this threshold, the related solution value is marked with a "*" symbol to highlight that this value is an upper bound of the optimal solution.

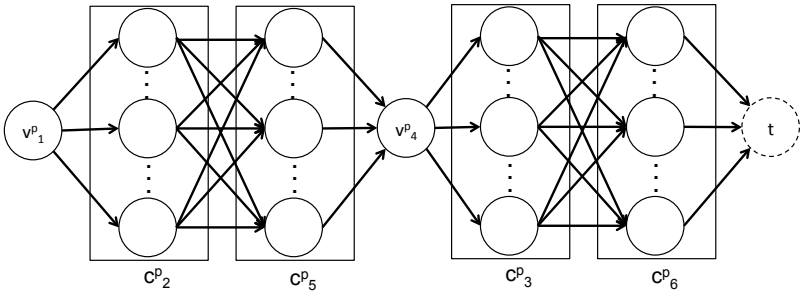
The following two subsections contain results for the ACSP and ACSP-UE problems, respectively.



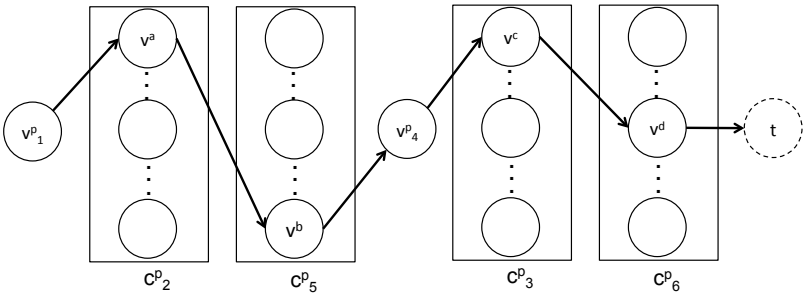
(a) Current high-level solution



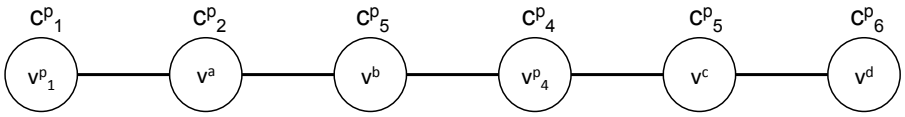
(b) Color sequence after 2-opt operation; new choices needed for c_2^p,c_5^p,c_3^p,c_6^p



(c) Auxiliary graph G^a (t=dummy destination)



(d) Shortest path identification



(e) High-level solution after the new vertex choices

Fig. 5 Neighbor construction through 2-opt Local Search ($i = 3$, $j = 5$)

5.1 Comparisons on the ACSP problem

In this subsection, we compare the effectiveness and the performance of our formulation and VNS metaheuristic with the formulation and heuristics proposed in [3] for the ACSP problem.

Instances			ILP2			ILP			GAP	
n	**m**	**k**	**LP**	**Obj**	**Time**	**LP**	**Obj**	**Time**	**LP**	**Obj**
50	189	10	37.03	40	0.40	26.01	40	10.53	42.39%	0.00%
50	152	20	101.00	101	0.06	79.35	101	3.20	27.29%	0.00%
50	154	25	129.34	132	0.21	106.36	132	6.08	21.60%	0.00%
100	338	25	128.81	138	3.04	94.33	138	86.91	36.55%	0.00%
100	330	40	207.62	220	1.25	148.55	220	19.82	39.76%	0.00%
100	373	50	229.28	233	0.95	192.91	233	35.74	18.85%	0.00%
200	734	50	179.69	223	158.22	119.95	223*	3612.17	49.81%	0.00%
200	746	75	373.51	399	87.47	310.25	399	2013.22	20.39%	0.00%

Table 1 Comparison of the ILP2 and ILP formulations for ACSP on the instances proposed in [3].

We start by comparing the performance of our ILP2 formulation with the formulation proposed in [3], named ILP. Both formulations were implemented by us, and compared on our testing environment.

A first comparison between ILP and ILP2 was carried out on the dataset of instances proposed in [3], having a number of vertices between 50 and 200 and a number of colors between 10 and 75. The instance files have been provided by the authors.

The results of this comparison are reported in Table 1. Under the *Instances* heading, we report the instances characteristics (number of vertices n, number of edges m and number of colors k). The next six columns report the root linear relaxation value (LP), the solution value (Obj) and the computational time ($Time$), in seconds, for ILP2 and ILP, respectively. Finally, under the GAP heading, we report the gap percentage between the linear relaxation values and between the solution values, respectively. These gaps are computed by using the formulas $100 \times \frac{LP(ILP2) - LP(ILP)}{LP(ILP)}$ and $100 \times \frac{Obj(ILP2) - Obj(ILP)}{Obj(ILP)}$, respectively.

The results under the GAP heading show that the linear relaxation of ILP2 is always better than the one of ILP, with gaps ranging from 18.85% to 49.81%. With respect to solutions quality, we note that ILP2 always finds the optimal one, while ILP reaches the time limit once (see the case with 200 nodes and 50 colors), hence it is not able to certify the optimality of the solution found in this case. Regarding computational times, ILP2 is always faster than ILP, solving 6 out of 8 instances in less than 4 seconds, and requiring about 158 seconds in the worst case. The computational times of ILP are significantly higher, and worse in all cases. Indeed, as already mentioned, in the worst case it reaches the time limit, on the last instance it requires about 2013 seconds, and on the remaining 6 instances the computational time ranges from 3 to 87 seconds. The results of Table 1 clearly show that ILP2 outperforms ILP on this dataset.

A second comparison among the two formulations was carried out on a new, larger set of instances. Our instances were generated with respect to 3 parameters: the number of vertices n, the number of edges m and the number of colors k. The vertices are randomly disposed in a square area of size 50x50. The value of n is chosen in the set $\{25, 50, 75, 100, 150\}$. The

number of edges m is chosen by using a density d that ranges in the set $\{0.2, 0.3, 0.4, 0.5\}$ ($m = \frac{n(n-1)}{2} \times d$). Finally, the number of colors k belongs to the set $\{\lceil 0.1n \rceil, \lceil 0.2n \rceil, \lceil 0.3n \rceil, \lceil 0.4n \rceil\}$. We generated 5 instances for each combination of parameters, discarding the case $n = 25$, $k = \lceil 0.1n \rceil = 3$, which resulted to be particularly trivial and in which it would not make sense to define the 2-opt operator (these instances were also used to test our VNS, as will be discussed in Section 5.2). Therefore, our dataset is composed in total of 76 different scenarios and 380 individual instances. For these tests, the node indexed with 0 was also assumed to be the ACSP source vertex. Our instances are available online[1].

Table 2 contains the results of the comparison between ILP2 and ILP on this new set of instances. Table headings have the same meaning that they have for Table 1, with the addition of column m which reports the value of the additional instance parameter. However, in this case we report average values for each scenario.

We can note that the linear relaxation values of ILP2 are again always better than the ones of ILP, with a percentage gap that ranges from about 1.5% to over 100%. In 56 out of 76 scenarios, the percentage gap is greater than 20%. As a consequence, we observe a remarkable difference between the effectiveness of ILP2 and ILP. Indeed, ILP2 reaches the time limit without finding the optimal solution only once (150 nodes, 4470 edges, 30 colors). On the other hand, ILP reaches the time limit 24 times, and the first failures occur on the instances with 75 nodes. The solution percentage gap is lower than -5% in 14 out these 24 cases, and it decreases down to about -35%. We note in particular that ILP never finds the optimal solution for the instances with $n = 150$. With respect to performances, ILP2 is most of the times an order of magnitude faster than ILP. We can note that the scenarios with up to 100 nodes are optimally solved by ILP2 within about 3.5 minutes. On the same scenarios, ILP reaches the time limit 8 times. In the 15 scenarios with 150 nodes solved to optimality by ILP2, the model requires up to about 18 minutes, while as mentioned the time limit is always reached by ILP.

A final comparison for the ACSP problem is carried out between our VNS metaheuristic and those proposed in [3] for the problem. We recall that in this work the authors present 3 metaheuristics, namely a simulated annealing (SA), an ant colony optimization (ACO) and a genetic algorithm (GA). Moreover, they describe 3 heuristics based on iterative rounding of a mathematical formulation for the problem that they develop. The heuristics differ with respect to the variables on which the rounding is performed, and are called LP_x, LP_f and $LP_{f/x}$ respectively. These 6 algorithms were tested on the same dataset of Table 1. The results for these algorithms are taken from [3].

Table 3 contains the results of this comparison, with the results of the 4 metaheuristics (that is, our VNS and the ones in [3]) reported in the (a) subtable, and the rounding heuristics in the (b) subtable. In [3], for each instance and algorithm, the obtained result is reported in terms of proportion with

[1] http://www.dipmat2.unisa.it/people/carrabs/www/DataSet/ACSP_Instances.zip

Instances			ILP2			ILP			GAP	
n	m	k	LP	Obj	Time	LP	Obj	Time	LP	Obj
25	60	5	67.97	72.40	0.08	47.31	72.40	0.30	43.69%	0.00%
25	90	5	57.33	61.40	0.07	39.36	61.40	0.49	45.65%	0.00%
25	120	5	39.66	47.40	0.04	32.80	47.40	0.33	20.90%	0.00%
25	150	5	39.19	46.00	0.10	29.96	46.00	0.52	30.79%	0.00%
25	60	8	116.64	117.20	0.11	86.73	117.20	1.04	34.48%	0.00%
25	90	8	98.22	100.20	0.03	74.97	100.20	1.40	31.02%	0.00%
25	120	8	82.26	89.20	0.13	68.63	89.20	1.66	19.86%	0.00%
25	150	8	80.25	85.20	0.10	66.46	85.20	1.64	20.76%	0.00%
25	60	10	149.80	157.20	0.01	137.37	157.20	0.44	9.05%	0.00%
25	90	10	118.16	137.40	0.04	107.68	137.40	1.55	9.73%	0.00%
25	120	10	109.14	110.00	0.07	99.76	110.00	1.09	9.41%	0.00%
25	150	10	81.21	104.20	0.11	79.96	104.20	4.42	1.56%	0.00%
50	245	5	37.67	44.40	0.24	21.85	44.40	3.84	72.36%	0.00%
50	367	5	28.84	39.00	0.21	17.30	39.00	6.28	66.74%	0.00%
50	490	5	23.87	37.40	0.33	15.21	37.40	8.26	56.94%	0.00%
50	612	5	24.25	29.60	0.55	12.27	29.60	7.90	97.61%	0.00%
50	245	10	78.73	96.60	0.28	62.62	96.60	9.10	25.73%	0.00%
50	367	10	73.89	80.40	0.33	51.44	80.40	9.93	43.65%	0.00%
50	490	10	66.12	81.00	0.80	44.98	81.00	36.40	47.00%	0.00%
50	612	10	54.61	69.80	0.90	36.82	69.80	25.07	48.30%	0.00%
50	245	15	152.49	176.20	0.34	125.39	176.20	10.38	21.61%	0.00%
50	367	15	129.13	136.60	0.41	102.33	136.60	14.19	26.19%	0.00%
50	490	15	112.05	134.20	0.74	88.59	134.20	31.59	26.49%	0.00%
50	612	15	100.14	111.60	0.96	76.12	111.60	28.93	31.55%	0.00%
50	245	20	223.05	228.60	0.28	188.52	228.60	6.47	18.32%	0.00%
50	367	20	193.58	205.00	0.48	165.63	205.00	14.56	16.88%	0.00%
50	490	20	148.33	161.40	0.65	125.47	161.40	22.71	18.22%	0.00%
50	612	20	157.06	171.60	0.80	133.60	171.60	22.24	17.56%	0.00%
75	555	8	48.51	58.60	1.05	23.79	58.60	20.18	103.93%	0.00%
75	832	8	41.99	62.80	2.55	27.91	62.80	82.42	50.46%	0.00%
75	1110	8	30.16	46.40	4.06	18.48	46.40	80.05	63.20%	0.00%
75	1387	8	34.45	43.60	2.71	20.04	43.60	44.49	71.87%	0.00%
75	555	15	97.19	121.40	0.95	79.93	121.40	40.24	21.59%	0.00%
75	832	15	94.64	111.80	5.12	70.67	111.80	145.38	33.91%	0.00%
75	1110	15	75.00	96.40	4.49	55.10	96.40	407.45	36.10%	0.00%
75	1387	15	60.35	90.80	7.68	47.56	90.80	701.17	26.88%	0.00%
75	555	23	187.52	222.40	3.53	162.47	222.40	154.83	15.42%	0.00%
75	832	23	158.94	178.00	2.94	127.54	178.00	402.89	24.62%	0.00%
75	1110	23	138.30	161.20	7.90	107.63	162.00*	973.69	28.49%	-0.49%
75	1387	23	131.89	147.00	3.70	105.30	147.00	254.37	25.25%	0.00%
75	555	30	291.01	307.20	1.70	237.69	307.20	109.54	22.43%	0.00%
75	832	30	204.28	234.00	1.45	181.19	234.00	68.79	12.74%	0.00%
75	1110	30	200.79	217.00	3.41	165.65	217.00	369.20	21.21%	0.00%
75	1387	30	186.20	208.40	8.96	153.63	209.00*	982.64	21.20%	-0.29%
100	990	10	51.71	70.80	4.94	26.39	70.80	258.00	95.95%	0.00%
100	1485	10	40.64	58.20	6.77	23.84	58.20	155.52	70.48%	0.00%
100	1980	10	29.90	52.00	10.54	22.68	52.00	381.85	31.85%	0.00%
100	2475	10	32.17	49.20	25.50	19.71	49.60*	1341.44	63.22%	-0.81%
100	990	20	128.05	149.60	5.48	91.30	149.60	343.70	40.26%	0.00%
100	1485	20	92.75	119.60	11.85	71.46	119.60	905.75	29.80%	0.00%
100	1980	20	76.00	121.40	22.04	64.89	122.40*	2300.55	17.12%	-0.82%
100	2475	20	64.00	103.60	23.79	56.79	108.60*	1947.42	12.70%	-4.60%
100	990	30	225.80	246.80	10.27	172.88	246.80	354.60	30.61%	0.00%
100	1485	30	184.57	211.80	8.67	149.75	211.80	440.12	23.25%	0.00%
100	1980	30	153.70	185.00	202.05	121.44	192.00*	2441.90	26.57%	-3.65%
100	2475	30	134.31	173.20	195.68	104.38	183.00*	3258.62	28.67%	-5.36%
100	990	40	315.86	341.00	3.77	270.11	341.00	232.40	16.94%	0.00%
100	1485	40	269.10	297.80	10.63	229.42	297.80	775.97	17.30%	0.00%
100	1980	40	215.75	237.60	7.72	179.59	237.60	325.57	20.14%	0.00%
100	2475	40	204.92	236.40	57.14	164.52	239.00*	3063.54	24.55%	-1.09%
150	2235	15	58.97	89.20	58.22	36.10	97.20*	3612.09	63.33%	-8.23%
150	3352	15	40.97	74.00	275.76	29.76	81.40*	3191.23	37.64%	-9.09%
150	4470	15	42.87	69.40	243.49	30.57	80.00*	3612.18	40.26%	-13.25%
150	5587	15	30.32	65.80	539.53	28.44	78.60*	3612.21	6.62%	-16.28%
150	2235	30	159.26	186.40	144.21	110.01	196.00*	3377.19	44.77%	-4.90%
150	3352	30	122.61	156.80	373.97	93.27	174.80*	3612.07	31.46%	-10.30%
150	4470	30	91.96	144.40*	1209.91	75.75	181.80*	3612.11	21.40%	-20.57%
150	5587	30	91.77	136.80	743.04	76.95	191.00*	3612.17	19.25%	-28.38%
150	2235	45	276.83	318.00	107.23	222.08	343.00*	3612.15	24.65%	-7.29%
150	3352	45	209.45	253.60	233.50	165.19	293.60*	3612.14	26.80%	-13.62%
150	4470	45	192.62	230.00	396.87	147.59	256.20*	3612.12	30.51%	-10.23%
150	5587	45	169.15	208.00	888.98	133.94	276.60*	3612.16	26.29%	-24.80%
150	2235	60	405.33	433.40	56.06	339.73	434.40*	2646.15	19.31%	-0.23%
150	3352	60	317.41	352.60	150.30	271.63	364.00*	2953.43	16.85%	-3.13%
150	4470	60	266.24	312.20	608.61	232.13	331.00*	3513.10	14.69%	-5.68%
150	5587	60	243.86	294.60	1066.29	198.13	454.25*	3612.22	23.08%	-35.15%

Table 2 Comparison of the ILP2 and ILP formulations for ACSP on the new set of instances.

(a) Metaheuristics results

Instances			VNS			SA [3]		
n	m	k	Avg Obj	Best Obj	Avg Time	Avg Obj	Best Obj	Avg Time*
50	189	10	1.00	1.00	0.03	1.00	1.00	27.62
50	152	20	1.01	1.00	0.13	1.14	1.11	38.85
50	154	25	1.01	1.01	0.21	1.17	1.12	45.99
100	338	25	1.01	1.00	1.09	1.23	1.18	168.00
100	330	40	1.03	1.00	4.13	1.33	1.22	215.50
100	373	50	1.02	1.00	8.95	1.39	1.32	276.76
200	734	50	1.05	1.03	32.30	1.51	1.46	870.14
200	746	75	1.05	1.04	154.88	1.62	1.55	1231.93

Instances			ACO [3]			GA [3]		
n	m	k	Avg Obj	Best Obj	Avg Time*	Avg Obj	Best Obj	Avg Time*
50	189	10	1.06	1.00	1.92	1.07	1.05	0.35
50	152	20	1.19	1.17	6.74	1.23	1.12	0.45
50	154	25	1.18	1.12	11.46	1.33	1.22	0.59
100	338	25	1.26	1.17	11.63	1.22	1.15	0.79
100	330	40	1.33	1.28	31.41	1.54	1.40	1.08
100	373	50	1.40	1.29	57.74	1.56	1.45	1.90
200	734	50	1.44	1.39	52.43	1.58	1.45	2.48
200	746	75	1.50	1.46	139.81	1.72	1.58	4.71

(b) Rounding heuristics results

Instances			LP_x [3]		LP_f [3]		$LP_{f/x}$ [3]	
n	m	k	Obj	Time*	Obj	Time*	Obj	Time*
50	189	10	3.15	1.46	1.15	0.85	2.08	0.80
50	152	20	1.86	2.10	1.16	1.12	1.19	0.99
50	154	25	1.90	2.28	1.32	1.48	1.36	0.98
100	338	25	2.14	4.80	1.21	2.59	1.51	2.79
100	330	40	2.00	7.43	1.32	3.46	1.46	3.35
100	373	50	1.81	7.18	1.16	3.32	1.12	2.91
200	734	50	2.35	33.78	1.39	19.38	1.69	19.47
200	746	75	1.96	42.78	1.33	25.39	1.48	19.15

Table 3 Comparison of heuristics for ACSP.
*Computational times reported in [3] are divided by 1.21

respect to the optimal objective function value, found using CPLEX. Furthermore, since the 3 metaheuristics are non deterministic, the authors perform 10 independent runs for each of them and report the best and average solutions found, respectively. In order to be comparable, we ran our tests and reported our results accordingly; these values are contained in the *Avg Obj* and *Best Obj* columns for each metaheuristic, while the average computational times in seconds can be found in the *Avg Time* columns. The results for a single run was instead reported for each instance and each of the 3 heuristics, and we report these results in Table 3(b). In order to improve the computational times comparability (in [3], an AMD Phenom II X4 810 machine running at 2.67 GHz with 2 GB of RAM was used), we referred to the CPU performance com-

parative table provided by the UC Berkeley SETI@home experiment website[2], based on Whetstone benchmarks. By comparing the GFLOPS/core values, we divided all computational times reported in [3] by 1.21.

We can see that our VNS appears to be remarkably more effective than the previous approaches. The optimal solution is found in all 10 runs in one case ($n = 50$, $k = 10$), and in the best case for 5 out 8 instances. In the computational tests performed in [3], only SA (the most time-intensive approach) was able to find the optimal solution for the same instance in all 10 runs. The ACO algorithm was able to find the optimal solution for this instance in the best case; no other instance was ever solved to optimality by any of their 6 proposed approaches.

On average, VNS found solutions diverging from the optimal one within 1% in 4 cases, 2% and 3% in one case each, and within 5% for the two largest instances with 200 vertices. In the best case, this threshold is never larger than 4%. On the other hand, the gap grow up to 46%, 55% or 58% for the previous 3 metaheuristics in the best case, and up to 50%, 62% or 72% in the average case. The overall best-performing rounding heuristic (LP_f) found a solution with an objective function gap equal to 15% for the instance with $n = 50$, $k = 10$, growing up to 39% for $n = 200$, $k = 50$.

With respect to the computational times, we note that GA algorithm appears to be the fastest one, running within 5 seconds on average. The ACO algorithm, while being slower than VNS on the smaller instances, appears to have roughly similar computational times on the largest ones; for $n = 200$, $k = 75$ the average computational time is 154.88 seconds for VNS and 139.81 seconds for ACO. The SA heuristic appears to be the most time intensive, being often at least one order of magnitude slower than VNS. The rounding heuristics have low computational times, being generally slower than GA and faster than ACO and VNS.

5.2 Comparisons on the ACSP-UE problem

In this section, we use our VNS algorithm to solve the ACSP-UE problem on our new instances, presented in the previous section. We compare the solution values found by VNS with the optimal values (or upper bounds) found by ILP2-UE. In order to better verify the stability of the VNS and be consistent with the previously presented tests, we performed 10 independent runs of our metaheuristic on each instance. Table 4 presents the collected results. The first 3 columns contain the instance characteristics; the following 4 columns contain the results for our formulation (*ILP2-UE* heading) and metaheuristic (*VNS* heading). Finally, the last column reports the gap value, in percentage, between the solutions of ILP2-UE and of VNS; in more detail, this value is computed as $100 \times \frac{Obj(VNS) - Obj(ILP2-UE)}{Obj(ILP2-UE)}$.

[2] https://setiathome.berkeley.edu/cpu_list.php

Instances			ILP2-UE		VNS		GAP
n	m	k	Obj	Time	Obj	Time	
25	60	5	51.40	0.05	51.40	0.00	0.00%
25	90	5	37.60	0.09	37.60	0.00	0.00%
25	120	5	38.40	0.13	38.40	0.00	0.00%
25	150	5	33.20	0.11	33.20	0.00	0.00%
25	60	8	81.00	0.05	81.00	0.00	0.00%
25	90	8	88.40	0.13	88.40	0.00	0.00%
25	120	8	80.00	0.16	80.00	0.00	0.00%
25	150	8	79.20	0.19	79.40	0.00	0.25%
25	60	10	141.80	0.05	141.80	0.00	0.00%
25	90	10	123.20	0.13	123.60	0.00	0.32%
25	120	10	103.20	0.09	103.20	0.01	0.00%
25	150	10	87.80	0.12	87.80	0.00	0.00%
50	245	5	30.60	0.35	30.80	0.00	0.65%
50	367	5	24.80	0.40	24.80	0.01	0.00%
50	490	5	25.80	0.61	25.80	0.00	0.00%
50	612	5	22.60	1.27	23.40	0.00	3.54%
50	245	10	85.20	0.42	85.20	0.03	0.00%
50	367	10	70.20	0.52	70.20	0.03	0.00%
50	490	10	69.20	1.30	69.26	0.03	0.09%
50	612	10	63.80	1.78	63.80	0.03	0.00%
50	245	15	153.00	0.51	153.60	0.08	0.39%
50	367	15	125.80	0.89	126.24	0.09	0.35%
50	490	15	125.40	1.31	125.40	0.08	0.00%
50	612	15	104.80	1.81	104.80	0.07	0.00%
50	245	20	211.80	0.42	212.72	0.14	0.43%
50	367	20	191.20	0.54	194.92	0.14	1.95%
50	490	20	154.20	0.76	155.32	0.14	0.73%
50	612	20	159.60	0.94	160.30	0.13	0.44%
75	555	8	51.40	1.93	51.40	0.05	0.00%
75	832	8	46.00	1.83	46.28	0.05	0.61%
75	1110	8	36.80	3.52	36.80	0.05	0.00%
75	1387	8	36.80	5.20	36.80	0.05	0.00%
75	555	15	111.40	1.95	111.52	0.18	0.11%
75	832	15	103.20	6.08	103.20	0.19	0.00%
75	1110	15	89.80	8.46	90.50	0.18	0.78%
75	1387	15	83.00	10.27	83.00	0.17	0.00%
75	555	23	203.00	2.46	206.18	0.51	1.57%
75	832	23	173.20	4.71	174.58	0.45	0.80%
75	1110	23	153.40	11.71	153.90	0.43	0.33%
75	1387	23	139.60	6.51	140.04	0.47	0.32%
75	555	30	289.80	1.81	292.90	1.12	1.07%
75	832	30	228.00	2.57	229.92	0.99	0.84%
75	1110	30	208.00	4.07	209.90	0.97	0.91%
75	1387	30	198.00	7.53	199.40	0.92	0.71%
100	990	10	59.40	5.28	59.46	0.16	0.10%
100	1485	10	46.80	9.80	46.92	0.16	0.26%
100	1980	10	42.40	16.84	42.52	0.16	0.28%
100	2475	10	42.40	34.81	42.40	0.15	0.00%
100	990	20	138.40	11.63	140.10	0.64	1.23%
100	1485	20	111.00	34.22	111.54	0.60	0.49%
100	1980	20	110.60	48.98	111.40	0.60	0.72%
100	2475	20	96.80	40.00	98.02	0.57	1.26%
100	990	30	236.80	7.79	240.70	1.96	1.65%
100	1485	30	202.80	11.95	205.08	1.78	1.12%
100	1980	30	174.40	48.78	176.28	1.71	1.08%
100	2475	30	163.80	109.93	165.00	1.59	0.73%
100	990	40	324.40	5.78	329.42	4.67	1.55%
100	1485	40	287.60	12.42	291.84	4.37	1.47%
100	1980	40	229.80	17.77	231.90	3.82	0.91%
100	2475	40	228.40	40.50	230.94	3.89	1.11%
150	2235	15	77.00	47.50	77.46	1.00	0.60%
150	3352	15	66.60	150.67	67.12	0.97	0.78%
150	4470	15	60.20	300.98	60.26	0.93	0.10%
150	5587	15	56.80	383.15	57.46	0.92	1.16%
150	2235	30	180.00	182.17	181.96	4.52	1.09%
150	3352	30	145.60	210.39	146.60	4.27	0.69%
150	4470	30	134.80*	1042.86	136.82	4.15	1.50%
150	5587	30	130.00	1322.56	131.56	3.92	1.20%
150	2235	45	307.20	91.18	317.04	15.83	3.20%
150	3352	45	248.40	413.83	253.56	14.24	2.08%
150	4470	45	222.00	501.15	224.58	12.50	1.16%
150	5587	45	200.80	750.11	203.32	13.15	1.25%
150	2235	60	424.60	78.92	436.40	43.06	2.78%
150	3352	60	344.40	191.68	352.32	38.00	2.30%
150	4470	60	303.40	771.11	309.20	37.39	1.91%
150	5587	60	289.60*	1742.00	294.46	34.80	1.68%

Table 4 Computational results of VNS and ILP2-UE for the ACSP-UE problem on the new set of instances.

For each row, we present average values (in terms of objective function value and computational times in second) of all the computational tests performed for each scenario and for each of the two approaches.

The results reported in the GAP column show the effectiveness of VNS, that often finds either optimal solutions or very close ones. In particular, given the 74 scenarios that are solved by optimality by ILP2-UE, we can see that the same solution is also found by VNS 23 times. Moreover, the gap value is lower than 1% for 53 out of 76 scenarios, and lower than 2% for 71 out of 76 scenarios. The gap value is higher than 3% only twice.

It is worth noting that on the smallest scenarios, with up to 75 nodes, the gap value is lower than 1% on 44 out of 46 scenarios; a single scenario ($n = 50$, $m = 612$, $k = 5$) among them has a peak corresponding to 3.54%, however we can note that the related solution values are small and therefore, in absolute terms, the solution values are not very far also in this case (22.60 for ILP2-UE, 23.40 for VNS). Overall, the instance characteristics do not appear to influence the VNS performances in this case.

On the largest scenarios, gap value peaks occur instead on sparse scenarios with a number of colors equal to $\lceil 0.3n \rceil$ or $\lceil 0.4n \rceil$. In particular, the highest gaps can be noted for the following scenarios: $n = 100$, $m = 990$, $k = 30$ (1.65%); $n = 100$, $m = 990$, $k = 40$ (1.55%); $n = 150$, $m = 2235$, $k = 45$ (3.20%); $n = 150$, $m = 2235$, $k = 60$ (2.78%). The easiest scenarios for VNS are generally the ones containing less colors, where the gap values are almost always lower than 1%.

Regarding the performances, we can see that VNS runs in less than 5 seconds for the scenarios with up to 100 nodes, while for the largest ones it runs within 44 seconds. The parameter that mainly affects the performance is the number of colors. This was expected, since a higher number of colors leads to longer high-level solutions and therefore to a higher number of relocate and 2-opt operations. We can see that, for instance, all scenarios with $k = \lceil 0.1n \rceil$ are solved within 1 second, regardless of the number of nodes. The instances with a number of colors equal to $\lceil 0.2n \rceil$, $\lceil 0.3n \rceil$ and $\lceil 0.4n \rceil$ are instead solved within 5, 16 and 44 seconds, respectively.

6 Conclusions

In this work we presented a mathematical formulation and a VNS metaheuristic to solve the ACSP problem. The VNS algorithm takes advantage of the concept of high-level solution, a fixed-length representation of any feasible solution. Our approaches are also used to solve a variant of the problem with unconstrained endpoints (ACSP-UE). The computational results show that the VNS outperforms some previously introduced heuristics for ACSP, and that it is able to find accurate solutions in fast computational times for ACSP-UE. With respect to future developments, we intend to further study the problem and develop efficient exact approaches, possibly based on the branch & cut strategy. The development of new, more effective metaheuristics could also represent an interesting research direction.

Acknowledgements

The authors wish to thank H. Akcan, who provided the set of benchmark instances proposed in [3].

References

1. H. Akcan and C. Evrendilek. Complexity of energy efficient localization with the aid of a mobile beacon. *IEEE Communications Letters*, 22(2):392–395, 2018.
2. M. B. Akçay, H. Akcan, and C. Evrendilek. All colors shortest path problem on trees. *Journal of Heuristics*, DOI: 10.1007/s10732-018-9370-4, 2018.
3. Y. Can Bilge, D. Çagatay, B. Genç, M. Sari, H. Akcan, and C. Evrendilek. All colors shortest path problem. arXiv:1507.06865.
4. B. Bontoux, C. Artigues, and D. Feillet. A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem. *Computers and Operations Research*, 37(11):1844–1852, 2010.
5. F. Carrabs, R. Cerulli, P. Festa, and F. Laureana. *On the Forward Shortest Path Tour Problem*, volume Optimization and Decision Science: Methodologies and Applications: ODS, Sorrento, Italy, September 4-7, 2017, pages 529–537. Springer International Publishing, Cham, 2017.
6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
7. V. Dimitrijević and Z. Šarić. An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs. *Information Sciences*, 102(1-4):105–110, 1997.
8. M. Dror, M. Haouari, and J. Chaouachi. Generalized spanning trees. *European Journal of Operational Research*, 120(3):583 – 592, 2000.
9. C. Feremans, M. Labbé, and G. Laporte. A comparative analysis of several formulations for the generalized minimum spanning tree problem. *Networks*, 39(1):29–34, 2002.
10. C. Feremans, M. Labbé, and G. Laporte. The generalized minimum spanning tree problem: Polyhedral analysis and branch-and-cut algorithm. *Networks*, 43(2):71–86, 2004.
11. P. Festa, F. Guerriero, D. Laganà, and R. Musmanno. Solving the shortest path tour problem. *European Journal of Operational Research*, 230(3):464–474, 2013.
12. M. Fischetti, J. J. Salazar González, and P. Toth. The symmetric generalized traveling salesman polytope. *Networks*, 26(2):113–123, 1995.
13. M. Fischetti, J. J. Salazar González, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.
14. B. Golden, S. Raghavan, and D. Stanojević. Heuristic search for the generalized minimum spanning tree problem. *INFORMS Journal on Computing*, 17(3):290–304, 2005.
15. M. Haouari, J. Chaouachi, and M. Dror. Solving the generalized minimum spanning tree problem by a branch-and-bound algorithm. *Journal of the Operational Research Society*, 56(4):382–389, 2005.
16. B. Hu, M. Leitner, and G. R. Raidl. Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. *Journal of Heuristics*, 14(5):473–499, 2008.
17. E. Ihler, G. Reich, and P. Widmayer. Class steiner trees and vlsi-design. *Discrete Applied Mathematics*, 90(1–3):173 – 194, 1999.
18. G. Laporte, H. Mercure, and Y. Nobert. Generalized travelling salesman problem through n sets of nodes: the asymmetrical case. *Discrete Applied Mathematics*, 18(2):185 – 197, 1987.
19. Y.-S. Myung, C.-H. Lee, and D.-W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995.

20. T. Öncan, J.-F. Cordeau, and G. Laporte. A tabu search heuristic for the generalized minimum spanning tree problem. *European Journal of Operational Research*, 191(2):306–319, 2008.
21. P. C. Pop, W. Kern, and G. Still. A new relaxation method for the generalized minimum spanning tree problem. *European Journal of Operational Research*, 170(3):900–908, 2006.
22. X.H. Shi, Y.C.Liang, H.P.Lee, C.Lu, and Q.X.Wang. Particle swarm optimization-based algorithms for TSP and generalized TSP. *Information Processing Letters*, 103(5):169–176, 2007.
23. L. V. Snyder and M. S. Daskin. A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research*, 174(1):38–53, 2006.