RAIRO Operations Research
Will be set by the publisher

A REDUCTION HEURISTIC FOR THE ALL-COLORS SHORTEST PATH PROBLEM

Francesco Carrabs[1], Raffaele Cerulli[1] and Andrea Raiconi[1]

Abstract. The All-Colors Shortest Path is a recently introduced NP-Hard optimization problem, in which a color is assigned to each vertex of an edge weighted graph, and the aim is to find the shortest path spanning all colors. The solution path can be not simple, that is it is possible to visit multiple times the same vertices if it is a convenient choice. The starting vertex can be constrained (ACSP) or not (ACSP-UE). We propose a reduction heuristic based on the transformation of any ACSP-UE instance into an Equality Generalized Traveling Salesman Problem one. Computational results show the algorithm to outperform the best previously known one.

Keywords: All-Colors Shortest Path Problem, Equality Generalized Traveling Salesman Problem, E-GTSP, Heuristic

Introduction

The All-Colors Shortest Path (ACSP) is a recently introduced combinatorial optimization problem. Given an undirected and edge-weighted graph, a logical attribute (defined *color*) is assigned to each of its vertices. The aim is to find a minimum-weight path, starting from a predefined source vertex, that spans at least a vertex for each color. Note that such a path could be non-simple, given that depending on the structure of the input graph, it may contain multiple occurrences of the same vertex. Multiple vertices for some colors may also be traversed.

The use of colors to add another layer of information on graphs has been formalized long ago, in its two main variants - vertex colored graphs and edge colored graphs, and their power in modeling different types of problems has been extensively discussed. In particular, the problem of finding special paths in edge colored graphs is discussed in [3], [4], [6].

The problem has several applications (see for instance [2], [5], [7]). In particular, it is suited to model path planning for routes on network topologies (such as urban roads) in which crossing the same location multiple times may be convenient or even necessary. We can consider, for instance, a problem related to the distribution or collection of items. In this scenario, vertices corresponding

...

[1] Department of Mathematics, University of Salerno. Email: {fcarrabs,raffaele,araiconi}@unisa.it

to the same color represent a natural way to model alternative stores or deposits where each class of items can be picked up or delivered. Alternatively, colors may represent alternative candidate locations in which such facilities may be constructed. In an outdoor exploration scenario, a mobile unit may have to collect samples or readings related to various terrain types, or other environmental conditions that can be found in sub-regions of a geographical area of interest. In this case, sub-regions with the same features would be marked by the same color. Finally, we note that in [1] the problem was linked to the modeling of optimal routes of a mobile beacon used to aid trilateration in a wireless sensor network offering a localization service.

ACSP was first proposed in [5]. In this work, the problem was proven to be NP-Hard and not approximable within a constant factor unless $P = NP$. Furthermore, the authors proposed an ILP formulation and six heuristics. In more detail, they presented three metaheuristics, using the Simulated Annealing, Ant Colony Optimization and Genetic paradigms respectively, and three iterative rounding heuristics based on the LP relaxation of the proposed formulation. In [2] the authors analyzed ACSP in the case in which the input graph is a tree. They proved that the problem remains difficult, and presented specialized heuristics.

In [7] the authors introduced a variant of ACSP named, All-Colors Shortest Path with Unconstrained Endpoints (ACSP-UE), in which the source vertex is unspecified, meaning that both endpoints may correspond to any vertex of the graph. The authors proposed a mathematical formulation and a Variable Neighborhood Search (VNS) metaheuristic, adaptable to solve both ACSP and ACSP-UE with simple modifications. The VNS algorithm was experimentally proven to outperform all algorithms proposed in [5] on their proposed dataset for ACSP. Furthermore, a new, larger dataset was created to test the performances of the VNS algorithm for ACSP-UE. A key concept underlying this algorithm is the one of *high-level* solution representation, which is resumed in Section 1. Furthermore, the authors provided a polynomial-time transformation from each of the two problems to the other, such that the optimal solution for ACSP (resp. ACSP-UE) can be trivially obtained from an optimal solution for ACSP-UE (resp. ACSP) on an appropriately built graph.

In this work, we prove that any ACSP-UE instance can be transformed, again in polynomial time, into a symmetric Equality Generalized Traveling Salesman Problem (E-GTSP, see [9]) one. The transformation takes advantage of the high-level solution concept. In particular, we show that from any feasible (resp. optimal) E-GTSP solution for the transformed problem we can easily derive a feasible (resp. optimal) ACSP-UE solution for the original one, with identical cost. Note that, while we focus on the ACSP-UE problem, the transformation from ACSP to ACSP-UE proposed in [7] also suggests a possible transformation from ACSP to E-GTSP.

The E-GTSP problem is a variant of the well-known symmetric Traveling Salesman Problem (TSP), in which vertices are partitioned into clusters, and the tour corresponding to any feasible solution has to include exactly one vertex for each cluster. In the symmetric Generalized Traveling Salesman Problem (GTSP) variant, the solution may possibly contain multiple vertices for each cluster. In the paper, we will always refer to the symmetric versions of these problems unless differently specified.

Both GTSP and E-GTSP are NP-Hard, since they contain TSP in the special case in which all clusters are singletons. Furthermore, the two problems are equivalent when the edge costs of the input graph satisfy the triangle inequality; indeed, in this case the optimal GTSP solution always contains exactly one vertex for each cluster (see [10]).

E-GTSP and GTSP have been both widely studied in the literature; see for instance [8], [9], [10], [14], [15], [17], [18], [19]. A heuristic method with remarkable performances for E-GTSP, called GLKH, was proposed in [13].

In this paper we propose a general resolution framework for ACSP-UE that involves the transformation of the input instance into an E-GTSP one and the resolution of the resulting problem with an appropriate algorithm. In particular, we show that when GLKH is used for this step, an effective heuristic for ACSP-UE can be obtained. Indeed, computational results show that this approach outperforms the VNS algorithm proposed in [7] in terms of both computational time and solution quality.

The rest of the work is organized as follows. A formal definition of ACSP-UE is provided in Section 1, along with a description of the concept of high-level solution and some related properties. The transformation from ACSP-UE to E-GTSP is described in Section 2. The proposed algorithm, based on this transformation, is discussed in Section 3. Section 4 resumes the results of our computational experiments, while final remarks are contained in Section 5.

1. Problem definition and properties

ACSP-UE is defined on connected, undirected, edge-weighted and vertex-colored graphs. Let $G = (V, E, C)$ be such a graph. Each vertex belonging to $V = \{v_1, \ldots, v_n\}$ is assigned to a color belonging to $C = \{c_1, \ldots, c_k\}$ $(k \leq n)$, while a positive weight $\omega(e_i) \in \mathbb{R}^+$ is associated to every edge e_i belonging to $E = \{e_1, \ldots, e_m\}$.

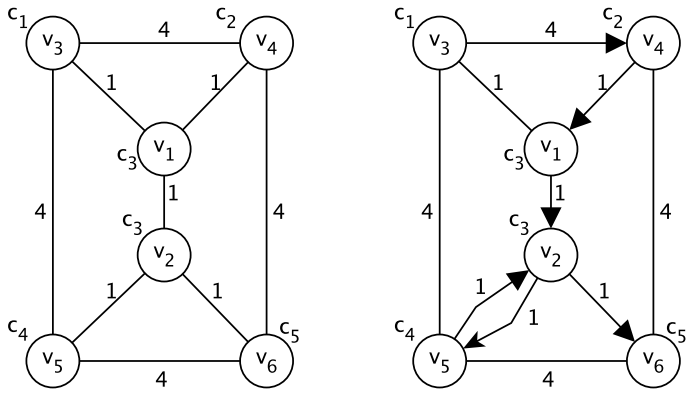
The ACSP-UE objective is to find a path $p = [v_1^p, \ldots, v_h^p]$ in G (that is, $\{v_i^p, v_{i+1}^p\} \in E \ \forall i \in \{1, \ldots, h-1\}$), such that the sum of the weights of its edges is minimized, and at least a vertex is traversed for each color of C.

As mentioned in the Introduction, feasible solutions for ACSP-UE do not contain a predefined number of vertices and edges, since they can correspond to non-simple paths with possibly multiple vertices for each color. However, a compact, fixed-length representation for any ACSP-UE solution was introduced in [7]. In more detail, given the solution $p = [v_1^p, \ldots, v_h^p]$, we define its *high-level representation* to be the sequence of $k = |C|$ vertices $\hat{p} =< v_1^{\hat{p}}, \ldots, v_k^{\hat{p}} >$, corresponding to the first occurrence of each color in p, in the same order in which they are encountered in the path. Note that $\hat{p}$ may be interpreted as a simple path defined on a complete graph with set of vertices V. We associate to $\hat{p}$ the same weight of the original solution p it refers to, and to each of its edges the sum of the weights of the edges that it replaces in p.

An example graph G with 6 vertices and 5 colors is shown in Figure 1(a), where weights and colors are reported next to the related edges and vertices, respectively. A feasible ACSP-UE solution is, for instance, $p = [v_3, v_4, v_1, v_2, v_5, v_2, v_6]$, with cost 9. The path p is shown in Figure 1(b), where edges belonging to the solution are shown as directed arcs to illustrate the visiting order of the vertices. The related high-level representation is $\hat{p} =< v_3, v_4, v_1, v_5, v_6 >$. The solution transformation in its high-level representation is shown in Figure 1(c).

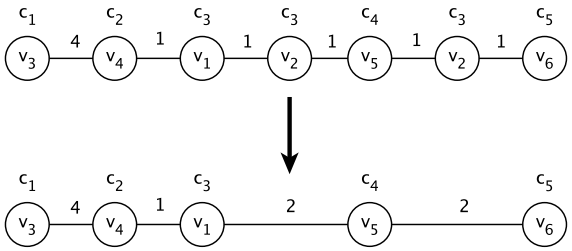
Finally, we present two properties proven in [7] regarding optimal solutions and their high-level representation:

Proposition 1. *Let $p = [v_1^p, \ldots, v_h^p]$ be an optimal ACSP-UE solution, and let $\hat{p} =< v_1^{\hat{p}}, \ldots, v_k^{\hat{p}} >$ be its high-level representation. The two paths share the same endpoints; that is, $v_1^p = v_1^{\hat{p}}$ and $v_h^p = v_k^{\hat{p}}$.*



(a) Example graph G

(b) ACSP-UE Solution



(c) High-level representation

FIGURE 1. ACSP-UE instance and feasible solution

Proposition 2. *Let $p = [v_1^p, \ldots, v_h^p]$ be an optimal ACSP-UE solution, and let $\hat{p} =< v_1^{\hat{p}}, \ldots, v_k^{\hat{p}} >$ be its high-level representation. Any two consecutive vertices $v_i^{\hat{p}}$ and $v_{i+1}^{\hat{p}}$ in $\hat{p}$ are connected in p by their shortest path computed on the input graph G.*

2. Transformation from ACSP-UE to E-GTSP

In this section we prove that it is possible to transform, in polynomial time, the ACSP-UE problem into the E-GTSP one. Moreover, we will describe a general approach to derive ACSP-UE solutions from E-GTSP ones that takes advantage of this transformation, as well as the heuristic for E-GTSP that we embedded in this algorithm.

We first report here the formal definition of E-GTSP. Let $G' = (V', E')$ be a complete and edge weighted graph in which the vertices are partitioned in mutually exclusive clusters. E-GTSP consists of finding a minimum-cost tour that starts and ends into a depot, and visits exactly one vertex for each cluster.

We note that, like high-level solutions for ACSP-UE, E-GTSP ones have fixed length, containing exactly one vertex for each cluster. This observation, along with the properties of optimal ACSP-UE solutions reported in Propositions 1 and 2, is the main intuition underlying the proposed transformation.

Let $G = (V, E, C)$ be an input graph for the ACSP-UE problem ($k = |C|$). We build a complete, undirected graph $G' = (V', E')$ with nonnegative edge weights, whose vertices are partitioned in $k + 1$ clusters, as follows:

- V' contains V plus one additional vertex, the depot v_0. Formally $V' = V \cup \{v_0\}$.
- The depot is directly connected to all the other vertices in V with edges whose cost is equal to zero.
- For each couple of vertices $v_i \in V, v_j \in V$, the weight of $\{v_i, v_j\}$ in G' is equal to the weight of their shortest path in G.
- Each vertex in G' is clustered according to its color in G. That is, if $v_i \in V$ is assigned to the j-th color in G ($j = 1, \ldots, k$), then v_i belongs to the j-th cluster in G'.
- The new vertex v_0 is the only element of the $(k + 1)$-th cluster.

It is straightforward to see that the construction of G' can be carried out in polynomial time. Moreover, it is worth noting that the size of G' is not significantly larger than the one of the original graph G, since it is a complete graph that contains just one additional vertex (the depot v_0). These properties are essential in order to obtain a competitive ACSP-UE algorithm based on the proposed transformation.

Another interesting property of G' is that, given that edge weights are obtained through shortest paths among the vertices of G, the triangle inequality always holds.

In the following, given a graph G_1 and a feasible ACSP-UE solution x on it, we use the notation ω^A_{x,G_1} to represent its cost. Similarly, if y is an E-GTSP solution on a graph G_2, its cost will be represented with the notation ω^E_{y,G_2}. We want to demonstrate the following result:

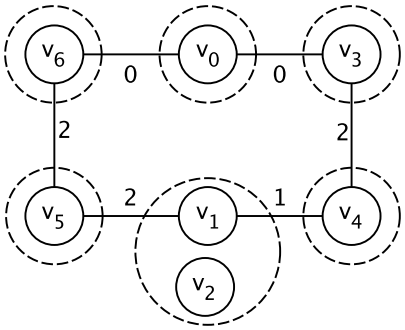
Theorem 1. *There exists a high-level ACSP-UE solution $\hat{p}$ in G with $\omega^A_{\hat{p},G} \leq r$ if and only if there exists a feasible E-GTSP solution t in G' with $\omega^E_{t,G'} \leq r$.*

Proof. $\boxed{\Longrightarrow}$ Let $\hat{p} =< v^{\hat{p}}_1, \ldots, v^{\hat{p}}_k >$ be the high-level representation of a feasible ACSP-UE solution in G, with $\omega^A_{\hat{p},G} \leq r$. Now, let us build the tour $t = [v_0, v^{\hat{p}}_1, \ldots, v^{\hat{p}}_k, v_0]$ in G'. Since t visits exactly one vertex for each cluster, it is a feasible E-GTSP solution. By construction, the edges $\{v_0, v^{\hat{p}}_1\}$ and $\{v^{\hat{p}_k}, v_0\}$ have weight zero. Furthermore, the weight of each edge $\{v^{\hat{p}}_i, v^{\hat{p}}_{i+1}\}$ in t is lower than or equal to the weight of the corresponding edge in $\hat{p}$, since weights in G' correspond to shortest path costs in G. It follows that $\omega^E_{t,G'} \leq \omega^A_{\hat{p},G} \leq r$.
$\boxed{\Longleftarrow}$ Let $t = [v_0, v^t_1, \ldots, v^t_k, v_0]$ be a feasible E-GTSP solution in G' with $\omega^E_{t,G'} \leq r$. We build from t a feasible ACSP-UE solution p in G as follows:

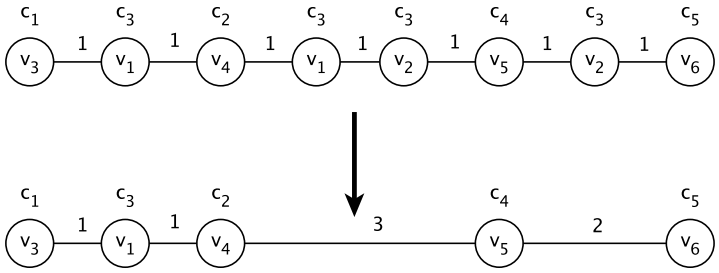
(1) Let v^t_1 be the first endpoint of p;
(2) For $i = 1, \ldots, k - 1$, append to p the shortest path between v^t_i and v^t_{i+1} in G.

It is easy to note that p spans all colors of C, and that it is therefore a feasible solution. Furthermore, by construction, the weight of each shortest path appended to p in Step 2 has the same weight of the edge between the same vertices in t. From p we derive the corresponding high-level solution $\hat{p}$ and we have that $\omega^A_{\hat{p},G} = \omega^E_{t,G'} \leq r$.

□



(a) E-GTSP solution in G'



(b) ACSP-UE solution in G

FIGURE 2. Obtaining an ACSP-UE solution from an E-GTSP one

We illustrate the transformation with an example. Let G' be a complete graph with 7 vertices and 6 clusters, computed as described starting from the graph G shown in Figure 1(a). Furthermore, let $\hat{p}_1$ be the high-level representation of the feasible ACSP-UE solution in G shown in Figure 1(c). The tour $t = [v_0, v_3, v_4, v_1, v_5, v_6, v_0]$ is the E-GTSP solution in G' built from $\hat{p}_1$ using the method described in the proof of Theorem 1, $\boxed{\Longrightarrow}$ implication. This solution is shown in Figure 2(a), where dashed circles represent clusters. We note that, in this case, $\omega^E_{t,G'} = 7 < \omega^A_{\hat{p}_1,G} = 9$. From t, we can build a new feasible ACSP-UE solution in G using the method described in the proof of Theorem 1, $\boxed{\Longleftarrow}$ implication. The new solution is $p_2 = [v_3, v_1, v_4, v_1, v_2, v_5, v_2, v_6]$ and its corresponding high-level solution is $\hat{p}_2 =< v_3, v_1, v_4, v_5, v_6 >$. We show p_2 and $\hat{p}_2$ in Figure 2(b). As expected, $\omega^A_{\hat{p}_2,G} = \omega^E_{t,G'} = 7$. In particular, in the case of this instance, p_2 and t are the optimal solutions for ACSP-UE in G and E-GTSP in G', respectively.

3. A NEW ALGORITHM FOR ACSP-UE

According to Theorem 1 and its proof, it is possible to solve any ACSP-UE problem instance by solving an appropriately defined E-GTSP one. This constitutes a promising line of research, given

the extensive amount of effective exact and heuristic algorithms available in the literature for this well-known problem. In more detail, a general resolution approach for ACSP-UE can be obtained as follows:

(1) Build the new graph G' from G as described in Section 2;
(2) Find a feasible E-GTSP solution t in G', by applying an appropriate algorithm;
(3) Obtain a feasible ACSP-UE solution in G from t, as described in the proof of Theorem 1, $\boxed{\Longleftarrow}$ implication.

For the second step of the algorithm, we opted for the recent heuristic proposed by Helsgaun ([13]), named GLKH. The idea behind this heuristic is to transform the E-GTSP input instance into an asymmetric TSP one, which is then solved using the Lin-Kernighan-Helsgaun (LKH) TSP solver ([11], [12]). LKH is an implementation and improvement of the Lin-Kernighan local search algorithm ([16]), based on variable k-opt neighborhoods, a generalization of 2-opt. GLKH has been proven to have impressive performances on E-GTSP benchmark instances[1]. The pseudocode of the resulting algorithm, that we call A-GLKH (short for ACSP-UE Algorithm embedding GLKH), is provided in Algorithm 1.

Algorithm 1: A-GLKH

Input: Graph $G = (V, E, C)$;
Output: A feasible ACSP-UE solution in G

```
1 G' ← buildNewGraph(G);      // build G' as described in Section 2
2 t ← GLKH (G');               // solve E-GTSP on G'
3 p ← buildPath (G,t);         // build p from t as described in Section 2
4 return p
```

Note that we can transform A-GLKH in an exact approach for ACSP-UE, by simply replacing the GLKH heuristic in step 2 with an exact algorithm for E-GTSP. It can also be noted that, since the triangle inequality always holds for G', and exact algorithm for GTSP can alternatively be used in this case.

4. Computational tests

In this section we compare our A-GLKH algorithm with the VNS algorithm proposed in [7] since, to the best of our knowledge, it is the best performing ACSP-UE algorithm proposed in the literature. A-GLKH is coded in C++, although the GLKH component is written in C. To better evaluate the performances of the heuristics, we also report results obtained by solving, using CPLEX 12.10, the ILP formulation proposed in [7]. In order to obtain as many optimal solutions as possible, we run CPLEX in multithread mode and with a time limit of 3 hours. All tests were performed on a machine with an Intel Core i7 processor running at 3.4 GHz and 8 GB of RAM.

We carried out comparisons on the whole dataset proposed in [7]. In these instances, the number of vertices n belongs to the set $\{25, 50, 75, 100, 150\}$. A density parameter d, ranging in the set $\{0.2, 0.3, 0.4, 0.5\}$, is considered. For a given choice of n and d, the resulting number of edges m is equal to $\frac{n(n-1)}{2} \times d$. Furthermore, the number of colors k belongs to the set $\{\lceil 0.1n \rceil, \lceil 0.2n \rceil, \lceil 0.3n \rceil, \lceil 0.4n \rceil\}$. For each combination of parameters, 5 different random instances

[1]for details, see `http://webhotel4.ruc.dk/~keld/research/GLKH/`

Instances				VNS			A-GLKH		
n	**m**	**k**	**Opt**	**Obj**	**Time**	**Gap**	**Obj**	**Time**	**Gap**
25	60	5	51.4	51.4	0.00	0.00%	51.4	0.47	0.00%
25	90	5	37.6	37.6	0.00	0.00%	37.6	0.37	0.00%
25	120	5	38.4	38.4	0.00	0.00%	38.4	0.50	0.00%
25	150	5	33.2	35.0	0.00	5.42%	33.2	0.45	0.00%
25	60	8	81.0	81.0	0.01	0.00%	81.0	0.78	0.00%
25	90	8	88.4	88.4	0.01	0.00%	88.4	0.99	0.00%
25	120	8	80.0	80.8	0.01	1.00%	80.0	0.92	0.00%
25	150	8	79.2	79.4	0.01	0.25%	79.2	0.97	0.00%
25	60	10	141.8	141.8	0.01	0.00%	141.8	0.80	0.00%
25	90	10	123.2	123.2	0.01	0.00%	123.2	1.19	0.00%
25	120	10	103.2	103.6	0.01	0.39%	103.2	1.22	0.00%
25	150	10	87.8	87.8	0.01	0.00%	87.8	1.72	0.00%
50	245	5	30.6	30.6	0.01	0.00%	30.6	0.70	0.00%
50	367	5	24.8	24.8	0.01	0.00%	24.8	0.65	0.00%
50	490	5	25.8	25.8	0.01	0.00%	25.8	0.77	0.00%
50	612	5	22.6	22.6	0.01	0.00%	22.6	0.85	0.00%
50	245	10	85.2	85.2	0.05	0.00%	85.2	2.23	0.00%
50	367	10	70.2	70.2	0.05	0.00%	70.2	2.11	0.00%
50	490	10	69.2	69.2	0.05	0.00%	69.2	2.42	0.00%
50	612	10	63.8	63.8	0.05	0.00%	63.8	2.85	0.00%
50	245	15	153	153.0	0.12	0.00%	153.0	3.61	0.00%
50	367	15	125.8	125.8	0.13	0.00%	125.8	4.29	0.00%
50	490	15	125.4	126.2	0.12	0.64%	125.4	4.17	0.00%
50	612	15	104.8	105.8	0.10	0.95%	104.8	4.23	0.00%
50	245	20	211.8	214.8	0.21	1.42%	211.8	4.81	0.00%
50	367	20	191.2	191.8	0.22	0.31%	191.2	5.05	0.00%
50	490	20	154.2	154.8	0.21	0.39%	154.2	5.10	0.00%
50	612	20	159.6	159.6	0.21	0.00%	159.6	5.84	0.00%
75	555	8	51.4	51.4	0.07	0.00%	51.4	2.21	0.00%
75	832	8	46.0	46.4	0.07	0.87%	46.0	2.25	0.00%
75	1110	8	36.8	36.8	0.07	0.00%	36.8	2.27	0.00%
75	1387	8	36.8	36.8	0.06	0.00%	36.8	2.38	0.00%
75	555	15	111.4	112.0	0.24	0.54%	111.4	5.02	0.00%
75	832	15	103.2	103.4	0.27	0.19%	103.2	5.31	0.00%
75	1110	15	89.8	89.8	0.26	0.00%	89.8	6.46	0.00%
75	1387	15	83.0	83.6	0.24	0.72%	83.0	6.57	0.00%
75	555	23	203.0	205.8	0.70	1.38%	203.0	8.00	0.00%
75	832	23	173.2	175.0	0.62	1.04%	173.2	7.30	0.00%
75	1110	23	153.4	157.4	0.64	2.61%	153.4	11.26	0.00%
75	1387	23	139.6	139.6	0.66	0.00%	139.6	8.66	0.00%
75	555	30	289.8	294.2	1.57	1.52%	289.8	9.46	0.00%
75	832	30	228.0	230.0	1.41	0.88%	228.0	11.03	0.00%
75	1110	30	208.0	208.4	1.37	0.19%	208.0	12.08	0.00%
75	1387	30	198.0	200.2	1.23	1.11%	198.0	10.12	0.00%

TABLE 1. Computational results on the small instances (1).

were generated. The simplest case $n = 25$, $k = \lceil 0.1n \rceil = 3$ was discarded. Hence, the dataset considered in [7] is composed of 76 scenarios (parameter choices), and 380 individual instances.

Moreover, for this work we considered a new dataset of larger instances, generated by choosing n in the set $\{200, 300, 400\}$, and using the same values for the other parameters. This new dataset is therefore composed of 48 scenarios and 240 instances. In the following, we refer to the original instances (with $n \leq 150$) as *small*, and to the new ones as *large*.

Instances				VNS			A-GLKH		
n	**m**	**k**	**Opt**	**Obj**	**Time**	**Gap**	**Obj**	**Time**	**Gap**
100	990	10	59.4	60.0	0.20	1.01%	59.4	3.89	0.00%
100	1485	10	46.8	46.8	0.19	0.00%	46.8	4.36	0.00%
100	1980	10	42.4	42.4	0.20	0.00%	42.4	4.22	0.00%
100	2475	10	42.4	43.2	0.19	1.89%	42.4	5.06	0.00%
100	990	20	138.4	139.8	0.85	1.01%	138.4	9.10	0.00%
100	1485	20	111.0	112.0	0.81	0.90%	111.2	10.19	0.18%
100	1980	20	110.6	110.6	0.82	0.00%	110.6	11.19	0.00%
100	2475	20	96.8	97.6	0.78	0.83%	96.8	13.28	0.00%
100	990	30	236.8	238.8	2.56	0.84%	237.6	11.95	0.34%
100	1485	30	202.8	205.4	2.32	1.28%	202.8	12.70	0.00%
100	1980	30	174.4	177.0	2.25	1.49%	174.4	14.05	0.00%
100	2475	30	163.8	165.2	2.08	0.85%	163.8	14.90	0.00%
100	990	40	324.4	332.0	5.97	2.34%	324.8	12.06	0.12%
100	1485	40	287.6	290.2	5.34	0.90%	287.8	17.44	0.07%
100	1980	40	229.8	231.6	4.55	0.78%	229.8	14.56	0.00%
100	2475	40	228.4	231.6	4.95	1.40%	228.6	21.87	0.09%
150	2235	15	77.0	77.8	1.11	1.04%	77.0	10.82	0.00%
150	3352	15	66.6	67.0	1.06	0.60%	66.6	13.60	0.00%
150	4470	15	60.2	60.4	1.05	0.33%	60.2	13.25	0.00%
150	5587	15	56.8	56.8	1.04	0.00%	56.8	16.25	0.00%
150	2235	30	180.0	180.6	5.33	0.33%	180.0	19.92	0.00%
150	3352	30	145.6	146.2	5.03	0.41%	146.0	18.12	0.27%
150	4470	30	134.8	136.4	5.02	1.19%	135.0	26.69	0.15%
150	5587	30	130.0	130.4	4.55	0.31%	130.2	24.92	0.15%
150	2235	45	307.2	315.4	17.23	2.67%	308.0	28.61	0.26%
150	3352	45	248.4	254.6	15.53	2.50%	249.4	26.73	0.40%
150	4470	45	222.0	225.8	14.16	1.71%	222.4	32.72	0.18%
150	5587	45	200.8	203.2	15.04	1.20%	201.2	32.99	0.20%
150	2235	60	424.6	433.4	45.87	2.07%	425.0	32.93	0.09%
150	3352	60	344.4	352.0	38.81	2.21%	344.8	36.91	0.12%
150	4470	60	303.4	308.6	39.19	1.71%	303.8	39.40	0.13%
150	5587	60	288.6	298.4	37.93	3.40%	289.6	41.79	0.35%

TABLE 2. Computational results on the small instances (2).

Tables 1 and 2 contain the results of the comparison between VNS and A-GLKH on the small instances. All values reported in the tables are averages computed on the 5 instances corresponding to the same scenario. Under the *Instances* heading, we report the instances characteristics (number of vertices n, number of edges m and number of colors k). The next column (Opt) reports the optimal solution values, provided by CPLEX. Indeed, all small instances were solved to optimality.

The following three columns report, for VNS, the solution value (Obj), the computational time ($Time$) and the percentage gaps (Gap) between the objective function values returned by the algorithm and the Opt values. These gaps are computed by using the formula $100 \times \frac{Obj(VNS) - Opt}{Opt}$. The final three columns report analogous data for A-GLKH.

Looking at the gap values, it can be noticed that both algorithms are very effective, since they find solutions that are either optimal or close to the optimal ones. VNS finds the optimal solutions only for 28 out of 76 scenarios. For 52 scenarios the gap is within 1%, while it is greater than 2% for 8 out of 76 scenarios, with a peak equal to 5.42% ($n = 25$, $m = 150$, $k = 5$). Much better results are obtained by A-GLKH that finds the optimal solutions for 60 out of 76 scenarios. On the

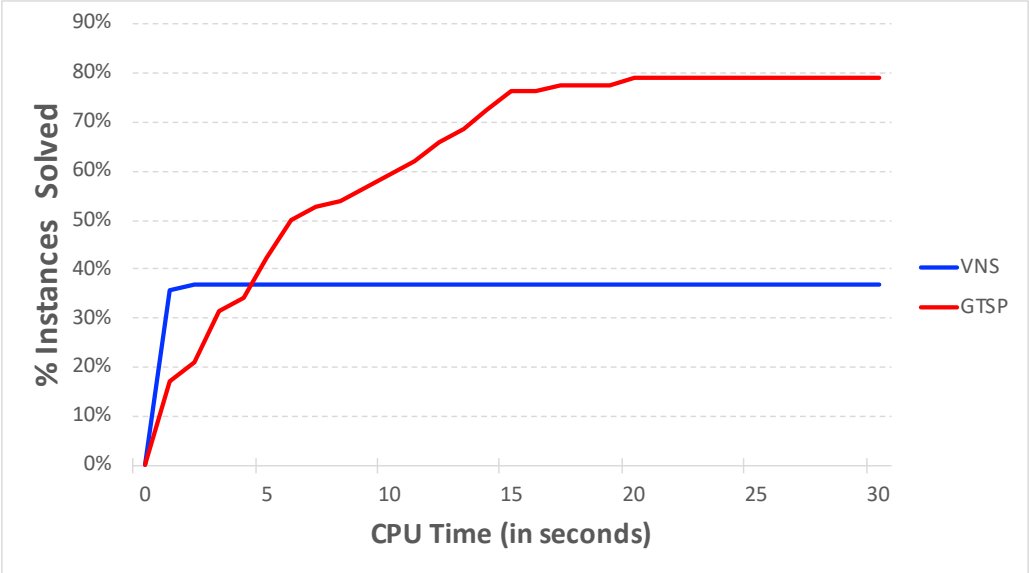


FIGURE 3. Performance comparison between VNS and A-GLKH on the small instances.

remaining 16 cases, its gap is always lower than 0.5%. These results prove A-GLKH to generally outperform VNS in terms of solution quality.

Regarding the computational times of the two algorithms, we note that VNS is always faster than A-GLKH, except for the instances with $n = 150$ and $k = 60$, where the performances of the algorithms are almost identical. However, both algorithms are very fast, always running within 45 seconds.

It is worth noting that the CPU time of both algorithms increases as the number of colors increases. For instance, on the scenarios with 150 vertices we can see that increasing the number of colors from 15 to 60 raises the VNS computational times from ~1 second to ~40 seconds, and the A-GLKH ones from ~10 to ~40 seconds. As also noted in [7], this is an expected behavior for VNS, since its 2-opt and relocate neighborhoods have a size that depends on high-level solutions length, which in turn is equal to k. With respect to A-GLKH, an increase in the number of colors corresponds to an increase in the number of clusters in the related transformed E-GTSP instance. As will be discussed in the comments related to large instances, when the size of the problem grows the performances of VNS get much more affected by the value of k in comparison with A-GLKH.

To conclude the comments on the small scenarios, in Figure 3 we give a representation of the algorithms capability to find optimal solutions. The horizontal axis represents CPU time in seconds, while the vertical axis represents the overall percentage of optimally solved instances. The blue curve is associated to VNS, while the red one is associated to A-GLKH. We note that, overall, around 40% of the instances are solved to optimality by VNS within 3 seconds, but no other instance is solved to optimality by this algorithm. On the other hand, A-GLKH reaches the same threshold (i.e. 40% of optimally solved instances) in 5 seconds, but it manages to solve to optimality almost 80% of the instances in around 20 seconds. This comparison further highlights the higher effectiveness of A-GLKH, which provided optimal solutions for around double the instances with respect to VNS.

The results of the comparison on large instances are reported in Table 3.

For these instances, CPLEX is able to provide optimal solutions only in some cases, mainly corresponding to instances with 200 vertices. For $n = 200$ and $n = 300$, whenever an instance is not solved to optimality, the best solution found (upper bound) is used to compute the average. These CPLEX solution values are reported under the *Opt/UB* heading; scenarios not solved to optimality are marked with the "*" symbol. The values under the *Gap* headings for the two heuristics, for these instances, are computed using the formula $100 \times \frac{Obj(heu) - Opt/UB}{Opt/UB}$, where heu is VNS or A-GLKH. For $n = 400$, the CPLEX upper bounds are not reported, being very far from the optimal solutions and therefore not meaningful. For these instances, gaps are evaluated according the formula $100 \times \frac{Obj(heu) - Obj(minValue)}{Obj(minValue)}$, where *minValue* is the minimum among solution values of VNS and A-GLKH. All other table headings have the same meaning discussed for Tables 1 and 2.

Only 5 instances with 200 vertices, corresponding to 4 scenarios, are not optimally solved by CPLEX within the time limit. We note that A-GLKH finds optimal solutions for all 4 scenarios with $k = 20$. In all other cases, except one (in which it is equal to 1.06%) the gap is below 1%. In one case ($n = 200$, $m = 9950$, $k = 40$) the A-GLKH solution is better than the upper bound provided by CPLEX. On the other hand, VNS finds the optimal solutions for 2 scenarios, and has gaps above 1% in 13 out of 16 cases. For 3 scenarios, the gap is greater than 3%, with a peak of 3.61%.

For $n = 300$, no scenarios are solved to optimality by the solver. We still report the obtained upper bounds as a reference for the quality of the two heuristics. In particular, A-GLKH finds a solution that is better than the upper bound in 12 scenarios, and in the remaining 4 scenarios the gap is below 1.7%. Conversely, VNS finds solutions below the upper bounds in 6 cases. The gap is above 2% in 6 of the remaining 10 cases, with a peak of 4.54%.

Finally, looking at $n = 400$, we note that A-GLKH finds better solutions than VNS for 10 out of 16 scenarios, and worse solutions in the remaining 6. In these 6 cases, the maximum A-GLKH gap is equal to 2.06%, while the gap for VNS grows up to 3.88%. Overall, looking at Table 3, we observe that A-GLKH finds better solutions than VNS for 38 out of 48 scenarios, and solutions of the same quality in 2 scenarios.

The results further highlight the higher effectiveness of A-GLKH. Furthermore, in terms of computational efficiency, the two algorithms behave very differently, in particular when the number of colors increases. For instance, when $n = 400$, $m = 15960$ and k is equal to 40, 80, 120 or 160 the VNS computational times are equal to around 90, 627, 2990 or 10718 seconds, respectively. On the instances with the highest number of vertices and colors, that is $n = 400$, $k = 160$, the VNS computational times are consistently high, being between around 9221 and 10718 seconds. Similar patterns can be seen for the other values of n, where the instances corresponding to the highest k values require the longest computational times. Overall, the results point out that VNS is unfit to solve large instances with a significant number of colors.

On the other hand, A-GLKH appears to be much more scalable, running within 240 seconds in 45 out of 48 scenarios, and in around 400 seconds in the worst case. Overall, the number of colors is again the factor that most influences the performances of the algorithm, but computational times increase at a much slower pace with respect to VNS. Figure 4 highlights this comparison, by showing how average computational times are affected by the k values for the instances with $n = 400$. It is worth noting that, when $k = 160$, A-GLKH is an order of magnitude faster than VNS.

Instances				VNS			A-GLKH		
n	**m**	**k**	**Opt/UB**	**Obj**	**Time**	**Gap**	**Obj**	**Time**	**Gap**
200	3980	20	87.0	88.2	3.36	1.38%	87.0	17.60	0.00%
200	5970	20	81.6	81.6	3.43	0.00%	81.6	23.78	0.00%
200	7960	20	76.0	76.0	3.38	0.00%	76.0	24.24	0.00%
200	9950	20	68.6	69.6	3.27	1.46%	68.6	25.20	0.00%
200	3980	40	218.4	224.8	19.61	2.93%	219.4	36.09	0.46%
200	5970	40	189.0	193.4	18.67	2.33%	190.6	39.51	0.85%
200	7960	40	170.6*	173.6	16.83	1.76%	172.4	43.65	1.06%
200	9950	40	156.2*	157.6	16.23	0.90%	155.8	43.32	-0.26%
200	3980	60	338.2	348.6	67.10	3.08%	340.6	53.09	0.71%
200	5970	60	296.4	305.8	65.07	3.17%	298.0	57.08	0.54%
200	7960	60	257.2*	263.6	58.82	2.49%	258.4	62.19	0.47%
200	9950	60	235.6	241.0	54.44	2.29%	237.4	51.62	0.76%
200	3980	80	465.6	482.4	200.86	3.61%	467.0	105.51	0.30%
200	5970	80	395.0	406.4	179.89	2.89%	396.4	89.44	0.35%
200	7960	80	360.8	370.0	169.05	2.55%	362.2	77.46	0.39%
200	9950	80	332.8*	341.0	165.13	2.46%	335.4	106.25	0.78%
300	8970	30	122.4*	123.2	22.72	0.65%	122.0	42.46	-0.33%
300	13455	30	108.4*	104.0	22.59	-4.06%	105.4	45.03	-2.77%
300	17940	30	116*	99.2	20.97	-14.48%	98.6	53.49	-15.00%
300	22425	30	107*	89.0	21.47	-16.82%	88.6	59.05	-17.20%
300	8970	60	279.4*	286.4	149.29	2.51%	281.0	80.94	0.57%
300	13455	60	240.6*	241.8	137.04	0.50%	239.0	76.51	-0.67%
300	17940	60	214.2*	209.8	139.50	-2.05%	210.8	93.37	-1.59%
300	22425	60	207.8*	198.0	126.27	-4.72%	195.2	94.01	-6.06%
300	8970	90	431*	450.2	616.66	4.45%	438.2	163.52	1.67%
300	13455	90	359.2*	375.4	550.64	4.51%	361.6	131.00	0.67%
300	17940	90	341.8*	342.8	538.44	0.29%	336.2	157.57	-1.64%
300	22425	90	334.2*	329.6	487.06	-1.38%	325.6	141.71	-2.57%
300	8970	120	599.4*	626.6	1996.07	4.54%	604.4	221.99	0.83%
300	13455	120	504.8*	522.2	1856.80	3.45%	504.4	239.56	-0.08%
300	17940	120	463.4*	478.0	1769.19	3.15%	461.2	238.25	-0.47%
300	22425	120	437.4*	446.0	1744.56	1.97%	434.8	235.27	-0.59%
400	15960	40		145.0	90.00	0.00%	146.4	73.54	0.97%
400	23940	40		124.0	85.15	0.49%	123.4	80.85	0.00%
400	31920	40		120.0	81.19	0.00%	120.6	85.28	0.50%
400	39900	40		112.2	78.03	0.72%	111.4	86.89	0.00%
400	15960	80		323.6	627.08	0.00%	326.2	128.10	0.80%
400	23940	80		279.0	583.05	0.00%	284.0	131.99	1.79%
400	31920	80		261.6	567.92	0.00%	267.0	130.33	2.06%
400	39900	80		248.4	515.75	0.00%	251.8	129.87	1.37%
400	15960	120		550.8	2989.98	3.22%	533.6	248.00	0.00%
400	23940	120		460.6	2805.81	1.59%	453.4	223.67	0.00%
400	31920	120		408.4	2612.49	2.30%	399.2	208.50	0.00%
400	39900	120		378.0	2674.47	1.78%	371.4	231.02	0.00%
400	15960	160		770.2	10717.60	3.41%	744.8	373.86	0.00%
400	23940	160		621.2	9928.98	3.88%	598.0	375.08	0.00%
400	31920	160		564.4	9537.74	2.99%	548.0	359.97	0.00%
400	39900	160		523.6	9221.54	2.99%	508.4	400.17	0.00%

TABLE 3. Computational results on the large instances.

5. Conclusion

In this work we proposed a method to transform any ACSP-UE instance into an E-GTSP one, and proved its correctness. Based on this transformation, we proposed a general resolution scheme for ACSP-UE which involves the resolution of the resulting E-GTSP instance. We showed that when the latter is solved through the GLKH algorithm, we obtain a fast and effective heuristic that

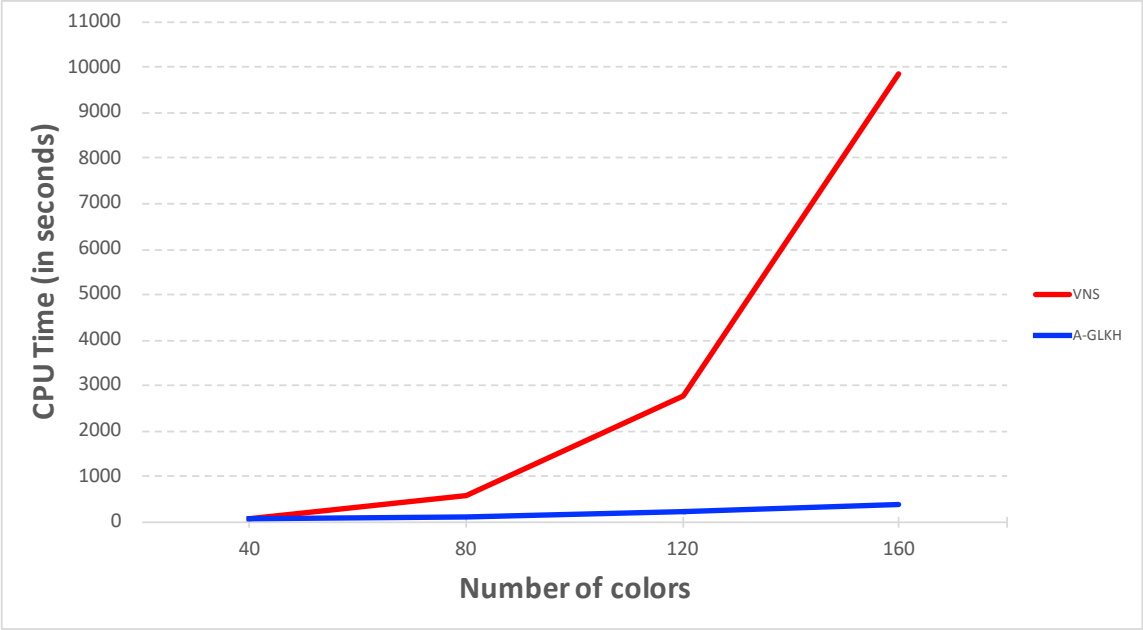


FIGURE 4. Computational times of VNS and A-GLKH on instances with $n = 400$ when the number of colors varies.

outperforms the best one currently known in the literature. The proposed framework would also allow to obtain optimal solutions, by replacing GLKH with any exact algorithm for either E-GTSP or GTSP.

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*, 24(4):617–644, 2018.

[3] A. Benkouar, Y. Manoussakis, V. Paschos, and R. Saad. On the complexity of finding alternating hamiltonian and eulerian cycles in edge-coloured graphs. In *Lecture Notes Computer Science*, volume 557, pages 190–198, 1991.

[4] A. Benkouar, Y. Manoussakis, V.Th. Paschos, and R. Saad. Hamiltonian problems in edge-colored complete graphs and eulerian cycles in edge-colored graphs: Some complexity results. *RAIRO Recherche Operationnelle*, 30(4):417–438, 1996.

[5] Y. Can Bilge, D. Çagatay, B. Genç, M. Sari, H. Akcan, and C. Evrendilek. All colors shortest path problem. arXiv:1507.06865.

[6] F. Carrabs, R. Cerulli, G. Felici, and G. Singh. Exact approaches for the orderly colored longest path problem: Performance comparison. *Computers and Operations Research*, 101:275–284, 2019.

[7] 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.

[8] 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.

[9] M. Fischetti, J. J. Salazar González, and P. Toth. The symmetric generalized traveling salesman polytope. *Networks*, 26(2):113–123, 1995.

[10] 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.

[11] K. Helsgaun. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[12] K. Helsgaun. General k-opt submoves for the Lin–Kernighan TSP heuristic. *Mathematical Programming Computation*, 1(2-3):119–163, 2009.

[13] K. Helsgaun. Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun algorithm. *Mathematical Programming Computation*, 7(3):269–287, 2015.

[14] I. Kara, H. Guden, and O.N. Koc. New formulations for the generalized traveling salesman problem. In *Proceedings of the 6th International Conference on Applied Mathematics, Simulation, Modelling (ASM '12)*, pages 60–65, 2012.

[15] G. Laporte and Y. Nobert. Generalized traveling salesman through n sets of nodes: An integer programming approach. *INFOR*, 21(1):61–75, 1983.

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

[17] 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.

[18] J. Silberholz and B. Golden. The generalized traveling salesman problem: A new genetic algorithm approach. In E.K. Baker, A. Joseph, A. Mehrotra, and M.A. Trick, editors, *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies. Operations Research/Computer Science Interfaces Series*, volume 37, pages 165–181, 2007.

[19] 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.