J Heuristics (2018) 24:617–644
https://doi.org/10.1007/s10732-018-9370-4

All Colors Shortest Path problem on trees

Mehmet Berkehan Akçay[1] · **Hüseyin Akcan**[1] ·
Cem Evrendilek[2]

Received: 25 September 2017 / Revised: 5 March 2018 / Accepted: 28 March 2018 /
Published online: 30 March 2018

Abstract Given an edge weighted tree $T(V, E)$, rooted at a designated base vertex $r \in V$, and a color from a set of colors $C = \{1, \ldots, k\}$ assigned to every vertex $v \in V$, *All Colors Shortest Path problem on trees* (*ACSP-t*) seeks the shortest, possibly non-simple, path starting from r in T such that at least one node from every distinct color in C is visited. We show that *ACSP-t* is NP-hard, and also prove that it does not have a constant factor approximation. We give an integer linear programming formulation of *ACSP-t*. Based on a linear programming relaxation of this formulation, an iterative rounding heuristic is proposed. The paper also explores genetic algorithm and tabu search to develop alternative heuristic solutions for *ACSP-t*. The performance of all the proposed heuristics are evaluated experimentally for a wide range of trees that are generated parametrically.

Keywords NP-hardness · Inapproximability · Integer linear programming · Linear programming relaxation · Genetic algorithm · Tabu search

✉ Hüseyin Akcan
huseyin.akcan@ieu.edu.tr

Mehmet Berkehan Akçay
berkehan.akcay@ieu.edu.tr

Cem Evrendilek
cem.evrendilek@ieu.edu.tr

[1] Department of Software Engineering, Izmir University of Economics, 35330 Izmir, Turkey

[2] Department of Computer Engineering, Izmir University of Economics, 35330 Izmir, Turkey

1 Introduction

In this paper, the All Colors Shortest Path problem on trees (*ACSP-t*) is introduced and explored with respect to its computational characteristics. *ACSP-t* is a variant of the All Colors Shortest Path (*ACSP*) problem first introduced in Bilge et al. (2015). While *ACSP* operates on graphs, *ACSP-t* operates on trees. Given an edge weighted, rooted tree with each node assigned apriori to a color from a set of known colors, *ACSP-t* aims at finding the shortest, possibly non-simple, path starting from the root and visiting every color at least once.

Constraining the input specifically to trees in *ACSP-t* is strongly motivated by both a theoretical interest and the wealth of real world applications. It is well known that many computational problems lend themselves to polynomial time solutions when the input is restricted to trees. Interestingly enough, however, we prove in this paper that this is not the case for *ACSP-t*. Given the fact that *ACSP* is known to be NP-hard (Bilge et al. 2015), *ACSP-t* also turns out to be NP-hard contrary to the expectations, as demonstrated in this paper. Besides, in many real world applications, the underlying network infrastructure is naturally modeled by a tree instead of a full fledged graph. Examples of such networks are road networks, data networks, and aggregation trees in wireless sensor networks. Moreover, employing instead a tree spanning the nodes of a graph is a common approach to reduce the infrastructure cost.

One typical scenario where *ACSP-t* finds application is related to item collection. In this scenario, an agent located at a specific base location is assumed to collect instances of known items with one or more instances of each item distributed randomly among known locations connected in the form of a tree. The objective is to collect at least one instance of each item by traveling the minimum distance from the base location. Once the agent has them all, it needs not move any further.

In another motivational scenario, we have a mobile agent which explores an outdoor area where various terrain types exist. These terrain types might be muddy terrains, roads, sand, meadows, forests with different types of trees, swamps, lakes, etc. The map of the area is known, and the objective of the mobile agent would be, starting from an initially known position, to explore the area, and to collect sensor readings from each of the available terrain types by following the shortest path.

Our main contributions in this paper are:

1. We formally define a computationally unique problem, namely *ACSP-t*, which is a generic problem finding an application in many domains.
2. We show that *ACSP-t* is NP-hard, which is quite an interesting result considering that many computational problems lend themselves to polynomial time solutions when the input graph is restricted to a tree.
3. We prove that there is no constant factor approximation algorithm for *ACSP-t*.
4. An Integer Linear Programming (*ILP*) formulation of *ACSP-t* is provided.
5. Several heuristic solutions based on iterative rounding of a Linear Programming (*LP*) relaxation, genetic algorithm, and tabu search are developed for *ACSP-t*.
6. We conduct an intense experimental study to perform a comparative analysis of the proposed heuristics.

7. Through experimentation, we observe and report a correlation between the node to color ratio and the difficulty of different instances of the problem.

The rest of the paper is organized as follows: In Sect. 2, we discuss the related work and position our paper with respect to the state of the art. In Sect. 3, we introduce the problem formally and prove its NP-hardness along with an inapproximability result. The *ILP* formulation of the problem and a heuristic based on its relaxation to *LP* is presented in Sect. 4. In Sect. 5, two metaheuristic solutions for *ACSP-t* are developed based on genetic algorithm and tabu search. In Sect. 6, we present and compare the results of all the proposed heuristics. Finally, the paper is concluded in Sect. 7.

2 Related work

The *ACSP* problem on general graphs has been shown to be NP-hard and not to admit a constant factor approximation algorithm in Bilge et al. (2015). Restricting the underlying graph to a tree is generally expected to change the computational characteristic of the problem so that it lends itself to a polynomial time solution. Contrary to the expectations, *ACSP-t* is shown to remain NP-hard in this paper.

Although there are several similar problems examined in the literature, *ACSP-t* is computationally unique. Generalized Minimum Spanning Tree (GMST) introduced by Myung et al. (1995) is probably the closest problem. Given an undirected graph $G = (V, E)$ with its vertex set partitioned into m clusters *GMST* is defined to be the problem of finding the Minimum Spanning Tree that visits exactly one node from every cluster. This problem has been shown to be NP-hard in Myung et al. (1995). Some inapproximability results for it are presented by Pop (2004). In Feremans et al. (2002), Pop et al. (2006, 2001), Integer Linear Programming formulations of *GMST* are also proposed. Feremans et al. (2002) study the polytope associated with the *GMST* problem. A recent study in Pop et al. (2018) proposes a two level solution to *GMST* advancing the state of the art. *GMST* problem when the underlying graphs is restricted to trees has been studied by Pop, and has been shown to be NP-hard in Pop (2002). Another variant by Dror et al. (2000), called *l-GMST*, relaxes *GMST*, and allows more than one node from every cluster to be visited. They also present different heuristic solutions including a genetic algorithm. Although *l-GMST* appears similar to *ACSP-t*, they differ computationally in various ways. In Fig. 1, solutions of *l-GMST* and *ACSP-t* for a given problem instance are presented. Each node, in this figure, is labeled with i/c where i corresponds to the node and c to the color assigned to it. The tree is rooted at r to which color 0 is assigned. All the edge weights are assumed to be equal to one. When the given figure is viewed as an instance of *l-GMST*, the optimal solution is the subtree enclosed in a dashed rectangle with a cost of 4 as shown in the figure. The best that could be obtained from this optimal as also a solution to *ACSP-t* when the subtree is appropriately interpreted will have a cost of 7. Yet, the optimal solution to *ACSP-t* as shown to the right of Fig. 1 has cost 5. It is observed through this simple construction that the shape of the solution, in other words, whether the minimum cost combinatorial structure that is sought is a tree or a walk has a considerable impact on how it is computed.

Another problem similar to *ACSP-t* is the Generalized/Group Steiner Tree Problem (*GSTP*) introduced by Reich and Widmayer (1990). *GSTP* is defined on a complete,

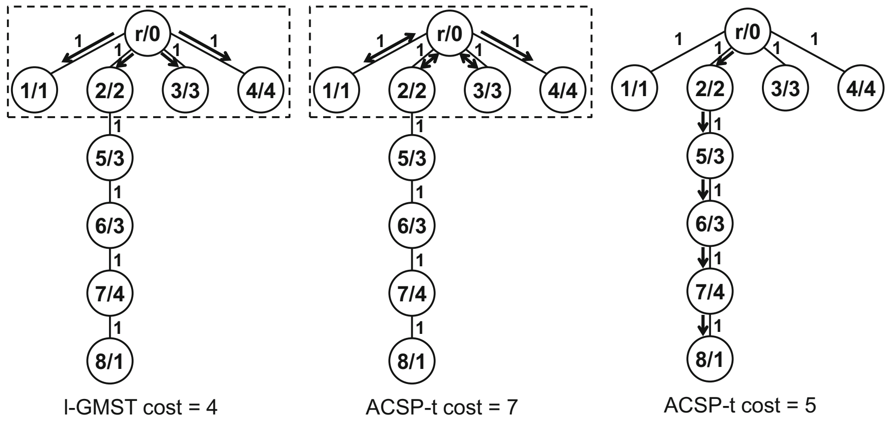

Fig. 1 The optimal solution to the *l-GMST* instance shown in the rectangular area (left), the corresponding solution to the corresponding *ACSP-t* instance as obtained from the solution above (middle), and the optimal solution to the corresponding *ACSP-t* instance (right)

edge weighted, undirected graph G with a subset of nodes S partitioned into m clusters to find the minimum cost tree in G that contains at least one node from each cluster. This NP-hard problem is shown to be a direct generalization of the set cover problem in Ihler (1992), Klein and Ravi (1995), and Slavik (1997). Ihler et al. (1999) show that the problem is NP-hard even on trees. Garg et al. (2000) introduce a polylogarithmic approximation algorithm for this problem. In Halperin et al. (2003), *GSTP* is proved to be not approximable to within $\Omega(\log^{2-\epsilon} n)$ unless NP admits a quasi-polynomial time Las Vegas algorithm.

The Generalized Traveling Salesman Problem (*GTSP*) formulated by Labordere (1969) is another problem that has similar features to the *ACSP-t* problem. Given a graph with the vertex set partitioned into m disjoint clusters, *GTSP* is, then, finding the shortest Hamiltonian tour containing exactly one or at least one node from each cluster. Laporte and Nobert (1983) prove that "exactly one" version corresponds to "at least one" node version when the distance matrix is Euclidean. They also develop the first ILP formulation for *GTSP*. A dynamic programming formulation is proposed as a solution procedure in Srivastava et al. (1969). In Laporte et al. (1987), an ILP formulation to this problem is presented when the distance matrix is asymmetrical. A hybrid diploid genetic based algorithm for solving *GTSP* has also been proposed in Pop et al. (2017). Lien et al. (1993) show that a given instance of *GTSP* can be transformed into an instance of the standard Traveling Salesman Problem (Lawler 1985) efficiently with the same number of nodes.

3 Definition and computational complexity of *ACSP-t*

In this section, we give a formal definition of the *ACSP-t* problem, prove its NP-hardness, and make some observations leading to an inapproximability result for *ACSP-t*.

Definition 1 Given a tree $T(V, E)$ rooted at $r \in V$, a function $w : E \rightarrow \mathbb{R}^+$ assigning positive weights to the edges, and another function $color : V \rightarrow C$ mapping vertices in $V = \{1, \ldots, n\}$ to colors in $C = \{1, \ldots, k\}$, *ACSP-t* is then to find the shortest, possibly non-simple, path starting from the root $r \in V$ such that every distinct color gets visited at least once.

For instances of *ACSP-t* where each node is colored with a distinct color ($k = n$), the problem reduces to finding a shortest non-simple path that traverses all the nodes in the tree. In that case, the solution can be found in polynomial time as suggested by Proposition 1 below.

Proposition 1 *Given an edge weighted tree $T(V, E)$ rooted at a node $r \in V$, a shortest possibly non-simple path Π starting at r and visiting all the nodes can be constructed by a depth-first traversal initiated at r in such a way that the nodes on the way to the leaf farthest away from r are visited last in Π. Furthermore, the length of Π is strictly less than twice the cost of the tree T.*

Proof The depth-first traversal has the effect of traversing every edge in T twice with the exception of the edges on the way from the root r to the last visited leaf, which are traversed only once. The total length of any such path is thus $2 \cdot \sum_{e \in E} w_e - \sum_{e \in \Pi'} w_e$, where Π' is the unique path from r to the leaf that gets visited last. This expression is obviously minimized when the second term is maximized since the first term corresponds simply to the cost of the tree T irrespective of the order in which the nodes are visited.

As there exists at least one edge that is visited exactly once given that the return to the base is not performed upon hitting the last leaf node in T, the second term is always non-zero. $\square$

We prove that *ACSP-t* is NP-hard by a polynomial time reduction from the Hitting Set Problem (*HSP*) (Garey and Johnson 1979). *HSP* is known to be NP-hard (Garey and Johnson 1979), and also a variant of the well-known Set Cover (*SC*) problem (Vazirani 2001).

HSP: Given $X = \{x_1, x_2, \ldots, x_n\}$ as a base set, $k \in N^+$, and a collection of m sets $S_1, S_2, \ldots, S_m$ with $S_i \subseteq X$, the objective of *HSP* is to find $Y \subset X$ such that $|Y| \leq k$ and $\forall i \; S_i \bigcap Y \neq \emptyset$ hold.

A given instance of *HSP* can be transformed into the corresponding instance of *ACSP-t* as follows. The color set C is initialized to have $n + m + 1$ colors as $C = \{c_0, c_1, \ldots, c_{n+m}\}$. We first create the root r such that $color(r) = c_0$. For each $x_i \in X$, two new nodes x_i and x_i' are created, with the same color, c_i, assigned to both as $color(x_i) = color(x_i') = c_i$. Lastly, for each $S_i \in \{S_1, \ldots, S_m\}$ in the given instance of *HSP*, we create $|S_i|$ nodes. This replication of the nodes ensures that a tree structure is maintained in the subsequent construction. For each element $x_j \in S_i$, the corresponding node $S_{i,j}$ is created ($S_{i,j} = x_j$ iff $x_j \in S_i$). All the nodes $S_{i,j}$ for a given i, are colored by the same color c_{n+i}.

Once the nodes with the corresponding colors are created, the tree in the corresponding instance of *ACSP-t* is constructed by connecting them as shown in Fig. 2.

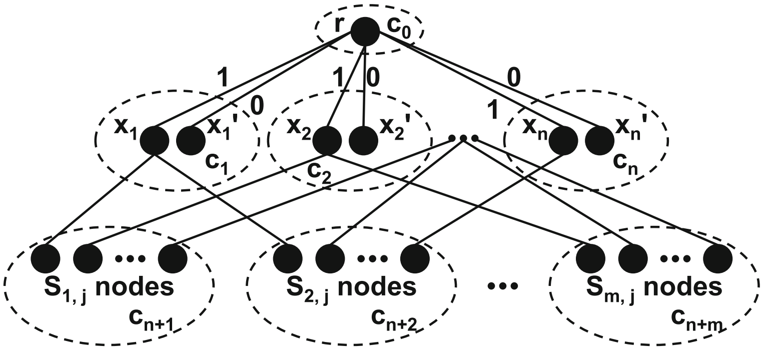

Fig. 2 Reduction from Hitting Set to *ACSP-t*

Each node x_i is connected to the root r with an edge of cost (weight, distance) one while node x_i' is connected to r with an edge of cost zero. For each node $S_{i,j}$, an edge between $S_{i,j}$ and x_j as its parent is created with weight zero.

All the colors must be visited at least once. The color 0 is visited at the root r. For colors c_1 through c_n, either x_i or x_i' can be visited. As there are no edges from nodes x_i' to any $S_{j,i}$ for $j \in \{1 \ldots m\}$, however, at least one or more nodes, without a prime, labeled x_i must be visited in order to get to the colors c_{n+1} through c_{n+m}.

This transformation is obviously polynomial in the size of the given *HSP* instance. A total of $1 + 2n + mn$ nodes including the root are created in the worst case, assuming each S_i covers all the elements in the base set X. So the entire transformation to construct the corresponding tree takes $O(mn)$ time which is directly proportional to the size of $S = \{S_1, S_2, \ldots, S_m\}$ in the given instance of *HSP*.

Lemma 1 *A given instance of HSP has a solution with size less than or equal to k if and only if the corresponding instance of ACSP-t has a solution of path length less than or equal to* $2k - 1$.

Proof (If part): If there is a solution P in the corresponding instance of *ACSP-t*, obtained through the transformation described, with path length less than or equal to $2k - 1$, then by choosing $Y = \{x_i | x_i \in P\}$ we obtain a *hitting set* with size less than or equal to k.

(Only if part): If a *hitting set* Y with $|Y| \leq k$ exists in the given instance of HSP, a *DFS* (Depth First Search) traversal of the subtree starting at r constrained to the nodes $x_i \in Y$, and their descendants only, has a cost less than or equal to $2k - 1$. □

Theorem 1 *ACSP-t is NP-hard.*

Proof The transformation is polynomial. This coupled with Lemma 1 readily proves the theorem. □

It is shown in Ihler et al. (1999) that *l-GMST*, referred to as the *CLASS TREE* problem there, does not admit a constant factor polynomial time approximation algorithm, even when the underlying graph is restricted to be a tree. Equipped with this knowledge, we can make the following similar observation for *ACSP-t*, in the same way it has been previously formulated for *ACSP* in Bilge et al. (2015).

Observation 2 *For a given valid instance I of l-GMST on trees (l-GMST-t),*

$$OPT_{\text{l-GMST-t}}(I) \leq \min_{J \in V}\{OPT_{\text{ACSP-t}}(I_j)\} < 2 * OPT_{\text{l-GMST-t}}(I)$$

where I_j is the corresponding instance of ACSP-t obtained by designating $j \in V$ as the root.

Proof We prove the two inequalities separately for the given expression. Let us assume, by contradiction, that $OPT_{\text{l-GMST-t}}(I) > \min_{j \in V}\{OPT_{\text{ACSP-t}}(I_j)\}$. In this case, *l-GMST-t* can simply adopt the solution that gives the minimum over all such instances for *ACSP-t*. All it takes is to cast the non-simple path to a tree by disregarding any duplicate edges, and hence a contradiction.

Let us assume the latter inequality does not, once more, hold, and

$$\min_{j \in V}\{OPT_{\text{ACSP-t}}(I_j)\} \geq 2 * OPT_{\text{l-GMST-t}}(I).$$

But we know that the optimal solution of *l-GMST-t* is a tree spanning all colors, and a *DFS* traversal of all nodes in it gives a non-simple path with length strictly less than twice the cost of this tree. This obviously is at least as good as a solution to one of the instances I_j of *ACSP-t*, contradicting the assumption. □

Theorem 3 *There is no constant factor polynomial time approximation (apx) for ACSP-t unless $P = NP$.*

Proof Let us assume, contrary to the theorem, that there is such an algorithm $apx_{\text{ACSP-t}}$ satisfying $apx_{\text{ACSP-t}}(I) \leq c * OPT_{\text{ACSP-t}}(I)$ for all valid instances I, and a constant $c > 1$. Now, given an instance I of *l-GMST-t*, let us feed I_j obtained by designating j as the root in the corresponding *ACSP-t* instance into apx_{ACSP-t} for each $j \in V$. We know, by Observation 2, that $OPT_{\text{l-GMST-t}}(I) \leq \min_{J \in V}\{OPT_{\text{ACSP-t}}(I_j)\} < 2 * OPT_{\text{l-GMST-t}}(I)$.

As $\forall j \in V \; apx_{\text{ACSP-t}}(I_j) \leq c * OPT_{\text{ACSP-t}}(I_j)$ holds by the assumption made,

$$\min_{j \in V}\{apx_{\text{ACSP-t}}(I_j)\} \leq c * \min_{j \in V}\{OPT_{\text{ACSP-t}}(I_j)\} < 2 * c * OPT_{\text{l-GMST-t}}(I)$$

is readily obtained. This, by definition, indicates the existence of a $2c$ approximation for *l-GMST-t*, and hence a contradiction as it certainly takes polynomial time to run apx_{ACSP-t} n times to record the minimum over all the possible instances of *ACSP-t*, each of which is rooted at a distinct node. □

4 Integer linear programming formulation of ACSP-t

In this section, an *ILP* formulation is first developed for *ACSP-t*. Then, we relax it to *LP* and propose a heuristic based on iterative rounding of this *LP* relaxation.

4.1 ILP model

In order to give a compact *ILP* formulation, we transform each undirected edge $\{i, j\} \in E$ into two directed edges (i, j) and (j, i). The binary variable $x_{i,j}$ is then set to 1 if and only if the directed edge (i, j) is visited in the solution. Both of these directed edges (i, j) and (j, i) are assigned the same weight $w_{\{i,j\}}$. We also introduce two more nodes as the source and the sink. While the source is denoted by 0, the sink is numbered as $n + 1$. These two nodes are assigned to a brand new color 0. We add a directed edge $(0, r)$ from the source to the original root of weight zero, as well as edges $(i, n+1)$ for all $i \in V$ each with a weight of zero. This ensures that the sink node $n + 1$ is the last node visited in any feasible solution. When all the colors are visited, the edge to node $n+1$ is taken, and the path terminates. Now the transformed instance has $C' = C \cup \{0\}$, $V' = V \cup \{0, n + 1\}$, $E' = \{(0, r)\} \cup \{(i, n + 1)|i \in V\} \cup \{(i, j), (j, i)|\{i, j\} \in E\}$, and the weight and color functions, using the same notation as before, have been augmented so that $color(0) = color(n + 1) = 0$, $w_{0,r} = 0$, $w_{i,n+1} = 0\ \forall i \in V$, and finally $w_{i,j} = w_{j,i} = w_{\{i,j\}}\ \forall\{i, j\} \in E$.

The ILP formulation then follows:

$$\text{minimize} \sum_{(i,j):(i,j)\in E'} x_{i,j} w_{i,j} \tag{4.1}$$

subject to

$$\sum_{j:(j,i)\in E'} x_{j,i} - \sum_{j:(i,j)\in E'} x_{i,j} = 0, \quad \forall i \in V \tag{4.2}$$

$$\sum_{(i,j):(i,j)\in E' \wedge color(j)=c} x_{i,j} \geq 1, \quad \forall c \in C \tag{4.3}$$

$$x_{parent(parent(i)),parent(i)} \geq x_{parent(i),i}, \quad \forall i \in V - \{r\} \tag{4.4}$$

$$x_{i,j} \in \{0, 1\}, \quad \forall (i, j) \in E' \tag{4.5}$$

Our objective function in 4.1 computes the length of a feasible path in an effort to minimize it. The result is the sum of the costs of the selected edges. In order to restrict the shape of the solution to a continuous single non-simple path, Constraints (4.2) are used to ensure that the number of the edges that enter into a node is equal to the number of the edges that leave. In order to overcome the difficulty of dealing with exceptional nodes such as the root r and the last node on a feasible path, the source and the sink have been introduced. Constraints (4.3) ensure that each color is visited at least once. Constraints (4.4) are used to enforce the connectivity of the nodes. It is actually a class of sub-tour elimination constraints. This class of constraints ascertains that a node cannot be visited before its parent. It should be noted that, for all $i \in V - \{r\}$, the $parent(i)$ is assumed to be already defined and unique. The parent of the root, on the other hand, is set to be the source node ($parent(r) = 0$) for Constraints (4.4) to work correctly. Constraints (4.5) dictate that all the decision variables are either 0 or 1 in any feasible solution.

This *ILP* formulation is readily relaxed to an *LP* formulation by replacing the integrality constraints (4.5) with a weaker constraint class (4.6) as shown below to ensure that each variable is in the [0, 1] interval.

$$0 \leq x_{i,j} \leq 1, \quad \forall (i, j) \in E' \tag{4.6}$$

4.2 LP-relaxation based heuristic

We propose a *LP* relaxation based iterative rounding heuristic. In the heuristic which we name *LP-iterative*, the rounding of variables are done in a decreasing order of their values iteratively. We find the largest $x_{i,j}$ with a non-integral value, round it to 1, and add $x_{i,j} = 1$ to the current *LP* modal before a subsequent call to *LP* is issued. This process is repeated until all $x_{i,j}$ values become either 0 or 1. In the worst case, *LP* is called $O(n)$ times. If there are edges with equal $x_{i,j}$ values, we break the ties in favor of those minimizing the $distance/\#colors$ ratio where $distance$ is the total distance to get to this edge, and $\#colors$ is the number of distinct colors on this path.

5 Metaheuristic algorithms

In this section, we present two metaheuristic approaches for solving the *ACSP-t* problem. In Sect. 5.1, we propose a genetic algorithm based heuristic for *ACSP-t* while, in Sect. 5.2, we present another heuristic based on tabu search.

5.1 Genetic algorithm based heuristic

The Genetic Algorithm based heuristic for *ACSP-t* (*GA*) is presented in four stages, as typical of any genetic algorithm, namely, chromosome encoding, crossover, mutation, and solution construction. The details of these stages are summarized below.

For encoding a chromosome representing a feasible solution for a given *ACSP-t* instance, we use an array of size k, where k is equal to the number of colors. The root r and its corresponding color are not included in the chromosome, since the root is always a part of the solution. Therefore, a chromosome is simply a list of nodes with distinct colors to be visited. Given a chromosome representation, we should be able to efficiently compute the corresponding shortest path. This shortest path obviously corresponds, by Proposition 1, to a depth-first traversal of the minimum cost subtree spanning all the nodes specified in the chromosome representation along with the root. The minimum cost subtree, on the other hand, is simply the union of all the paths connecting the nodes in the chromosome to the root.

The second stage of *GA* is the crossover stage. In this stage, the Roulette Wheel Selection algorithm (Goldberg 1989) is used to find the two parent chromosomes. With a predetermined probability, the two nodes with the same color in the parents are swapped to perform the crossover. This operation is repeated for each individual color and therefore completed in time linear in the number of colors. As the output of the crossover stage, two child chromosomes are created from the two parents.

```
input: $ACSP\text{-}t$ instance identified with $T(V, E)$ rooted at r.
output: Best cost path in population
1: function GA($T, r$)
2:     $Population \leftarrow \{\}$;
3:     for $i = 1$ to populationSize do
4:         $chromosome \leftarrow$ createChromosome(T);
5:         calculateFitness($T, r, chromosome$);
6:         $Population \leftarrow Population \cup chromosome$;
7:     end for
8:     for $i = 1$ to iterationsize do
9:         $selectedParents \leftarrow$ rouletteWheelSelection($Population$);
10:        $children \leftarrow$ Crossover($selectedParents$);
11:        for each child c in $children$ do
12:            r $\leftarrow$ random(0,1) ;
13:            if r $< mutationRate$ then
14:                mutate($c$) ;
15:            end if
16:            calculateFitness($T, r, c$);
17:        end for
18:        remove the 2 worst $chromosomes$ from $Population$;
19:        $Population \leftarrow Population \cup children$;
20:    end for
21:    $bestSolution \leftarrow$ best solution in $Population$;
22:    formPath($T, r, bestSolution$);
23: end function
```

Fig. 3 Genetic algorithm based heuristic for *ACSP-t* (*GA*)

In the mutation phase, with a predefined mutation probability, a subset of the colors are selected randomly and the nodes with that particular color in the chromosome are replaced with another random node of the same color.

The last stage of the genetic algorithm is the solution generation stage. Upon the completion of the iterations in the genetic algorithm, the chromosome with the best fitness value is selected as a solution to the given *ACSP-t* instance.

The pseudo-code of *GA* is given in Fig. 3. The population is generated in lines 3–7. The algorithm iterates as many times as dictated by *iterationSize* global to the algorithm. Parent chromosomes are selected using the Roulette Wheel Selection algorithm in line 9 and crossover is performed in line 10. Two of the worst chromosomes are removed from the population in line 18 and the newly created child chromosomes are added to the population in line 19. Finally, when the iterations are over, the best chromosome is selected in line 21 and the solution is generated in line 22 using a depth-first search traversal as already described.

5.2 Tabu search based heuristic

The proposed tabu search based heuristic is called *Tabu*. A feasible solution denoted by s in *Tabu* is represented as an array of size k, where k is the number of distinct colors. $A(s)$ denotes the set of selected nodes for the particular solution s. Therefore, each node in the solution array is a node with a distinct color. The shortest path

```
input: $ACSP\text{-}t$ instance identified with $T(V,E)$ rooted at $r$
output: Best cost path
 1: function TABU($T,r$)
 2:     $colorselectionFrequency[i] = 0 \ \forall i \in V$;
 3:     $tabuList \leftarrow \varnothing$;
 4:     $tabuTenure \leftarrow \varnothing$;
 5:     $initialSolution \leftarrow$ createInitialSolution(T) ;
 6:     $bestSolution \leftarrow initialSolution$;
 7:     for i = 0 to iterationSize do
 8:         sc ← selectColors();
 9:         update $colorselectionFrequency$ for all colors in $sc$;
10:         $bestCandidate \leftarrow$ neighborhoodSearch($T,sc,initialSolution$);
11:         if $cost(bestCandidate)$ ¡ $cost(bestSolution)$ then;
12:             $bestSolution \leftarrow bestCandidate$;
13:         end if
14:         $initialSolution \leftarrow bestCandidate$;
15:         update $tabulist$;
16:         update $tabuTenure$;
17:     end for
18:     FORMPATH($T,r,bestSolution$);
19: end function
```

Fig. 4 Tabu search based heuristic for *ACSP-t* (*Tabu*)

corresponding to this representation can be found using a depth-first traversal as in *GA*.

The crux of *Tabu* is the neighborhood search, which is performed in two steps: identifying the colors for which the nodes are to be changed and then finding the new paths passing through these new candidate nodes. In the iterative neighborhood search, a lower cost solution is sought using the tabu mechanism. First, a total of c colors is selected from the possible set of all colors and the search is restricted to the nodes having these colors only, denoted by $N^c(s)$. The selection frequency for each color is maintained and the colors are selected with probability proportional to the inverse of their selection frequencies. As a result, a frequently selected color is given a lower chance for selection on the next iteration.

In our implementation, the neighborhood size is selected as $N^3(s)$. Therefore, all $(|c_p|-1) \times (|c_q|-1) \times (|c_r|-1)$ solutions are evaluated as possible neighbors where $|c_i|$ denotes the number of nodes with color c_i, and c_p, c_q, c_r are three distinct colors. By evaluating the neighbors, a candidate list of the nodes is generated, among which the lowest cost solution is selected. A node is included in the candidate list if the move is not a tabu, or the move meets the aspiration criteria. In this version of the algorithm, a simple aspiration criteria is employed in which the tabu moves are accepted if the particular move leads to a solution better than the best solution known. After selecting this best candidate, the tabu list and the tabu tenure are updated to reflect the changes.

The pseudocode of the Tabu Search algorithm is given in Fig. 4. The input to the algorithm is an instance of the *ACSP-t* as a rooted tree, and the output is a valid feasible solution to the problem. Lines 2–4 initializes the parameters. Initial solution is created in line 5, and best solution is initialized in line 6. The algorithm iterates for a predefined number of iterations specified by *iterationSize*. In each iteration, the colors to be modified are selected in line 8 and the color selection frequencies are updated in line 9. The neighborhood search is performed in line 10 and the best solution is updated in lines 11–12. In lines 14–16, the initial solution, the tabu list, and the tabu tenure are updated. Finally, the corresponding shortest, possibly, non-simple path is formed in line 18.

6 Experimental study

In this section, we present an experimental evaluation of the heuristics proposed, based on LP-relaxation, genetic algorithm, and tabu search for the *ACSP-t* problem. All algorithms are implemented using C++ on a computer with an Intel i7 2.79 GHz CPU, 8 GB 1333 MHz DDR 3 RAM, and running on Windows 7 operating system. For solving *ILP* and *LP* models, IBM ILOG CPLEX Optimizer (2015) is used. We conduct our experiments with several types of datasets parametrically generated. The details of the datasets used for the experimentation are presented in Sect. 6.1 and the experimental results are reported in Sect. 6.2. A discussion of the experimental results are then given in Sect. 6.3.

6.1 Datasets

In addition to the number of the nodes and the colors, the trees used in our experiments are classified based on two main criteria, namely the bushiness type and the edge weight distribution. When the number of nodes in a tree is fixed, the bushiness corresponds to the average branching factor and has a direct impact on the height of the tree generated. Three different bushiness types are considered in this paper as listed below:

1. Random (R): It is used to generate the trees in which the branching factor of the nodes are randomly selected. When the bushiness type is random, the branching factor is set to a random value between 5 and 20 for each individual node. The tree height is determined dynamically in this case.
2. Deep (D): The deep trees with a low average branching factor are generated. For deep trees, the branching factor is set to 2 for all the nodes, and a balanced deep tree is created.
3. Shallow (S): It is used to generate the trees with a relatively small height and a high average branching factor that is fixed between 9 and 20 for all the nodes in the tree.

For each bushiness type, four different types of edge weight distributions are used:

1. randomly distributed edge weights (R),
2. all edge weights are set to one (1),
3. decreasing weights from the root to the leaves (D),
4. increasing weights from the root to the leaves (I).

The weights associated with the edges for each edge weight distribution are presented in Table 1. Furthermore, in the experiments, in order to observe the effect of the node to color ratio (n/k), for each tree type generated, three different node to color ratios (2, 4, and 10) are used.

Table 1 Permissible values of edge weights for each edge weight distribution

Distribution	Permissible values	Description
R	1–10	Edge weights are set randomly
1	1	All edge weights are set to one
D	1–10	Starting from level 0, edge weights decrease gradually with depth
I	1–10	Starting from level 0, edge weights increase gradually with depth

Table 2 The complete list of parameters used in generating different types of trees

Parameter	Description	Values
n	Number of nodes	211–1111
k	Number of colors	26–556
b	Bushiness type	S (shallow), D (deep), R (random)
bf	Branching factor	2–20
w	Edge weight distribution	Random (R), 1 (all 1), I (increasing), D (decreasing)

The complete list of the parameters used in generating different types of trees are given in Table 2. In order to uniquely identify the dataset used, each specific type of the tree used throughout the experiments is labeled with a string of the form $nX_1kX_2bX_3bfX_4wX_5$, where X_i with $i \in \{1 \ldots 5\}$ corresponds to the value of the parameter that precedes, used in generating the corresponding dataset.

6.2 Experimental results

The results of the experiments with the algorithms on all the datasets are presented in three parts. In Sect. 6.2.1, we present the results for the trees when the bushiness type is random (random trees). In Sect. 6.2.2, we report the results for shallow trees and in Sect. 6.2.3, we give the results for deep trees.

Along with the conventional tabu search algorithm, *Tabu*, proposed in our paper, we also compare the results of a modified version of the tabu search algorithm by Öncan et al. (2008) for the *GMST* problem. In this new version, which we call *Tabu-VF*, the neighborhood size is changed dynamically by cycling over $N^1(s)$, $N^2(s)$, and $N^3(s)$ as the feasible neighborhood sizes during the neighborhood search.

For *GA*, *Tabu*, and *Tabu-VF*, various parameter settings are evaluated to determine the best values for each parameter. For brevity, however, the parameter tuning section is omitted in this paper. The details of it can be found in Akçay (2015). As a result of the parameter tuning, for *GA*, the iteration size is set to 30,000, the population size is selected as 1000, the crossover rate is set to 0.6, and the mutation rate is set to 0.5. Similarly, for *Tabu* and *Tabu-VF*, the iteration size is set to 50,000 and the tabu tenure

Table 3 The parameters along with their values used in the metaheuristic algorithms

Metaheuristic	Parameter	Value
GA	Iteration size	30,000
	Population size	1000
	Crossover rate	0.6
	Mutation rate	0.5
Tabu, Tabu-VF	Iteration size	50,000
	Tabu tenure	10

is selected as 10. All the experiments are performed using these settings, which are listed in Table 3 for a quick reference. It should be noted that the iteration size values for *GA*, *Tabu*, and *Tabu-VF* have been chosen as a result of the observations made during the parameter tuning phase as described in Akçay (2015). They correspond to the values in the range tested that best account for the solution quality within a feasible amount of time.

For each combination of the bushiness type and the edge weight distribution, we present the comparative results in two different types of tables with respect to (1) the best path costs and (2) the average runtimes of the heuristics. In order to avoid using the long dataset names in referring to individual rows of these tables, we assign a unique acronym to the datasets, which also indexes the rows, as given in the first column of the tables. We repeat the individual tests 10 times for each metaheuristic algorithm. While the best solution returned out of 10 is used in the best path cost tables, the average of the 10 runtimes are reported in the average runtime tables. In an attempt to easily compare the quality of the heuristics, the cost of the solutions they return are normalized with respect to the optimal value presented in the *ILP* column for each dataset. The execution times reported in all the respective tables are in seconds. When the *ILP* runs last longer than that can be afforded, they are terminated and instead the results associated with the corresponding *LP* executions are reported. Such results are appropriately marked with an asterisk in the tables.

The rest of this section presents the experimental results in detail for a versatility of the combinations of the dataset parameters. Therefore, those interested in seeing the overall discussion earlier can safely skip to Sect. 6.3.

6.2.1 Results for random trees

In this section, the results for random trees are presented for two different edge weight distributions, namely random trees with random edge weights, and random trees with all edge weights set to one.

For the best path costs shown in Table 4, *GA*, *Tabu*, and *Tabu-VF* have values that are worse off on the average by a factor of 1.053, 1.055, and 1.082 of the optimal respectively. Although this points at a slightly better solution quality for *GA* compared to *Tabu*, a closer inspection of the table reveals that their relative performance relies heavily on the value of the n/k ratio. As the n/k ratio decreases, the quality

Table 4 Best path costs as a factor of the optimal solution for random trees with randomly distributed weights

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
RR-1	n255k26bRbf5wR	155	1.6839	1.1548	1.1097	**1.1032**
RR-2	n255k64bRbf5wR	511	1.4716	1.0509	**1.0313**	1.0470
RR-3	n255k128bRbf5wR	1131	1.1034	**1.0000**	**1.0000**	1.0053
RR-4	n421k43bRbf7wR	262	1.7137	1.1069	**1.0534**	1.1374
RR-5	n421k106bRbf7wR	780	1.4064	1.0526	1.0269	**1.0218**
RR-6	n421k211bRbf7wR	1933	1.1107	**1.0021**	1.0041	1.0072
RR-7	n511k52bRbf7wR	368	1.7554	1.1332	**1.0000**	1.1196
RR-8	n511k128bRbf7wR	984	1.3923	1.0244	**1.0142**	1.0366
RR-9	n511k256bRbf7wR	2227	1.1486	1.0031	**1.0009**	1.0085
RR-10	n820k82bRbf8wR	557	1.6535	1.0610	**1.0323**	1.0467
RR-11	n820k205bRbf8wR	1659	1.4286	1.0536	**1.0289**	1.0573
RR-12	n820k410bRbf8wR	3954	1.1477	**1.0086**	1.1088	1.1179
RR-13	n1023k103bRbf15wR	685	1.6350	1.1255	**1.0847**	1.0876
RR-14	n1023k256bRbf15wR	2157	1.3848	1.0408	**1.0315**	1.0890
RR-15	n1023k512bRbf15wR	4989	1.1189	**1.0072**	1.1812	1.1920
RR-16	n1111k112bRbf20wR	750	1.8213	1.0813	**1.0347**	1.1107
RR-17	n1111k278bRbf20wR	2340	1.2872	**1.0376**	1.0436	1.0897
RR-18	n1111k556bRbf20wR	5499	1.1200	**1.0105**	1.2035	1.2042

The best quality solutions are shown in bold for each dataset

of the solutions obtained by *GA* gets better compared to *Tabu* and *Tabu-VF*. This is observed specifically on the datasets *RR-6*, *RR-12*, *RR-15*, and *RR-18* when the n/k ratio is 2 in Table 4. Interestingly enough, when $n/k = 10$ *Tabu* obtains better quality solutions compared to *GA*, consistently over the respective datasets. *LP-iterative* returns solutions that are, on the average, farther from the optimal by a factor of 1.710, 1.395, and 1.125 when the n/k ratios are 10, 4, and 2 respectively.

When the n/k ratio increases, the runtime of *ILP* increases dramatically on the datasets *RR-10*, *RR-13*, *RR-14*, *RR-16*, and *RR-17* as shown in Table 5. The average of the runtimes for these five datasets is 2192 s while the average for the rest is only 4.9 s. As *LP-iterative* calls the LP solver multiple times, its average runtime is 319 s, which is high compared to the metaheuristics. *GA*, *Tabu*, and *Tabu-VF* have average runtimes of 39.4, 28.5, and 8.5 s respectively. The runtime of *GA* increases with the number of colors (k) due to the crossover operation. As a result of the neighborhood search in *Tabu*, and *Tabu-VF*, on the other hand, the runtime decreases whenever the n/k ratio decreases.

Random trees with all edge weights set to one We could obtain the optimal values using *ILP* in mostly a feasible time except for the datasets *R1-10*, *R1-13*, and *R1-16*. For these datasets, marked with an asterisk (*) in the corresponding rows, we present

Table 5 Average runtimes for random trees with randomly distributed weights

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
RR-1	n255k26bRbf5wR	**1.641**	17.321	18.322	19.341	8.432
RR-2	n255k64bRbf5wR	**2.145**	36.598	23.455	3.508	2.541
RR-3	n255k128bRbf5wR	**1.293**	79.658	31.558	1.411	1.495
RR-4	n421k43bRbf7wR	**3.841**	41.278	22.613	33.906	8.112
RR-5	n421k106bRbf7wR	**2.988**	101.774	23.802	8.234	5.854
RR-6	n421k211bRbf7wR	**2.227**	167.013	50.488	3.716	2.341
RR-7	n511k52bRbf7wR	**14.637**	62.587	35.744	57.603	17.718
RR-8	n511k128bRbf7wR	**4.726**	143.692	51.953	10.455	6.715
RR-9	n511k256bRbf7wR	**2.048**	255.504	48.584	6.502	6.513
RR-10	n820k82bRbf8wR	750.046	154.392	41.545	101.617	**18.757**
RR-11	n820k205bRbf8wR	16.718	344.059	70.345	18.809	**7.577**
RR-12	n820k410bRbf8wR	**2.741**	618.323	64.361	7.234	7.318
RR-13	n1023k103bRbf15wR	3960.33	248.196	32.643	88.725	**20.289**
RR-14	n1023k256bRbf15wR	491.993	574.221	33.851	13.825	**6.007**
RR-15	n1023k512bRbf15wR	4.497	972.131	52.596	5.218	**4.451**
RR-16	n1111k112bRbf20wR	3941	302.541	29.674	111.567	**15.189**
RR-17	n1111k278bRbf20wR	1818.82	589.667	28.598	14.554	**6.069**
RR-18	n1111k556bRbf20wR	**4.575**	1034.56	48.421	7.605	6.757

Fastest runtimes are reported in bold

the lower bounds obtained by *LP* instead of the actual optimal values in Table 6 and the execution times taken by a call to the respective *LP* in Table 7. *LP-iterative* returns solutions that are, on the average, farther from the optimal by a factor of 1.407, 1.20, and 1.118 when the n/k ratios are 10, 4, and, 2 respectively. For the best path costs shown in Table 6, *GA*, *Tabu*, and *Tabu-VF* have values that are on the average within factor 1.025, 1.042, and 1.056 of the optimal values respectively. It is observed, however, from Table 6 that when $n/k = 10$, *Tabu* outperforms *GA* in solution quality on all the datasets with the exception of the dataset *R1-10*. An inspection of Table 7 reveals that the average running time for *ILP* is 31.3 s excluding the datasets *R1-14* and *R1-17* as well as the ones already marked with an asterisk. *LP-iterative* has a runtime of 411.3 s on the average. *GA*, *Tabu*, and *Tabu-VF* have on the other hand average runtimes of 39, 26.6, and 8.6 s respectively.

6.2.2 *Results for shallow trees*

In this section, we present the results for shallow trees for four different edge weight distributions, namely, randomly distributed edge weights, all edge weights set to one, edge weights decreasing with depth, and edge weights increasing with depth.

Table 6 Best path costs for random trees with all weights set to one

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
R1-1	n255k26bRbf5w1	42	1.4762	1.0238	**1.0000**	1.0238
R1-2	n255k64bRbf5w1	120	1.2167	1.0417	**1.0083**	1.0250
R1-3	n255k128bRbf5w1	256	1.0938	**1.0078**	1.0117	1.0234
R1-4	n421k43bRbf8w1	73	1.2877	1.0411	**1.0137**	1.0411
R1-5	n421k106bRbf8w1	203	1.2808	**1.0197**	1.0443	1.0394
R1-6	n421k211bRbf8w1	410	1.1024	**1.0049**	1.0146	1.0146
R1-7	n511k52bRbf8w1	92	1.4565	1.0326	**1.0217**	**1.0217**
R1-8	n511k128bRbf8w1	240	1.1542	1.0250	**1.0167**	1.0333
R1-9	n511k256bRbf8w1	512	1.1445	1.0195	**1.0156**	1.0391
R1-10	n820k82bRbf8w1	81*	2.3457	**1.9012**	1.9630	1.9630
R1-11	n820k205bRbf8w1	397	1.1864	**1.0327**	1.0353	1.0680
R1-12	n820k410bRbf8w1	824	1.1092	**1.0158**	1.0704	1.0777
R1-13	n1023k103bRbf15w1	104*	2.5000	2.0000	**1.9412**	2.0098
R1-14	n1023k256bRbf15w1	498	1.1807	**1.0442**	1.0542	1.0783
R1-15	n1023k512bRbf15w1	1032	1.1231	**1.0165**	1.1231	1.1231
R1-16	n1111k112bRbf20w1	114*	2.4234	1.9820	**1.9550**	1.9910
R1-17	n1111k278bRbf20w1	541	1.1811	**1.0370**	1.0536	1.0702
R1-18	n1111k556bRbf20w1	1116	1.1344	**1.0197**	1.1452	1.1541

The best quality solutions are shown in bold for each dataset

Shallow trees with randomly distributed edge weights Table 8 presents the optimal values along with the best path costs attained by the heuristics. *LP-iterative* returns solutions that are, on the average, a factor of 1.431, 1.094, and 1.003 of the optimal when the n/k ratios are 10, 4, and, 2 respectively. *GA*, *Tabu*, and *Tabu-VF* have values within a factor of 1.015, 1.038, and 1.051 of the optimal respectively. In Table 9, the average runtimes are reported as 5.1 s for *ILP*, 349.6 s for *LP-iterative*, 39.4 s for *GA*, 22.8 s for *Tabu*, and 6.7 s for *Tabu-VF*.

Shallow trees with all edge weights set to one The optimal values are obtained using *ILP* in a feasible time except for the datasets *S1-4* and *S1-7*. *LP-iterative* returns solutions that are, on the average, a factor of 1.021 and 1.008 away from the optimal when the n/k ratios are 4 and 2 respectively. For the best path costs shown in Table 10, *GA* returns the optimal or the best for all the datasets while *Tabu* and *Tabu-VF* have values worse off by a factor 1.007 and 1.012 of the optimal respectively. In Table 11, the average runtimes are reported as 528.3 s for *LP-iterative*, 37.6 s for *GA*, 22.1 s for *Tabu*, and 8.2 s for *Tabu-VF*.

Shallow trees with edge weights decreasing with depth Among the datasets involved in this case, we could not obtain the optimal values using *ILP* in a feasible time for the datasets *SD-4*, *SD-7*, and *SD-8*. As can be computed from Table 12, *LP-iterative*

Table 7 Average runtimes for random trees with all weights set to one

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
R1-1	n255k26bRbf5w1	**6.591**	18.607	34.957	29.412	13.592
R1-2	n255k64bRbf5w1	**4.006**	45.678	48.245	5.694	5.781
R1-3	n255k128bRbf5w1	**1.171**	87.783	39.295	2.402	2.139
R1-4	n421k43bRbf8w1	15.503	52.293	25.750	38.851	**12.756**
R1-5	n421k106bRbf8w1	10.819	127.487	28.451	6.710	**3.651**
R1-6	n421k211bRbf8w1	2.579	256.484	43.312	3.174	**2.846**
R1-7	n511k52bRbf8w1	23.692	99.524	28.489	47.182	**12.759**
R1-8	n511k128bRbf8w1	10.751	230.505	38.414	8.333	**4.127**
R1-9	n511k256bRbf8w1	**2.968**	38.234	41.985	3.650	3.991
R1-10	n820k82bRbf8w1	2.456*	254.953	21.997	73.859	**13.743**
R1-11	n820k205bRbf8w1	281.860	573.437	50.517	14.117	**7.983**
R1-12	n820k410bRbf8w1	18.353	952.115	54.513	5.124	**4.037**
R1-13	n1023k103bRbf15w1	6.671*	376.979	36.739	110.652	**22.815**
R1-14	n1023k256bRbf15w1	48,268	718.946	48.352	18.315	**8.124**
R1-15	n1023k512bRbf15w1	20.251	1095.26	63.751	7.002	**5.331**
R1-16	n1111k112bRbf20w1	7.001*	428.337	24.862	84.623	**20.541**
R1-17	n1111k278bRbf20w1	7393.44	822.434	28.615	14.721	**6.011**
R1-18	n1111k556bRbf20w1	8.252	1224.04	44.642	4.571	**4.322**

Fastest runtimes are reported in bold

Table 8 Best path costs as a factor of the optimal solution for shallow trees with randomly distributed edge weights

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
SR-1	n421k43bSbf20wR	152	1.3026	**1.0132**	1.0263	**1.0132**
SR-2	n421k106bSbf20wR	573	1.0244	1.0035	**1.0000**	1.0070
SR-3	n421k211bSbf20wR	1592	**1.0000**	**1.0000**	**1.0000**	**1.0000**
SR-4	n820k82bSbf9wR	420	1.4405	1.0381	**1.0143**	1.0595
SR-5	n820k205bSbf9wR	1322	1.1362	**1.0045**	1.0151	1.0272
SR-6	n820k410bSbf9wR	3382	1.0065	**1.0006**	1.0716	1.0585
SR-7	n1111k112bSbf10wR	552	1.5489	1.0453	**1.0308**	1.0815
SR-8	n1111k278bSbf10wR	1849	1.1201	**1.0249**	**1.0249**	1.0552
SR-9	n1111k556bSbf10wR	4532	1.0033	**1.0013**	1.1615	1.1593

The best quality solutions are shown in bold for each dataset

Table 9 Average runtimes for shallow trees with randomly distributed edge weights

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
SR-1	n421k43bSbf20wR	**2.036**	38.681	23.542	23.455	5.853
SR-2	n421k106bSbf20wR	**1.037**	51.822	43.653	3.856	2.452
SR-3	n421k211bSbf20wR	**1.162**	55.836	40.145	1.765	1.784
SR-4	n820k82bSbf9wR	**10.072**	330.252	35.931	76.133	16.131
SR-5	n820k205bSbf9wR	**3.472**	342.550	45.123	12.933	6.318
SR-6	n820k410bSbf9wR	**2.701**	575.765	62.400	5.710	5.663
SR-7	n1111k112bSbf10wR	14.803	351.998	30.567	66.442	**12.023**
SR-8	n1111k278bSbf10wR	7.024	701.217	28.642	11.234	**6.051**
SR-9	n1111k556bSbf10wR	3.791	698.401	44.675	**3.476**	3.648

Fastest runtimes are reported in bold

Table 10 Best path costs as a factor of the optimal solution for shallow trees with all weights equal to one

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
S1-1	n421k43bSbf20w1	82	**1.0000**	**1.0000**	**1.0000**	**1.0000**
S1-2	n421k106bSbf20w1	208	1.0192	**1.0000**	**1.0000**	**1.0000**
S1-3	n421k211bSbf20w1	418	1.0048	**1.0000**	1.0048	1.0048
S1-4	n820k82bSbf9w1	82*	2.0370	**1.9877**	**1.9877**	**1.9877**
S1-5	n820k205bSbf9w1	405	1.0296	**1.0000**	**1.0000**	1.0099
S1-6	n820k410bSbf9w1	815	1.0147	**1.0000**	1.0147	1.0196
S1-7	n1111k112bSbf10w1	111*	2.0270	**1.9730**	**1.9730**	**1.9730**
S1-8	n1111k278bSbf10w1	551	1.0145	**1.0000**	1.0073	1.0145
S1-9	n1111k556bSbf10w1	1109	1.0054	**1.0000**	1.0252	1.0325

The best quality solutions are shown in bold for each dataset

returns solutions that are, on the average, farther from the optimal by a factor of 1.076. For the best path costs shown in Table 12, *GA*, *Tabu*, and *Tabu-VF* return solutions that are on the average within factor 1.028, 1.036, and 1.044 of the optimal respectively. In Table 13, while the average runtime of *LP-iterative* is reported as 408.9 s, *GA* runs in 35.1 s, *Tabu* in 19.9 s, and *Tabu-VF* in 8.5 s.

Shallow trees with edge weights increasing with depth We could not obtain the optimal value for the dataset *SI-4* using *ILP* in a feasible time. *LP-iterative*, in Table 14, returns solutions that are, on the average, farther from the optimal by a factor of 1.019, 1.009, and 1.005 when the n/k ratios are 10, 4, and 2 respectively. For the best path costs shown in Table 14, *GA* always returns the optimal for all datasets while *Tabu* and *Tabu-VF* are within factor 1.006 and 1.008 of the optimal respectively. In Table 15,

Table 11 Average runtimes for shallow trees with all weights set to one

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
S1-1	n421k43bSbf20w1	**5.694**	59.403	24.866	25.591	15.554
S1-2	n421k106bSbf20w1	4.758	135.023	39.031	5.179	**3.713**
S1-3	n421k211bSbf20w1	**1.108**	56.737	41.668	2.371	2.590
S1-4	n820k82bSbf9w1	6.596*	257.926	31.887	58.810	**14.540**
S1-5	n820k205bSbf9w1	19.748	712.413	35.741	9.641	**5.039**
S1-6	n820k410bSbf9w1	**3.812**	772.628	48.703	4.852	4.711
S1-7	n1111k112bSbf10w1	7.126*	360.772	32.786	76.705	**15.584**
S1-8	n1111k278bSbf10w1	26.341	1138.25	33.431	11.281	**6.942**
S1-9	n1111k556bSbf10w1	5.037	1261.77	50.435	**4.649**	4.967

Fastest runtimes are reported in bold

Table 12 Best path costs as a factor of the optimal solution for shallow trees with edge weights decreasing with depth

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
SD-1	n421k43bSbf20wD	520	1.2000	**1.0462**	**1.0462**	1.0615
SD-2	n421k106bSbf20wD	1316	1.0851	**1.0426**	1.0547	1.0699
SD-3	n421k211bSbf20wD	2640	**1.0133**	1.0136	1.0136	1.0136
SD-4	n820k82bSbf9wD	486*	2.4527	2.2881	2.2798	**2.2305**
SD-5	n820k205bSbf9wD	2604	1.0998	1.0445	**1.0292**	1.0461
SD-6	n820k205bSbf9wD	5200	1.0385	**1.0100**	1.0285	1.0269
SD-7	n1111k112bSbf10wD	666*	2.5766	2.2583	**2.2342**	2.2402
SD-8	n1111k278bSbf10wD	2230*	1.6915	1.6556	**1.6502**	1.6771
SD-9	n1111k556bSbf10wD	7024	1.0216	**1.0125**	1.0433	1.0450

The best quality solutions are shown in bold for each dataset

the average runtimes are reported as 397.9 s for *LP-iterative*, 36.5 s for *GA*, 20.9 s for *Tabu*, and 8.1 s for *Tabu-VF*.

6.2.3 Results for deep trees

In this section, we present the results for deep trees for four different edge weight distributions, namely, randomly distributed edge weights, all edge weights set to one, edge weights decreasing with depth, and edge weights increasing with depth.

Deep trees with randomly distributed edge weights The optimal values can be obtained using *ILP* except for the dataset *DR-7*. *LP-iterative* can be seen in Table 16 to return solutions that are, on the average, farther from the optimal by a factor 1.803, 1.314, and

Table 13 Average runtimes for shallow trees with edge weights decreasing with depth

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
SD-1	n421k43bSbf20wD	**6.274**	65.172	32.356	26.482	17.978
SD-2	n421k106bSbf20wD	5.528	133.621	43.212	4.957	**3.131**
SD-3	n421k211bSbf20wD	**1.144**	55.552	45.683	2.886	1.915
SD-4	n820k82bSbf9wD	5.001*	153.442	23.924	53.044	**13.365**
SD-5	n820k205bSbf9wD	142,651	553.068	30.726	7.888	**6.438**
SD-6	n820k205bSbf9wD	5.761	519.719	40.844	**2.934**	4.812
SD-7	n1111k112bSbf10wD	10.965*	272.692	24.211	67.462	**14.544**
SD-8	n1111k278bSbf10wD	8.245*	1045.22	28.902	10.447	**8.734**
SD-9	n1111k556bSbf10wD	9.172	881.823	46.15	**3.444**	5.347

Fastest runtimes are reported in bold

Table 14 Best path costs as a factor of the optimal solution for shallow trees with weights increasing with depth

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
SI-1	n421k43bSbf20wI	278	**1.0000**	**1.0000**	**1.0000**	**1.0000**
SI-2	n421k106bSbf20wI	900	1.0044	**1.0000**	**1.0000**	1.0022
SI-3	n421k211bSbf20wI	1942	1.0010	**1.0000**	1.0010	1.0010
SI-4	n820k82bSbf9wI	371*	1.5768	**1.5067**	**1.5067**	**1.5067**
SI-5	n820k205bSbf9wI	1675	1.0143	**1.0000**	**1.0000**	1.0072
SI-6	n820k410bSbf9wI	3709	1.0097	**1.0000**	1.0135	1.0151
SI-7	n1111k112bSbf10wI	777	1.0386	**1.0000**	**1.0000**	**1.0000**
SI-8	n1111k278bSbf10wI	2337	1.0077	**1.0000**	1.0051	1.0077
SI-9	n1111k556bSbf10wI	5083	1.0035	**1.0000**	1.0315	1.0311

The best quality solutions are shown in bold for each dataset

1.121 when the n/k ratios are 10, 4, and 2 respectively. *GA*, *Tabu*, and *Tabu-VF* can come as close to the optimal as by a factor of 1.019, 1.030, and 1.049 respectively. As computed from Table 17, the average runtimes are reported as 391.5 s for *LP-iterative*, 37.3 s for *GA*, 275 s for *Tabu*, and 5.6 s for *Tabu-VF*.

Deep trees with all edge weights set to one The optimal values could not be obtained using *ILP* in a feasible time for the datasets *D1-7* and *D1-8*. *LP-iterative* returns solutions that are, on the average, farther from the optimal by a factor 1.333, 1.146, and 1.087 when the n/k ratios are 10, 4, and 2 respectively. For the best path costs shown in Table 18, *GA*, *Tabu*, and *Tabu-VF* have values worse off by a factor 1.018, 1.032, and 1.051 of the optimal respectively. In Table 19, the average runtimes are

Table 15 Average runtimes for shallow trees with weights increasing with depth

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
SI-1	n421k43bSbf20wI	**2.587**	82.174	31.124	34.515	18.267
SI-2	n421k106bSbf20wI	3.907	158.575	34.415	5.123	**3.806**
SI-3	n421k211bSbf20wI	**1.123**	63.885	53.214	3.034	3.021
SI-4	n820k82bSbf9wI	7.362*	150.788	20.124	49.872	**12.741**
SI-5	n820k205bSbf9wI	6.967	460.999	40.875	8.277	**4.119**
SI-6	n820k410bSbf9wI	3.851	469.717	45.131	3.599	**2.995**
SI-7	n1111k112bSbf10wI	138.324	255.284	24.746	69.284	**18.523**
SI-8	n1111k278bSbf10wI	8.124	840.613	33.311	10.727	**5.382**
SI-9	n1111k556bSbf10wI	4	1099.42	45.341	4.044	**3.713**

Fastest runtimes are reported in bold

Table 16 Best path costs as a factor of the optimal solution for deep trees with randomly distributed edge weights

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
DR-1	n255k26bDbf2wR	201	2.0647	**1.0000**	**1.0000**	**1.0000**
DR-2	n255k64bDbf2wR	561	1.3939	1.0160	**1.0053**	1.0374
DR-3	n255k128bDbf2wR	1244	1.1270	1.0048	**1.0032**	1.0064
DR-4	n511k52bDbf2wR	396	1.5404	1.0202	**1.0000**	1.0480
DR-5	n511k128bDbf2wR	1150	1.2183	**1.0270**	**1.0270**	1.0287
DR-6	n511k256bDbf2wR	2486	1.1279	1.0068	**1.0032**	1.0193
DR-7	n1023k103bDbf2wR	140*	11.5071	7.4500	**7.1429**	7.1643
DR-8	n1023k256bDbf2wR	2441	1.3290	1.0610	**1.0414**	1.0901
DR-9	n1023k512bDbf2wR	5357	1.1088	**1.0134**	1.1572	1.1581

The best quality solutions are shown in bold for each dataset

reported as 255.5 s for *LP-iterative*, 32.3 s for *GA*, 21.2 s for *Tabu*, and, 6.42 s for *Tabu-VF*.

Deep trees with decreasing edge weights The optimal values could be obtained using *ILP* except for the datasets *DD-7* and *DD-8*. *LP-iterative* returns, in Table 20, solutions that are, on the average, a factor of 1.6, 1.365, and 1.154 away from the optimal when the n/k ratios are 10, 4, and 2 respectively. *GA*, *Tabu*, and *Tabu-VF* have values worse off by a factor 1.072, 1.061, and 1.073 of the optimal respectively. In Table 21, the average runtimes are reported as 39.5 s for *GA*, 21.9 s for *Tabu*, and 6.6 s for *Tabu-VF*. *LP-iterative*, on the other hand, has a relatively higher average runtime of 161.6 s excluding the dataset *DD-9* which runs in 1655.38 s alone.

Table 17 Average runtimes for deep trees with randomly distributed edge weights

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
DR-1	n255k26bDbf2wR	**2.882**	22.651	21.358	20.816	8.127
DR-2	n255k64bDbf2wR	2.467	57.836	23.466	3.590	**2.323**
DR-3	n255k128bDbf2wR	**0.820**	101.757	31.706	1.543	1.541
DR-4	n511k52bDbf2wR	17.035	65.015	20.639	56.129	**8.498**
DR-5	n511k128bDbf2wR	13.391	145.161	29.156	9.517	**3.342**
DR-6	n511k256bDbf2wR	**1.503**	292.502	28.439	4.134	3.213
DR-7	n1023k103bDbf2wR	7.793*	479.848	41.761	121.993	**13.251**
DR-8	n1023k256bDbf2wR	9180.50	917.34	47.564	19.235	**5.819**
DR-9	n1023k512bDbf2wR	7.036	1441	91.322	6.474	**4.045**

Fastest runtimes are reported in bold

Table 18 Best path costs as a factor of the optimal solution for deep trees with all edge weights set to one

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
D1-1	n255k26bDbf2w1	45	1.4000	**1.0000**	**1.0000**	1.0444
D1-2	n255k64bDbf2w1	121	1.1653	1.0165	**1.0000**	**1.0000**
D1-3	n255k128bDbf2w1	257	1.0856	1.0078	**1.0000**	1.0156
D1-4	n511k52bDbf2w1	98	1.2653	1.0408	**1.0204**	1.0612
D1-5	n511k128bDbf2w1	254	1.1260	**1.0315**	1.0472	1.0630
D1-6	n511k256bDbf2w1	520	1.0769	**1.0115**	1.0231	1.0385
D1-7	n1023k103bDbf2w1	102*	2.3824	**2.0490**	2.1078	2.0882
D1-8	n1023k256bDbf2w1	255*	2.3569	**2.1294**	2.1451	2.2000
D1-9	n1023k512bDbf2w1	1061	1.0980	**1.0170**	1.1320	1.1357

The best quality solutions are shown in bold for each dataset

Deep trees with increasing edge weights Among the related datasets, we could not obtain the optimal value using *ILP* for the dataset *DI-7*. According to Table 22, *LP-iterative* returns solutions that are, on the average, farther from the optimal by a factor of 1.11, 1.064, and 1.042 when the n/k ratios are 10, 4, and 2 respectively. *GA*, *Tabu*, and *Tabu-VF* have values worse off by a factor 1.014, 1.025, and 1.034 of the optimal respectively. An inspection of Table 23 reveals that the average running time is 282.9 s for *LP-iterative*, 30.2 s for *GA*, 20 s for *Tabu*, and 6.2 s for *Tabu-VF*.

6.3 Discussion and analysis of the experimental results

For *ACSP-t* instances where each node has a distinct color, the optimal solution can be found in polynomial time by Proposition 1. In support of this fact, the following observation has been made throughout the experiments. The average runtime tables

Table 19 Average runtimes for deep trees with all edge weights set to one

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
D1-1	n255k26bDbf2w1	10.577	14.045	20.867	19.756	**8.533**
D1-2	n255k64bDbf2w1	5.217	40.891	23.571	3.697	**2.449**
D1-3	n255k128bDbf2w1	**1.146**	73.971	30.512	1.732	1.638
D1-4	n511k52bDbf2w1	39.302	57.207	23.856	40.851	**9.314**
D1-5	n511k128bDbf2w1	20.688	154.707	29.171	6.864	**3.775**
D1-6	n511k256bDbf2w1	9.454	275.352	45.181	3.881	**3.511**
D1-7	n1023k103bDbf2w1	7.347*	296.099	27.623	92.895	**16.952**
D1-8	n1023k256bDbf2w1	7.145*	679.947	39.912	16.092	**6.898**
D1-9	n1023k512bDbf2w1	40.712	707.155	49.841	5.054	**4.727**

Fastest runtimes are reported in bold

Table 20 Best path costs as a factor of the optimal solution for deep trees with edge weights decreasing with depth

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
DD-1	n255k26bDbf2wD	277	1.6931	1.0505	**1.0289**	1.0866
DD-2	n255k64bDbf2wD	707	1.4187	1.0396	**1.0226**	1.0820
DD-3	n255k128bDbf2wD	1371	1.1648	1.0160	1.0117	**1.0058**
DD-4	n511k52bDbf2wD	488	1.5082	1.2172	1.1475	**1.0984**
DD-5	n511k128bDbf2wD	1230	1.3106	1.1057	**1.0683**	**1.0683**
DD-6	n511k256bDbf2wD	2298	1.1340	1.0296	**1.0226**	1.0348
DD-7	n1023k103bDbf2wD	204*	5.8627	5.2843	**4.7745**	4.8725
DD-8	n1023k256bDbf2wD	512*	5.2773	4.4336	**4.2070**	4.4102
DD-9	n1023k512bDbf2wD	3652	1.1632	**1.0433**	1.1238	1.1369

The best quality solutions are shown in bold for each dataset

reveal that the average execution time of *ILP* decreases dramatically as the n/k ratio gets smaller. In fact, when $n/k = 2$, a rule of the thumb is to use *ILP* directly as it becomes the fastest for most datasets and reasonably fast for the rest.

It is interesting to note how the runtimes are effected by the change in n/k. When this ratio gets smaller, both *Tabu* and *Tabu-VF* as is the pattern with *ILP* run faster than they do for instances with a large n/k. For such instances, which might be classified as easy in the light of Proposition 1, *Tabu* and *Tabu-VF* can be told to tackle such instances relatively easier. In contrast, the runtimes of *GA* and *LP-iterative* are inversely proportional to this ratio.

Table 24 presents the rankings of the heuristic algorithms proposed with respect to their quality of the solutions. In each cell of the table, the count of the times a specific heuristic gets closest to the optimal is reported for distinct combinations of

Table 21 Average runtimes for deep trees with edge weights decreasing with depth

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
DD-1	n255k26bDbf2wD	**6.145**	12.995	35.464	28.712	7.891
DD-2	n255k64bDbf2wD	7.984	30.981	28.119	4.977	**2.876**
DD-3	n255k128bDbf2wD	2.102	109.687	42.432	**1.592**	1.638
DD-4	n511k52bDbf2wD	12.891	41.543	21.639	43.514	**8.573**
DD-5	n511k128bDbf2wD	36.372	186.061	39.766	6.958	**3.349**
DD-6	n511k256bDbf2wD	18.581	356.265	57.765	3.713	**3.214**
DD-7	n1023k103bDbf2wD	8.988*	445.844	23.725	89.402	**19.202**
DD-8	n1023k256bDbf2wD	10.741*	109.699	25.654	13.326	**5.803**
DD-9	n1023k512bDbf2wD	13.953	1655.38	80.886	**4.227**	6.724

Fastest runtimes are reported in bold

Table 22 Best path costs as a factor of the optimal solution for deep trees with edge weights increasing with depth

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
DI-1	n255k26bDbf2wI	172	1.0814	**1.0000**	**1.0000**	1.0116
DI-2	n255k64bDbf2wI	574	1.0418	1.0174	1.0139	**1.0000**
DI-3	n255k128bDbf2wI	1400	1.0271	**1.0000**	**1.0000**	**1.0000**
DI-4	n511k52bDbf2wI	462	1.1385	1.0216	**1.0000**	1.0173
DI-5	n511k128bDbf2wI	1464	1.0669	1.0164	1.0178	**1.0109**
DI-6	n511k256bDbf2wI	3342	1.0437	1.0132	**1.0108**	1.0197
DI-7	n1023k103bDbf2wI	600*	1.9450	1.8800	**1.8550**	1.8617
DI-8	n1023k256bDbf2wI	3475	1.0829	1.0265	**1.0224**	1.0541
DI-9	n1023k512bDbf2wI	7877	1.0551	**1.0135**	1.1353	1.1582

The best quality solutions are shown in bold for each dataset

the n/k ratio, the bushiness type (b), and the edge weight distribution (w). The very reason that the n/k ratio was incorporated into this table can be attributed to the observation that this ratio has an impact on both the average runtimes as well as the quality of the solutions attained. A quick inspection of Table 24 tells us that there is not a single heuristic that dominates for all combinations of the values of bushiness type, weight distribution, and n/k. *LP-iterative* is easily seen to be not promising as there is not a single occurrence of it in the table. This is actually in conformance with our expectations as *ACSP-t* has been proven in this paper to be constant factor inapproximable.

A more detailed inspection of the table, however, lets us make some additional observations. Drilling down on n/k, for example, lets us specify a different heuristic for distinct values of n/k. When $n/k = 10$, the counts associated with the heuristics

Table 23 Average runtimes for deep trees with edge weights increasing with depth

Dataset acronym	Dataset name	Algorithm				
		ILP	LP-iterative	GA	Tabu	Tabu-VF
DI-1	n255k26bDbf2wI	**3.289**	25.319	24.261	19.932	8.155
DI-2	n255k64bDbf2wI	2.915	38.865	25.727	3.888	**2.591**
DI-3	n255k128bDbf2wI	1.710	96.296	26.645	**1.467**	2.512
DI-4	n511k52bDbf2wI	16.251	71.713	21.178	45.839	**8.751**
DI-5	n511k128bDbf2wI	12.567	138.435	30.972	7.745	**3.495**
DI-6	n511k256bDbf2wI	4.154	371.571	35.492	**2.949**	5.335
DI-7	n1023k103bDbf2wI	6.751*	252.549	27.788	80.831	**12.402**
DI-8	n1023k256bDbf2wI	13.564	524.277	26.641	12.725	**5.319**
DI-9	n1023k512bDbf2wI	27.128	1027.38	53.491	**4.305**	6.873

Fastest runtimes are reported in bold

Table 24 Number of the best results achieved by heuristic algorithms

b	w											
	R			1			D			I		
n/k→	10	4	2	10	4	2	10	4	2	10	4	2
R	TB: 5 VF: 1	GA: 1 TB: 4 VF: 1	GA: 5 TB: 2	GA: 1 TB: 5 VF: 1	GA: 4 TB: 2	GA: 5 TB: 1	–	–	–	–	–	–
S	GA: 1 TB: 2	GA: 2 TB: 2	GA: 3 TB: 1 VF: 1	GA: 3 TB: 3 VF: 3	GA: 3 TB: 2 VF: 1	GA: 3	GA: 1 TB: 2 VF: 1	GA: 1 TB: 2	GA: 3 TB: 1 VF: 1	GA: 3 TB: 3 VF: 2	GA: 3 TB: 2	GA: 3
D	GA: 1 TB: 3 VF: 1	GA: 1 TB: 3	GA: 1 TB: 2	GA: 2 TB: 2	GA: 2 TB: 1 VF: 1	GA: 2 TB: 1	TB: 2 VF: 1	TB: 3 VF: 1	GA: 1 TB: 1 VF: 1	GA: 1 TB: 3	TB: 1 VF: 2	GA: 2 TB: 2 VF: 1

TB Tabu, *VF* Tabu-VF

can be rolled up as 13 for *GA*, 30 for *Tabu*, and 10 for *Tabu-VF*. Averaging the normalized best path costs over all the datasets with known optimal solutions (not marked with an asterisk) when $n/k = 10$ yields 1.0522 for *GA*, 1.0277 for *Tabu*, and 1.0531 for *Tabu-VF*. Obviously, *Tabu* takes the lead qualitywise as reflected by these numbers. However, in order to see the tradeoff between the execution time and the solution quality, we also compute the corresponding average runtimes over the related datasets as 27.6 for *GA*, 46.9 for *Tabu*, and 12.5 for *Tabu-VF*. We can therefore state that, although *Tabu* leads in quality, *Tabu-VF* is a good candidate as long as the runtime is critical and the solutions that are 5.3% away from the optimal are admissible as opposed to those that are 2.8% away.

Considering the datasets when $n/k = 4$, the counts associated with the heuristics can be reported as 17 for *GA*, 22 for *Tabu*, and 6 for *Tabu-VF*. The average best path costs over all the related datasets is calculated as 1.0284 for *GA*, 1.0240 for *Tabu*, and 1.0405 for *Tabu-VF*. Additionally, the corresponding average runtimes can be reported as 35.2 for *GA*, 8.0 for *Tabu*, and 4.3 for *Tabu-VF*. In this case, *Tabu* that gets closest to the optimal also happens to run in a reasonably short time. Hence, *Tabu* could be the preferred choice.

The counts when $n/k = 2$ can be reported as 28 for *GA*, 11 for *Tabu*, and 4 for *Tabu-VF*. The average best path costs over all the related datasets are calculated as 1.0092 for *GA*, 1.0531 for *Tabu*, and 1.0584 for *Tabu-VF*. Additionally, the corresponding average runtimes can be reported as 43.8 for *GA*, 3.6 for *Tabu*, and 3.4 for *Tabu-VF*. Even though in terms of quality *GA* is the best heuristic, the average runtime of *ILP* here is only 3.3 s. Therefore, as discussed above, *ILP* renders other alternatives impractical.

7 Conclusion

In this paper, we introduce a computationally interesting problem, namely the *ACSP* problem on trees, which is generic enough to find an application in many real world domains. We show that *ACSP-t* is NP-hard and prove that there is no constant factor approximation algorithm for it. An *ILP* formulation of *ACSP-t* as well as several heuristic algorithms based on iterative rounding of *LP* relaxation, genetic algorithm, and tabu search are developed. We conduct an extensive experimental study to perform a comparative analysis of all the proposed heuristics for various kinds of trees and edge weight distributions. It is observed from these experiments that there is not a single heuristic that dominates for all combinations of the values of the bushiness type, the weight distribution, and the node to color ratio. Another major observation is made by noting the correlation between the node to color ratio and the difficulty of different instances of the problem. As this ratio gets smaller, the *ACSP-t* instances become easier to solve. When the node to color ratio is relatively large, *Tabu-VF* is a good candidate as long as the runtime is critical. For average values of n/k, *Tabu* that gets closest to the optimal also happens to run in a reasonably short time. For small values of n/k, however, *ILP* renders its alternatives impractical.

As a future work, we plan to investigate new algorithms for the All Colors Shortest Path problem instances whose input is either a graph or a tree. Exploring new heuristics with guaranteed approximation bounds is left as an interesting open problem. Another interesting open problem would be to look for tighter inapproximability bounds.

References

Akçay, M.B.: All colors shortest path problem on trees. Master's thesis, Izmir University of Economics, Izmir (2015)

Bilge, Y.C., Çağatay, D., Genç, B., Sarı, M., Akcan, H., Evrendilek, C.: All colors shortest path problem (2015). arXiv:1507.06865

Dror, M., Haouari, M., Chaouachi, J.: Generalized spanning trees. Eur. J. Oper. Res. **120**(3), 583–592 (2000)

Feremans, C., Labbé, M., Laporte, G.: A comparative analysis of several formulations for the generalized minimum spanning tree problem. Networks **39**(1), 29–34 (2002)

Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)

Garg, N., Konjevod, G., Ravi, R.: A polylogarithmic approximation algorithm for the group Steiner tree problem. J. Algorithms **37**(1), 66–84 (2000)

Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1989)

Halperin, E., Krauthgamer, R.: Polylogarithmic inapproximability. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, pp. 585–594. ACM (2003)

IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/. Accessed 20 July 2015

Ihler, E.: The complexity of approximating the class Steiner tree problem. In: Graph-Theoretic Concepts in Computer Science, pp. 85–96. Springer, Berlin (1992)

Ihler, E., Reich, G., Widmayer, P.: Class Steiner trees and VLSI-design. Discrete Appl. Math. **90**(1), 173–194 (1999)

Klein, P., Ravi, R.: A nearly best-possible approximation algorithm for node-weighted Steiner trees. J. Algorithms **19**(1), 104–115 (1995)

Labordere, H.: Record balancing problem: a dynamic programming solution of a generalized travelling salesman problem. Rev. Fr. Inf. Rech. Oper. **3**(NB 2), 43 (1969)

Laporte, G., Nobert, Y.: Generalized traveling salesman problem through n-sets of nodes—an integer programming approach. Inf. Syst. Oper. Res. **21**(1), 61–75 (1983)

Laporte, G., Mercure, H., Nobert, Y.: Generalized travelling salesman problem through n sets of nodes: the asymmetrical case. Discrete Appl. Math. **18**(2), 185–197 (1987)

Lawler, E.L.: The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley-Interscience Series in Discrete Mathematics. Wiley, New York (1985)

Lien, Y.N., Ma, E., Wah, B.W.S.: Transformation of the generalized traveling-salesman problem into the standard traveling-salesman problem. Inf. Sci. **74**(1), 177–189 (1993)

Myung, Y.S., Lee, C.H., Tcha, D.W.: On the generalized minimum spanning tree problem. Networks **26**(4), 231–241 (1995)

Öncan, T., Cordeau, J.F., Laporte, G.: A tabu search heuristic for the generalized minimum spanning tree problem. Eur. J. Oper. Res. **191**(2), 306–319 (2008)

Pop, P.C.: The generalized minimum spanning tree problem. Ph.D. thesis, University of Twente (2002)

Pop, P.C.: New models of the generalized minimum spanning tree problem. J. Math. Model. Algorithms **3**(2), 153–166 (2004)

Pop, P.C., Kern, W., Still, G.: An approximation algorithm for the generalized minimum spanning tree problem with bounded cluster size. Technical report 1577, Department of Applied Mathematics, University of Twente (2001)

Pop, P.C., Kern, W., Still, G.: A new relaxation method for the generalized minimum spanning tree problem. Eur. J. Oper. Res. **170**(3), 900–908 (2006)

Pop, P.C., Matei, O., Sabo, C.: A hybrid diploid genetic based algorithm for solving the generalized traveling salesman problem. In: Hybrid Artificial Intelligent Systems—12th International Conference, HAIS 2017, Proceedings, La Rioja, Spain, 21–23 June 2017, pp. 149–160 (2017)

Pop, P.C., Matei, O., Sabo, C., Petrovan, A.: A two-level solution approach for solving the generalized minimum spanning tree problem. Eur. J. Oper. Res. **265**(2), 478–487 (2018)

Reich, G., Widmayer, P.: Beyond Steiner's problem: a VLSI oriented generalization. In: Graph-theoretic Concepts in Computer Science, pp. 196–210. Springer, Berlin (1990)

Slavik, P.: The errand scheduling problem. Technical report, Department of Computer Science, SUNY, Buffalo (1997)

Srivastava, S., Kumar, S., Garg, R., Sen, P.: Generalized traveling salesman problem through n sets of nodes. Can. Oper. Res. Soc. J. **7**, 97–101 (1969)

Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2001)