Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 28, no. 1, pp. 81–93 (2024)
DOI: 10.7155/jgaa.v28i1.2929

The Minimum Consistent Spanning Subset Problem on Trees

Ahmad Biniaz *Parham Khamsepour*

School of Computer Science,
University of Windsor, Canada

Submitted: January 2024 Accepted: April 2024 Published: May 2024

Article type: Regular paper Communicated by: A. Wolff

Abstract. Given a vertex-colored edge-weighted graph, the *minimum consistent subset* (MCS) problem asks for a minimum subset S of vertices such that every vertex $v \notin S$ has the same color as its nearest neighbor in S. This problem is NP-complete. A recent result of Dey, Maheshwari, and Nandy (2021) gives a polynomial-time algorithm for the MCS problem on two-colored trees. A *block* is a maximal connected set of vertices of the same color. We introduce a variant of the MCS problem, namely the *minimum consistent spanning subset* problem, for which we require the set S to contain a vertex from every block of the graph such that every vertex $v \notin S$ has a nearest neighbor in S that is in the same block as v. We observe that this problem is NP-hard on general graphs. We present a polynomial-time algorithm for this problem on trees.

1 Introduction

Let G be a simple undirected edge-weighted graph whose vertex set V is partitioned into sets $V_1, \ldots, V_k$. The *distance* between two vertices $u, v \in V$ is defined as the weight of the shortest path between u and v in G. The *Minimum Consistent Subset* (MCS) problem asks for a minimum cardinality subset S of V such that, for every $i \in \{1, \ldots, k\}$ and for every $v \in V_i$, a nearest neighbor of v in S belongs to V_i; see [4, 5, 6]. We stress that v could have multiple nearest neighbors in S but for the purpose of the MCS problem it suffices to have only one of them in V_i. The set S is a *representative* for the structure of the entire graph G. We may assume that the vertices of V are colored by k different colors such that the vertices in each V_i have the same color and the vertices in V_i and V_j, with $i \neq j$, have different colors. Hence the MCS problem asks for a minimum cardinality subset S of V such that for every $v \in V$ a nearest neighbor of v in S has the same color as v. See Fig. 1(a) for an example where all edges have the same weight.

The MCS problem was first introduced by Hart [9] (in 1968) for points in the Euclidean plane. In this version G is assumed to be a complete graph, and the vertices are represented by points in

Research supported by NSERC.

E-mail addresses: abiniaz@uwindsor.ca (Ahmad Biniaz) khamsep@uwindsor.ca (Parham Khamsepour)

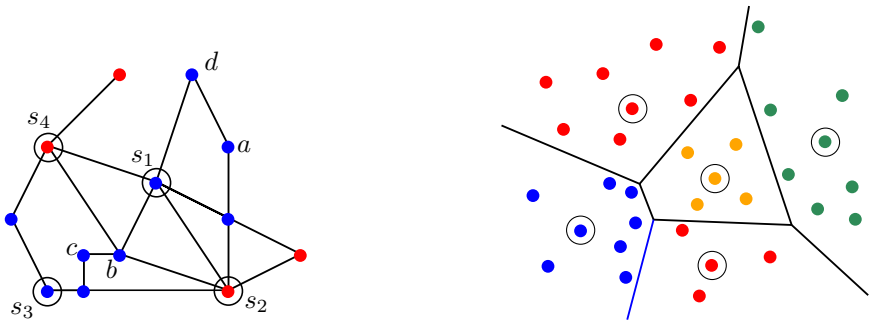

Figure 1: (a) An example of a 2-colored graph G where all edges have the same weight. The set $S = \{s_1, s_2, s_3, s_4\}$ is an MCS for G. The vertex s_1 is a nearest neighbor of a, b, c, and d in S. (b) The Euclidean MCS where the circled points belong to S.

the plane, and edge weights are Euclidean distances between the endpoints [2, 7, 8, 9]. As every vertex of V has the same color as its nearest neighbor in S, in the Voronoi diagram of S all points in each Voronoi cell have the same color as the center of the cell; see Fig. 1(b).

The MCS problem finds applications in solving nearest neighbor problems [3, 7, 8, 13], finding optimal number of clusters in k-clustering problems such as k-means and k-nearest neighbors [6], and finding optimal set of classifiers in classifying algorithms [10]. The MCS problem is also useful in the field of pattern recognition, such as speech and handwriting recognition [6, 11].

When all edges of G have the same unit weight, we say that G is *unweighted*. In this case the distance between two vertices u and v is the number of edges in the shortest path between u and v. The MCS problem is NP-complete even for two-colored unweighted planar graphs [1, 5]; this is shown by a reduction from the minimum dominating set problem. The Euclidean version of the MCS problem is also NP-complete [12] even for two-colored points [11].

There has not been much progress on the MCS problem from the algorithmic point of view. Recently, Dey, Maheshwari, and Nandy [4] solved this problem in polynomial time for some simple two-colored (also known as bicolored and bichromatic) unweighted trees such as paths, caterpillars, spiders, combs with respective running times $O(n)$, $O(n)$, $O(n^2)$, and $O(n^2)$.[1] In a companion paper [5] they present an $O(n^4)$-time algorithm for general two-colored unweighted trees. See Fig. 2(a) and 2(c) for examples of MCS on bicolored trees.

A minimum consistent subset for a bicolored tree may consist of only two vertices, no matter how large the tree is; see for example the tree in Fig. 2(c). For the purpose of clustering and classifications such a solution does not accurately reflect the structure of the entire tree. To capture the entire structure of the tree we need a stronger version of the MCS problem which we introduce below.

We define a *block* to be a maximal connected set of vertices of the same color in a tree. The tree in Fig. 2(b) consists of seven blocks denoted $B_1, \ldots, B_7$. The solution of the MCS problem may not contain representatives (i.e., vertices) from all blocks in the tree; see Fig. 2(a) and Fig. 2(c). Therefore, a minimum consistent subset may not capture the structure of the entire tree. In order to have a better representative for the tree we introduce a more constrained version of the MCS

[1] A caterpillar is tree in which every vertex is within distance one from a path called skeleton. A spider is a tree that is the union of disjoint paths connected to a vertex called center. A comb is a tree consisting of a path called skeleton in which each vertex is connected to another path.

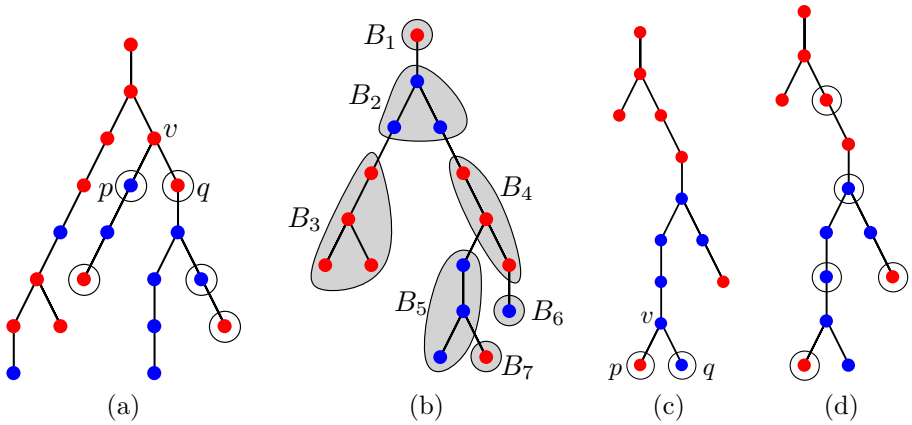

Figure 2: (a) An MCS for a two-colored unweighted tree. (b) Blocks in a tree. (c) An MCS for an unweighted tree that has size 2, and (d) an MCSS for the same tree.

problem. In this version, which we call the *minimum consistent spanning subset* (MCSS) problem, the solution S must contain at least one vertex (i.e., a representative) from each block of the tree such that every vertex $v \notin S$ has a nearest neighbor in S that is in the same block as v. A solution to the MCSS problem spans over all blocks in the tree. The constraint of having a nearest neighbor in the same block is natural for clustering purposes. See Fig. 2(d) for an example of an MCSS of a two-colored tree. In this paper we first observe that the MCSS problem is NP-hard for general graphs. Then we turn our attention to trees. We present a simple 2-approximation algorithm for the MCSS problem on trees. Our main contribution, which is summarized below, is a polynomial-time algorithm that solves the problem optimally on trees.

Theorem 1 *A minimum consistent spanning subset on a vertex-colored weighted tree with n vertices can be computed in $O(n^4)$ time.*

In Section 2 we review some related works and results. In Section 3 we present preliminaries for our algorithm. The algorithm itself is presented in Section 4. We describe our algorithm for general trees. For special trees such as paths, spiders, and combs we achieve better running times, namely $O(n)$, $O(n^2)$, and $O(n^3)$, respectively, in Section 5.

1.1 Hardness of the MCSS problem on general graphs

The MCSS problem on a vertex-colored edge-weighted general graph G can be defined in a similar fashion where each block is a maximal connected subset of vertices of the same color in G. We observe that this problem is NP-hard. This is implied from the NP-hardness proof of the MCS problem for two-colored edge-weighted general graphs, due to Banerjee, Bhore, and Chitnis [1]. They use a reduction from the dominating set problem in connected graphs as follows. Given an instance of the dominating set problem on a connected graph G, construct an instance of the MCS problem consisting of two copies of G, namely G_1 and G_2, and a vertex v. The edges of G_1 and G_2 have weight 1. Connect every vertex of G_1 to all vertices of G_2 by edges of weight $2 - 3\epsilon$, and every vertex of G_2 to v by edges of weight ϵ. Then color the vertices of G_1 red, and the vertices of G_2 and v blue.

In the above construction the vertices of the same color form a block, and any solution for the MCS problem must contain vertices from both blocks. This matches with the requirements of our MCSS problem. Thus the same reduction implies that the MCSS problem on general graphs is also NP-hard, even for graphs that consist of two blocks.

2 Previous Work

In this section we discuss some related works for both general graphs and geometric graphs. Assume that the number of vertices of the input graph is n.

In a recent study, Dey et al. [4] show how to solve the MCS problem on some simple two-colored unweighted trees such as paths, caterpillars, spiders, and combs with respective running times $O(n)$, $O(n)$, $O(n^2)$, and $O(n^2)$. In a companion paper [5] they show how to solve the MCS problem on two-colored unweighted trees in $O(n^4)$ using a dynamic programming algorithm. They reduce each instance of the problem to a shortest path problem in a graph. Their algorithm is highly dependent on specific subtrees called *gates* where each gate consists of three vertices p, q, v where p and q are of different colors and equidistant to a vertex v of degree larger than 2; see Fig. 2(a) and Fig. 2(c).

There are major differences between our MCSS algorithm and the MCS algorithm of [5] in terms of both objectives and features: (i) the two algorithms solve two different problems which have different objectives, (ii) our algorithm works for multicolored weighted trees while it is unclear how one could generalize the algorithm of [5] to more than two colors or to weighted trees without blowing the running time mainly due to the notion of gates, (iii) in contrast to that of [5] our algorithm is based on a recursive formulation of the problem and it does not transform the original problem to a shortest path problem nor uses any gates.

Recall that the MCS problem was originally introduced for points in the Euclidean plane by Hart [9]. In a recent study regarding the Euclidean MCS, Biniaz et al. [2] provide some complex algorithms to find the MCS of colored point sets in the plane. These include a sub-exponential algorithm for determining the MCS of points on a plane, an algorithm to compute the Euclidean MCS for collinear points in $O(n)$ time, and two dynamic programming algorithms to find the MCS for multi-colored and two-colored points lying on two parallel lines in the plane with the respective running time of $O(n^6)$ and $O(n^4)$. Furthermore, they propose an algorithm with time complexity of $O(n \log n)$ to determine whether the size of the MCS for a set of two-colored points is two, and to find such a subset if it exists [2]. In the next section, some preliminaries will be discussed before presenting our algorithm.

3 Preliminaries for the Algorithm

For simplicity we present our algorithm for unweighted trees. In the end we show how to extend the algorithm to weighted trees. It is easily seen that an algorithm for the MCSS problem on bicolored trees would also work on multicolored trees because any solution should contain a vertex from each block regardless of the color of neighboring blocks.[2] In other words one can think the tree as a collection of blocks. Therefore, in our figures (but not in the description of the algorithm) we only consider bicolored trees. In this section we discover some properties that will be used to design our algorithm in the subsequent section. Let T be a tree and let S be a consistent spanning

[2]Notice that this does not apply to the MCS problem.

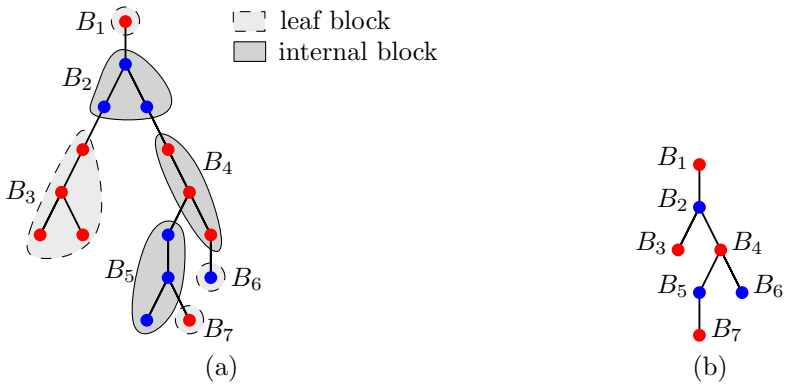

Figure 3: (a) Example of a tree T together with its blocks. (b) The block tree of T.

subset of T. We say that a vertex $v \in T$ is *covered* by the vertex $u \in S$ if u is a vertex of S that is closest to v. Analogously, we say that u *covers* v.

Observation 1 *If all vertices of T have the same color, i.e. T is monochromatic, then every vertex of T is a minimum consistent spanning subset for T.*

Recall the definition of block as a maximal subset of connected vertices of the same color in T. In view of Observation 1 we may assume that T has more than one block. Let $k \geq 2$ be the number of blocks in T. We define the *block tree* of T, denoted by $\mathcal{B}(T)$, as the tree with k vertices, each corresponding to a block of T, and there is an edge between any two vertices if their corresponding blocks are neighbors in T. In other words, $\mathcal{B}(T)$ is obtained by contracting all blocks of T. Notice that each vertex of $\mathcal{B}(T)$ corresponds to a block of T, and vice versa. We refer to a block of T as a *leaf block* if its corresponding vertex in $\mathcal{B}(T)$ is a leaf. A block of T that is not a leaf block is called an *internal block*, see Fig. 3.

We denote the shortest-path distance between two vertices u and v in T by $dist(u, v)$.

Lemma 1 *Any minimum consistent spanning subset of a tree T contains exactly one vertex from each leaf block of T.*

Proof: We use contradiction to prove this lemma. Consider a minimum consistent spanning subset S^* that contains more than one vertex from some leaf block, say B. Let N be the neighboring block of B. Let a be a vertex of S^* in B that is closest to N, as in Fig. 4. Let S' be the set obtained from S^* by removing all the vertices of B except for a. We claim that S' is a consistent spanning subset for T; this would contradict S^* being minimum.

To prove the above claim, it suffices to show that a is a vertex of S' that is closest to every vertex of B. Let n be a vertex of S^* in N that is closest to B. Let v be the last vertex of B on the path from a to n, as depicted in the figure to the right. As S^* is a consistent spanning subset, we have $dist(v, a) \leq dist(v, n)$. Now consider any vertex in $x \in B$. The path between x and n passes through v. Therefore $dist(x, a) \leq dist(x, n)$, which proves the claim. □

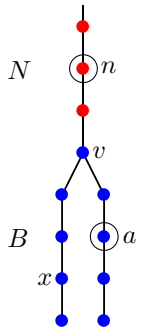

Figure 4: Illustration of Lemma 1.

3.1 An approximation algorithm

Recall the block tree $\mathcal{B}(T)$; see Fig. 3. Let S be the empty set. For every edge $e \in \mathcal{B}(T)$ our algorithm adds to S the two vertices of T that correspond to the endpoints of e. It is easily seen that S is a feasible solution for the MCSS problem. We claim that S is a 2-approximate solution. Let b denote the number of blocks in T, which is the same as the number of vertices of $\mathcal{B}(T)$. Since $\mathcal{B}(T)$ is a tree, it has $b-1$ edges. Therefore $|S| = 2b-2$. On the other hand the size of any optimal solution S^* is at least b because it must contain at least one vertex from each block of T, and hence $|S^*| \geq b$. Therefore

$$|S| = 2b - 2 \leq 2|S^*| - 2.$$

We note that this elementary analysis is the best possible for this algorithm, for example if T is a path with blocks of size three, then our algorithm picks two vertices for each block (except for the two leaf blocks) while the optimal solution picks one vertex (the middle vertex) from each block.

4 The Algorithm

Lemma 1 suggests a more constrained version of the MCSS problem, in the sense that we can fix a leaf block B and enforce exactly one vertex of B to be in the solution. As we do not know in advance which vertex of B is in the optimal solution, we try all of them and report the best answer.

Our algorithm employs a nontrivial dynamic programming approach. First we introduce the subproblems that will be generated throughout the algorithm and then we will show how to solve the subproblems recursively.

4.1 Defining the subproblems

We denote each subproblem by $T(a, c)$ where a and c are two given vertices of T. Consider the path δ between a and c in T and let x be the neighbor of c in δ (it might be the case that $a = x$). By removing the edge (x, c) from T we obtain two subtrees. Let T_c be the subtree containing c, see Fig. 5(a). Let T' be the union of δ and T_c as in Fig. 5(b). We define $T(a, c)$ to be the MCSS problem on T' with the following constraints:

- a must be in the solution, and

- all the vertices from a to x on δ must be covered by a.

These constraints imply that the vertices from a to x should have the same color.

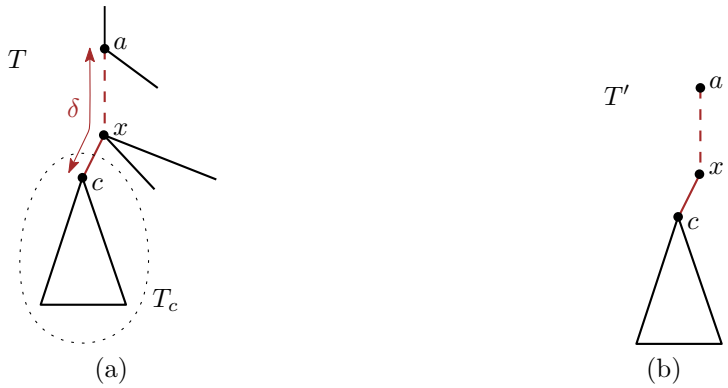

Figure 5: (a) The tree T, and (b) the tree T'.

4.2 Solving the subproblems

We denote the size of the (constrained) MCSS for $T(a, c)$ by $S(a, c)$. If $T(a, c)$ has no solution then we set $S(a, c) = +\infty$. To solve $T(a, c)$ we proceed as follows.

If T' is monochromatic, then (by Observation 1) we return a as the solution. In this case $S(a, c) = 1$. Assume that T' is not monochromatic. We root T' at a.

Lemma 2 *If $T(a, c)$ has a solution, then any solution of $T(a, c)$ contains a vertex z in the same block as a or in a neighboring block of a such that all vertices on the path from a to z are covered by a or by z.*

Proof: As T' is multicolored, any solution of $T(a, c)$ should contain at least two vertices. In particular it should contain at least one vertex from each neighboring block of a. In any solution of $T(a, c)$ a vertex that is closest to a satisfies the statement of the lemma for z. □

Let z be any vertex of T' that satisfies the constraints of Lemma 2, see Fig. 6 (It might be the case that $z = c$. Also z could be in a's block or in a's neighboring block.) If such a vertex z does not exist then $T(a, c)$ has no solution and thus we set $S(a, c) = +\infty$.

Let $a \rightsquigarrow z$ denote the path from a to z. By Lemma 2 all vertices of $a \rightsquigarrow z$ are covered by a or z. Since x must be covered by a (as imposed by the definition of $T(a, c)$), we must have $dist(x, a) \leq dist(x, z)$, and thus $dist(z, a) \geq 2 \cdot dist(x, a)$. Moreover if $a \rightsquigarrow z$ has $2k$ vertices (including a and z) then the first k vertices must be covered by a and the second k vertices by z. If $a \rightsquigarrow z$ has $2k + 1$ vertices then the first k vertices must be covered by a, the last k vertices by z, and the middle vertex say m must be covered by one of a and z that has the same color as m.

Now we are in a problem setting where both a and z must be in the solution, and all vertices on $a \rightsquigarrow z$ must be covered by a and z. We denote this more constrained version of $T(a, c)$ by problem $T(a, c, z)$ (we do not call this a subproblem for a reason to be determined later). In other words $T(a, c, z)$ is the MCSS problem on T' with the following constraints:

- z is in the same block as a or in a neighboring block of a,

- a and z must be in the solution,
- $dist(z, a) \geq 2 \cdot dist(x, a)$, and
- if $a \rightsquigarrow z$ has $2k$ or $2k+1$ vertices then the first k vertices must have the same color as a and the last k vertices must have the same color as z.

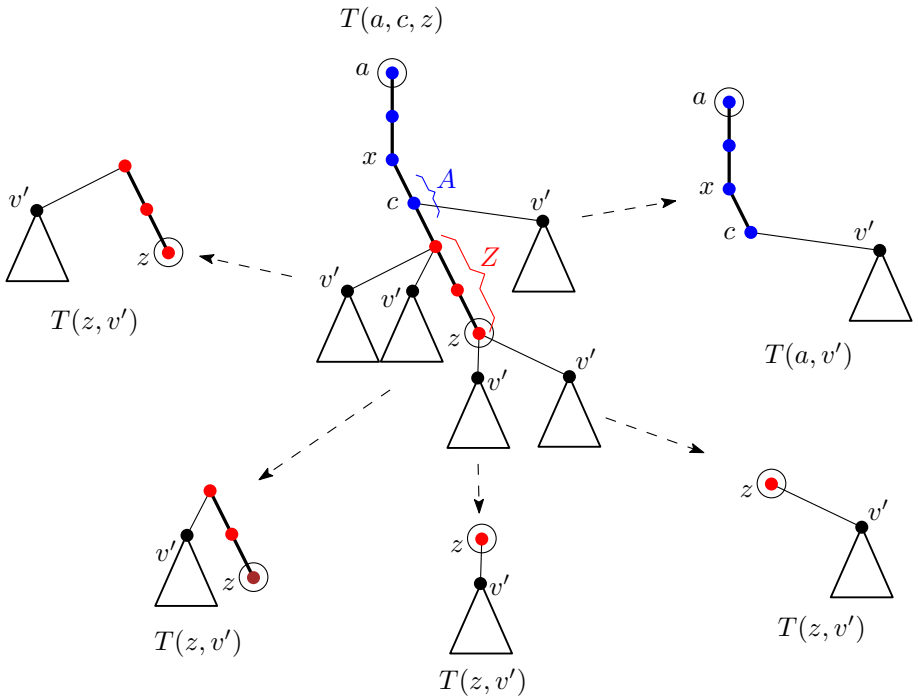

Figure 6: Solving $T(a, c)$ recursively in terms of $T(a, v')$ and $T(z, v')$ where z is a valid pair for a.

We refer to any vertex z that satisfies the above constraints as a *valid pair* for a. Now we show how to solve $T(a, c, z)$. We denote the size of the solution for $T(a, c, z)$ by $S(a, c, z)$. Let A be the set of all the vertices on the path $c \rightsquigarrow z$ that are closer to a than to z as in Fig. 6. Let Z be the set of all the vertices on $c \rightsquigarrow z$ that are closer to z than to a. If a vertex m has the same distance to a and z, then we put it in the set that has the same color as m.

To solve $T(a, c, z)$ we define two sets $\mathcal{A}$ and $\mathcal{Z}$ as follows. For each vertex v in A, we add to $\mathcal{A}$ all children of v that are not on the path $c \rightsquigarrow z$. For each vertex v in Z, we add to $\mathcal{Z}$ all children of v that are not on the path $c \rightsquigarrow z$. Then the solution of $T(a, c, z)$ is obtained by taking the union of $\{a, z\}$ with the solutions of $T(a, v')$ for all $v' \in \mathcal{A}$ and the solutions of $T(z, v')$ for all $v' \in \mathcal{Z}$. It might happen that some choices of z does not lead to a valid solution, but then a subsequent subproblem, e.g. $T(z, v')$ will be infeasible and $S(z, v') = +\infty$.

One might wonder that for two vertices $v'_1, v'_2 \in \mathcal{Z}$ the solutions of $T(z, v'_1)$ and $T(z, v'_2)$ may affect each other. However, this cannot happen because by the definition of $T(.,.)$ all vertices on paths from z to the parents of v'_1 and v'_2 must be covered by z, and thus any vertex in the solutions of $T(z, v'_1)$ and $T(z, v'_2)$ lies in the same level as z or in a lower level (i.e. higher depth) in T'. The same argument applies to vertices in $\mathcal{A}$. Notice that some vertex, say m, might have equal distance

to multiple vertices of different colors in the final solution. This is still okay for the purpose of the MCSS problem because both a and z will be nearest neighbors of m in the final solution and one of them has the same color as m.

Therefore

$$S(a,c,z) = 2 + \sum_{v' \in \mathcal{A}} S(a,v') + \sum_{v' \in \mathcal{Z}} S(z,v') - |\mathcal{A}| - |\mathcal{Z}|,$$

where the subtractive terms $|\mathcal{A}|$ and $|\mathcal{Z}|$ come from the fact that a and z are counted in each $S(a,v')$ and $S(z,v')$. Thus, we are able to solve $T(a,c,z)$ in terms of $T(a,v')$ and $T(z,v')$ which are smaller instances of $T(a,c)$.

Now we turn our attention back to the original problem $T(a,c)$. If we knew z then the solution of $T(a,c,z)$ would also be a solution for $T(a,c)$. But we do not know z in advance. Therefore we consider all valid pairs z for a and take one that gives the smallest solution. Let P denote the set of all valid pairs for a. Then

$$S(a,c) = \min\{S(a,c,z) \colon z \in P\}.$$

If P is the empty set, i.e. a has no valid pairs, then we set $S(a,c) = +\infty$. This is the end of our solution for the subproblem $T(a,c)$; the problem $T(a,c,z)$ was introduced to just simplify the description of our recursive solution.

4.3 Solving the original problem

In this section we show how to solve the original MCSS problem on the tree T. Let S denotes an optimal solution for this problem. If T is monochromatic then we return an arbitrary vertex as a solution. Assume that T has more than one colors. Let L be a leaf block of T, and let N be the (only) neighboring block of L. Let c be the vertex of N adjacent to a vertex of L. By Lemma 1 any optimal solution has exactly one vertex, say a, from L. Thus a is the closest vertex of S to all vertices in L. Hence the original problem is an instance of the $T(a,c)$ problem that was introduced earlier. Since we do not know a, we try all vertices of L. Therefore,

$$|S| = \min\{S(a,c) \colon \ a \in L\}.$$

The correctness of our algorithm is implied by Lemma 1, the fact that in any optimal solution all the vertices of L are closer to a than to any other vertex of S, and from the correctness of our solution for $T(a,c)$.

4.4 Running time analysis

In this section we analyze the running time of our algorithm for the MCSS problem on a tree T with n vertices. The algorithm follows a top-down dynamic programming approach and consists of subproblems of the form $T(a,c)$ where a and c are two vertices of T. In total we have $O(n^2)$ subproblems of this form. To solve each subproblem we examine $O(n)$ valid pairs for a. For each valid pair z of a we look up to solutions of $O(n)$ smaller subproblems, namely $T(a,v')$ and $T(z,v')$. Thus each subproblem $T(a,c)$ can be solved in $O(n^2)$ time. Therefore the total running time of the algorithm is $O(n^4)$.

One might wonder that for any fixed pair (a,c) if the size of solutions of $T(a,c,z)$, i.e. $S(a,c,z)$, is monotonic (increasing or decreasing) with respect to $dist(a,z)$ then we could save an $O(n)$ factor

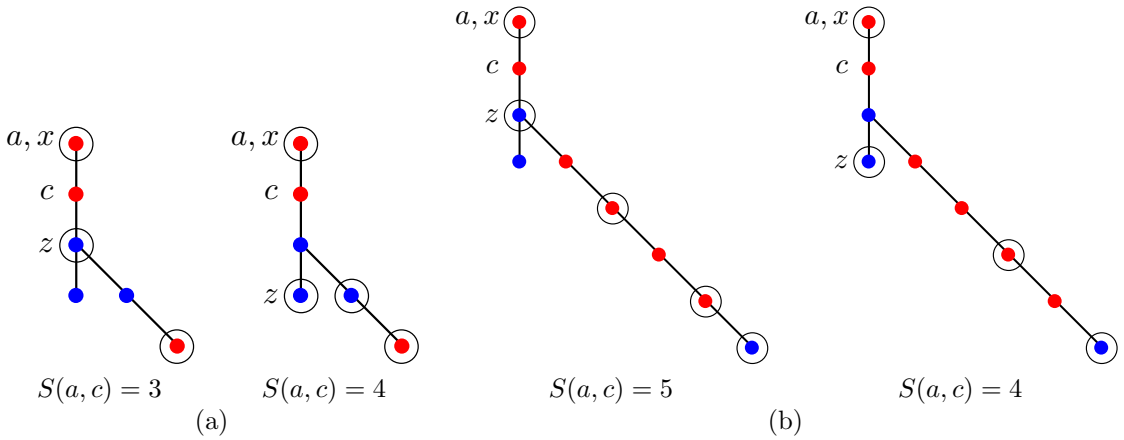

Figure 7: If we increase $d(a, z)$ then $S(a, c)$ can increase (a) or decreases (b).

in the computation of $T(a, c)$. However, this is not always the case; Fig. 7 shows that if we increase $dist(a, z)$ then $S(a, c, z)$ could increase (Fig. 7(a)) or decrease (Fig. 7(b)).

4.5 Extension to weighted trees

Our algorithm can simply be extended to solve the MCSS problem on weighted trees within the same time complexity. The only parts that need to be adjusted are the definitions of *distances* and *valid pairs*. The distance $dist(x, y)$ between two vertices a and y is now the the shortest distance in the weighted tree. The existence of a valid pair z for a (Lemma 2) can be shown by taking a vertex in the solution whose weighted distance to a is minimum. The validity of z is now verified by similar constraints, except for the last constraint, which we adjust as follows:

- All vertices on path $a \rightsquigarrow z$ that are closer to a (in terms of weighted distance) must have the same color as a and all vertices that are closer to z must have the same color as z.

5 Faster Algorithms for Simple Trees

In this section we show how to solve the MCSS problem on some simple trees faster than on general trees. We denote the input tree by T. A *spider* is a tree with one vertex of degree at least 3 called the *center* and all other vertices of degree at most 2. A *comb* is a tree consisting of a path called the *skeleton* where each of its vertices is connected to another path called the *dangling path*.

5.1 Paths

We transform path vertices to points on a line where the Euclidean distance between two points is identical to the weight of the edge between the corresponding vertices. Then we run the $O(n)$-time MCS algorithm of Biniaz et al. [2]. This gives a solution for the MCSS problem.

5.2 Spiders

Let B denote the block containing the center of T. Any solution must contain some vertex from B, among which one is closest to the center. We examine each vertex $b \in B$ as a vertex in the solution that is closest to the center. Then we break the tree into some paths each of which is defined by two endpoints that are b and a leaf of T. We solve the MCSS problem on each path independently (with the constraint that b must be in the solution) and then take the union of the solutions as a solution for the MCSS problem on T. The total running time is $|B| \cdot O(n) = O(n^2)$.

5.3 Combs

We use our algorithm of previous section (for general trees) and show how to save a linear factor from the running time.

Recall from subsection 4.2 that we solve $T(a, c, z)$ in linear time by looking up to solutions of $O(n)$ smaller subproblems of the form $T(a, v')$ and $T(z, v')$. We are going to solve this in constant time with the help of pre-processing. We take advantage of the fact that each vertex v on the path $a \rightsquigarrow z$ has at most one child that can be v' (because T is a comb) and the fact that each subproblem $T(a, v')$ is an MCSS instance on a path (again because T is a comb).

Let $s_1, \ldots, s_k$ be the vertices of the skeleton in this order. For each $i \in \{1, \ldots, k\}$ let l_i be the leaf of T in the dangling path at s_i. We define two $n \times k$ matrices $\mathcal{P}$ and $\mathcal{S}$ where the rows represent the n verties of T and the columns represent the k vertices of the skeleton. For each vertex $a \in T$ and each $i \in \{1, \ldots, k\}$ the entry $\mathcal{P}[a][i]$ stores the size of the solution for the MCSS problem on the path from a to l_i with the constraint that a is in the solution and all vertices from a to s_i are covered by a—this can be computed in linear time using path algorithm. Each entry $\mathcal{S}[a][i]$ stores the sum of $\mathcal{P}[a][j]$ for all indices j of skeleton vertices that lie on the path $a \rightsquigarrow s_i$. Thus $\mathcal{P}$ and $\mathcal{S}$ can be computed in $O(kn^2)$ and $O(k^2n)$ time, respectively, in a pre-processing step.

Let m be the index such that all vertices on the path $a \rightsquigarrow p_m$ have the same color as a and all vertices on the path $p_{m+1} \rightsquigarrow z$ have the same color as z (this corresponds to definitions of the sets A and Z in the original algorithm). Then $\mathcal{S}[a][m] = \sum_{v' \in \mathcal{A}} S(a, v')$ and $\mathcal{S}[z][m+1] = \sum_{v' \in \mathcal{Z}} S(z, v')$. Since $\mathcal{S}[a][m]$ and $\mathcal{S}[z][m+1]$ are already computed, we can solve $T(a, c, z)$ in constant time. Therefore, the total running time of the algorithm is $O(n^3)$.

6 Conclusions

We introduced the minimum consistent spanning subset problem. We showed that this problem is NP-hard on general graphs and can be solved in $O(n^4)$ time on trees, where n is the number of vertices of the tree. Two natural open problems are to (i) improve the running time for trees, and (ii) present approximation algorithms for the general case.

The version of the MCSS that we studied here requires every vertex $v \notin S$ to have a nearest neighbor in S that is in the same block as v. An interesting problem would be to drop the constraint of being in the same block. In other word every vertex $v \notin S$ must have a nearest neighbor in S that has the same color as v, and does not necessarily need to be in the same block. See for example Fig. 8. In this example, an optimal solution for the version that we have stidied in this paper picks two vertices from the block B_2. However, if we drop the constraint then the optimal solution picks one vertex from each block as shown the figure (the optimal solution here is unique).

The algorithm that we have presented in Section 4 does not extend to the new version because our structural lemma, Lemma 2, does not hold for this version. For example if we start our

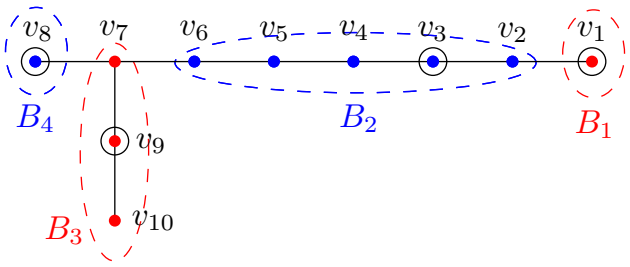

Figure 8: An optimal solution assuming that the constraint of having a nearest neighbor in the same block has been dropped. The vertex v_6 does not have a nearest neighbor in its own block, however it has a nearest neighbor, v_8, in a different block but of the same color.

algorithm from B_1 in Fig. 8, the first try would be $T(v_1, v_2)$ in which v_3 is chosen as a valid pair for v_1. Then the algorithm tries to solve $T(v_3, v_4)$ but the vertex $z = v_9$ in the neighboring block does not satisfy the lemma's outcome that all vertices from v_3 to v_9 are covered by v_3 or v_9, because v_6 is covered by v_8.

Acknowledgement. We are very grateful to the anonymous reviewer who meticulously verified our proof, and provided valuable feedback that clarified the constraints of the problem. In particular the new version that is introduced in Section 6, the example that is shown in Figure 8, and discovery of the limitation of our Lemma 2 for the new version are all due to the reviewer.

References

[1] Sandip Banerjee, Sujoy Bhore, and Rajesh Chitnis. Algorithms and hardness results for nearest neighbor problems in bicolored point sets. In *Proceedings of the 13th Latin American Symposium Theoretical Informatics (LATIN)*, volume 10807, pages 80–93. Springer, 2018. doi:10.1007/978-3-319-77404-6_7.

[2] Ahmad Biniaz, Sergio Cabello, Paz Carmi, Jean-Lou De Carufel, Anil Maheshwari, Saeed Mehrabi, and Michiel Smid. On the minimum consistent subset problem. *Algorithmica*, 83(7):2273–2302, 2021. doi:10.1007/s00453-021-00825-8.

[3] Belur V. Dasarathy. Minimal consistent set identification for optimal nearest neighbor decision systems design. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(3):511–517, 1994. doi:10.1109/21.278999.

[4] Sanjana Dey, Anil Maheshwari, and Subhas C. Nandy. Minimum consistent subset of simple graph classes. *Discrete Applied Mathematics*, 338:255–277, 2023. doi:10.1016/J.DAM.2023.05.024.

[5] Sanjana Dey, Anil Maheshwari, and Subhas C. Nandy. Minimum consistent subset problem for trees. In Evripidis Bampis and Aris Pagourtzis, editors, *Fundamentals of Computation Theory*, pages 204–216, 2021. doi:10.1007/978-3-030-86593-1_14.

[6] Byron J. Gao, Martin Ester, Jin-Yi Cai, Oliver Schulte, and Hui Xiong. The minimum consistent subset cover problem and its applications in data mining. In *Proceedings of the 13th*

ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), page 310–319, 2007. doi:10.1145/1281192.1281228.

[7] Lee-Ad Gottlieb, Aryeh Kontorovich, and Pinhas Nisnevitch. Near-optimal sample compression for nearest neighbors. *IEEE Transactions on Information Theory*, 64(6):4120–4128, 2018. doi:10.1109/TIT.2018.2822267.

[8] K. Chidananda Gowda and Gopal Krishna. The condensed nearest neighbor rule using the concept of mutual nearest neighborhood. *IEEE Transactions on Information Theory*, 25(4):488–490, 1979. doi:10.1109/TIT.1979.1056066.

[9] Peter Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(3):515–516, 1968. doi:10.1109/TIT.1968.1054155.

[10] Qing He, Xiu-Rong Zhao, and Zhong-Zhi Shi. Minimal consistent subset for hyper surface classification method. *International Journal of Pattern Recognition and Artificial Intelligence*, 22(01):95–108, 2008. doi:10.1142/S0218001408006132.

[11] Kamyar Khodamoradi, Ramesh Krishnamurti, and Bodhayan Roy. Consistent subset problem with two labels. In *Proceedings of the 4th International Conference on Algorithms and Discrete Applied Mathematics (CALDAM)*, volume 10743, pages 131–142. Springer, 2018. doi:10.1007/978-3-319-74180-2_11.

[12] Gordon Wilfong. Nearest neighbor problems. In *Proceedings of the Seventh Annual ACM Symposium on Computational Geometry (SoCG)*, page 224–233, 1991. doi:10.1145/109648.109673.

[13] Guanghua Yu, Jin Tian, and Minqiang Li. Nearest neighbor-based instance selection for classification. In *12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 75–80, 2016. doi:10.1109/FSKD.2016.7603154.