University of Windsor
Scholarship at UWindsor

Electronic Theses and Dissertations | Theses, Dissertations, and Major Papers

6-1-2023

The Minimum Consistent Spanning Subset Problem on Trees

Parham Khamsepour
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

 Part of the Computer Sciences Commons

Recommended Citation

Khamsepour, Parham, "The Minimum Consistent Spanning Subset Problem on Trees" (2023). *Electronic Theses and Dissertations*. 9348.
https://scholar.uwindsor.ca/etd/9348

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

The Minimum Consistent Spanning Subset Problem on Trees

By

Parham Khamsepour

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2023

©2023 Parham Khamsepour

The Minimum Consistent Spanning Subset Problem on Trees

by

Parham Khamsepour

APPROVED BY:

M. Sangani Monfared
Department of Mathematics and Statistics

D. Wu
School of Computer Science

A. Biniaz, Advisor
School of Computer Science

April 20, 2023

DECLARATION OF CO-AUTHORSHIP AND PREVIOUS PUBLICATION

I. Co-Authorship

I hereby declare that this thesis incorporates material that is the outcome of my research under the supervision of Dr. Ahmad Biniaz.

In particular the new results presented in chapters 3 and 4 of this thesis were co-authored by Ahmad Biniaz. All the co-authors contributed to finalizing the ideas and follow-up discussions. A. Biniaz contributed to this research with his initial thoughts and the main idea this research is based upon. Both P. Khamsepour and A. Biniaz contributed to design and analysis of the algorithms and writing/editing the contents of these chapters. All implementations, verifications, validations, and visualizations were performed by P. Khamsepour under the supervision of A. Biniaz.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work under the supervision of A. Biniaz.

II. Previous Publication

This thesis includes 1 original paper that has been submitted for publication in peer reviewed conference, as follows:

Thesis Chapter	Publication title/full citation	Publication Status
Chapter 3	Biniaz, A., and Khamsepour, P. (2023). The Minimum Consistent Spanning Subset Problem on Trees. *49th International Workshop on Graph-Theoretic Concepts in Computer Science.*	Submitted

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as a graduate student at the University of Windsor.

III. General

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

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 not in S has the same color as its nearest neighbor in S. This problem has applications in clustering and classification algorithms, specially in finding the optimal number of clusters in k-clustering algorithms. This problem is NP-complete. A recent result of Dey, Maheshwari, and Nandy (2021) gives a polynomial-time algorithm for the MCS problem on trees. In thesis we study the MCS problem on different settings, and discuss some of the shortcomings of the MCS problem for trees. We then introduce a variant of the MCS problem, namely the *minimum consistent spanning subset* (MCSS) problem, for which we require the set S to contain a vertex from every *block* of the graph, where a block is a maximal connected vertices of the same color. We observe that this problem is NP-hard on general graphs. We present an $O(n^4)$-time algorithm to find the MCSS on multi-colored weighted trees with n vertices. We also improve the running time for simple classes of trees.

DEDICATION

I hereby dedicate this thesis to my esteemed parents, *Sima* and *Amir*, and my brother, *Peyman*, with immense gratitude and respect. Their constant and unwavering support, encouragement, love, and understanding throughout my academic journey have been invaluable to me.

ACKNOWLEDGEMENTS

I would like to sincerely express my most profound gratitude towards my supervisor Dr. Biniaz, whose input helped me immensely. With his input, I was able to look at my research with a different perspective and a more critical eye.

Secondly, I would like to express my gratitude to my thesis committee members, Dr. Wu and Dr. Sangani Monfared, for their valuable advice and suggestions.

I humbly extend my thanks to the School of Computer Science and all concerned people who helped me in this regard.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

This thesis focuses on the Minimum Consistent Subset (MCS) and Minimum Consistent Spanning Subset (MCSS) problems on graphs. To introduce these problems and the corresponding algorithms, we first provide some basic definitions and terminologies.

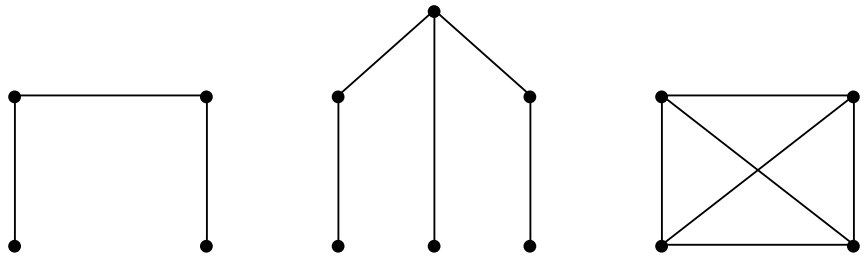
1.1 Basic definitions and terminologies

A *graph* $G = (V, E)$ consist of a set V of vertices, and a set E of edges, where each edge connects two distinct vertices together. Now we introduce the different classes of graphs, and some other terminologies:

- **Undirected graph:** is a graph where each edge is a set of two vertices. In other words, each edge is an unordered pair of vertices. All graphs in this thesis are undirected unless otherwise specified.
- **Directed graph:** is a graph where each edge of G has a direction from one vertex to another. In other words, each edge is an ordered pair of vertices.
- **Weighted graph:** is a graph such that each edge has a weight assigned to it, that shows the cost of going from one vertex to another. If all the edges have the same weight, we consider it as an unweighted graph.
- **Path:** in a graph is an alternating sequence of vertices and edges connecting a starting vertex to an ending vertex, without repeating any vertices, or edges

such that each edge is incident to its succeeding and proceeding vertices, see Fig. 1.1.1(a).

- **Cycle:** is a path that starts and ends at the same vertex. In other words, a cycle is a closed path.
- **Acyclic graph:** is a graph with no cycles. In other word, in an acyclic graph, its not possible to start at a vertex and follow a path along the edges that eventually gets back to the starting vertex, see Fig. 1.1.1(a) and Fig. 1.1.1(b).
- **Complete Graphs :** are undirected graphs in which every pair of distinct vertices is connected by exactly one edge. A complete graph with n vertices is shown as K_n, Fig. 1.1.1(c) shows the graph K_4.
- **Connected graph:** is a graph in which exists a path between any pair of vertices. All the figures in Fig. 1.1.1 show examples of connected graphs.
- **Tree:** is a connected acyclic graph, see Fig. 1.1.1(a) and Fig. 1.1.1(b).

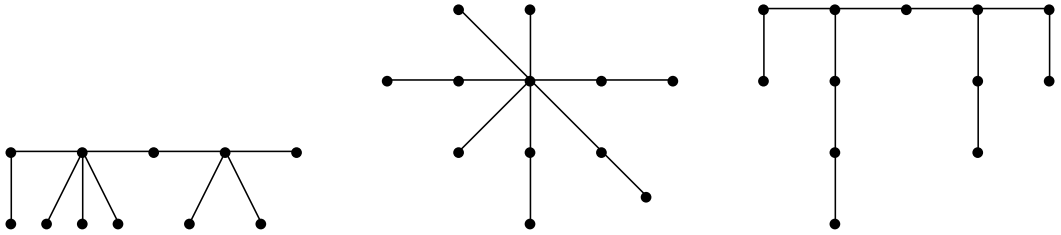


(a) (b) (c)

Fig. 1.1.1: (a) Example of a path, (b) an acyclic connected graph (tree), and (c) the graph K_4.

- **Caterpillar:** is a tree consist of a path, called *skeleton*, and a set of vertices each of which, called a *dangling* vertex, is connected to a vertex of the skeleton by an edge, as shown in Fig. 1.1.2(a).
- **Spider:** is a tree consist of multiple paths that share a single vertex, called the *center*, and do not share any other vertices. In a spider tree only the center vertex has more than two neighbors, see Fig. 1.1.2(b).

- **Comb:** is a tree consist of a path called the skeleton, and a set of disjoint paths each connected to a unique vertex on the skeleton. Fig. 1.1.2(c) shows an example of a comb.



(a) (b) (c)

Fig. 1.1.2: (a) Example of a caterpillar, (b) spider, and (c) comb tree.

- **Leaf:** is a vertex of a graph that has only one edge connected to it.

- **Voronoi Diagram**: Consider a set P of points in the Euclidean plane. The Voronoi diagram of P is a geometric structure that partitions the plane into $|P|$ cells, one for each point p, called the Voronoi cell of p, such that any point in the Voronoi cell of p is closer (in terms of Euclidean distance) to p than any other point in the point set, see Fig. 1.1.3.

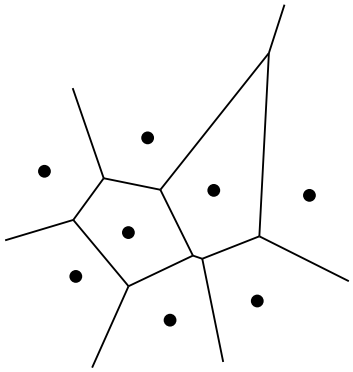


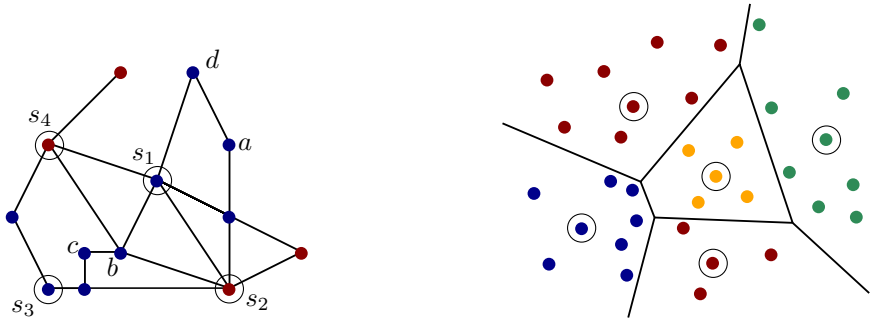
Fig. 1.1.3: Voronoi diagram for a given point set.

- **NP-hardness**: A problem is considered NP-hard if it is at least as hard as the hardest problems in the complexity class NP (Non-deterministic Polynomial time). NP problems are those that can be solved in polynomial time by a non-

deterministic Turing machine, or verified in polynomial time by a deterministic Turing machine.

1.2 The MCS problem

Let G be a simple undirected edge-weighted graph where its vertex set V is partitioned into $V_1, \ldots, V_k$. 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, 12]. 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. 1.2.1(a) for an example where all edges have the same weight.



(a) (b)

Fig. 1.2.1: (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 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 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. 1.2.1(b).

The MCS problem finds applications in solving nearest neighbor problems [3, 7, 8, 15], finding optimal number of clusters in k-clustering problems such 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 [14] even for two-colored points [11].

There has not been much progress on the MCS problem from the algorithmic point of view. Recently, Dey et al. solved this problem in polynomial-time for some simple two-colored (also known as bicolored and bichromatic) unweighted trees such paths, spiders, caterpillars, and combs [4]. In a companion paper [5] they presented an algorithm for general two-colored unweighted trees. See Fig. 1.2.2(a) and Fig. 1.2.2(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. 1.2.2(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 subset of vertices of the same color in a tree. The tree in Fig. 1.2.2(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 e.g. Fig. 1.2.2(a) and Fig. 1.2.2(c). Therefore, a minimum consistent subset may not capture the structure of the entire tree. In order to have

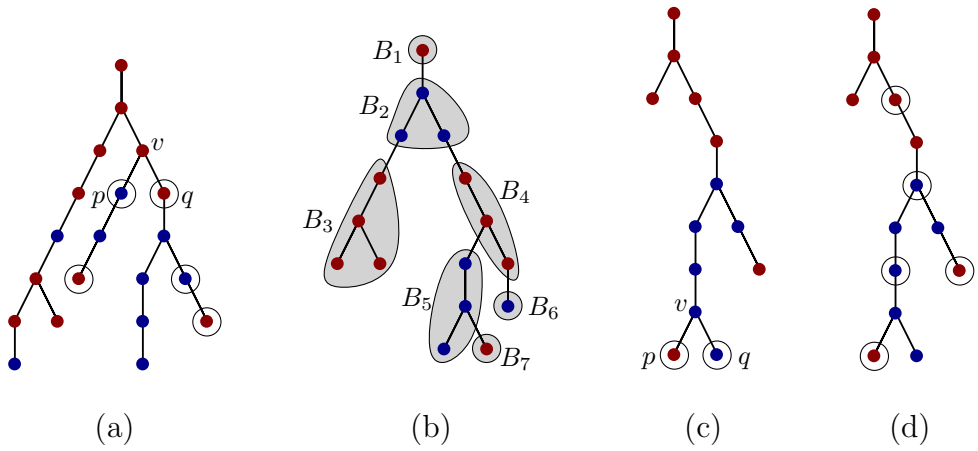


Fig. 1.2.2: (a) An MCS for a two-colored unweighted tree. (b) Blocks in a tree. (c) An MCS for an unweighted tree that has size 2, and (d) an MCSS for the same tree.

a better representative for the tree we introduce a more constrained version of the MCS problem.

1.3 The MCSS problem

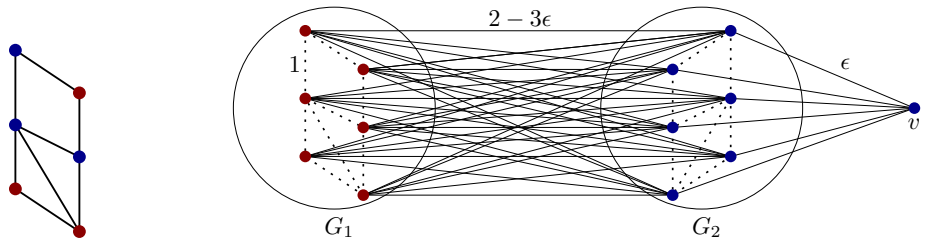
The *minimum consistent spanning subset* (MCSS) is a more constrained version of the MCS problem. In this version, which we call it the MCSS problem, the solution S must contain at least one vertex (i.e. a representative) from each block of the tree. A solution to the MCSS problem spans over all blocks in the tree. See Fig. 1.2.2(d) for an example of an MCSS on a two-colored tree.

1.3.1 Problem definition

We formally define The MCSS problem on trees as follows. Given an undirected vertex-colored edge-weighted graph $G = (V, E)$, the MCSS problem asks for a subset $S \subseteq V$ of minimal size such that for a vertex $v \in V$ the nearest neighbor of v in S has the same color as v. Furthermore, each block of G must have at least one representative in S. In the MCSS problem, we consider the distance between two vertices, denoted by $dist(a, b)$, as the weight of edges in the shortest path between a and b in the tree.

1.3.2 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 on a graph 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 wight 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, see Fig. 1.3.1.



(a) (b)

Fig. 1.3.1: (a) Example of the the graph G, and (b) its constructed graph for the MCS problem.

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.

Now that the formal problem definition for MCSS problem and its hardness are known, we discuss the organization of this thesis in the next part.

1.4 Contributions and thesis organization

The following theorem summarizes our main result and contribution of this thesis.

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

Although our MCSS algorithm shares some similarity with the MCS algorithm of [5] in terms of running time and the main approach which is dynamic programming, there are major differences: (i) the two algorithms 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) our algorithm is based on a recursive formula and it does not transform the original problem to a shortest path problem nor uses any gates.

We also show how to solve the MCSS problem faster on simple trees such as paths, spiders, combs and caterpillars with the respective running time of $O(n)$, $O(n^2)$, $O(n^3)$, and $O(n^3)$.

The rest of this thesis is consist of the following four sections:

Chapter 2: In this chapter we discuss the previous work, and results regarding the MCS problem for graphs, and points in the plane. This is followed by applications of the MCS problem.

Chapter 3: In this chapter, we present our main algorithm, and prove Theorem 1. Our algorithm uses a dynamic programming approach to find the MCSS on trees.

Chapter 4: In this chapter, we discuss the MCSS on simpler classes of trees, such as paths, spiders, combs and caterpillars. For these trees we obtain the MCSS in a better running time.

Chapter 5: This chapter concludes the thesis, and discusses some open problems, and future research direction.

CHAPTER 2

Related works

In this chapter we first discuss some related works to solve the MCS for both general graphs and geometric graphs. Then we discuss some applications of the MCS.

Let n be the number of vertices of the input graph. In a recent study, Dey, Maheshwari, and Nandy [4] showed how to solve the MCS problem on some simple two-colored unweighted trees such as paths, spiders, caterpillars, and combs with respective running times $O(n)$, $O(n^2)$, $O(n)$, and $O(n^2)$. In a companion paper [5] they showed 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 structures called *gates*. Manna et al. [12] also proposes an algorithm to find the MCS for k-colored spiders in $O(n^{k+2})$ using the concept of gates. We will discuss gates in detail in the Section 2.1.

The MCS problem was originally introduced for points in the Euclidean plane by Hart [9]. The Euclidean version of this problem is also well studied [1, 2, 14]. Banerjee et al. [1] solved the MCS problem for collinear points in $O(n^2)$ time. In another study, Wilfong et al. [14] proposed a complex algorithm to find the MCS for a set of points where one point is colored red, and the rest are blue in $O(n^2)$ time. Recently, Biniaz et al. [2] provided 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 with the respective

running times of $O(n^6)$ and $O(n^4)$. Furthermore, they proposed 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 following sections of this chapter, we discuss some of the above algorithms that solve the MCS problem on different settings in more detail.

2.1 The MCS problem on graphs

As previously discussed some algorithms have been recently proposed to solve the MCS problem on specific classes of unweighted trees. Before discussing these algorithms, it is useful to understand the concept of *gates*.

In a two-colored unweighted tree a gate is defined as a tuple (p, q, v). Let v be a vertex with two equidistant children p and q of opposite colors, such that all the vertices on the path $v \rightsquigarrow p$ (and $v \rightsquigarrow q$ respectively) all have the same color (except v itself), see Fig. 2.1.1(a) and Fig. 2.1.1(b). While computing the MCS for two-colored unweighted trees by choosing a gate (p, q, v), assume that p and q are in the solution. Now, all the vertices in the subtrees connected to the each child of v that does not contain p or q, will be closer to either vertex p or q depending on their color, and are considered solved [4, 5, 12], see Fig. 2.1.1(c). In general a gate with k colors can be formed for a vertex v, if v has k different equidistant children $p_1, \cdots, p_k$ where each vertex $p_1, \cdots, p_k$ has a unique color and all the vertices on each path $v \rightsquigarrow p_i$ (except v) all have the same color [12], see Fig. 2.1.1(d). Gates are widely used in [4, 5, 12] to find the MCS of different unweighted trees.

2.1.1 The MCS problem on simple unweighted trees

In this section we discuss some previous work regarding finding the MCS for simple classes of trees. We assume that any tree T considered in this section has n vertices.

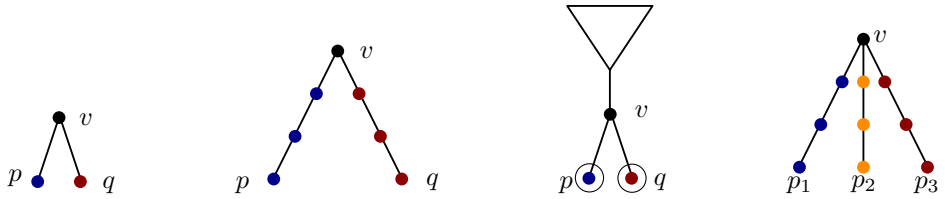


Fig. 2.1.1: (a) and (b) Example of a gate (p, q, v). (c) Example showing all the vertices in the triangle connected to v are closer to either p or q depending of their color, and are solved. (d) Example of a 3-colored gate.

2.1.1.1 Paths

Dey et al. [4] first propose an algorithm to find the MCS for a simple unweighted path, denoted by T, in $O(n)$ time assuming that the blocks of T, say $B_1, \cdots, B_k$, are given from left to right, see Fig. 2.1.2.

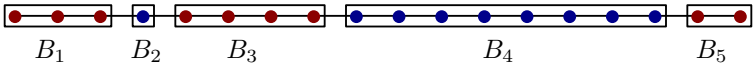


Fig. 2.1.2: Example of a path and all of its blocks.

They show that that only one vertex is enough from B_1 and B_k to be in the MCS, while all the other blocks require at most two vertices. Thus the MCS for T must include at least one vertex from each block. This means that the MCS and the MCSS problem on a path is identical.

The algorithm then transforms the MCS problem on a path T to a shortest path problem, and solves it as such. Let $\mathcal{T}$ be a graph with all the vertices of T. To help with selecting correct vertices for the MCS of T, a dummy vertex is added to $\mathcal{T}$ for each block of T, denoted by $d_1, \cdots, d_k$, see Fig. 2.1.3.

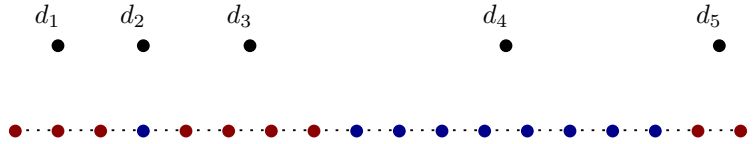


Fig. 2.1.3: Example of the vertices of the new overlay graph $\mathcal{T}$ while solving a path.

For each vertex v there exists at most three vertices v' in the next block such that any vertex u between v and v' is closer to one of v and v' that has the same color

as u. Such (v, v') is called a *valid pair*. For each v a directed edge with the weight of 0 is added to $\mathcal{T}$ from v to all the vertices of the next block that form a valid pair with v. Furthermore, to help with selecting more than one vertex from each block to be in the solution, a directed edge is added from each vertex $v \in B_k$ to the dummy vertex d_k. It should be noted for each vertex v another edge is added in the opposite direction from d_k to v as well. These edges between each vertex and the dummy vertices have the weight 1 for all the vertices in $B_2, \cdots, B_{k-1}$, and will have the weight of 0 for B_1 and B_k. To find the MCS for T, shortest path between d_1 and d_k such that the vertices are selected from left to right for $\mathcal{T}$ is computed. By removing the dummy vertices from the shortest path, a MCS of T is obtained, see Fig. 2.1.4.

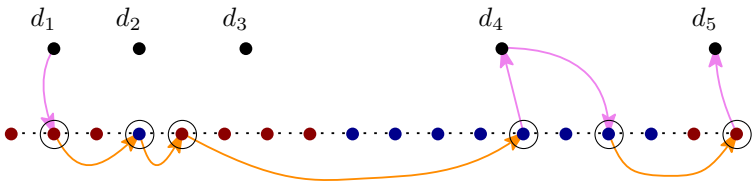


Fig. 2.1.4: Example of the shortest path and the MCS of a path.

2.1.1.2 Two-colored spiders

Here we discuss the algorithm of Dey et al. [4] to solve the MCS for a two-colored unweighted spider tree T in $O(n^2)$ time.

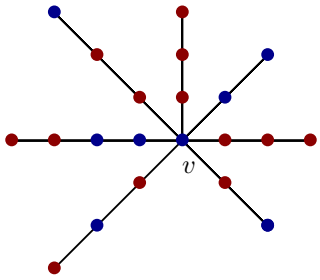


Fig. 2.1.5: Example of a two-colored spider.

Let v denote to the center of T, see Fig. 2.1.5. The algorithm considers three cases. Case(i): v's color is different from all of its adjacent vertices. In this case, the center vertex v is added to the solution as it forms a block of size 1, and all of

the the paths connected to v are then solved using the algorithm for paths with the assumption that v is already in the solution, see Fig. 2.1.6.

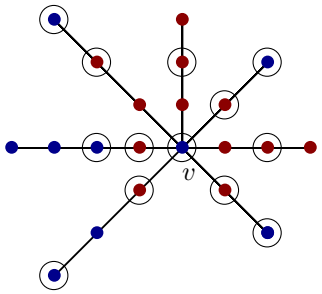


Fig. 2.1.6: Example of the MCS for a two-colored spider where the center vertex is colored differently than all the other adjacent vertices to it.

Case(ii): v has the same color as all of its adjacent vertices. Let B be the block containing v. The algorithm guesses the vertex v' in the optimal solution that is closest to v. To do so, the algorithm considers each vertex v' in B to be the closest vertex to v in the solution. Then the MCS for each path from v' to any leaf is computed with the assumption that v' is the closest vertex to v in the solution. The final solution is obtained by considering all vertices of B and choosing the vertex resulting in the smallest consistent subset, see Fig. 2.1.7.

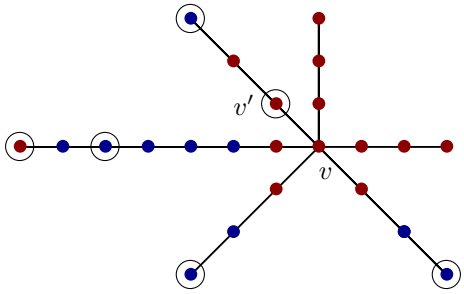


Fig. 2.1.7: Example of the MCS for a two-colored spider where the center vertex is colored the same as all the other adjacent vertices to it.

Case(iii): v has neighboring vertices of both colors. In this case, v and some its neighbors form gates. The algorithm considers all the possible gates (p, q, v), and chooses the gate resulting in the consistent subset of smallest size. For each gate, all the vertices on the paths connected to v, except the paths containing p and q, are closer to vertex p or q depending of their color. Let δ_1 and δ_2 be the respective paths

from p and q to the end of the path that contains p and respectively q. The consistent subset base on each gate (p, q, v) contains p and q, and the MCS for paths δ_1 and δ_2 with the assumption that p and q are already in the solution. This algorithm computes the consistent subset base on all gates, and finds the MCS for the spider in $O(n^2)$ time, see Fig. 2.1.8.

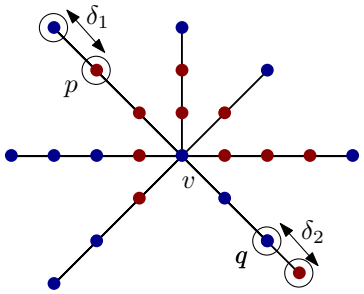


Fig. 2.1.8: Example of the MCS for a two-colored spider using the gate (p, q, v).

2.1.1.3 Multi-colored spiders

Recently Manna et al. [12] solved the MCS problem for multi-colored unweighted spiders. To find the MCS for a multi-colored unweighted spider, a very similar approach to the two-colored spiders is taken. The algorithm works by first checking the center vertex, and the corresponding block containing the center vertex. Here they also consider 3 cases. Case(i) where v has different color from its neighbors, and case(ii) where v has the same color as all of its neighbors are handled similarly to 2-colored spiders [4]. The main difference between the algorithm in [12], and the algorithm proposed in [4] for two-colored spiders, is the case where some gates can be formed around the center vertex. In a k-colored spider where the neighboring vertices to the center v are colored differently, two scenarios might happen (i) some gates with all k colors can be formed around v, or (ii) no gate with all k colors can be formed around v. For the first case, the algorithm considers all gates with all k colors, and the solution for the paths connected to the vertices of the gate are computed. In this case, the gate resulting in the smallest size consistent subset is the MCS of the spider, see Fig. 2.1.9(a). For the second case, where not all k colors exist in the neighboring blocks of v, the algorithm considers all combinations of different colored gates of all

sizes. For each gate the consistent subset includes the vertices of the gate, the solution for the paths connected to the vertices of the gates, and some other vertices obtained as follows. For each path δ (δ does not contain a vertex of the gate) connected to v that contain vertices of different color than the ones in the gate, let $\phi(\delta)$ be the first block of δ. The solution for δ is computed base on a vertex of the gate already in the solution that has the same color as vertices of $\phi(\delta)$. Let p_i be such a vertex, and let δ' be the union of δ and the path $p_i \rightsquigarrow v$. The solution for δ' is computed as a path with the assumption that p_i is the closest to vertex to v in the solution. By adding the solution for each path δ to the solution of the gate, the size of the consistent subset base on that gate is obtained. By checking all gates, and choosing the one resulting in smallest consistent subset the MCS for the spider is found, see Fig. 2.1.9(b). For each of these cases, the algorithm requires $O(n^{k+2})$ to find the MCS of the k-colored spider.

(a) (b)

Fig. 2.1.9: (a) Example of the MCS of 4-colored spider where the center v and $p_1, \cdots, p_4$ form a gate. (b) Example of the MCS of 7-colored spider where the center v and $p_1, \cdots, p_4$ form a gate.

2.1.1.4 Caterpillars

Let T be a two-colored unweighted caterpillar. An algorithm of Dey et al. [4] propose to find a MCS of T in $O(n)$ time using gates. The algorithm first finds a vertex v on the skeleton such that it has two children of opposite color. In this case, he MCS for the entire tree is $\{p, q\}$, see Fig. 2.1.10.

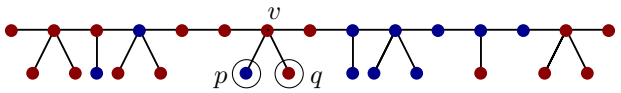


Fig. 2.1.10: Example of the MCS for a caterpillar of size two.

Now consider the case where none of the vertices on the skeleton have two children of opposite color to form a gate. To find the MCS in this case, the algorithm traverses the tree to find gates that are formed between a vertex on the skeleton, and a dangling vertex. For a vertex v on the skeleton, a *left* gate is defined as a tuple (p, q, v) such that p is the vertex to the left of v on the skeleton, and q is a dangling child of v of opposite color than p. For a left gate (p, q, v) all the vertices to the right of v are closer to either p or q of the same color. A *right* gate is defined in the similarly, where p is to the right of v in the gate (p, q, v), and q is the dangling child of v with an opposite color to p. Consider a caterpillar with only left gates in it, then by selecting the leftmost left gate (p, q, v) and adding p and q to the solution, all the vertices to the right of v will be closer to either p or q of the same color in the solution. The MCS for the unsolved portion to the left of p is then solved in $O(n)$ as a path, see Fig. 2.1.11. A similar process happens if the tree only contains right gates by choosing the rightmost right gate.

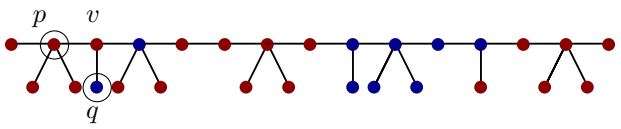
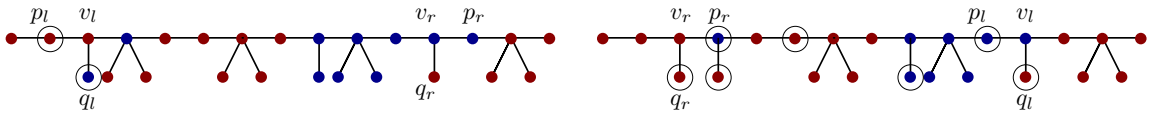


Fig. 2.1.11: Example of the MCS of a caterpillar containing only left gates.

Lastly, in case both left gates and right gates are present in a two-colored unweighted caterpillar two cases are considered. (i) In this case the leftmost left gate (p_l, q_l, v_l) is to the left of the rightmost right gate (p_r, q_r, v_r). The algorithm finds a consistent subset base on each gate, and chooses the smallest one as the MCS of the caterpillar, see Fig. 2.1.12(a). (ii) The leftmost left gate (p_l, v_l, q_l) is to the right of the rightmost right gate (p_r, v_r, q_r). The MCS in this case consists of all four vertices $\{p_l, q_l, p_r, q_r\}$, and the MCS for the unsolved middle portion between p_r and p_l is then solved with a similar approach to the path, as it does not contain anymore

gates. Notice that in the algorithm if any of the vertices of the gates, have children of different colors, all of those vertices need to be included in the solution as well, see Fig. 2.1.12(b). This algorithm finds the MCS for a given two-colored unweighted caterpillar with n vertices in $O(n)$ time.



(a) (b)

Fig. 2.1.12: (a) Example when the left most left gate (p_l, q_l, v_l) is to the left of the right most right gate (p_r, q_r, v_r). (b) Example of the MCS of a caterpillar when the left most left gate (p_l, q_l, v_l) is to the right of the right most right gate (p_r, q_r, v_r).

2.1.1.5 Combs

The last algorithm discussed by Dey et al. [4] is to find the MCS for a two-colored unweighted comb. To solve this problem, a similar approach to the caterpillar is taken. Using the leftmost left gate, and rightmost right gate the problem is broken down to smaller problems that need to be solved. Each unsolved portion, is solved with a similar approach to a path, by transforming it to a shortest path problem. Unlike the shortest path problem for the paths, in this problem the weight of the edges added between two vertices in the overlay graph consists of the sizes of the MCS for the unsolved paths connected to the vertices between the two end of the edge. To help find the weight of these edges in the shortest path problem, the sizes of the solutions for each path connected to the vertices of the skeleton is calculated base on all the vertices in the same block and stored in the vertices, so they can be accessed when needed. This algorithm can correctly compute the MCS for a comb with n vertices in $O(n^2)$ time, see Fig. 2.1.13.

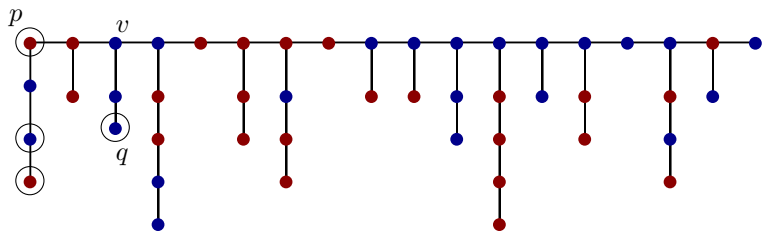


Fig. 2.1.13: Example of the MCS for a two-colored comb base on the left most left gate (p, q, v).

2.1.2 The MCS problem on two-colored unweighted trees

In a another study Dey et al. [5] proposed an algorithm to find the MCS for two-colored unweighted trees using a bottom-up dynamic programming approach in $O(n^4)$ time. In this section we summarize their algorithm. This study uses the same ideas discussed in the previous section to find the MCS of simple two-colored unweighted trees. A *useful gate* in two-colored unweighted trees is defined as follows (see also [5]). Assume T is rooted at a leaf. Let (p, q, v) be a gate in a given two-colored unweighted tree T. The gate (p, q, v) is called useful if there are no other gates in the subtrees connected to the vertices along the paths $v \rightsquigarrow p$ and $v \rightsquigarrow q$ in a deeper level. Let (p, q', v') be a gate containing a useful gate (p, q, v). Let $\mathcal{S}'$ be the set of solved vertices in the subtrees that will be closer to either p or q' from the gate (p, q', v') assuming that p and q' are in the solution. Moreover, let $\mathcal{S}$ be the set of solved vertices in the subtrees that will be closer to either p or q from the useful gate (p, q, v) assuming p and q are added to the solution. Since v is a descendant of v', $\mathcal{S}'$ is always a subset of $\mathcal{S}$. This implies checking the size of the consistent subset for the gate (p, q', v') would be redundant as it would never be smaller than the size of the consistent subset for the useful gate (p, q, v), see Fig. 2.1.14.

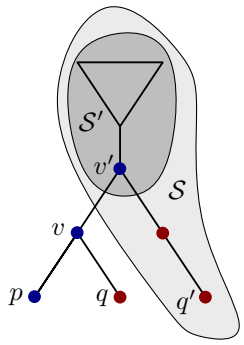


Fig. 2.1.14: Example of a useful gate (p, q, v). All the vertices in the area $\mathcal{S}'$ will be closer to either p or q' base on their color in the solution, while the vertices in $\mathcal{S}$ will be closer to either p or q.

Now, to find the MCS of a two-colored unweighted tree T Algorithm 1 in [5] works by generating a new anchored tree base on each vertex and adding it to a set. Moreover, for each edge in T, a fictitious vertex is added on the edge, and a new tree anchored at the fictitious vertex is generated and added to the same set as well. Then the algorithm checks all the trees in the resulting set and removes all the trees that are not anchored at any useful gate. The algorithm then computes the size of the consistent subset for all of the remaining anchored trees. For each anchored tree, it might be the case that the tree is anchored at several useful gates of different sizes. The algorithm finds the MCS base on each useful gate for all of the anchored trees, and chooses the one with the smallest consistent subset as the MCS of the tree. Let T be a two-colored unweighted tree anchored at useful gate (p, q, v). As previously discussed due to (p, q, v) being a useful gate all of the vertices on subtrees connected to v, except the subtrees including p and q, are closer to either p or q in the solution with the same color. However, the subtrees connected to the vertices along the path $v \rightsquigarrow p$ and $v \rightsquigarrow q$ are not solved, see Fig. 2.1.15.

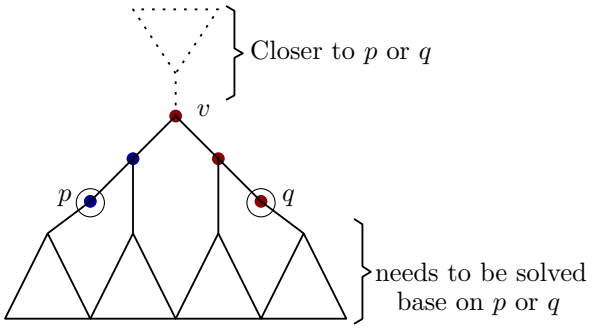


Fig. 2.1.15: Example of a useful gate (p, q, v).

The solution for the subtree connected to each vertex u along the path $v \rightsquigarrow p$ is computed with the assumption that no vertex is closer to u than p in the solution for that subtree. This is done to ensure the consistency of the subset in other branches. The same approach is taken for the vertices along the path $v \rightsquigarrow q$ with respect to vertex q in the solution. Consider T_u as the unsolved subtree connected to vertex u along the path $v \rightsquigarrow p$. Let T' be the union of T_u and the path from $p \rightsquigarrow u$. To find the MCS for T' with the assumption that p is the closest vertex to u in the solution the algorithm works as follows. Consider all the paths from p to a leaf in T', and compute a consistent subset for T' base on each path. The solution for T' base on each path is computed by transforming it to a shortest path problem. In the new shortest path problem, similarly to the comb discussed in the previous section, the weight of an edge between two vertices (regardless of whether its connecting two vertices of the same or opposite colors) is considered as the summation of sizes of the solutions for the unsolved subtrees connected the vertices between the two end of the edge. To find the size of the solution for each unsolved subtree while computing the weight of the edges in the shortest path problem further recursions happen that can be solved with the same approach. The algorithm computes the size of the consistent subset base on each path in T', and chooses the one with smallest one as the MCS of T'. The same process is done to find the MCS for all the unsolved subtrees connected to the vertices on the path $v \rightsquigarrow p$ and $v \rightsquigarrow q$. The size of consistent subset base on each useful gate is computed by computing the MCS sizes for all of the unsolved subtrees, and adding them together. To help with the computations that happen while solving

the MCS base on each useful gate in an anchored tree, the solutions for all required subtrees in an anchored tree is computed using a bottom-up dynamic programming approach and stored in the vertices to be used when needed. The algorithm repeats this process for all of the useful gates in all of the anchored trees, and chooses the smallest consistent subset as the MCS of the given two-colored unweighted tree in $O(n^4)$ time, see Fig. 2.1.16.

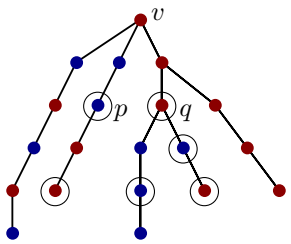


Fig. 2.1.16: Example of the MCS of a two-colored tree using the useful gate (p, q, v).

2.2 The Euclidean MCS problem

In a recent study Biniaz et al. [2] proposed different algorithm to solve the MCS problem for points in the Euclidean plane in different settings and formations.

The first algorithm discussed in [2] is to solve the decision version of the MCS problem for points in the Euclidean plane. Consider P as a set of n colored points in the Euclidean plane. To find the MCS of these points, the trivial approach is to check all the possible subsets of P to find the smallest consistent subset. This approach has an exponential running time. The decision version of this problem, trying to check if a consistent subset of size k exist, can also be solved trivially by checking all the possible subsets of size k with the time complexity of $n^{O(k)}$. The algorithm presented in [2] runs in $n^{O(\sqrt{k})}$. It uses dynamic programming to divide the problem into smaller portions that can be solved recursively. Assume S is the a consistent subset for the points P of size k. Let $\mathcal{V}$ be the Voronoi diagram for the points of S. Every point of P in the plane is in the Voronoi cell of a point $s \in S$, and closer to s than any other point of S. Notice as every point of P 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. 2.2.1.

A *balanced curve separator* for a triangulated planar graph is a cycle that splits the graph to two portions, inside and outside of the cycle, where the number of vertices in each portion is balanced in regards to the total number of vertices. A variation of $\mathcal{V}$ has a balanced curve separator of size $O(\sqrt{k})$, such that it divides the problem to two parts inside and outside of the curve. The vertices of a balanced curve separator alternate between the points of P and the vertices of the Voronoi diagram that are introduced by three points of P. For each balance curve separator the points from P that the curve passes, and the three points that introduced each Voronoi vertex that the curve passes are added to the consistent subset, see Fig. 2.2.1. The solutions for the problems inside and outside of the separator are also computed similarly by finding another balanced curve separator, and adding the points in the solution to the consistent subset. This process continues until the consistent subset of size k is found.

The issue is that S of size k is not known for the algorithm to compute $\mathcal{V}$, and the balanced curve separator. Each vertex of $\mathcal{V}$ is introduced by 3 points of P. This means that in total there exist $n^{O(\sqrt{k})}$ curve separators and Voronoi diagrams that all need to be checked. This algorithm finds if a point set P has a consistent subset of size k in sub-exponential time of $n^{O(\sqrt{k})}$. To find the MCS for a set of points, the algorithm starts from $k = 1$, and increases k until the smallest k with a solution is found.

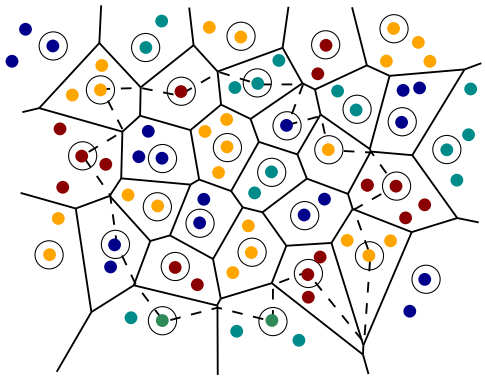


Fig. 2.2.1: Example of the solution S along with the Voronoi diagram of S and one of the balance curve separator for a set of points.

The next algorithm of Biniaz et al. [2] is a simple algorithm to find a MCS of size two for a set of two colored points in the Euclidean plane (if such a MCS exists). Let P be a set of two-colored points in the plane. Notice that if P has a MCS of size two, that implies that one red point r and one blue point b are forming the MCS. These two points are chosen such that all the red points in P are closer to r, and all of the blue points of P are closer to b. Consider two points r and b in the plane, notice that the perpendicular bisector of r and b denoted by ℓ separates the plane to two regions. Every point on the same region as r is closer to r, while every point on the region of b is closer to b. Using this approach the MCS of size two for a set of two-colored points can be found trivially by checking each pair (r, b) of opposite color, and computing the perpendicular bisector for them to check if the pair makes a valid MCS. Notice that if the pair (r, b) does form a valid MCS, the perpendicular bisector divides the red and blue points such that all the red points are in the same region as r and closer to r, while all of the blue points are in the same region as b and closer to b, see Fig. 2.2.2. This algorithm is able to find the MCS of size two by checking all the pairs, and checking the validity of the solution in $O(n^2 \log n)$ time for n points. Biniaz et al. [2] propose a nontrivial algorithm that projects the points in the three-dimensional environment, and uses a point cone incident approach to solve this problem in $O(n \log n)$ time for a point set with n points.

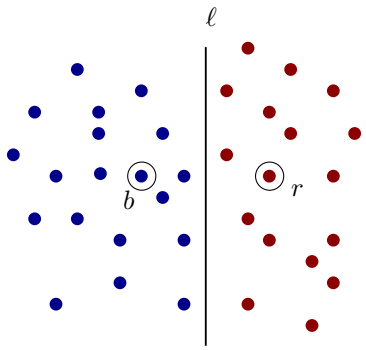


Fig. 2.2.2: Example of MCS of size two for a set of two-colored points in the Euclidean plane.

Biniaz et al. [2] also studies the MCS problem for restricted Euclidean point sets such as points on a line or on two parallel lines. For multi-colored points on a line

they propose an algorithm that solves the MCS problem in $O(n)$ time. This improves the previous $O(n^2)$ running time of Banerjee et al. [1]. To find the MCS for a set of collinear points P, the concepts of blocks for a set of collinear colored points on the x-axis is defined similarly to the blocks defined for graphs. Let the points of P be $p_1, \cdots, p_n$ from left to right, and let the blocks of P be $B_1, \cdots, B_m$ from left to right, see Fig. 2.2.3.

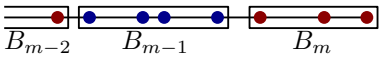


Fig. 2.2.3: Example of the blocks on a collinear point set.

This algorithm stores a table $T(.)$ for all the points on the line. For a point p_k the value of $T(k)$ shows the size of MCS for the portion of the line to the left p_k. While computing $T(k)$ for each point p_k in a block B_m, a point to left of p_k from the block B_m or B_{m-1} must be in the solution. The algorithm checks all these possible points as a pair for p_k to find the valid points p_i such that all the points between p_i and p_k are closer to either p_i or p_k that has the same color as them. Then by choosing the p_i with the smallest $T(i)$, the value for $T(k)$ is obtained, see Fig. 2.2.4. Instead of checking all the points in B_m and B_{m-1} to compute $T(k)$ in $O(n)$ time, by storing some additional information in each point, $T(k)$ can be computed in constant time. For each point p_k by storing the index of the previous point in the same block as p_k with smaller $T(.)$, the value $T(k)$ can be correctly computed in constant time. This implies in total it would take $O(n)$ time to compute the table for all the points starting from the left most point using a bottom-up dynamic programming approach.

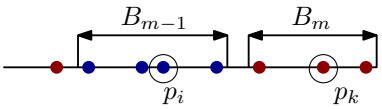


Fig. 2.2.4: Example of valid p_i for the point p_k while computing $T(k)$.

The last problem studied by Biniaz et al. [2] is to find the MCS for a set of points in the Euclidean plane lying on two parallel lines. Two different variation of this problem are studied for multi-colored and two-colored points. To find the MCS for

a set of multi-colored points lying on two parallel line, the algorithm first assumes that the points in the MCS are chosen only from the points of one line. Let P and Q be the set of multi-colored point on each line. With the assumption that the MCS for the points of $P \cup Q$ consists of only points from P, the algorithm projects the points of Q to P. Then using a similar approach to the collinear points, the MCS for the new line is found in $O(n \log n)$ time. In this new line the points of P are the only ones considered to be in the MCS, while all points of $P \cup Q$ are considered for the validity of the solution. A similar process happens if the MCS only contains the point from the line Q. Lastly, to find the MCS by selecting points from both lines, a dynamic programming approach is used to break the problem into smaller subproblems. The algorithm chooses one point from each line already in the solution that have the smallest distance to each other. This pair will divide the problem to two other similar instances, one to the right and one to the left of the chosen pair. Each subproblem is then solved recursively. To find the pair of points that are closest together in the solution, the algorithm checks all the pairs of points lying in different lines, and chooses the closest pair resulting in the smallest solution. The recursions continue until the MCS for the entire point set is found. This algorithm solves the MCS problem for a set of multi-colored points lying on two parallel lines in $O(n^6)$ time using dynamic programming where $|P \cup Q| = n$.

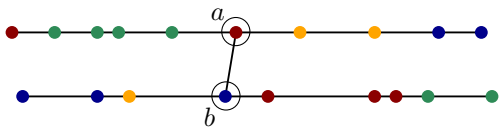


Fig. 2.2.5: Example showing the closest pair (a, b) in the solution. The unsolved portions to the the left and right of (a, b) are solved independently and recursively.

To solve the two-colored variant of this problem the algorithm of Biniaz et al. [2] works as follows. Let R be the set of all the red points lying in a single line in the plane, and let B be the set of all the blue points in another line parallel to the line passing through points of R. The algorithm works by choosing the pair (r, b) that are the rightmost pair of opposite color points in the solution, the problem can be broken down to a smaller subproblem to the left of the chosen pair. The new subproblem

is solved similarly by finding the rightmost pair of points of opposite color in the solution. To find each pair, the algorithm solves the MCS base on all the possible pairs of different color, and chooses the rightmost pair resulting the smallest solution. The recursions continue until the MCS for all the points is found. This algorithm correctly computes the desired MCS using a dynamic programming approach in the running time of $O(n^4)$ where $|B \cup R| = n$.

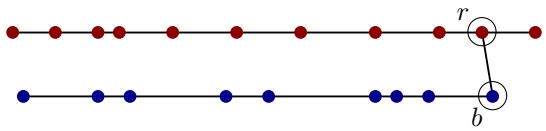


Fig. 2.2.6: Example showing the rightmost pair (r, b) in the solution. The unsolved portion to the the left (r, b) is solved recursively.

2.3 Applications of the MCS problem

Recall that the MCS problem was introduced by Hart [9] with the motivation to solve the nearest neighbour problem. Since then the MCS problem has shown to be useful in problems that require solving the nearest neighbour problem [3, 7, 8, 15]. Moreover, Gao et al. [6] and He et al. [10] discuss that many clustering algorithms that require finding the number of clusters before handling new test data, can benefit from using the MCS problem as well. Using the MCS problem a set of representatives for the training data can be obtained. The number of the samples obtained using the MCS is the optimal number of clusters for the given data set. This approach has been proven to be useful in k-clustering algorithms such as k-means and k-Nearest Neighbors.

Gao et al. [6] and Khodamoradi et al. [11] have shown that the MCS problem is useful in pattern recognition problems such as hand or speech recognition. The MCS problem can be used to find the frequent patterns that exist in a set data. By passing similar data to the MCS problem, the obtained MCS from each sample will also be similar. This approach can be used to find the frequent patterns that exist in a set of data.

Lastly, He et al. [10] shows that the MCS problem is applicable in complex classifying models such as Hyper Surface classification. Hyper Surface classification is mostly used when the size of dataset is large, and the data are in two or three dimensional space. The Hyper Surface classification algorithm requires a set of representatives from the training data set, so it can classify the new incoming test samples base on the labels of the existing representatives. They suggest an approach that uses the MCS problem to find the needed set of representatives for the model. They show that by passing the entire training data set to the MCS algorithm, the obtained MCS has optimal number of representatives in it. They further test this approach by creating a real life classifying model on a breast cancer dataset using Hyper Surface Classification, and the MCS problem. The MCS problem here is used to find the set of representatives from the training set needed to classify the incoming test data. They show as a result of the MCS having the smallest number of needed representatives, adding or removing any other samples to the MCS to be used in the classification will result in a lower overall accuracy for the predictions.

CHAPTER 3

Algorithm for the MCSS problem

In this chapter we present our algorithm to find the MCSS for any given tree, and prove our main result mentioned in Theorem 1. Notice that by taking the two end-point vertices of each bichromatic edge, a 2-approximation algorithm for the MCSS problem on trees can be obtained. Our proposed exact algorithm uses dynamic programming to break the MCSS problem to smaller subproblems, and solves it in $O(n^4)$ time for a tree with n vertices. Before presenting our algorithm and its details, we first need to discuss some preliminaries for the algorithm.

3.1 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 multi-colored trees.[1] 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 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.*

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

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 it's 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. 3.1.1.

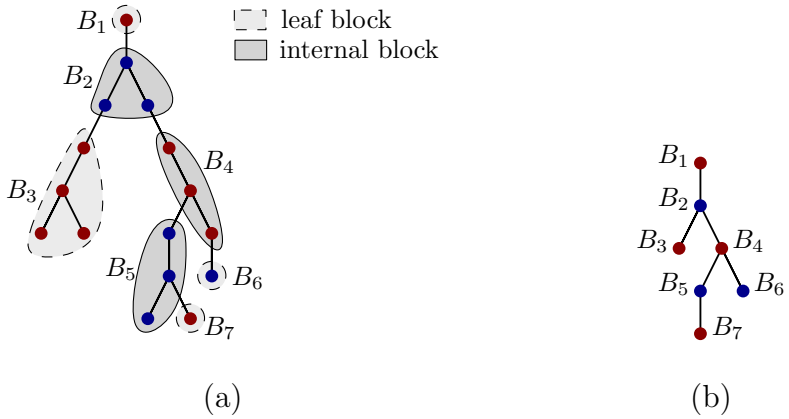
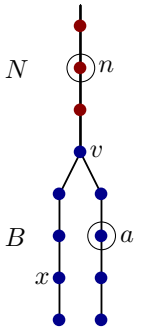


Fig. 3.1.1: (a) Example of a tree T together with its blocks. (b) The block tree of T.

We denote the shortest-path distance 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. Let S' be the set obtained from S^* by removing all the vertices of B except for a. We claim that 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. □

3.2 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.

3.2.1 Defining 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. 3.2.1(a). Let T' be the union of δ and T_c as in Fig. 3.2.1(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.

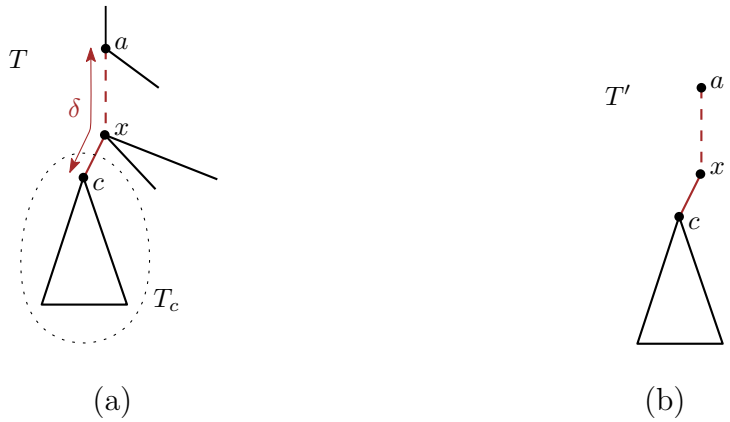


Fig. 3.2.1: (a) The tree T, and (b) the tree T'.

3.2.2 Solving the subproblems

We denote the size of the (constrained) MCSS for $T(a, c)$ by $S(a, c)$. 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. 3.2.2 (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.

We refer to any vertex z that satisfies the above constraints by 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. 3.2.2. Let Z be the set 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 a 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 $v' \in \mathcal{Z}$. Therefore

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

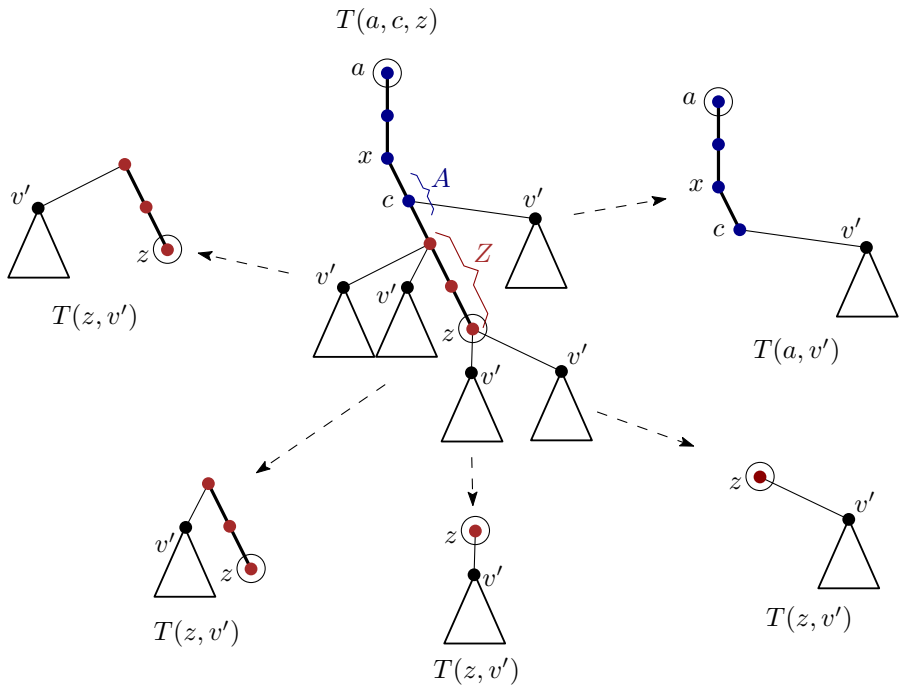


Fig. 3.2.2: Solving $T(a, c)$ recursively in terms of $T(a, v')$ and $T(z, v')$ where z is a valid pair for a.

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

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

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.

3.2.3 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,

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

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 $T(a, c)$.

3.2.4 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 in the computation of $T(a, c)$. However, this is not always the case; Fig. 3.2.3 shows that if we increase $dist(a, z)$ then $S(a, c, z)$ could increase (Fig. 3.2.3(a)) or decrease (Fig. 3.2.3(b)).

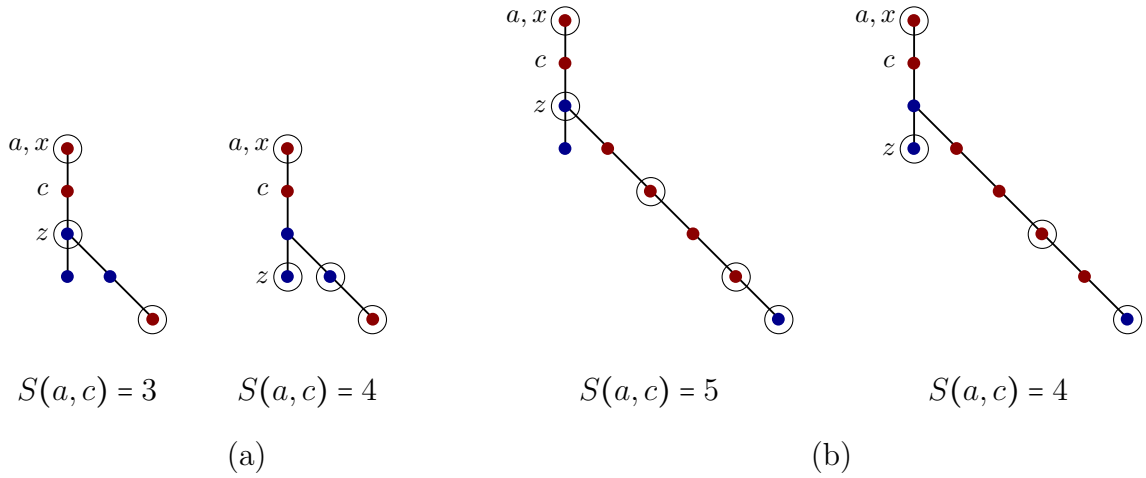


Fig. 3.2.3: If we increase $d(a, z)$ then $S(a, c)$ can increase (a) or decreases (b).

3.2.5 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.

3.2.6 Details of implementation

Let T be a tree with n vertices fixed at a leaf block. To find the MCSS of T in $O(n^4)$ we need to access the solution sizes for the subproblems of type $T(a, c)$ that happen in constant time. In this section we abuse the notation so that a vertex v refers to its index among all n points.

We store two different arrays A and B for each vertex. For each vertex x we store $|children(x)|$ arrays, each one $A_x[c][a]$ showing the sizes of the solution for

$T(a, c)$. Furthermore, we define $B_x[a]$ for a vertex x as the sum of all the answers of all its children, recognizing a is already in the solution covering x. By subtracting the values of $B_x[a]$ and $A_x[c][a]$ the size of the solution for all children of a vertex x where a certain child c should not be present is obtained in constant time (Notice that besides the summed solution size of the children, the number of subproblems without a solution must also be kept in $B_x[a]$ to correctly do the computations).

When computing the arrays for a vertex x in T we keep track of the MCSS sizes base on all the vertices in the same block as x. Let k be the size of the block containing x. We initialize each array $A_x[c][.]$ and $B_x[.]$ to have the size k one cell for each vertex in the block, and compute the cells of the arrays accordingly during the execution of the algorithm. This ensures that if a vertex a in the same block as x is in the solution and is covering x, the sizes of the MCSS for each child of x based on vertex a is accessible from the arrays in constant time.

At each level of T all the subtrees connected to the vertices of that level are disjoint. As a result, we start from the deepest level of our rooted tree T, and solve all the subproblems needed to fill the arrays for the vertices of that level, and use them to compute the arrays for the vertices in the previous level. The proposed algorithm computes the arrays, and finds the MCSS for T in $O(n^4)$ time.

An implementation for this algorithm has been done using p5.js framework [13].

CHAPTER 4

The MCSS problem on simple trees

In this section we introduce some algorithms with faster running times to solve the MCSS problem on simpler classes of trees. In the following sections, assume n is the total number of vertices in each tree. In this chapter we present algorithms to find the MCSS for weighted multi-colored paths, spiders, combs and caterpillars with respective running time of $O(n)$, $O(n^2)$, $O(n^3)$ and $O(n^3)$.

4.1 Paths

To find the MCSS for a path T first consider the case where T is unweighted. Recall that the MCS and the MCSS for an unweighted path are identical as a result of one vertex being in the solution from each block. Using the algorithm by Dey et al. [4] for the MCS problem on paths, the MCSS for a multi-colored unweighted path is computed in $O(n)$ time.

Now consider the case where T is weighted, in this case we transform the 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. Now, by computing the MCS for the set of multi-colored collinear points using the algorithm provided for collinear points by Biniaz et al. [2], we are able to find the MCSS for a multi-colored weighted path in $O(n)$ time.

4.2 Spiders

Let T be a spider centred at vertex v, and let B be the block T that contains v. To find the MCSS for T, we consider each vertex $v' \in B$ to be the closest vertex to v in the solution (v' can be v itself). Now, the size of the solution based on v' can be computed by finding the MCSS for all the paths from v' to all the leaf vertices that exist in T with the assumption that v' is the closest vertex to v in the MCSS of each path. It takes $O(n)$ to find the size of the consistent spanning subset in each path. This implies that we can compute the size of a consistent spanning subset for T in $|B| \times O(n)$ time. Notice that if v is the only vertex in B, then we can find the MCSS for T in $O(n)$ time. However, we might have $O(n)$ vertices in B which implies the running times of $O(n^2)$ for this algorithm in the worst case, see Fig. 4.2.1.

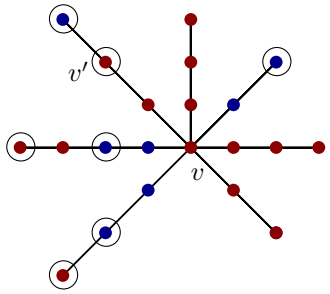


Fig. 4.2.1: Example of a consistent spanning subset for the spider, assuming v' is the closest vertex to v in the solution.

4.3 Combs

In this section we presenet an algorithm to find the MCSS of a comb. Let T be a comb, and let $p_1, \cdots, p_k$ to be the k vertices of its skeleton from left to right. Our algorithm solves instances of path problems (stores their solution values in a matrix M), and computes the sum of the values (in a matrix $\mathcal{S}$). Then it runs the algorithm of Chapter 3 using the information stored in M and $\mathcal{S}$. For each vertex p_i, let $\delta(p_i)$ be dangling path of p_i, and let q_i be the leaf of $\delta(p_i)$ (it might be that $q_i = p_i$). Let $\phi(p_i)$ be the block of $\delta(p_i)$ which contains p_i. Let a be any vertex in the block of p_i (it could be that $a = p_i$), and let $\delta(a, q_i)$ be the path from a to q_i, see Fig. 4.3.1.

Moreover, let $S(a, q_i)$ be the size of a MCSS for $\delta(a, q_i)$ with the constraint that a is in the solution, and no vertex in the solution is closer to p_i than a (i.e. a is covering p_i in the solution). We solve all the possible path problems $\delta(a, q_i)$ for each a in the block of each vertex p_i, and store the sizes of the obtained solutions in a matrix denoted by M with dimensions $n \times k$, where $M[a][i]$ (here we abuse the notation so that a refers to its index among all n points) has the value $S(a, q_i)$. If $\delta(a, q_i)$ does not have a solution we set $M[a][i] = +\infty$. To compute entries of M, we have to solve $O(kn)$ instances of path problem. This implies a running time of $O(kn^2)$ to compute all entries of M.

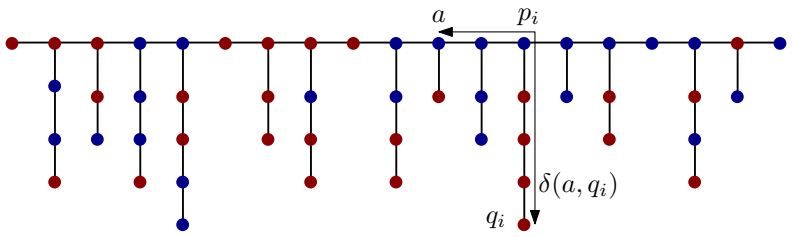


Fig. 4.3.1: Example of a comb T and a path $\delta(a, q_i)$.

Now that all the cells of M are computed, we create another matrix $\mathcal{S}$ with dimensions $n \times k$, whose entries are the sum of values in M. Take any p_i. Let x be any vertex in $\phi(p_i)$. We define $\mathcal{S}[x][i]$ to be the sum of values $M[x][j]$ where p_j is any skeleton vertex on the path from x to p_i. To compute $\mathcal{S}[x][j]$ we have to lookup the solution of $O(k)$ paths from M. Since the dimension of our matrix $\mathcal{S}$ is $O(nk)$, we can compute all the cells of $\mathcal{S}$ in $O(k^2 n)$ time.

Now we use $\mathcal{S}$ and M, and a modified version of our algorithm proposed in Chapter 3 to solve the MCSS problem for T. Let $\mathcal{T}$ be a subtree of T consisting of $\phi(p_1) \cup \cdots \cup \phi(p_k)$. $\mathcal{T}$ has only the blocks that contain a skeleton vertex. We also add two vertices p_0 and p_{k+1} to $\mathcal{T}$ with unique colors respectively as vertices on the skeleton to the left of p_1 and to the right of p_k, see Fig. 4.3.2. Notice that any MCSS of $\mathcal{T}$ contains $\{p_0, p_{k+1}\}$.

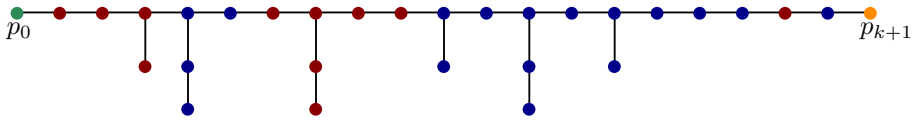


Fig. 4.3.2: The tree $\mathcal{T}$.

Let tree $\mathcal{T}$ be fixed at the leaf block containing p_0 . While executing the algorithm of Chapter 3, we consider any vertex of $\phi(p_1)$ to be a valid pair for p_0, and p_{k+1} to be a valid pair for any vertex in $\phi(p_k)$. Now, let a and c be two vertices of $\mathcal{T}$. Recall to solve the subproblem $T(a, c)$ we have to solve $O(n)$ instances of $T(a, c, z)$ for each valid pair z of a. Notice that while executing our algorithm on $\mathcal{T}$, for each $T(a, c, z)$ only one recursion of type $T(z, v')$ will happen. Let p_m be the last vertex with same color as a on the path between $a \rightsquigarrow z$. This implies that all the vertices from $p_{m+1} \rightsquigarrow z$ have the same color as z in $T(a, c, z)$. To account for the size of the MCSS for the paths connected to vertices $a \rightsquigarrow z$, the sizes of $\mathcal{S}[a][m]$ and $\mathcal{S}[z][m+1]$ are added to the size of the solution for $T(a, c, z)$ in constant time, see Fig. 4.3.3.

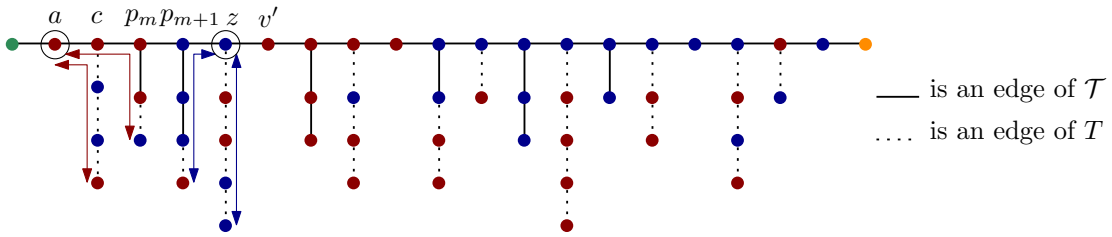


Fig. 4.3.3: An overlay image of T, $\mathcal{T}$ and $T(a, c, z)$. The sum of solution sizes for all the paths colored blue and red is respectively stored in $\mathcal{S}[a][m]$ and $\mathcal{S}[z][m+1]$.

By executing this modified algorithm on $\mathcal{T}$, and removing $\{p_0, p_{k+1}\}$ from any MCSS of $\mathcal{T}$ the size of the MCSS for T is obtained. These modifications imply to solve each subproblem of type $T(a, c)$ we have to check $O(n)$ valid pairs for a. Since only one recursion of type $T(z, v')$ happens, the size of the MCSS for each subproblem $T(a, c, z)$ can be computed in constant time using $\mathcal{S}$ and M. This implies by checking $O(n^2)$ subproblems of type $T(a, c)$ we can find the size of the MCSS for the comb in $O(n^3)$ time.

4.4 Caterpillars

In this section we present and algorithm to solve the MCSS problem on a caterpillar. This algorithm is similar to the algorithm proposed for combs in section 4.3 with a small modification in its the approach in computing the matrix M. This algorithm computes two matrices M and $\mathcal{S}$, and use them to run a modified version of the algorithm from Chapter 3 to find the MCSS.

Let T be a caterpillar, and let the k vertices of its skeleton to be $p_1, \cdots, p_k$ from left to right. Let a be any vertex in the block of p_i (it might be the case that $a = p_i$). Let $\delta(a, p_i)$ denote to the subtree containing all the dangling vertices of p_i colored differently than p_i together with the path from $a \rightsquigarrow p_i$. We define $S(a, p_i)$ as the size of the constrained MCSS on $\delta(a, p_i)$ with the assumption that a is in the solution, and a is covering all vertices on the path $a \rightsquigarrow p_i$. Since dangling vertices colored differently than the closest vertex to them on the skeleton form a block of size 1, they must be present in any consistent spanning subset of T. Thus, to find $S(a, p_i)$ we need to check all the dangling vertices of p_i in $\delta(a, p_i)$ to verify that by adding them to the solution p_i is still covered by a. We compute the size of MCSS for all $\delta(a, p_i)$, and store the values in a matrix M with dimensions $n \times k$. If $\delta(a, p_i)$ does not have a solution with the above constrains we set $M[a][i] = +\infty$. In the case that all the dangling vertices of p_i colored differently than p_i are added to the solution, and all vertices of $a \rightsquigarrow p_i$ are still covered by a, then we consider the number of dangling vertices for p_i colored differently than p_i as $S(a, p_i)$, and the value of $M[a][i]$, see Fig. 4.4.1.

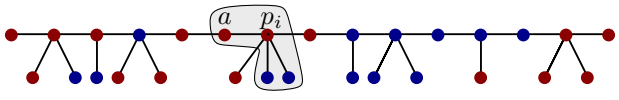


Fig. 4.4.1: The colored section shows $\delta(a, p_i)$, and $M[a][i] = 2$.

This process requires finding the solution for $O(kn)$ constrained MCSS for each $\delta(a, p_i)$ to compute all entries of the matrix M. In weighted caterpillars, the constrained MCSS problem on each $\delta(a, p_i)$ is solved by checking the validity for all $O(n)$ dangling vertices of p_i with different color than p_i, which implies a total running time

of $O(kn^2)$ for this part. In unweighted caterpillars, the solution for the MCSS problem on each $\delta(a, p_i)$ can be found in constant time by checking the validity of just one of the dangling vertices of p_i with different color than p_i, which implies a running time of $O(kn)$ for this part. Now, we continue the rest of the algorithm as mentioned in the section 4.3 for combs, and compute the matrix $\mathcal{S}$ using the matrix M, and find the MCSS of T in $O(n^3)$ time.

CHAPTER 5

Conclusions and future works

This thesis provides a comprehensive study pn different aspects of the MCS problem. In this problem, given a vertex-colored edge-weighted graph G the objective is to find a minimal sized subset S of the vertices of G, such that the closest neighbor to each vertex $v \in G$, in the chosen subset S is a vertex of same color.

We introduced the MCS problem on different settings such as general graphs and geometric graphs, and discussed its applications in clustering and classification algorithms. We also discussed some of the shortcomings of the MCS problem on trees, and proposed a new variant of this problem for trees called the Minimum Consistent Spanning Subset problem to get a better representative of the entire tree. We showed that the MCSS problem is NP-hard for general graphs, and we presented an algorithm that is able to find the MCSS for a multi-colored weighted tree with n vertices using a dynamic programming approach in $O(n^4)$ time. Using our main algorithm we also presented other algorithms to compute the MCSS for some simple trees such as path, spider, comb and caterpillars with respective running time of $O(n)$, $O(n^2)$, $O(n^3)$ and $O(n^3)$.

5.1 Future works

Two natural open problems regarding the minimum consistent spanning subsets would be to (i) improve the running time for trees, and (ii) present approximation algorithms for the general case.

REFERENCES

[1] Banerjee, S., Bhore, S., and Chitnis, R. (2018). Algorithms and hardness results for nearest neighbor problems in bicolored point sets. In Bender, M. A., Farach-Colton, M., and Mosteiro, M. A., editors, *LATIN 2018: Theoretical Informatics*, pages 80–93, Cham. Springer International Publishing.

[2] Biniaz, A., Cabello, S., Carmi, P., De Carufel, J.-L., Maheshwari, A., Mehrabi, S., and Smid, M. (2021). On the minimum consistent subset problem. *Algorithmica*, 83(7):2273–2302.

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

[4] Dey, S., Maheshwari, A., and Nandy, S. C. (2021). Minimum consistent subset of simple graph classes. In Mudgal, A. and Subramanian, C. R., editors, *Algorithms and Discrete Applied Mathematics*, pages 471–484, Cham. Springer International Publishing.

[5] Dey, S., Maheshwari, A., and Nandy, S. C. (2021). Minimum consistent subset problem for trees. In Bampis, E. and Pagourtzis, A., editors, *Fundamentals of Computation Theory*, pages 204–216, Cham. Springer International Publishing.

[6] Gao, B. J., Ester, M., Cai, J.-Y., Schulte, O., and Xiong, H. (2007). 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 '07, page 310–319, New York, NY, USA. Association for Computing Machinery.

[7] Gottlieb, L.-A., Kontorovich, A., and Nisnevitch, P. (2018). Near-optimal sample compression for nearest neighbors. *IEEE Transactions on Information Theory*, 64(6):4120–4128.

[8] Gowda, K. and Krishna, G. (1979). The condensed nearest neighbor rule using the concept of mutual nearest neighborhood (corresp.). *IEEE Transactions on Information Theory*, 25(4):488–490.

[9] Hart, P. (1968). The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516.

[10] HE, Q., ZHAO, X.-R., and SHI, Z.-Z. (2008). Minimal consistent subset for hyper surface classification method. *International Journal of Pattern Recognition and Artificial Intelligence*, 22(01):95–108.

[11] Khodamoradi, K., Krishnamurti, R., and Roy, B. (2018). Consistent subset problem with two labels. In Panda, B. and Goswami, P. P., editors, *Algorithms and Discrete Applied Mathematics*, pages 131–142, Cham. Springer International Publishing.

[12] Manna, B. and Roy, B. (2023). Some results on minimum consistent subsets of trees. *arXiv preprint arXiv:2303.02337*.

[13] The implementation of the Minimum Consistent Spanning Subset problem (2023). `https://parhamkhamsepour.com/mcss`.

[14] Wilfong, G. (1991). Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SCG '91, page 224–233, New York, NY, USA. Association for Computing Machinery.

[15] Yu, G., Tian, J., and Li, M. (2016). Nearest neighbor-based instance selection for classification. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 75–80.

VITA AUCTORIS

NAME:	Parham Khamsepour
PLACE OF BIRTH:	Qazvin, Iran
YEAR OF BIRTH:	1998
EDUCATION:	IKIU, Qazvin, Iran, 2016-2020, Bachelor's of Software Engineering
	University of Windsor, Windsor, Ontario, Canada, 2021-2023, M.Sc. in Computer Science