On Finding Optimal Sub-structures in Graphs

By

Sanjana Dey

A thesis presented for the award of the degree of
Doctor of Philosophy
in
Computer Science
at the
Indian Statistcal Institute, Kolkata

Under the supervision of
Professor Subhas C. Nandy
Advanced Computing and Microelectronics Unit
Indian Statistical Institute
203, B. T. Road, Kolkata, India 700 108
July 2022

Publications[1]

⋆ Subhadeep R Dev, Sanjana Dey, Florent Foucaud, Ralf Klasing and Tuomo Lehtila. *The Red-Blue Separation problem on graphs.* International Workshop on Combinatorial Algorithms (IWOCA), LNCS 13270(285-298), Springer, 2022.

⋆ Sanjana Dey, Anil Maheshwari and Subhas C. Nandy. *Minimum Consistent Subset Problem for Trees.* International Symposium on Fundamentals of Computation Theory (FCT), LNCS 12867(204-216), Springer, 2021.

⋆ Sanjana Dey, Anil Maheshwari and Subhas C. Nandy. *Minimum Consistent Subset of Simple Graph Classes.* International Conference on Algorithms and Discrete Applied Mathematics (CALDAM), LNCS, 12601(471–484), Springer, 2021.

⋆ Sanjana Dey, Florent Foucaud, Subhas C. Nandy and Arunabha Sen. *Discriminating Codes in Geometric Setups.* International Symposium on Algorithms and Computation (ISAAC), LIPIcs, 181(24:1-24:16), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

• Kaustav Basu, Sanjana Dey, Subhas C. Nandy and Arunabha Sen. *Sensor Networks for Structural Health Monitoring of Critical Infrastructures Using Identifying Codes.* International Conference on the Design of Reliable Networks (DRCN), (43-50), IEEE, 2019.

[1]The publications which are part of the thesis are highlighted with ⋆.

- Sanjana Dey, Ramesh K. Jallu and Subhas C. Nandy. *Minimum Spanning Tree of Line Segments.* Computing and Combinatorial Conference (COCOON), LNCS, 10976(529-541), Springer, 2018.

⋆ Sanjana Dey, Florent Foucaud, Subhas C. Nandy and Arunabha Sen. *Complexity and Approximation for Discriminating and Identifying Code Problems in Geometric Setups.* Revised draft submitted in February 2022. Submitted to Algorithmica in June 2021 (Manuscript No: ALGO-D-21-00098R1).

⋆ Sanjana Dey, Anil Maheshwari and Subhas C. Nandy. *Minimum Consistent Subset Problem for Trees.* Submitted to Journal of Combinatorial Optimization in June 2022 (Manuscript No: JOCO-S-22-00269).

⋆ Sanjana Dey, Anil Maheshwari and Subhas C. Nandy. *Minimum Consistent Subset of Simple Graph Classes.* Submitted to Discrete Applied Mathematics in July 2021 (Manuscript No: DA13552).

Acknowledgement

Having have a doctorate was a long standing dream for me. Hence the decision to pursue Ph.D. It wasn't an easy decision to live through. What one never knows before doing a Ph.D. is the number of failures are much higher than the number of successes. It is sheer grit and will-power that gets you through. But the one thing that helps one survive and successfully complete is the immense support of one's peers. Now that I am finally in the last phase of the journey, I would like to take the opportunity to thank everyone whose help and love made this thesis possible.

An advisor is an undetachable part in completion of a thesis and mine has been the same. Professor Subhas C Nandy has been one of the most important pillars behind this thesis. He has shown immense patience in handling a naive and simpleton like me. He has been a steady influence throughout my Ph.D. career. He has oriented and supported me with promptness and care, and has always been encouraging in times of difficulties. His hard work has set an example for me. He has even made me listen to Nazrul sangeet sometimes to lighten the mood around us and shared his experiences accumulated across his years. Without his encouragement and guidance this thesis would not have materialized.

After advisor comes the role of co-authors who also play an important part in shaping one's thesis. I would like to thank Dr. Florent Foucaud for working with me and teaching me a lot of tricks on the way. He has been a friend and mentor. He has supported me, guided me and helped me a great deal all through. Because of his initiative, I even got the opportunity to work in LIMOS, Clermont Ferrand

which was an experience on its own for me.

Another indispensible person whose role cannot be ignored is Professor Anil Maheshwari. He has been a mentor all through my journey. I have always been in his awe by the way he handles a new problem. He has always been a calm and patient listener and it has been an absolute delight working with him.

I would also wish to extend my gratitude to Professor Ralf Klasing for inviting me to LaBRI, Bordeaux. It was an amazing experience working with him and getting to know a dedicated and funny person like him.

I would also like to thank all my other co-authors Prof. Arunabha Sen, Kaustav Basu, Dr. Ramesh Jallu and Dr. Tuomo Lehtila for all their help.

I acknowledge, each and every member of Advanced Computing and Microelectronics unit (ACMU); teachers, workers and students alike; for a relaxed, family-like, but nonetheless research-oriented environment. I am also grateful to the Indian Statistical Institute (ISI), Kolkata, for providing me the required funds for various visits, as well as for the purchase of reference materials, and technical pieces of equipment during the course of my research.

Personally, I owe a huge deal to Subhadeep and Suranjana for taking care of me and being my personal cheerleaders. They have ensured my mental health throughout the whole time. They have made me laugh at times when smiling seemed difficult.

Last but not least, I am indebted to my parents for being my life-support and tolerating all my tantrums with love.

I hope you enjoy reading this thesis as much I enjoyed writing it.

Sanjana Dey

Abstract

In computer science, a problem is said to have an optimal sub-structure if an optimal solution can be constructed from optimal solutions of its sub-problems. These optimal sub-structures are computed in the classical graph-theoretic setting where the graph is a structure with a set of vertices and edges. In computational geometry, the vertex set is usually represented by a set of geometric objects like unit disks, etc., and the edge set is represented by the intersection of these geometric structures. In this thesis, three problems are investigated namely minimum discriminating codes, red-blue separation, and minimum consistent subset.

In the minimum discriminating codes problem, we handle some geometric structures like unit intervals and arbitrary intervals in $\mathbb{R}$ and axis parallel unit squares in $\mathbb{R}^2$. We prove the hardness of the problem in both one-dimensional and two-dimensional planes. We also propose PTAS for the unit interval case and a 2-factor approximation algorithm for the arbitrary interval case. In polynomial time we have given approximation algorithms producing constant-factor solution in $\mathbb{R}^2$ with axis parallel unit square objects. We have also studied a similar problem known as the minimum identifying codes in some geometric settings.

In the red-blue separation problem, we consider a graph whose vertices are colored red or blue. We study the computational complexity in some graph classes. We design polynomial-time algorithms when one of the colored classes is bounded by a constant. We also give some tight bounds on the cardinality of the optimal solution.

In the minimum consistent subset problem, we work with simple graph classes like paths, caterpillars, trees, etc. For each of these graphs, we have designed optimal algorithms. We have also considered both undirected and directed versions for a few of the graphs.

Keywords: dominating sets, discriminating codes, identifying codes, red-blue separation, consistent subset, graphs, approximation algorithms.

Table of Contents

List of Figures

CHAPTER 1

Introduction

Contents

1.1 Introduction

With the cognitive evolution of the species of homo sapiens, few of the most interesting inventions have been developed in the field of sciences. Under close observation, it can be claimed that computer science as a field has improved and grown by leaps and bounds. In the field of computer science the progress has been nothing less than evolutionary in the last century with the discovery of the 'bit'. The technology of computer science has shrunk the size of the world manifolds with the transfer of information being established in a jiffy. But as we all know, with great power comes great responsibility. Such huge amount of information

cannot be used raw. It needs processing and in some cases, even refining. Thus, arises the need to study structures such as graphs, which are modelled from real world scenarios, and deeply understand the sub-structures in such graphs for better processing and utilizing these vast array of information.

In mathematics, or more specifically in the field of graph theory, a *graph* is a structure that amounts to a set of objects in which some pairs of the objects can be said to be in some sense "related". The objects correspond to mathematical abstractions called *vertices* and each of the related pairs of vertices is called an *edge*. In diagrammatic form, a graph is typically depicted as a set of dots or circles for the vertices, joined by lines or curves for the edges.

The edges can be of two types: directed or undirected. For example, if the vertices represent people at a party, and there exists an edge between any two people if they have shaken hands, then this graph is undirected because a person p_1 can shake hands with another person p_2 only if p_2 also shakes his/her hands with p_1. In contrast, if any edge from a person p_1 to a person p_2 corresponds to p_1 owes money to p_2, then this graph is directed. The former type of graph is referred as an *undirected graph* while the latter type of graph is called a *directed graph*.

An *intersection graph* is a graph where each vertex is associated with a set and the vertices are connected by edges whenever the corresponding sets have a nonempty intersection. If the sets are geometric objects, the resulting graph is a *geometric graph*. For instance, the intersection graph of disks in the plane is a disk graph (shown in Figure 1.1).

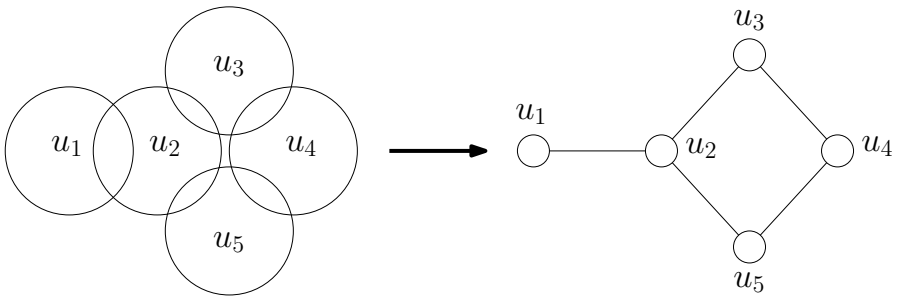


Figure 1.1: An intersection graph of unit disks.

This thesis deals with problems related to finding optimal sub-structures in graphs. Now that we know what we mean by a graph, let us give a formal definition of what do we mean by optimal sub-structure. Let a "problem" be a collection of "alternatives" $\mathcal{A}$, and let each alternative a have an associated cost $c(a)$. The task is to find a set of alternatives $A \subseteq \mathcal{A}$ that minimizes $\sum_{a \in A} c(a)$. Suppose that the alternatives can be partitioned into subsets, i.e. each alternative belongs to only one subset and each subset has its own cost function. The minima of the global cost function can be found restricted to the subset. A few important sub-structures in graphs which make them an indispensable asset in computer science are vertex cover, edge cover, independent set, maximum clique, maximum matching etc. Each of these sub-structures have their unique set of applications in the real world.

In order to judge the efficiency of the method of computing an optimal sub-structure of a graph, one needs to analyze the time and space complexities of executing the corresponding algorithm. If the running time of an algorithm is upper bounded by a polynomial expression in the size of the input for the algorithm, i.e., $T(n) = O(n^k)$ for some positive constant k then we call it a *polynomial time algorithm*. Some examples of polynomial time algorithms in a graph are maximum matching, edge cover, etc. If $k = 0$, then the algorithm is known as a *constant time algorithm*. All the basic arithmetic operations (addition, subtraction, multiplication, division, and comparison) can be done in constant time.

Problems for which a deterministic polynomial time algorithm exists belong to the complexity class P. The concept of polynomial time leads to several complexity classes in computational complexity theory. One of the most important classes defined using polynomial time is NP (nondeterministic polynomial time). NP is the complexity class of decision problems[1] that can be solved on a non-deterministic Turing machine in polynomial time [KT05]. The two main types of problem widely studied with respect to class NP are:

NP-hard problem: A problem H is NP-hard when every problem L in NP can

[1] A decision problem is a problem that can be posed as a yes–no question of the input values.

be reduced in polynomial time to an instance of H; that is, assuming a solution for the reduced instance of H one can produce the solution of the problem L in polynomial time [VL91, Knu74].

NP-complete problem: A problem is NP-complete when it is both NP-hard and in NP.

Simple example of NP-hard problems are the minimum vertex cover problem, the maximum independent set problem, etc. whereas the independent set decision problem is NP-complete.

In complexity theory, co-NP is another complexity class. A decision problem X is a member of co-NP if and only if its complement $\overline{X}$ is in the complexity class NP. The polynomial hierarchy is a hierarchy of complexity classes that generalize the classes NP and co-NP.

Oracle Definition of Polynomial Hierarchy

Definition 1.1. "For the oracle definition of the polynomial hierarchy, define

$$\Delta_0^{\mathsf{P}} := \Sigma_0^{\mathsf{P}} := \Pi_0^{\mathsf{P}} := \mathsf{P},$$

where P is the set of decision problems solvable in polynomial time. Then for $i \geq 0$ define

$$\Delta_{i+1}^{\mathsf{P}} := \mathsf{P}^{\Sigma_i^{\mathsf{P}}}$$

$$\Sigma_{i+1}^{\mathsf{P}} := \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$$

$$\Pi_{i+1}^{\mathsf{P}} := \mathsf{coNP}^{\Sigma_i^{\mathsf{P}}}$$

where P^{A} is the set of decision problems solvable in polynomial time by a Turing machine augmented by an oracle for some complete problem in class A."

The classes NP^{A} and $\mathsf{coNP}^{\mathrm{A}}$ are defined analogously. For example, $\Sigma_1^{\mathsf{P}} = \mathsf{NP}, \Pi_1^{\mathsf{P}} = \mathsf{coNP}$, and $\Delta_2^{\mathsf{P}} = \mathsf{P}^{\mathsf{NP}}$ is the class of problems solvable in polynomial time with an oracle for some NP-complete problem.

In the field of theoretical computer science as a consequence of the widely believed P $\neq$ NP conjecture, a wide class of optimization problems (in particular NP-hard problems) cannot be solved exactly in polynomial time. Thus, arises the need of approximate solutions in place of optimal solutions which gives us the following definition:

Approximation algorithm [WS11]

Definition 1.2. "An approximation algorithm for a problem is an efficient (most preferably polynomial time) algorithm that given any arbitrary instance of the problem, one can find a solution to that optimization problem with provable guarantee on the difference of the returned solution to the optimal solution of the problem for that given instance."

By now there are several established techniques to design approximation algorithms. These include greedy algorithms, local search, enumeration and dynamic programming etc.

Polynomial-Time Approximation Scheme (PTAS)

Definition 1.3. "A PTAS is an algorithm which takes an instance of an optimization (minimization/maximization) problem and a parameter $\epsilon > 0$ and produces a solution that is within a factor $(1 + \epsilon)$ of the optimal solution of the minimization problem or $(1 - \epsilon)$ of the optimal solution of the maximization problem."

For example, for the Euclidean traveling salesman problem, a PTAS would produce a tour with length at most $(1 - \epsilon)L$, where L is the length of the shortest tour for the given instance. The running time of a PTAS is required to be polynomial in the problem size i.e. $T(n) = f(n)g(n, \epsilon)$. Thus, an algorithm running in time $O(n^{1/\epsilon})$ or even $O(n^{exp(1/\epsilon)})$ counts as a PTAS. Although getting a PTAS is sometimes not possible.

APX

Definition 1.4. "In computational complexity theory, the class APX is the set of NP optimization problems that allow polynomial-time approximation algorithms with approximation ratio bounded by a constant."

The traveling salesman problem when the distances in the graph satisfy the conditions of a metric belongs to the class of APX. It is also to be noted that there exists problems for which finding a constant factor approximation algorithm might not be possible. An instance would be the set cover problem.

In computer science, *parameterized complexity* is a class of computational complexity theory which focuses mainly on classifying computational problems according to their inherent difficulty with respect to multiple parameters of the input or the output. The complexity of a problem is measured as a function of those parameters. This has allowed the classification of NP-hard problems on a finer scale than in the classical setting, where the complexity of a problem is only measured as a function of the number of bits in the input.

The existence of efficient, exact, and deterministic solving algorithms for NP-complete, or otherwise NP-hard, problems is considered unlikely, if input parameters are not fixed; all known solving algorithms for these problems require time that is exponential in the total size of the input.

Fixed Parameter Tractable algorithm [DF99]

Definition 1.5. "Some problems can be solved by algorithms that are exponential only in the size of a fixed parameter while polynomial in the size of the input. Such an algorithm is called a fixed-parameter tractable (FPT-)algorithm, because the problem can be solved efficiently for small values of the fixed parameter."

FPT problems are those that can be solved in time $f(k) \cdot |x|^{O(1)}$ for some computable function f, k is the given parameter and $|x|$ is the size of the input x of the problem. Typically, this function is thought of as single exponential, such as

$2^{O(k)}$ but the definition admits functions that grow even faster. A vertex cover of size k in a graph of n vertices can be found in time $O(2^k n)$, so this problem is in FPT [CKX10]. The problem of finding the maximum clique is both fixed parameter intractable and hard to approximate. However, for sparse graphs (graphs in which the number of edges is at most a constant times the number of vertices in any subgraph), the maximum clique has bounded size and may be found exactly in linear time [CN85].

1.2 Scope of the thesis

In this thesis, we have studied three special types of optimal sub-structures in graph settings and in geometric settings which are closely related to a fundamental problem of finding a dominating set in a graph (which has been defined later). Apart from theoretical interest, all of these problems have huge practical applications as well. The specific problems which have been focused in this thesis are enlisted.

Minimum Discriminating and Identifying Codes: Given a set of points P and a set of objects S, we study the minimum discriminating code and the minimum identifying code problem in some geometric set-ups. In the minimum discriminating code problem, the objective is to select a minimum size subset $S' \subseteq S$ such that all the points are covered by at least one object in S' and for any pair of points p_i, p_j, the subset of objects covering p_i and the subset of objects covering p_j are different. The obtained results of the problem of minimum discriminating codes in various geometric set-ups are listed below:

- We have shown that when the set of points P are on a horizontal line and the set of objects S are arbitrary length intervals, then finding the minimum discriminating code problem is NP-complete.

- We have designed a 2-factor approximation algorithm for the above problem.

- When the set of points P are on a horizontal line and the set of objects S are unit length intervals, then we have designed a PTAS.
- Also, when the set of points P are on the $2D$ plane and the set of objects S are axis parallel unit squares, then finding the minimum discriminating code is shown to be NP-complete.
- With the above setting, we have also given constant factor approximation algorithms.

For the identifying code problem, we are given a set of objects S and the objective is to choose a subset of objects $S' \subseteq S$ such that each object in S have non-empty intersection with some object in S' and the subset of objects of S' having non-empty intersection with each object in S is unique. In our study, the objects in S are axis parallel unit squares, and we have designed a constant factor approximation algorithm for the minimum identifying code problem.

Red-Blue Separation: We study two variations of the separation problem. In the first variation, we are given a graph whose vertices are colored either red or blue, and we are to find a subset V' of the vertex set of minimum cardinality such that no red-blue vertex pair is dominated by the same subset of vertices in V'. The results we have discussed regarding the minimum red-blue separation are as follows:

- We have shown that the problem is NP-complete in several restricted graph classes.
- When the smaller color class is of unit size, the problem is hard to approximate with a factor of $(1 - \epsilon) \ln n$ even for split graphs.
- On general graphs, the problem has an algorithm that can produce a $2 \ln n$-factor approximation result.
- When the smaller color class is bounded by a constant, the problem can be solved in polynomial time in triangle free graphs and bounded-degree graphs.

The other variation studied is where the coloring of the vertices is not specified and we study the problem of finding the maximum value achievable for the minimum red-blue separation over all possible colorings of the vertices of the graph. In this max-version of the separation problem we have discussed the following results:

- We show the existence of tight bounds on the size of V' in terms of the number of vertices of a graph which are $\lfloor \log_2 n \rfloor$ for general graphs and $\frac{2n}{3}$ for trees.
- We show that the problem can be approximated within a factor of $O(\ln^2 n)$.

Minimum Consistent Subset: This problem is studied in a graph theoretic setting. We are given a graph $G = (V, E)$ whose vertices are colored red or blue. The objective is to find a subset of vertices $V' \subseteq V$ vertices such that for any vertex $v \in V$, the vertex $v' \in V'$ closest to v is of the same color as that of v. We have given polynomial time algorithms for various graph classes, listed below. We have studied the problem for the directed versions of some graph classes.

- For paths (both undirected and directed), we have given linear time algorithms.
- For spiders (both undirected and directed), we have designed quadratic and linear time algorithms respectively.
- For undirected caterpillars we have give an $O(n)$ time algorithm.
- For undirected trees we have designed an $O(n^4)$ time algorithm.

1.3 Organization of the thesis

The thesis is organized in a total of seven chapters. In Chapter 2, we give a detailed overview of related works. In Chapter 3, we study the minimum discriminating

and minimum identifying code problem in various geometric set-ups. In Chapter 4, we study the problem of red-blue separation in bi-coloured graphs. In Chapter 5 and 6, the minimum consistent subset problem for various graph classes have been studied. Finally, in Chapter 7, concluding remarks on our studies and possible future directions of research are illuminated.

CHAPTER 2

Review and Related Works

This thesis studies three types of problems for extracting optimal sub structures in a graph:

(i) minimum discriminating/identifying codes,
(ii) red-blue separation and
(iii) minimum consistent subset.

The existing study of each of these three problems in the literature has been described in details in these sections.

Contents

2.1 Discrimination and Identification

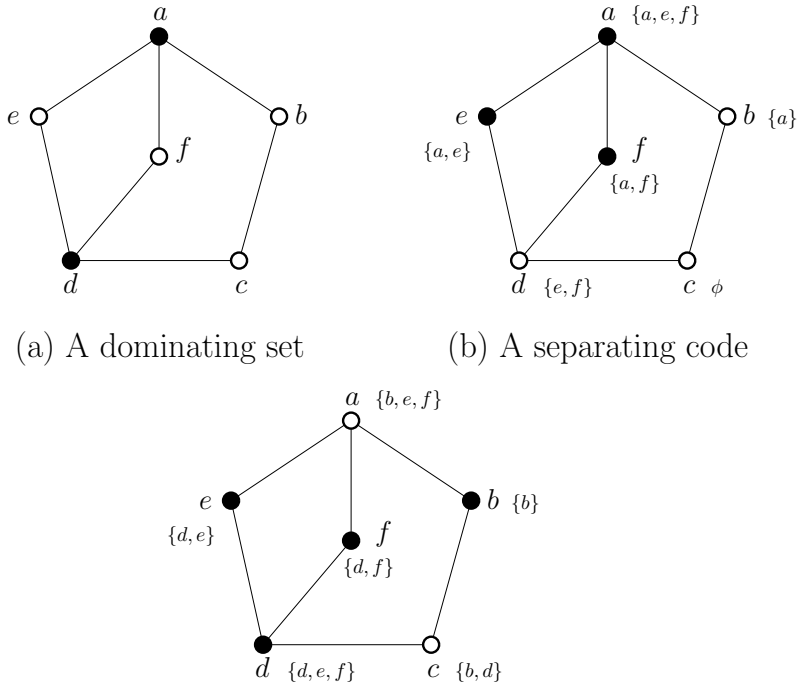
Separation problems for discrete structures have been studied extensively from various perspectives. In the 1960s, Rényi [Ré61] introduced the SEPARATION problem for set systems (a set system is a collection of sets over a set of elements), which has been rediscovered by various authors in different contexts, see e.g. [Bon72, CCHL08, HY14, MS85]. In this problem, one aims at selecting a solution subset $\mathcal{S}$ of sets from the input set system to separate every pair of elements, such that the subset of $\mathcal{S}$, which correspond to the sets to which each element belongs to, is unique. The graph version of this problem (where the sets of the input set system are the closed neighborhoods of a graph), called IDENTIFYING CODE [KCL98], is also extensively studied. The two problems that are primarily discussed in this section are minimum discriminating codes and minimum identifying codes. We first discuss the problem of minimum identifying codes as it can be considered as a parent problem for minimum discriminating codes.

A graph G is henceforth defined as a tuple (V, E) where V is the set of vertices and E is the set of edges. The closed neighborhood $N[v]$ of a vertex v in a graph is the set of all the vertices adjacent to v including v itself. An identifying code of a graph G is a subset of vertices of G that allows one to distinguish each vertex of G by means of its neighborhood within the identifying code. This notion, defined by Karpovsky, Chakrabarty and Levitin in 1998 [KCL98], has been widely studied since then. In order to define an identifying code formally the notion of dominating set and separating set is required.

Dominating set

Definition 2.1. A dominating set of a graph $G = (V, E)$ is a subset $\mathcal{D}$ of vertices of G such that for each vertex $v \in V$, $\mathcal{D} \cap N[v] \neq \phi$. The size of the dominating set is known as domination number and it denoted by $\gamma(G)$.

An example of a dominating set is shown in Figure 2.1(a). It is to be noted that dominating sets and many of their variants have been studied extensively in two



(a) A dominating set (b) A separating code

(c) An identifying code

Figure 2.1: A graph with (a) dominating set, (b) separating code and (c) identifying code; highlighted in black vertices.

classic textbooks [HHS98b] and [HHS98a].

Separating set

Definition 2.2. A separating set is a subset $\mathcal{S}$ of vertices of $G = (V, E)$ such that for each pair u, v of distinct vertices in V, we have $N[u] \cap \mathcal{S} \neq N[v] \cap \mathcal{S}$.

We can assign a code to each vertex with respect to a separating set which we call as the *separating code* of the vertex. The code of each vertex $v \in V$, with respect to $\mathcal{S}$, is the combination of the vertices of $N[v] \cap \mathcal{S}$. An example of a separating code is depicted in Figure 2.1(b) where the code of each vertex is shown.

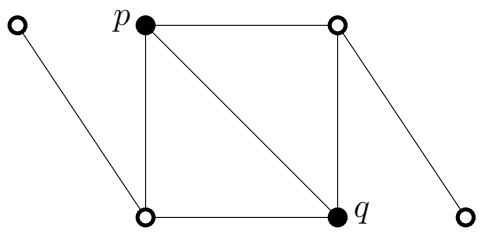


Figure 2.2: Twins in a graph.

Identifying code [KCL98]

Definition 2.3. Given a graph G, a subset $\mathcal{C}$ of V is an identifying code of G if $\mathcal{C}$ is both a dominating set and a separating set of G i.e.

- for a vertex $v \in V$, $\mathcal{C} \cap N[v] \neq \phi$ and
- for a pair of vertices u, v, $N[u] \cap \mathcal{C} \neq N[v] \cap \mathcal{C}$.

An identifying code has been represented in Figure 2.1(c) where the code of each vertex is also given. It should be noted that both the dominating set and the separating set may not produce the identifying code for a graph, as shown in Figure 2.1.

An important observation on a graph is that: not all graphs can admit the identifying code of the vertices of the graph.

Twins

Definition 2.4. Two distinct vertices u and v are called twins if $N[u] = N[v]$. Here u and v cannot be separated as mentioned in Observation 2.1.

An example with two twin vertices p and q is shown in Figure 2.2.

Observation 2.1

A graph has an identifying code if and only if it is twin-free.

In fact, in a twin-free graph G, $\mathcal{C} = V$ is itself an identifying code of G, $\mathcal{C}$ is definitely a dominating set, and for each vertex $v \in V$, we also have $N[v] \cap \mathcal{C} =$

$N[v]$. Because of the twin-freeness of G, all these sets are unique, and $\mathcal{C}$ is also a separating code of G. All the graphs considered in this thesis are *twin-free*. Twin free graphs have also been studied independently in [CHHL07, Aug08, LHH+09, ACHL10, CHL12].

The upper and lower bound of identifying codes in general graphs is shown to be $(n-1)$ in [Ber01, GM07] and $\lfloor \log_2(n+1) \rfloor$ in [KCL98] respectively. The tightness of the bounds has been proved in [Fou12] and [Mon06] respectively. The same bounds are true for bipartite graphs. The upper bound for both interval graphs and unit interval graphs is $(n-1)$ [Ber01, GM07]. The lower bounds for interval graphs is $\sqrt{2n+\frac{1}{4}}-\frac{1}{2}$ and that of unit interval graphs is $\frac{n+1}{2}$ [Fou12]. The identifying code problem has been shown to be NP-complete for bipartite graphs in [CHL03]. Even in planar bipartite unit disk graph the problem remains NP-complete [MS09]. It was shown that Min-ID-Code (and thus Min-Disc-Code) for graphs is log-APX hard [LT08], and this holds even for split graphs, bipartite graphs, co-bipartite graphs [Fou15], and for bipartite graphs of girth 6 [BLL+15]. However, for line graphs and planar graphs, Min-ID-Code remains NP-complete but constant factor approximation algorithms are available (see [FGN+13] and [BFS19, SR84], respectively).

The notion of geometric separator in computational geometry comes from [DHMS01]. Let us assume having k finite disjoint sets $C_1, \ldots, C_k$ of $\mathbb{R}^2$. A finite set of curves $\mathcal{S}$ in the plane is a separator for the sets $C_1, \ldots, C_k$ if every connected component in $\mathbb{R}^2 \setminus \mathcal{S}$ contains points from only one set C_i. Finding separators has been a classical problem of computational geometry, in particular for image analysis. The most studied case is $k = 2$, i.e., there are two types of points and the separation is to be done with lines or circles [AHM+00]. In [GP19], the problem of identifying n points in the plane using disks, i.e., minimizing the number of disks so that each point is contained in a disk and no two points are in exactly the same set of disks, is considered. In [GP19], it is proved that if there are no three colinear points nor four co-circular points, then $\lceil \frac{n}{6} \rceil + 1$ disks are enough, improving the known bound of $\lceil \frac{n+1}{2} \rceil$ when no three points are colinear. This problem was mentioned in [GT13], which considered a more general separation with convex sets. In par-

ticular, they proved that $\lfloor \frac{n}{2} \rfloor$ circles are enough to separate n points even if they are in a general position (i.e. no three colinear points). In [BU95] an algorithm is given with time complexity $O(n \log n)$ to find a family of $\frac{n}{2}$ lines that separates any set of n points in a general position. In [CDKW05] it is proved that in the case, particularly where the lines are parallel to the axis, the problem is NP-complete and gave a polynomial time constant factor approximation algorithm for this case. The identifying code problem in the grid i.e. in $\mathbb{Z}^2$, using Euclidean balls is studied in [JL11]. The underlying graph has a subset of grid points in $\mathbb{Z}^2$ as vertices and the closed neighborhoods are given by the Euclidean balls of a fixed radius r. The authors give lower and upper bounds on the sizes of minimum identifying codes in this graph as a function of r. An implementation of indoor location detection systems based on identifying codes with some experimental evidence is shown in [UTS04].

A problem that is much related to identifying codes and which has been studied for several decades is the test cover problem, which generalizes the separating set. Let $\mathcal{I}$ be a set of elements and $\mathcal{A}$ be a set of subsets of $\mathcal{I}$. Members of $\mathcal{I}$ and $\mathcal{A}$ are referred to as individuals and attributes respectively. We say that an attribute $a \in \mathcal{A}$ separates two distinct elements i, i' of $\mathcal{I}$ if a is contained in exactly one of i or i'. A test cover of the set system $(\mathcal{I}, \mathcal{A})$ is a set $\mathcal{T} \subseteq \mathcal{A}$ such that each pair of distinct elements of $\mathcal{I}$ is separated by some subset of $\mathcal{T}$. Note that, as in the case of separating sets, a test cover may only exist if all pairs of individuals can actually be separated. In that case, we say that the set system $(\mathcal{I}, \mathcal{A})$ is *identifiable*. The notion of a test cover has appeared in a large number of papers under different contexts (test cover in [dBHH$^+$03], test collection in [GJ02], and test set in [MS85]). In [Kog95], the essential test set of a matrix, which is the intersection of all the test sets, has been studied and a relationship between the size of a matrix and the cardinality of the essential test set is derived. In [ACHL13], the notion of watching systems in graphs is introduced, which is a generalization of identifying codes. A watching system in a graph $G = (V, E)$ is a finite set $\mathcal{W} = \{w_1, w_2, \ldots, w_k\}$ where each w_i is a tuple $w_i = (v_i, Z_i)$ where v_i is a vertex in V and the edges are defined by the set $Z_i = N[v_i]$ such that $\{Z_1, Z_2, \ldots, Z_k\}$ is an identifiable system. The authors give some basic properties of watching systems, and also study the cases of

the paths and cycles, and give some complexity results. Similar problems are also sometimes called *shattering* problems, see [NAH02]. The problem of IDENTIFYING CODE is extensively studied (see [Lob] for an online bibliography with almost 500 references as of January 2022).

Theorem 2.1: [Fou12]

Let $(\mathcal{I}, \mathcal{A})$ be an identifiable set system. Then there is a test cover of $(\mathcal{I}, \mathcal{A})$ of at most $|\mathcal{I}| - 1$ elements of $\mathcal{A}$.

The analogy between test covers and identifying codes is however limited to some extent: the test cover problem does not ask for each individual to actually belong to an attribute of the test cover (i.e. there is no domination condition). However we have the following notion, which slightly differs from the one of a test cover. Given an $\mathcal{I}$-identifiable set system $(\mathcal{I}, \mathcal{A})$, a subset $\mathcal{C}$ of $\mathcal{A}$ is a discriminating code of $(\mathcal{I}, \mathcal{A})$ if it is a test cover of $(\mathcal{I}, \mathcal{A})$ and each element of $\mathcal{I}$ belongs to some set of $\mathcal{C}$. The notion of a discriminating code is defined below.

Discriminating code [CCCH06]

Definition 2.5. Let $G = (I \cup A, E)$ be a bipartite undirected graph. A subset $\mathcal{C}$ of A is said to be a discriminating code of G if:

- $\forall i \neq j : \mathcal{C} \cap N[i] \neq \mathcal{C} \cap N[j]$, and
- $\forall i \in I : \mathcal{C} \cap N[i] \neq \phi$.

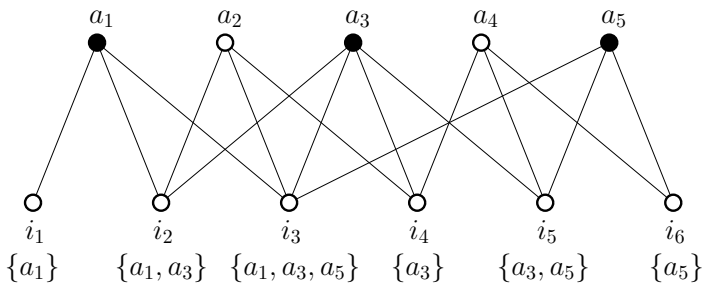


Figure 2.3: A graph with the members of the discriminating code is highlighted in black vertices.

A discriminating code has been represented in Figure 2.3 where the code of each

vertex is also given. For the simplification of understanding, I can be viewed as a set of individuals and A as a set of attributes, with an edge between $i \in I$ and $a \in A$ if attribute a belongs to individual i; a discriminating code is then a set of attributes sufficient to distinguish all the individuals. Discriminating codes are closely related to locating-dominating codes [CSS87].

Thus, two individuals are called twins if their neighborhoods are equal (i.e. two individuals have the same set of attributes). The graph $G = (I \cup A, E)$ is said to be twin-free if no elements of I are twins.

Observation 2.2

A bipartite graph has a discriminating code if and only if it is twin-free.

In [CCHL08], it is shown that the decision version of the discriminating code problem for bipartite graphs i.e. does there exist a discriminating code of size k, is NP-complete. In [CCHL08], a polynomial time algorithm is given for computing discriminating code when the graph G is a tree. In [CCHL07], it is shown that the discriminating codes and identifying codes defined for general graphs are equivalent in the case of the Hamming space[1]. Some properties of discriminating codes are studied in [CCC+08]. In particular, they give bounds on the minimum size of these codes, investigate graphs where minimal discriminating codes have size close to the upper bound, or give the exact minimum size in particular graphs. More generally, for an integer $r \geq 1$, they define $B_r(v)$, which can be seen as a ball of radius r centred at v, as the set of vertices within distance r from v, where the distance $d(x, y)$ between two vertices x and y is the smallest possible number of edges in any path between them. They also give an NP-completeness result for the existence of an r-discriminating code[2] $\mathcal{C} \subseteq A$ of size at most k where $r \geq 1$ is odd. In [MRT14], open-out separating codes[3] are handled which are

[1] A Hamming space can be defined over an alphabet set Q as the set of different words of a fixed length N with elements from Q.

[2] A code $C \subseteq A$ for a bipartite graph $G = (I \cup A, E)$ is called r-discriminating if for each $v \in I$ all the sets $C \cap B_r(v)$ are nonempty and distinct.

[3] Given a digraph D and $C \subseteq V(D)$ we say that C is an open-out-separating code if for distinct $u, v \in V(D)$ it holds $N^+(u) \cap C = N^+(v) \cap C$ where $N^+(v)$ denotes the open-out neighbor of a vertex $v \in V$ excluding v itself.

similar to discriminating codes. [LHC20] presents a survey of the major results on identification and on locating-domination[4].

More references on several coding mechanisms on graphs based on different applications, namely locating-dominating sets, open locating dominating sets, metric dimension, etc, and their computational hardness results are available in [Fou12, FMN+17].

2.2 Red-Blue Separation

The Red-Blue-Separation problem is well studied in Computational Geometry. Here two sets of points R and B are given in $\mathbb{R}^d$ where the points in R are colored "red" and the points in B are colored "blue". The objective is to use a set $\mathcal{A}$ of minimum number of hyperplanes in $\mathbb{R}^d$ such that each cell in the arrangement of the hyperplanes in $\mathcal{A}$ is either empty or contains point(s) of same color. In $\mathbb{R}^2$, if the question is *does there exists a single line that separates the points in R from the points in B*, then it can be solved in $O(n^2 \log n)$ time as follows: generate a set S of $O(|R| \times |B|)$ line segments by joining each point in R with every point in B, and execute the $O(n \log n)$ time algorithm of [EMP+82] for computing a transversal of that set of line segments S, if it exists.

For example say we are given a set R of red points and a set B of blue points in the plane, the Red-Blue Separation problem asks if there are at most k axis parallel lines that separate R from B, or in other words, each cell induced by the lines of the solution is either empty or monochromatic (containing point(s) of only one color, see Figure 2.4). A more natural variation of this problem targets separating the red and blue points by a set of k lines of arbitrary orientation which has been shown to be $W[1]$ Hard in [BGL19]. Red-Blue Separation has been studied in the geometric setting of red and blue points in the Euclidean plane [BGL19, CDKW05, MMS20]. In this problem, one wishes to select a small

[4] A set C of vertices in a graph $G = (V, E)$ is a locating-dominating code if it is dominating and any two vertices of $V \setminus C$ are dominated by distinct set of codes.

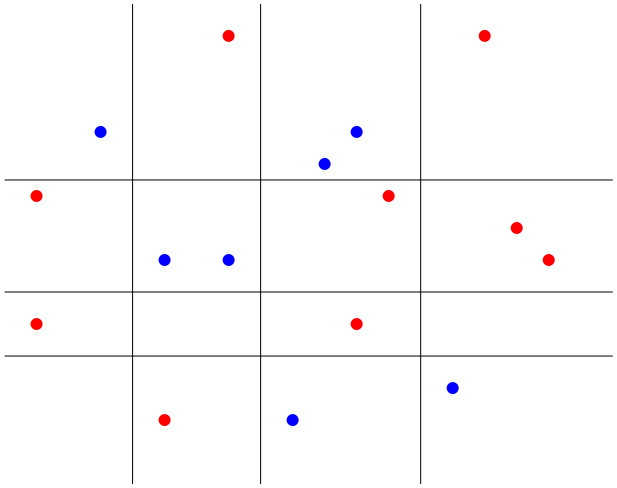


Figure 2.4: An illustration for the geometric red-blue separation.

set of (axis-parallel) lines such that any pair of red and blue points lie on the two sides of one of the solution lines.

In [CH98], it is shown that the problem of computing a RED-BLUE-SEPARATOR[5] of minimum cardinality in $\mathbb{R}^2$ is NP-hard. Gaur et al. proposed a 2-factor approximation algorithm for stabbing a set of axis-parallel rectangles using minimum number of axis-parallel lines [GIK02]. As this problem is same as the problem of hitting a given set of segments of arbitrary orientation using minimum number of axis-parallel line, we can use it to produces a 2-factor approximation solution for the RED-BLUE-SEPARATION problem using axis-parallel lines. It is also shown that the problem is fixed parameter tractable when parameterized by the size of the smallest color class and the solution size [KMM+21]. The RED-BLUE-SEPARATION problem using arbitrary lines in $\mathbb{R}^2$ can easily be formulated as a set cover problem, and hence a $O(\log n)$ factor approximation is obvious. Consider each line segment joining a pair of red and blue points; consider its dual, which is an wedge. Thus, we have the set $\mathcal{W}$ of wedges corresponding to the duals of these line segments. Consider the arrangement $\mathcal{A}_{\mathcal{W}}$ of the wedges in $\mathcal{W}$. Each cell of $\mathcal{A}_{\mathcal{W}}$ is the intersection of a subset $\chi \in \mathcal{W}$ of wedges. Thus choosing a point in that cell implies the corresponding line in the primal plane separates each pair of

[5] A set of lines that separates each point in R from every point in B

points corresponding to the members in χ. Thus, we have $O(|R| \times |B|)$ red-blue pairs, and each cell in $\mathcal{A}_W$ defines a subset of red-blue pairs. We need to choose minimum number of subsets to cover all these red-blue pairs.

Motivated by machine learning applications, the following RED-BLUE SEPARATION problem has been studied in the literature, where the points are given in the plane and the sets of $\mathcal{S}$ are defined by half-planes [CH98]. Geometric RED-BLUE SEPARATION problem can be considered in higher dimension also. The motivation of RED-BLUE SEPARATION problem stems from the DISCRETIZATION problem for two classes of points in machine learning, where each point has multiple features represented by its coordinates. Here the color of a point represents a data class. The problem is useful in a preprocessing step to transform the continuous features into discrete ones, with the aim of classifying the data points [CH98, KMM+21, KE07]. In 2D, this problem was shown to be NP-hard [CH98] but 2-approximable [CDKW05] and fixed-parameter tractable when parameterized by the size of a smallest color class [BGL19] and by the solution size [KMM+21]. A polynomial-time algorithm for a special case, where the points are on a circle, was recently given in [MMS20].

Another similar problem is the RED-BLUE-SET-COVER problem, where given a red point set, a blue point set, and a set of objects, the objective is to choose a subset of objects to cover all the blue points, while minimizing the number of red points covered. In [CH15], it has been proved that the problem is NP-hard even when the objects are unit squares in 2D, and the first polynomial-time approximation scheme (PTAS) for this case has been given. In [MNP21] variations of the geometric RED-BLUE-SET-COVER problem in the plane using various geometric objects has been studied.

The SEPARATION problem for set systems (also known as TEST COVER) was introduced in the 1960s [Ré61] and widely studied from a combinatorial point of view [BS07] as well as from the algorithmic perspective for the settings of classical, approximation and parameterized algorithms [CGJ+16, MS85]. Geometric versions of SEPARATION have been studied as well [DFNS20, GP19, HPJ20]. The

SEPARATION problem is also closely related to the VC DIMENSION problem [VC15] which is very important in the context of machine learning. In VC DIMENSION, for a given set system $(V, \mathcal{S})$, one is looking for a (large) set X of vertices that is *shattered*, that is, for every possible subset of X, there is a set of $\mathcal{S}$ whose trace on X is the subset. This can be seen as "perfectly separating" a subset of $\mathcal{S}$ using X; see [BLL+15] for more details on this connection. It thus seems natural to study the graph version of RED-BLUE SEPARATION.

We study the RED-BLUE SEPARATION problem for graphs. The problem is motivated from its geometric counterpart, which is the red blue separation of points in the Euclidean plane. The RED-BLUE SEPARATION problem which we study here is a red-blue colored version of SEPARATION defined in the previous section as separating code, where instead of all pairs we only need to separate red vertices from blue vertices in the graph. The formal definition of the problem is given below.

RED-BLUE SEPARATION

Definition 2.6. Given a graph $G = (V, E)$ where each vertex is colored either blue or red, the objective is to choose a subset $\mathcal{C} \subseteq V$ of minimum cardinality such that the code of every pair of red and blue vertices is different. The code assigned to each vertex $v \in V$ is the combination of the vertices in $\mathcal{C}$ which are in the closed neighborhood of v.

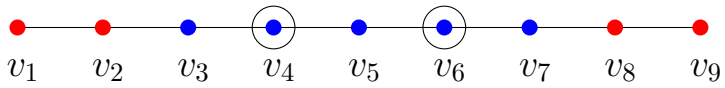


Figure 2.5: A path graph whose highlighted vertices give a red-blue separation.

In Figure 2.5 the RED-BLUE SEPARATION problem is demonstrated using a path graph. Here, if the vertices v_4 and v_6 are chosen in the red-blue separating set then all the red vertices are separated from the blue vertices as the red vertices have empty code.

2.3 Consistent Subset Problem

In the classification problems, there are two extremes of knowledge: either complete statistical knowledge of the underlying distribution of the observation o and the true category c, or no knowledge of the underlying distribution except that which can be inferred from samples [CH67].

Nearest Neighbor Rule [CH67]

Definition 2.7. A decision to classify an observation o into the category c is allowed to depend only on a collection of n correctly classified samples $o_1, \ldots, o_n$ into their corresponding categories $c_1, \ldots, c_n$, and the simplest decision procedure is the nearest neighbor (NN) rule, which classifies an observation o in the category of its nearest neighbor. See Figure 2.6.

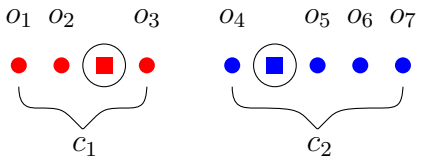


Figure 2.6: Classification by nearest neighbor rule: the representatives of different classes are shown using square points and the sample elements are shown using circular points.

The NN rule is such that it assigns an unclassified sample to the same class as the nearest of n stored, correctly classified samples. In other words, given a collection of n reference points, where each of them is classified by some external source, a new point is assigned to the same class as its nearest neighbor among those reference points. From a practical point of view the NN rule is not a good choice for many applications because of the storage requirements it imposes. In [Har68], the following concept of Condensed Nearest Neighbor (CNN) rule is introduced which retains the basic approach of the NN rule without imposing such stringent storage requirements as each point classified in the NN rule by some external source, a new point is assigned to the same class as its nearest neighbor.

The CNN rule defines the notion of a consistent subset of a sample set which

can correctly classify all of the remaining objects in the sample set. A minimal consistent subset is a consistent subset with a minimum number of elements.

Minimum Consistent Subset [Har68]

Definition 2.8. Given a set P of n points in the plane that is partitioned into $P_1, ..., P_k$, with $k \geq 2$, and the goal is to find an smallest set $S \subseteq P$ such that for every $i \in \{1, ..., k\}$ it holds that if $p \in P_i$ then the nearest neighbor of p in S belongs to P_i. It is implied by the definition that S should contain at least one point from every P_i.

Figure 2.7: Consistent subset of a point set: different classes are shown using different colors.

Note the difference between Figure 2.6 and 2.7. In the former figure two new points apart from P were introduced as representatives to the two sets of points whereas in the later the representatives are chosen from the point set P itself. A further modification to the nearest neighbor decision rule by [Har68] is the reduced nearest neighbor rule which is introduced in [Gat72]. It is an extension of the CNN rule. Since every set is trivially a consistent subset of itself, every set has a consistent subset.

Since the inception of this problem in 1968, from the algorithmic point of view a significant progress has not been observed. There had been several attempts of developing algorithms, but those were either not optimal [Wil91] or had an exponential running time [RWLI75]. In [RWLI75], a procedure is introduced to approximate nearest neighbor decision boundaries. The algorithm in the paper produces a selective subset of the original data so that (1) the subset is consistent, (2) the distance between any sample and its nearest neighbor in that subset is less than the distance from the sample to any sample of the other members of the subset, and (3) the subset is the smallest possible. A two-stage iterative algorithm for selecting a subset of a training set of samples to be used in a condensed nearest neighbor (CNN) decision rule was introduced in [GK79]. The k-center problem [MIH81] is to place k objects on the plane so that the distance from a client (a

point) to its closest object is not greater than a given number r. Apart from the algorithmic study, the theory of fuzzy sets is introduced into the k-nearest neighbor technique to develop a fuzzy version of the algorithm in [KGG85].

In [Wil91] it has been proved that the consistent subset problem is NP-complete if the input points are colored by at least three colors; the proof is based on the NP completeness of the disc cover problem [MIH81]. In the same paper, a technically-involved $O(n^2)$-time algorithm is presented for a special case of two-colored input points where one point is red and all other points are blue. [Wil91] showed that this problem, even with two colors, is also NP-complete. In [Cla94] output-sensitive geometric algorithms are proposed for the nearest-neighbor classification problem.

Traditionally, the nearest-neighbor (NN) search has been based on two basic indexing approaches: object-based indexing and solution-based indexing. The former's construct is based on the locations of data objects. The latter is built on a precomputed solution space. Thus, NN queries can be reduced to and processed as simple point queries in the solution space. Both approaches exhibit some disadvantages, especially when employed for wireless data broadcast in mobile computing environments. In [ZXLL04], a new index method is introduced, called the grid-partition index. The grid-partition index is constructed using the Voronoi diagram. This method has two distinctive characteristics. First, the solution space is divided into grid cells such that a query point can be efficiently mapped into a grid cell around which the nearest object is located which significantly reduces the search space. Second, the grid-partition index stores the objects that are potential NNs of any query falling within the cell. The storage of objects, instead of the Voronoi cells, makes the grid-partition index a hybrid of the solution-based and object-based approaches. In [Tou02], a variety of open problems are mentioned, which are basically of a computational geometric nature that arise in instance based learning. In [Tou05], geometric proximity graphs such as Voronoi diagrams and their many relatives are discussed which provide elegant solutions to data mining problems such as outlier detection.

In [Ang05, Ang07] a novel algorithm is proposed, called the fast condensed nearest

neighbor (FCNN) rule, for computing a training-set-consistent subset for the nearest neighbor decision rule. The objective is to show that condensation algorithms for the nearest neighbor rule can be applied to huge collections of data. [HLT22] has worked on the nearest neighbor representation of Boolean functions.

In [AF07], the parallel fast condensed nearest neighbor (PFCNN) rule is presented which is a distributed method for computing a consistent subset of a very large data set for the nearest neighbor classification rule. Different variants of the basic PFCNN method are introduced in [AF07] in order to cope with the communication overhead typical of distributed environments and to reduce memory requirements. An analysis of spatial cost, CPU cost, and communication overhead is accomplished for all the algorithms. They indeed scale up well and are efficient in memory consumption, confirming the theoretical analysis, and achieve noticeable data reduction and good classification accuracy.

The general definition of the MCS problem was given in [GEC+07] as follows. A ground set X and a constraint t is given. The objective is to compute a subset $X' \subseteq X$ that satisfy the constraint t. They proposed the following application for reducing the data communication: *if a constraint t and the consistent subset X' with respect to the constraint t are communicated to a user, then the user can classify each element of the ground set X using the subset X' and the constraint t.* In its geometric variation in real domain some *distance* measure serves as the constraint assuming that the distances between each pair of objects is distinct.

In [AB08], the SAT–CNN algorithm is introduced, which is a method for computing a minimum size consistent subset using the Nearest Neighbor rule via SAT encodings. It exploits a suitable encoding of the CNN problem in a sequence of SAT problems in order to exactly solve it, provided that enough computational resources are available. Comparison of SAT–CNN with well-known greedy methods have shown that SAT–CNN is able to return a better solution.

Recently it has been proved that the consistent subset problem with points having two colors is also NP-complete using a reduction from the planar rectilinear mono-

tone 3-SAT [KKR18]. It must be noted that the one color version of the problem is trivial as every single point in the set P is a consistent subset. [ADBH+15] has studied a class of geometric optimization problems closely related to the 2-center problem: Given a set S of n pairs of points in the plane. For every pair, one has to assign red color to a point of the pair and blue color to the other point in order to optimize the radii of the minimum enclosing ball of the red points and the minimum enclosing ball of the blue points. More recently, [BBC18] showed that the consistent subset problem on collinear points, i.e., points that lie on a straight line, can be solved optimally in $O(n^2)$ time. A sub-exponential time algorithm for the consistent subset problem in $\mathbb{R}^2$ is also available in [BCC+19]. It is also shown that in $O(n \log n)$ time one can test whether the size of the MCS of a bi-colored point set in $\mathbb{R}^2$ is 2 or not. In the same paper, an $O(n)$ time algorithm is presented for the collinear points, thereby improving the previous running time by a factor of $\Theta(n)$. They also propose an $O(n^6)$ time dynamic programming algorithm for points arranged on two parallel lines.

We will study the graph-theoretic version of the consistent subset problem with nearest neighbor (with respect to the hop-distance) as the constraint.

Minimum Consistent Subset in Graphs

Definition 2.9. For a set of colored vertices V in the graph $G = (V, E)$, a set $\mathcal{C} \subseteq V$ is a consistent subset if for each vertex $v \in V \setminus \mathcal{C}$, the closest vertex of $v \in \mathcal{C}$ has the same color as that of v. The consistent subset problem requires a consistent subset with minimum cardinality.

A related problem is recently studied in [BCMPR20, BCMPR21], where the *inverse Voronoi diagram* (IVD) in graphs is defined as follows. A graph $G = (V, E)$ with positive edge weights, and a sequence of subsets $\{V_1, V_2, \ldots, V_k\}$, $V_i \subseteq V$ for $i = 1, 2, \ldots, k$, are given where each subset V_i is connected in G, and $\cup_{i=1}^{k} V_i = V$. The objective is to identify the existence of a subset $X = \{x_1, x_2, \ldots, x_k\}$, where each $x_i \in V_i$ is such that for every element $v \in V_i$ its nearest neighbor in X is x_i. Here, by distance of a pair of vertices $u, v \in V$, we mean the shortest path distance with respect to the edge weights in G. In [BCMPR20, BCMPR21], it

is shown that the IVD problem for planar graphs is NP-complete. For trees, the IVD problem can be solved in $O(N + n\log^2 n)$ time [BCMPR21], where n is the number of vertices in the tree and $N = n + \sum_{i=1}^{k} |V_i|$.

2.4 Applications and Motivations

The three types of problems that has been studied in this thesis are motivated from a myriad of applications. The applications have been divided into three parts specific to the problems.

Discrimination and Identification: Identification problems in general have a wide range of applications in situations which involve different variants of testing. For example, test covers can be used for biological identification of individuals according to their attributes, the diagnosis of faults or diseases, or pattern recognition [dBHH+03, MS85]. Identifying codes have also been applied to scenarios where one wishes to detect failures in a computer network. Some variants, such as $(1, \leq \ell)$-identifying codes allow one to handle the case of several simultaneous failures. This type of situation arises when the network is a complex of rooms and corridors, and detectors are e.g. fire alarms or motion sensors. Detectors placed as an identifying code, allow one to detect and locate a fire in the building or an intruder. This idea has been explained in e.g. [RUP+03, SS10] and a real experimental motion sensor system based on identifying codes has been implemented and discussed in [UTS04]. Identifying codes have been known to be used in routing problems in networks. Given two computers which form a part of a network, the objective is to send a message from one to the other under certain constraints depending on the network. The message usually transits through specific computers of the network called routers. In some applications, these routers form a dominating set. In [LTCS09] the problem has been solved using identifying codes and domination-based routing schemes, by using the fact that identifying codes are dominating sets and that they induce unique identifiers

to each network node. Another application of identifying codes have been used in comparing secondary RNA structures (viewing these molecules as graphs) [HKSZ06]. Experiments have indeed shown that the values of domination parameters for RNA molecules help to give a good description of the molecular properties of these structures. Identifying code also has several applications in the field of localization and contamination detection [RSTU04].

Discrimination has also been related to many real life problems. Using wireless sensor networks, the Structural Health Monitoring (SHM) problem for critical infrastructures, such as bridges, buildings, electric power equipments, has received considerable attention in the research community in recent years [NAE+17]. Sensors placed in the deployment area have two functions: (i) to sense a target function such as temperature, pressure, vibration, etc., and (ii) to transmit the sensed data either directly or through multiple other sensor nodes (which serves as relays) to the control station, for the analysis of the sensed data. While the first function relates to coverage of the sensing region, the second function relates to the connectivity aspects of the network formed by the sensors. In [BZSG19], a novel terror network monitoring approach is proposed that claims to significantly reduce the resource requirement of law enforcement authorities, but still provide the capability of uniquely identifying a suspect in case the suspect becomes active in planning a terrorist attack. The approach relies on the assumption that, when an individual becomes active in planning a terrorist attack, his/her friends/associates will have some hints of the individual's plan. Accordingly, even if the individual is not under active surveillance by the authorities, but the individual's friends/associates are, then the individual planning the attack, can be uniquely identified. Very recently, in [BS21] two types of networks related to drug trafficking organizations and terrorist organizations are considered, and they have presented a methodology for the surveillance of individuals associated with these networks based on the notion of discriminating codes. In [Bel18], the problem of traffic monitoring is studied which consists in differentiating a set of walks on a directed graphs by placing sensors on as few arcs as possible. Traffic monitoring presents new challenges such as taking

into account the multiplicity and order of the arcs in a walk. A new and stronger model of separation based on languages that generalizes the traffic monitoring problem is introduced in their work.

Red-Blue Separation: Separation problems are closely related to identification and discrimination problems. The separation problem has numerous applications in areas such as monitoring and fault-detection in networks [UTS04], biological testing [MS85], and machine learning [KE07]. It has applications to fault-tolerant multi-modal sensor fusion in the context of embedded sensor networks [KSPSV02]. When using linear arrangements for separation, there are applications in the domain of manufacturing, constructive solid geometry and statistical classification [Fre00].

The notion of separation or separating codes has many applications in a wide range of domains. In each case a diagnosis has to be delivered with limited or expensive access to information. Notable examples in such scenarios include visualization and pattern detection [dBHH+03, NF77], routing [LTCS09] or fault detection [DDSM76] in telecommunication networks. It also has applications in many areas of bio-informatics, such as analysis of molecular structures [HKSZ06] or in medical diagnosis, where test covers are the core of diagnostic tables (see [WL72]) and are therefore important for blood sampling or bacterial identification (see [WLH80] for a survey on different methods). Separating codes have also been studied under the name of *sieves* in the context of logic characterizations of graphs; the size of a minimal separating code determines the complexity of the first-order logic formula required to describe a graph [KPSV05].

Consistency: A new and highly parallel method of exploiting continuity has been introduced in [Ull74] for the problem of character recognition in a class of hand-printed characters using the concept of nearest neighbor. In [Pat71], an approach to interactive pattern recognition is dealt with using a-priori problem knowledge. The a-priori knowledge is either in the form of uncertain correlation information among features or new features which are nonlinear functions of original features. In [ZXLL04], the grid-partition index is introduced to support NN search in both on-demand access and periodic

broadcast modes of mobile computing. In [Das17] the concept is applied on datasets related to image classification of hand digits and face recognition dataset.

In addition to being a natural variant of the fundamental 2-center problem from facility location, our problem is motivated by a problem in "chromatic clustering", called the Chromatic Cone Clustering problem. It arises in certain applications in biology, as studied by Ding and Xu [DX11]. Another view of motivation of our problem comes from a transportation problem, in which case there are origin/destination pairs of points between which the traffic flows. There is the option of establishing a special high-priority traffic corridor, which can be modeled as a straight segment, where the knowledge about the traffic flow is required for going between pairs of points. The goal is to locate the corridor in a way such that the off-corridor travel is minimized when traffic between origin/destination pairs utilizes the corridor. Models dealing with alternative transportation systems have been suggested in location theory [BP88], and simplified mathematical models have been widely studied in order to investigate basic geometric properties of urban transportation systems [AHS+03]. Recently, there has been an interest in facility location problems derived from urban modeling. In many of the cases one might be interested in locating a highway that optimizes some given function that depends on the distance between elements of a given point set [AAA+07, CCH+08, DBKPLV13, KT08]. In [ADBH+15] an application in air traffic management has been studied specifically with the use of "flow corridors" (or "tubes").

2.5 Contributions

The contributions of the thesis is categorized as follows:

Discrimination and Identification: We study geometric variations of the discriminating code problem. In the *discrete version* of the problem, a finite

set of points P and a finite set of objects S are given in $\mathbb{R}^d$. The objective is to choose a subset $S^* \subseteq S$ of minimum cardinality such that for each point $p_i \in P$ the subset $S_i^* \subseteq S^*$ covering p_i, satisfy $S_i^* \neq \emptyset$, and each pair $p_i, p_j \in P$, $i \neq j$, satisfies $S_i^* \neq S_j^*$. In the *continuous version* of the problem, the solution set S^* can be chosen freely among a (potentially infinite) class of allowed geometric objects.

We first study the 1-dimensional case ($d = 1$), the points in P are placed on a line L, and the objects in S are finite-length line segments aligned with L (called intervals). We show that the discrete version of this problem is NP-complete. This is somewhat surprising as the continuous version is known to be polynomial-time solvable. This is also in contrast with most geometric covering problems, which are usually polynomial-time solvable in one dimension. However, we have proposed a polynomial-time 2-approximation algorithm for this discrete version of the problem. We also design a PTAS for both discrete and continuous versions in one dimension, for the restriction where the intervals are all required to have the same length. We then study the 2-dimensional case ($d = 2$) for axis-parallel unit square objects. We show that both continuous and discrete versions are NP-complete, and design polynomial-time approximation algorithms that produce $(16 + \epsilon)$ and $(128 + \epsilon)$ approximate solutions respectively, using rounding of suitably defined integer linear programming problems.

Finally, we apply our techniques to a related variant of the discrete problem, where instead of points and geometric objects we just have a set S of objects which are axis-parallel unit squares. The goal is to select a small subset S^* of objects so that all objects of S are discriminated by their intersection with the objects of S^*. This problem can be viewed as a graph problem by stating it in terms of the vertices of the geometric intersection graph of S; under this graph-theoretical form, it is known as the *identifying code problem*. Our previously mentioned reduction for $d = 1$ can be adapted for the identifying code problem on interval graphs. We show that the identifying code problem for unit square intersection graphs ($d = 2$) can also be solved in the same manner as for the discrete version of the discriminating code problem for

unit square objects described above, and all our approximation results still hold in this setting.

Red-Blue Separation: Here, we introduce and study the RED-BLUE SEPARATION problem on graphs, where we a graph is given whose vertices are colored either red or blue, and we want to select a (small) subset of vertices, called *red-blue separating set*, such that for every red-blue pair of vertices, there is a vertex from the separating set whose closed neighborhood contains exactly one of the two vertices of the pair. This problem was previously studied in a geometric setting (where red and blue points in the plane are to be separated by lines), motivated by applications in machine learning.

We study the computational complexity of RED-BLUE SEPARATION on graphs, in which one asks whether a given red-blue vertex colored graph has a red-blue separating set of size at most a given integer. The decision problem is NP-complete even for restricted graph classes such as planar bipartite sub-cubic graphs, in the setting where the two color classes have equal size. We also show that the optimization problem is NP-hard to approximate within a factor of $(1-\epsilon)\ln n$ for every $\epsilon > 0$, even for split graphs of order n, and when one color class has size 1. On the other hand, it is always approximable in polynomial-time within a factor of $2\ln n$. In contrast, for triangle-free graphs and for graphs of bounded maximum degree, RED-BLUE SEPARATION is solvable in polynomial-time when the size of the smaller color class is bounded by a constant (using algorithms that are in the parameterized class XP, with the size of the smaller color class as parameter). However, on general graphs, the problem is $W[2]$-hard even when parameterized by the solution size plus the size of the smaller color class.

We also consider the problem MAX RED-BLUE SEPARATION where the coloring is not part of the input. In this problem, given an input graph G, we want to determine the smaller integer k such that, *for every possible red-blue-coloring* of G, there is a red-blue separating set of size at most k. We study tight bounds on the cardinality of an optimal solution of MAX RED-BLUE SEPARATION on graphs, showing that it can range from logarithmic in the number of vertices, up to the number of vertices minus one. We also give

bounds with respect to related parameters. For trees however we prove an upper bound which is two-thirds the number of vertices in the graph. We then show that MAX RED-BLUE SEPARATION is NP-hard, even for graphs of bounded maximum degree, but can be approximated in polynomial time within a factor of $O(\ln^2 n)$.

Consistency: We study the minimum consistent subset (MCS) problem on graphs. We are given a connected simple undirected graph $G = (V, E)$ whose each vertex is colored by one of the colors $\{c_1, c_2, \ldots, c_k\}$. The objective is to compute a subset $\mathcal{C} \subseteq V$ such that for each vertex $v \in V$, its set of nearest neighbors in $\mathcal{C}$ (with respect to the hop-distance) contains at least one vertex of the same color as v. The decision version of the MCS problem is NP-complete for general graphs. Even for planar graphs, the decision version of the MCS problem is NP-complete. We will fisrt consider some simple graph classes like k-chromatic path, k-chromatic spider, bi-chromatic caterpillar, bi-chromatic comb, and propose polynomial-time algorithms for solving the problem on those graphs. We then discuss the minimum consistent problem on trees. Even on trees, the problem is non-trivial. However, we propose an involved polynomial-time algorithm for computing a minimum consistent subset of bi-chromatic trees. We also give a simple approximation algorithm by extending the idea from Steiner trees.

CHAPTER 3

Discrimination and Identification

Contents

3.1 Organization

In this chapter we study the geometric version of the MIN-DISC-CODE problem introduced in [BDNS19], which will be referred to as G-MIN-DISC-CODE. Here we have two sets of nodes in the bipartite graph $G = (U \cup V, E)$ where $U = S$, a set of geometric objects, and $V = P$, a set of points in $\mathbb{R}^d$; a vertex $u \in U$ has edges to a subset $V_u \subseteq V$ in E if the object S_u corresponding to u contain the points in P corresponding to the vertices in V_u. Each object in S has its label. We assign an identification(id) of a point which is the label of all the objects containing that point. Given an instance (P, S), two points $p_i, p_j \in P$ are called *twins* if each member in S that contains p_i also contains p_j, and vice-versa. An instance (P, S) of G-MIN-DISC-CODE is *twin-free* if no two points in P are twins. Geometrically, if we consider the arrangement $\mathcal{A}$ of the geometric objects S, then the instance (P, S) is twin-free if each cell of $\mathcal{A}$ contains at most one point of P. As mentioned earlier, for a twin-free instance, a subset of S that can uniquely assign id's to all the points in P is said to *discriminate* the points of P and is called a *discriminating code* or *disc-code* in short. In the discrete version of the problem the set S of objects is given along with the set of points P as the input; the objective is to find a subset $S^* \subseteq S$ of minimum cardinality that is a disc-code for the points in P. In the continuous version, we can freely choose the position of objects S^* in $\mathbb{R}^d$ such that each point gets a unique id, and the size of the set S^* is minimum. The two problems are formally stated as follows.

Problem: DISCRETE-G-MIN-DISC-CODE
Input: A point set P to be discriminated, and a set of objects S to be used for the discrimination.
Output: A minimum-size subset $S^* \subseteq S$ that discriminates all points in P.

A related problem, namely Minimum Identifying Code (MIN-ID-CODE) is defined

in the literature. Here a set S of objects are given.

Problem: Continuous-G-Min-Disc-Code
Input: A point set P to be discriminated.
Output: A minimum-size set S^* of objects, placed *anywhere* in the region under consideration, that discriminates the points in P.

Problem: Min-ID-Code
Input: A set of objects S to be identified.
Output: A minimum-size subset $S^* \subseteq S$ that identifies all objects in S.

The following proposition from [HPV98] (which uses the technique of *partition refinement* for computing twins) will be useful.

Proposition 3.1: [HPV98]

Checking whether a given instance (P, S) of Discrete-G-Min-Disc-Code with $|P| = n$ and $|S| = m$ is twin-free can be done in time $O(m + n)$.

We will study the computation hardness issues of different variations of Min-Disc-Code problems and propose approximation algorithms. The chapter is organized as follows. Section 3.2 deals with the G-Min-Disc-Code in 1D where the objects are intervals. By using a polynomial time reduction from a restricted version of the 3-SAT problem, we show that Discrete-G-Min-Disc-Code problem for discriminating points on a real line by interval objects of arbitrary length, is NP-complete. Here, the challenge is to overcome the linear nature of the problem and to transmit the information across the entire construction without affecting intermediate regions as an interval can stretch over many other smaller intervals as well as points and complicate the construction of an instance. This result is in contrast with Continuous-G-Min-Disc-Code problem in 1D, which is polynomial-time solvable [GP19]. This is also in contrast with most geometric covering problems, which are often polynomial-time solvable in 1D [DBRDG17]. We also propose a simple polynomial-time 2-factor approximation algorithm for

Object Type	Continuous-G-Min-Disc-Code		Discrete-G-Min-Disc-Code	
	Hardness	Algorithm	Hardness	Algorithm
1D intervals unbounded	-	Polynomial-time solvable ([GP19])	NP-hard (Thm. 3.1)	2-approximable (Thm. 3.2)
1D intervals bounded	-	Polynomial-time solvable ([GP19])	Open	2-approximable (Thm. 3.2)
1D unit intervals	Open	PTAS (Cor. 3.1)	Open	PTAS (Thm. 3.3)
2D axis-parallel unit squares	NP-hard (Thm. 3.4)	$(16OPT+1)$-approximable (Thm. 3.5)	NP-hard (Thm. 3.4)	$(128OPT+1)$-approximable (Thm. 3.6)

Table 3.1: Summary of our results on G-Min-Disc-Code problems.

Discrete-G-Min-Disc-Code in 1D.

We also design a PTAS for both Discrete-G-Min-Disc-Code and Continuous-G-Min-Disc-Code problems in 1D, when all the objects are required to have the same length. In this context, it needs to be mentioned once again that the Continuous-G-Min-Disc-Code problem for arbitrary intervals is polynomially solvable [GP19].

In Section 3.3 we study both problems in 2D for axis-parallel unit square objects, which is a natural extension of 1D intervals to the 2D setting. The continuous version is known to be NP-complete for unit disks [GP19], and we show that the reduction can be adapted to our setting, for both the continuous and discrete cases.

We then design polynomial-time constant-factor approximation algorithms for both problems in that setting. The approximation factors are 16 and 128 for the continuous and discrete problem respectively.[1] To obtain these algorithms, we re-formulate our problems into a problem of *stabbing* line segments in 2D, which can be reduced to a geometric Hitting Set problem.

Our results on Discriminating Code problems are summarized in Table 3.1.

Finally, in Section 3.4, we consider the related Min-Id-Code problem restricted to unit square graphs (geometric intersection graphs for 2D axis-parallel unit

[1]Note that, the $(4+\epsilon)$ factor approximation algorithms presented in the conference version of this paper [DFNS20] were wrong and the algorithms have been corrected here.

squares). We show that MIN-ID-CODE for unit square graphs can be solved in the same manner as the DISCRETE-G-MIN-DISC-CODE problem for axis-parallel unit square objects, and our approximation results for DISCRETE-G-MIN-DISC-CODE still hold for MIN-ID-CODE on this class of graphs.

3.2 The G-MIN-DISC-CODE problem in 1D

It has been shown that CONTINUOUS-G-MIN-DISC-CODE is polynomial-time solvable in 1D [GP19]. Thus, in this section we focus on DISCRETE-G-MIN-DISC-CODE.

An instance (P, S) of the DISCRETE-G-MIN-DISC-CODE problem is a set $P = \{p_1, \ldots, p_n\}$ of points and a set S of m intervals of arbitrary lengths placed on a real line $\mathbb{R}$. Assuming that the points are sorted with respect to their x-coordinate values, we define $n+1$ *gaps* $\mathcal{G} = \{g_1, \ldots, g_{n+1}\}$, where $g_1 = (-\infty, p_1)$, $g_i = (p_{i-1}, p_i)$ for $2 \leq i \leq n$, and $g_{n+1} = (p_n, \infty)$.

Observe that (i) if both endpoints of an interval $s \in S$ lie in the same gap of $\mathcal{G}$, then it can not discriminate any pair of points; thus s is *useless*, and (ii) if more than one interval in S have both their endpoints in the same two gaps, say $g_a = (p_a, p_{a+1}), g_b = (p_b, p_{b+1}) \in \mathcal{G}$, then both of them discriminate exactly the same point-pairs. Thus, they are *redundant* and we need to keep only one interval among them. In a linear scan, we can first eliminate the useless and redundant intervals. From now onwards, m will denote the number of intervals, none of which are useless or redundant. Hence, m may be $O(n^2)$ in the worst case.

3.2.1 NP-completeness

The DISCRETE-G-MIN-DISC-CODE problem is in NP, since given a subset $S' \subseteq S$, in linear time one can test whether the problem instance (P, S') is twin-free (i.e.,

whether the id of every point in P induced by S' is unique) by Proposition 3.1.

We prove the NP-hardness of DISCRETE-G-MIN-DISC-CODE using a polynomial-time reduction from the 3-SAT-2L problem (defined below), which is known to be NP-complete [Tov84].

Problem: 3-SAT-2L

Input: A collection of m clauses $C = \{c_1, c_2, \ldots, c_m\}$ where each clause contains at most three literals, over a set of n Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$, and each literal appears at most twice.

Output: A truth assignment of X such that each clause is satisfied (if it exists).

Given an instance (X, C) of 3-SAT-2L, we construct in polynomial time an instance $(P, S) = \Gamma(X, C)$ of the DISCRETE-G-MIN-DISC-CODE problem on the real line $\mathbb{R}$. The main challenge of this reduction is to be able to connect variable and clause gadgets, despite the linear nature of our 1D setting. The basic idea is that we will construct an instance where some specific set of *critical* point-pairs will need to be discriminated (all other pairs being discriminated by some partial solution forced by our gadgets). Let us start by describing our basic gadgets.

Covering Gadget

Definition 3.1. A *covering gadget* Π consists of three intervals I, J, K and four points p_1, p_2, p_3 and p_4 satisfying $p_1 \in I$, $p_2 \in I \cap J$, $p_3 \in I \cap J \cap K$ and $p_4 \in J \cap K$ as in Figure 3.1. Every other interval of the construction will either contain all four points, or none. There may exist a set of points in $K \setminus \{I \cup J\}$, depending on the need of the reduction (see Figure 3.1).

The idea of the covering gadget is to forcefully discriminate the points placed in $K \setminus \{I \cup J\}$ by the other intervals in the construction, so that they are all covered by K, are discriminated from p_1, p_2, p_3, p_4, and also discriminated from all other points not in $I \cup J \cup K$.

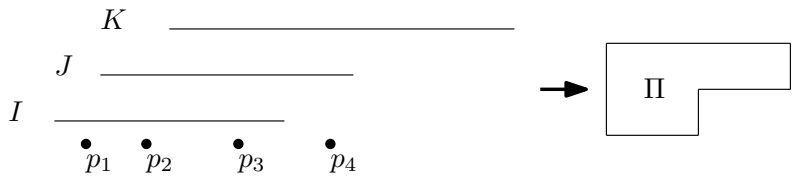


Figure 3.1: A covering gadget Π, and its schematic representation.

Observation 3.1

The points p_1,p_2,p_3,p_4 in a covering gadget can only be discriminated by choosing all three intervals I, J, K in the solution.

Proof. Follows from the fact that none of the intervals in $\Gamma(X, C)$, that is not a member of the covering gadget Π can discriminate the four points in Π. Moreover, if we do not choose I, then p_3, p_4 are not discriminated. If we do not choose J, p_1, p_2 are not discriminated. If we do not choose K, p_2, p_3 are not discriminated (see Figure 3.1). □

Let us now define the gadgets modeling the clauses and variables of the 3-SAT-2L instance.

Clause Gadget

Definition 3.2. Let c_i be a clause of C. The *clause gadget* for c_i, denoted $G_c(c_i)$, is defined by a covering gadget $\Pi(c_i)$ along with two points p_{c_i}, p'_{c_i} placed in $K \setminus \{I \cup J\}$ (see Figure 3.2).

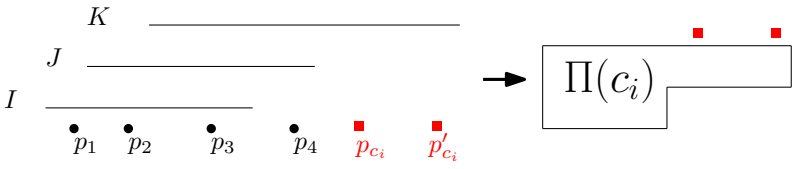


Figure 3.2: A clause gadget $\Pi(c_i)$, and its schematic representation.

The idea behind the clause gadget is that some interval that ends between points p_{c_i}, p'_{c_i} will have to be taken in the solution, so that this pair gets discriminated.

Variable Gadget

Definition 3.3. Let x_j be a variable of X. The *variable gadget* for x_j, denoted $G_v(x_j)$, is defined by a covering gadget $\Pi(x_j)$, and five points $p^1_{x_j}, \ldots, p^5_{x_j}$ placed consecutively in $K \setminus \{I \cup J\}$. We place six intervals $I^0_{x_j}$, $I^1_{x_j}$, $I^2_{x_j}$, $I^0_{\overline{x_j}}$, $I^1_{\overline{x_j}}$, $I^2_{\overline{x_j}}$, as in Figure 3.3.

- Interval $I^0_{x_j}$ starts between $p^1_{x_j}$ and $p^2_{x_j}$, and ends between $p^3_{x_j}$ and $p^4_{x_j}$.
- Interval $I^0_{\overline{x_j}}$ starts between $p^2_{x_j}$ and $p^3_{x_j}$, and ends between $p^4_{x_j}$ and $p^5_{x_j}$.
- Interval $I^1_{x_j}$ starts between $p^2_{x_j}$ and $p^3_{x_j}$, and ends after $p^5_{x_j}$.
- Interval $I^2_{x_j}$ starts between $p^4_{x_j}$ and $p^5_{x_j}$, and ends after $p^5_{x_j}$.
- Interval $I^1_{\overline{x_j}}$ starts between $p^1_{x_j}$ and $p^2_{x_j}$, and ends after $p^5_{x_j}$.
- Interval $I^2_{\overline{x_j}}$ starts between $p^3_{x_j}$ and $p^4_{x_j}$, and ends after $p^5_{x_j}$.

(The end-point of the four intervals, namely $I^1_{x_j}$, $I^1_{\overline{x_j}}$, $I^2_{x_j}$, $I^2_{\overline{x_j}}$, will be determined at the time of construction of the whole instance.)

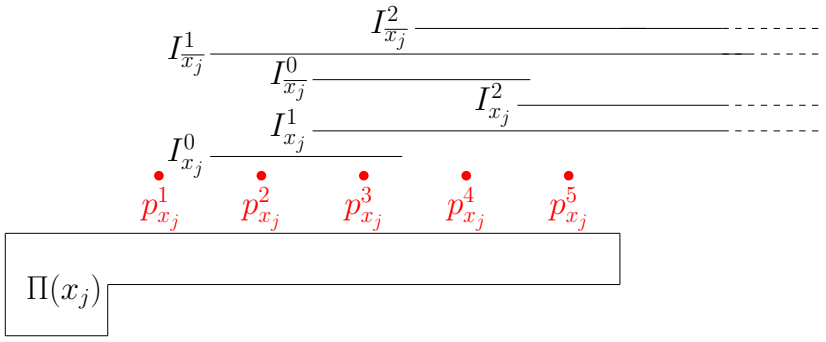


Figure 3.3: Variable gadget for variable x_j.

In a variable gadget $G_v(x_j)$, the intervals $I^1_{x_j}$ and $I^2_{x_j}$ represent the two possible occurrences of literal x_j because a literal can occur at most twice, while $I^1_{\overline{x_j}}$ and $I^2_{\overline{x_j}}$ represent the two possible occurrences of $\overline{x_j}$ in the 3-SAT formula. The right end points of each of these four intervals will be in the clause gadget of the clause where the occurrence of that literal takes place.

An example for the construction of $\Gamma(X, C)$ is shown in Figure 3.4. We assume that every literal appears in at least one clause[2]. Now the construction of the G-MIN-DISC-CODE instance $\Gamma(X, C)$ is as follows:

- For each variable $x_i \in X$, $\Gamma(X, C)$ contains a variable gadget $G_v(x_i)$.
- The gadgets $G_v(x_1), G_v(x_2), \ldots, G_v(x_n)$ are positioned consecutively, in this order, without overlap.
- For each clause $c_j \in C$, $\Gamma(X, C)$ contains a clause gadget $G_c(c_j)$.
- The gadgets $G_c(c_1), G_c(c_2), \ldots, G_c(c_m)$ are positioned consecutively, in this order, to the right of the placement of the variable gadgets, without overlap.
- For every variable x_i, assume x_i appears in clauses c_{i_1} and c_{i_2}, and $\overline{x_i}$ appears in c_{i_3} and c_{i_4} (possibly $i_1 = i_2$ or $i_3 = i_4$). Then, we extend the intervals $I^1_{x_i}$, $I^1_{\overline{x_i}}$, $I^2_{x_i}$, $I^2_{\overline{x_i}}$ so that $I^1_{x_i}$ ends between $p_{c_{i_1}}$ and $p'_{c_{i_1}}$, $I^2_{x_i}$ ends between $p_{c_{i_2}}$ and $p'_{c_{i_2}}$, $I^1_{\overline{x_i}}$ ends between $p_{c_{i_3}}$ and $p'_{c_{i_3}}$, and $I^2_{\overline{x_i}}$ ends between $p_{c_{i_4}}$ and $p'_{c_{i_4}}$.

Thus, our geometric instance $(P, S) = \Gamma(X, C)$ of the DISCRETE-G-MIN-DISC-CODE problem instance has $|P| = 6m + 9n$ points and $|S| = 3m + 9n$ intervals.

Let $\mathcal{S}^\Pi$ be the union of the discriminating codes (i.e., all intervals of type I, J, K, as mentioned in Observation 3.1) of all covering gadgets of type $\Pi(x_i)$ and $\Pi(c_j)$ of $\Gamma(X, C)$.

Consider any covering gadget Π_1. By Observation 3.1, the points p_1, p_2, p_3, p_4 of each covering gadget used in the reduction are discriminated among each other by the set $\mathcal{S}^\Pi$, and they are discriminated from all other points of $\Gamma(X, C)$, since they are the only ones to be covered by one of the intervals I, J from Π_1. Moreover, all the points covered by the interval K from Π_1 are discriminated from all the points not covered by K. Thus, overall, all point-pairs of $\Gamma(X, C)$ are discriminated by $\mathcal{S}^\Pi$, except the following *critical* point-pairs:

[2]If a literal does not appear in any clause, then we can assign a truth value to its variable so that all its occurrences are true, and further ignore this variable.

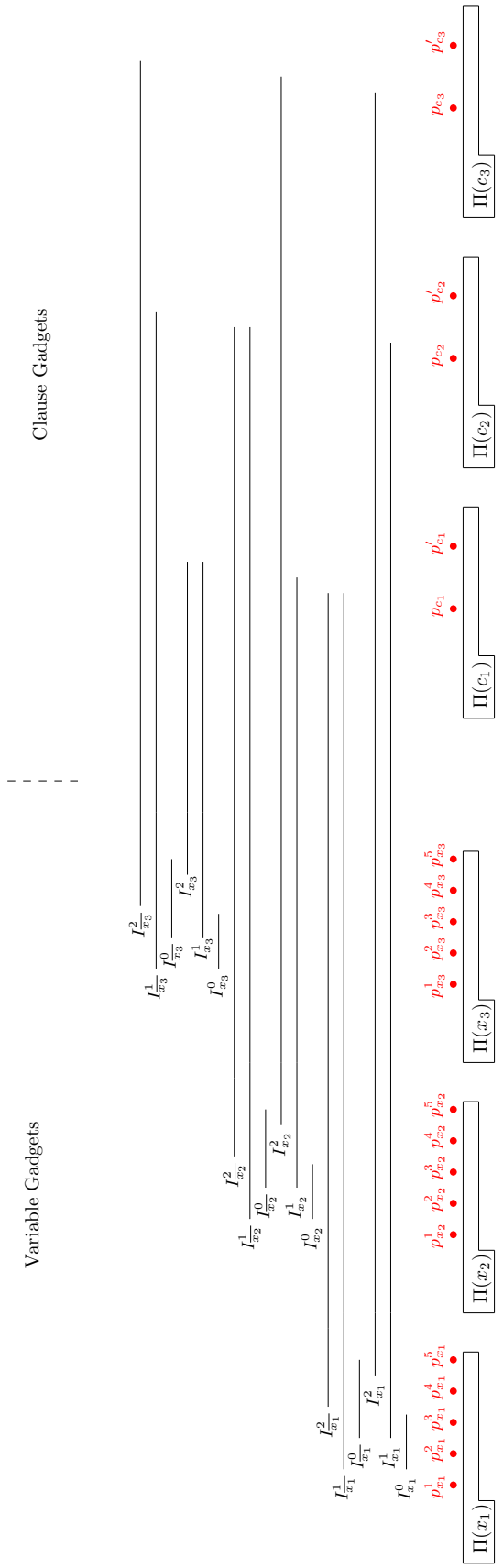


Figure 3.4: The instance $\Gamma(X, C)$ for the formula $(X, C) = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3)$.

- the point-pairs among the five points $p^1_{x_i}, \ldots, p^5_{x_i}$ of each variable gadget $G_v(x_i)$, and
- the point-pair $\{p_{c_j}, p'_{c_j}\}$ of each clause gadget $G_c(c_j)$.

In the proof of the following main result of this section, we will demonstrate, in particular, that if there exists a truth assignment of the variables in X such that all the clauses in C are satisfied, then the critical point-pairs are also discriminated.

Theorem 3.1

DISCRETE-G-MIN-DISC-CODE in 1D is NP-complete.

Proof. We prove that (X, C) is satisfiable if and only if $\Gamma(X, C)$ has a discriminating code of size $6n + 3m$. In both directions of the proof, we will consider the set $\mathcal{S}^\Pi$ defined above. Each variable gadget and clause gadget contains one covering gadget. Thus, $|\mathcal{S}^\Pi| = 3(n + m)$.

($\Longrightarrow$) Consider first some satisfying truth assignment of X. We build a solution set S^* as follows. First, we put all intervals of $\mathcal{S}^\Pi$ in S^*. Then, for each variable x_i, if x_i is true, we add intervals $I^0_{x_i}$, $I^1_{x_i}$ and $I^2_{x_i}$ to S^*. Otherwise, we add intervals $I^0_{\overline{x_i}}$, $I^1_{\overline{x_i}}$ and $I^2_{\overline{x_i}}$ to S^*. Notice that $|S^*| = 6n+3m$. As observed before, it suffices to show that S^* discriminates the point-pair $\{p_{c_j}, p'_{c_j}\}$ of each clause gadget $G_c(c_j)$, and the points $p^1_{x_i}, \ldots, p^5_{x_i}$ of each variable gadget $G_v(x_i)$. (All other pairs are discriminated by $\mathcal{S}^\Pi$.)

Since the assignment is satisfying, each clause c_j contains a true literal $l_i \in \{x_i, \overline{x_i}\}$. Then, one interval of $G_v(x_i)$ is in S^* and discriminates p_{c_j} and p'_{c_j}. Furthermore, consider a variable x_i. Point $p^1_{x_i}$ is discriminated from $p^2_{x_i}, \ldots, p^5_{x_i}$ as it is the only one not covered by any of $I^0_{x_i}$, $I^1_{x_i}$, $I^2_{x_i}$, $I^0_{\overline{x_i}}$, $I^1_{\overline{x_i}}$, and $I^2_{\overline{x_i}}$. If x_i is true, $p^2_{x_i}$ is covered by $I^0_{x_i}$; $p^3_{x_i}$ is covered by $I^0_{x_i}$ and $I^1_{x_i}$; $p^4_{x_i}$ is covered by $I^1_{x_i}$; $p^5_{x_i}$ is covered by $I^1_{x_i}$ and $I^2_{x_i}$. If x_i is false, $p^2_{x_i}$ is covered by $I^1_{\overline{x_i}}$; $p^3_{x_i}$ is covered by $I^0_{\overline{x_i}}$ and $I^1_{\overline{x_i}}$; $p^4_{x_i}$ is covered by $I^0_{\overline{x_i}}$, $I^1_{\overline{x_i}}$ and $I^2_{\overline{x_i}}$; $p^5_{x_i}$ is covered by $I^1_{\overline{x_i}}$ and $I^2_{\overline{x_i}}$. Thus, in both cases, the five points are discriminated, and S^* is discriminating, as claimed.

($\Longleftarrow$) For the converse, assume that S^* is a discriminating code of $\Gamma(X, C)$ of size $6n + 3m$. By Observation 3.1, $\mathcal{S}^{\Pi} \subseteq S^*$. Thus there are $3n$ intervals of S^* that are not in $\mathcal{S}^{\Pi}$.

First, we show that $S^* \setminus \mathcal{S}^{\Pi}$ contains exactly three intervals of each variable gadget $G_v(x_i)$. Indeed, it cannot contain less than three, otherwise we show that the points $p^1_{x_i}, \ldots, p^5_{x_i}$ cannot be discriminated. To see this, note that each consecutive pair $\{p^s_{x_i}, p^{s+1}_{x_i}\}$ $(1 \leq s \leq 4)$ must be discriminated, thus S^* must contain one interval with an endpoint between these two points. There are four such consecutive pairs in $G_v(x_i)$, thus if $S^* \setminus \mathcal{S}^{\Pi}$ contains at most two intervals of $G_v(x_i)$, it must contain $I^0_{x_i}$ and $I^0_{\overline{x_i}}$. But now, the points $p^1_{x_i}$ and $p^5_{x_i}$ are not discriminated, a contradiction.

Let us now show how to construct a truth assignment of (X, C). Notice that at least one of $I^0_{x_i}$ and $I^0_{\overline{x_i}}$ must belong to S^*, otherwise some points of $G_v(x_i)$ cannot be discriminated. If $I^0_{x_i} \in S^*$ but $I^0_{\overline{x_i}} \notin C$, then necessarily $I^1_{x_i} \in S^*$ to discriminate $p^2_{x_i}$ and $p^3_{x_i}$, and $I^2_{x_i} \in S^*$ to discriminate $p^4_{x_i}$ and $p^5_{x_i}$. In this case, we set x_i to true. Similarly, if $I^0_{\overline{x_i}} \in S^*$ but $I^0_{x_i} \notin S^*$, then necessarily $I^1_{\overline{x_i}} \in S^*$ to discriminate $p^1_{x_i}$ and $p^2_{x_i}$, and $I^2_{\overline{x_i}} \in S^*$ to discriminate $p^3_{x_i}$ and $p^4_{x_i}$. In this case, we set x_i to false. Finally, if both $I^0_{x_i}$ and $I^0_{\overline{x_i}}$ belong to S^*, the third interval of $S^* \setminus \mathcal{S}^{\Pi}$ in $G_v(x_i)$ may be any of the four intervals covering $p^5_{x_i}$. If this third interval is $I^1_{x_i}$ or $I^2_{x_i}$, we set x_i to true; otherwise, we set it to false.

Observe that when we set x_i to true, none of $I^1_{\overline{x_i}}$ and $I^2_{\overline{x_i}}$ belongs to S^*; likewise, when we set x_i to false, none of $I^1_{x_i}$ and $I^2_{x_i}$ belongs to S^*. Thus, our truth assignment is coherent. As for every clause c_j, the point-pair $\{p_{c_j}, p'_{c_j}\}$ is discriminated by S^*, one interval corresponding to a true literal discriminates it. The obtained assignment is satisfying, completing the proof. □

3.2.2 A 2-approximation algorithm

We next design a 2-approximation algorithm for Discrete-G-Min-Disc-Code in 1D by carefully choosing at most n intervals to discriminate the n points of P.

First, we will need the following proposition, already observed in [GP19].

Proposition 3.2: [GP19]

Any solution of Discrete-G-Min-Disc-Code and Continuous-G-Min-Disc-Code in 1D for inputs of n points has size at least $\frac{n+1}{2}$.

Proof. In order to discriminate the consecutive points in P, for any feasible solution SOL for P, every gap between two consecutive points will contain an end-point of at least one interval in SOL. There exist $n-1$ gaps for the n points in P. But we must also have intervals covering the first point and the last point, which amounts to $n+1$ positions for the end-points of intervals of SOL. Thus, any solution has size at least $(n+1)/2$. □

We now describe an iterative algorithm for the Discrete G-Min-Disc-Code problem. Let (P, S) be an instance of G-Min-Disc-Code problem where the points in P are sorted in increasing order with respect to their x-coordinate values. For the first step, we let S_1 consist of any interval that contains p_1. At Step i, our partial solution $S_i \subseteq S$ will have the property that it covers and discriminates all the points in $\{p_1, \ldots, p_i\}$ (using at most i intervals). Thus, at step n, S_n is a valid solution for (P, S) of size at most n.

We need to execute n iterations steps for $p_1, \ldots, p_n$. We assume that i iterations have already been executed, and thus we have correctly computed a set S_i that discriminates $P_i = \{p_1, \ldots, p_i\}$. We now consider the set $P_{i+1} = \{p_1, \ldots, p_{i+1}\}$. We distinguish three cases as follows.

Case 1: The id of p_{i+1} using the intervals in S_i is non-null and is different from the id of every point $p \in P_i$. Here, simply let $S_{i+1} = S_i$, that is, S_i is already a feasible solution for P_{i+1}.

Case 2: The id of p_{i+1} using the intervals in S_i is null, that is, no interval of S_i covers p_{i+1}. Here, we choose any arbitrary interval $s \in S$ that covers

p_{i+1}. Thus, we have $S_{i+1} = S_i \cup \{s\}$. As the members in P_i are already discriminated up to i-th iteration, they remain discriminated by S_{i+1} (even if the new interval covers a subset of P_i).

Case 3: The id of p_{i+1} using the intervals in S_i is non-null, but is the same as the id of some $p_j \in P_i$ $(j < i)$. Note that, in such a case the id of p_{i+1} can match with at most one element of P_i as the id's of P_i are all distinct with respect to S_i. Here, we choose an interval s of S that can discriminate p_j and p_{i+1}. Such an interval always exists as we have already checked that (P, S) is twin-free. Thus, $S_{i+1} = S_i \cup \{s\}$ is our valid partial solution.

As we have inserted at most one interval at each iteration, $|S_n| \leq n$. By Proposition 3.2, $|S_n| \leq 2OPT - 1$. Thus, we have a 2-approximation factor.

We now analyze the time and space complexity. We will use two tree data structures for processing the points in P and the intervals in S efficiently. These are a *height-balanced binary tree* $\mathcal{T}_H$, and a *priority search tree* $\mathcal{T}_P$.

$\mathcal{T}_H$: It is a binary tree in which the depth of the two subtrees of every node does not differ by more than 1 [Knu97]. A height-balanced binary tree with n nodes has height $\Theta(\log n)$. Each operation (lookup, insertion or deletion) takes time $\Theta(\log n)$ in the worst case.

$\mathcal{T}_P$: A priority search tree [McC85, dBCvKO08] is a hybrid of a priority queue and a binary search tree. It stores a set of 2-dimensional points (a pair of real numbers) for the efficient answering of 1.5-dimensional queries in a one-side open query box of the form $(-\infty, a] \times [b, c]$. In other words, it can report or count the points whose x-coordinate is smaller than a, and y-coordinate lies in the range $[b, c]$. The preprocessing time and space complexities of this data structure are $O(n \log n)$ and $O(n)$ respectively; the time complexity for reporting and counting a 1.5-dimensional query is $O(s + \log n)$ and $O(\log n)$ respectively, where s is the number of points returned by the search.

Before the start of the algorithm, we compute a $\mathcal{T}_P$ data structure with a set of pairs of reals $(\ell(s), r(s))$ corresponding to the segments in S, where $\ell(s)$ and $r(s)$ denote the coordinates of the left and right end-point of the interval $s \in S$ (on the x-axis). The preprocessing time and space required for $\mathcal{T}_P$ are $O(m \log m)$ and $O(m)$ ($m = |S|$) respectively. Identifying the intervals in S that contains a point $p \in P$ and does not contain a point $q \in P$ is equivalent to a 1.5-dimensional range query with the query box $(-\infty, x(p)] \times (x(p), x(q))$, where $x(p)$ denotes the x-coordinate of the point p.

The height-balanced binary tree $\mathcal{T}_H$ stores the *groups* generated after processing the intervals in S_i, and is updated at the end of each iteration i. A *group* is a maximal set of pairwise intersecting intervals of S_i. These groups are totally ordered in the sense that a pair of consecutive groups share only their common end-point. Each group contains at most one point of P_i as S_i discriminates P_i at the end of the i^{th} iteration. Thus, each group is attached with the corresponding point of P_i, if exists. Since $|S_i| \leq i$, and the number of groups created with $|S_i|$ is $2 \times |S_i| + 1$, the size of the data structure $\mathcal{T}_H$ for storing the groups during the entire execution is $O(n)$. While processing p_{i+1}, this tree structure is used to identify an appropriate interval to cover the point p_{i+1} (depending on Cases 2 and 3 of the algorithm stated below) in $O(\log n)$ time. As a result, we also know which case to follow for discriminating p_{i+1}. The cost of maintenance of $\mathcal{T}_H$ after each iteration is also $O(\log n)$

If Case 2 happens, we need to identify an interval $s \in S$ that covers p_{i+1}, i.e., a segment with $\ell(s) < x(p_{i+1}) < r(s)$, or in other words, a pair of reals $(\ell(s), r(s))$ that lies in the range box $(-\infty, x(p_{i+1})] \times [x(p_{i+1}), \infty)$.

If Case 3 happens, then we need to choose a segment $s \in S$ satisfying either $x(p_j) < \ell(s) < x(p_{i+1}) < r(s)$ i.e., $(\ell(s), r(s)) \in (x(p_j), x(p_{i+1})] \times [x(p_{i+1}), \infty)$ or $\ell(s) < x(p_j) < r(s) < x(p_{i+1})$ i.e., $(\ell(s), r(s)) \in (-\infty, x(p_j)] \times [x(p_j), x(p_{i+1}))$.

As mentioned earlier, in either of the cases such an element can be found in the data structure $\mathcal{T}_P$ in $O(\log m)$ time. Thus, the only task that remains is to modify

the data structure $\mathcal{T}_H$ after inserting $s = [a, b] \in \mathcal{T}_H$. Let a and b lie in the groups $g_\alpha = [\theta_1, \theta_2]$ and $g_\beta = [\psi_1, \psi_2]$, respectively. Now, g_α and g_β is to be deleted from $\mathcal{T}_H$ and four intervals $[\theta_1, a]$, $[a, \theta_2]$, $[\psi_1, b]$ and $[b, \psi_2]$ need to be inserted in $\mathcal{T}_H$. If $= [a, b]$ lies entirely in the same group $[\theta_1, \theta_2]$, then $[\theta_1, \theta_2]$ is split into three groups $[\theta_1, a]$, $[a, b]$ and $[b, \theta_2]$. Surely, we need to attach the points p_j and p_{i+1} to the appropriate interval groups to which they belong. This needs another $O(\log n)$ time. Thus, processing p_{i+1} requires $O(\log m + \log n)$ time in the worst case.

The construction of $\mathcal{T}_P$ needs $O(m \log m)$ time and $O(m)$ space [Lee04, McC85]. Processing n points requires $O(n(\log n + \log m))$ time as earlier. As $n = O(m)$, we thus have the following result:

Theorem 3.2

There exists a 2-factor approximation algorithm solving Discrete-G-Min-Disc-Code in 1D, that runs in $O(m \log m)$ time and $O(m)$ space.

3.2.3 A PTAS for the unit interval case

We now design a PTAS for the 1D case where all intervals in S have the same length. For the sake of simplicity let us assume that the endpoints of the intervals in S are distinct.

The following observation (which was also made in the related setting of identifying codes of unit interval graphs [Fou12, Proposition 5.12]) plays an important role in designing our PTAS.

Observation 3.2: [Fou12]

In an instance (P, S) of Discrete-G-Min-Disc-Code in 1D, if the objects in S are intervals of the same (unit) length, then discriminating all the pairs of *consecutive* points in P is equivalent to discriminating *all* the pairs of points in P.

Proof. Assume that we have a set $S' \subseteq S$ that covers all the points and discriminates all consecutive point-pairs in P, but there exists a pair of non-consecutive points p_i and p_j $(i < j)$ which are not discriminated. Since p_i and p_j are covered by the same set of intervals of S' and the intervals are of unit length, they must be at a distance at most 1 apart. Now, since they are not consecutive, p_{i+1} lies between p_i and p_j. Since S' discriminates p_i and p_{i+1}, there is an interval $I \in S'$ with an endpoint in the gap $g_i = [p_i, p_{i+1}]$. If it is a right endpoint, I covers p_i but not p_j, a contradiction. Thus, it must be the left endpoint. But since the distance between p_i and p_j is at most 1, I contains p_j (but not p_i), again a contradiction. □

For a given $\epsilon > 0$, we choose $\lceil \frac{n\epsilon}{4} \rceil$ points, namely $q_1, q_2, \ldots, q_{\lceil \frac{n\epsilon}{4} \rceil} \in P$, called the *reference points*, as follows: q_1 is the $\lceil \frac{2}{\epsilon} \rceil$-th point of P from the left, and for each $i = 1, 2, \ldots, \lfloor \frac{n\epsilon}{4} \rfloor$, the number of points in P between every consecutive pair (q_i, q_{i+1}) is $\lceil \frac{4}{\epsilon} \rceil$ (both inclusive). The number of points to the right of $q_{\lceil \frac{n\epsilon}{4} \rceil}$ may be less than or equal to $\lceil \frac{2}{\epsilon} \rceil$. For each *reference point* q_i, we choose two intervals $I_i^1, I_i^2 \in S$ such that both I_i^1, I_i^2 contain (span) q_i, and the left (resp. right) endpoint of I_i^1 (resp. I_i^2) have the minimum x-coordinate (resp. maximum x-coordinate) among all intervals in S that span q_i. Observe that all the points in P that lie in the range $G_i = [\ell(I_i^1), r(I_i^2)]$ are *covered*, where $\ell(I_i^1)$ and $r(I_i^2)$ are the x-coordinates of the left endpoint of I_i^1 and the right endpoint of I_i^2, respectively. These ranges will be referred to as *group-ranges*. Since the endpoints of the intervals are distinct, the span of a *group-range* is strictly greater than 1.

We now define a *block* as follows. Observe that the ranges G_i and G_{i+1} may or may not overlap. If several consecutive ranges $G_i, G_{i+1}, \ldots, G_k$ are pairwise overlapping, then the horizontal range $[\ell(I_i^1), r(I_k^2)]$ forms a block. The region between a pair of consecutive blocks will be referred to as a *free region*. We use $B_1, B_2, \ldots, B_l$ to name the blocks in order, and $F_0, F_1, \ldots, F_l$ to name the free regions (from left to right). The points in each block are covered. Here, the remaining tasks are (i) for each block, choose intervals from S such that consecutive pairs of points in that block are discriminated, and (ii) for each free region, choose intervals from S such that all its points are covered, and the pairs of consecutive points are

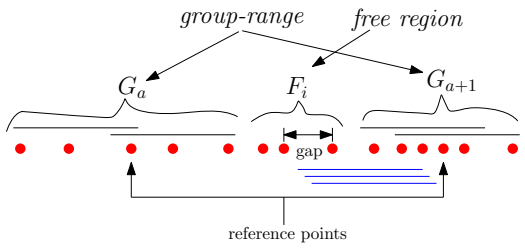


Figure 3.5: Demonstration of redundant edges in a free region which are non-redundant in the problem instance (P, S).

discriminated.

Observation 3.3

There exists no interval $I \in S$ that contains both a point in F_i and a point in F_{i+1}.

Proof. Note that, F_i and F_{i+1} are separated by the block B_{i+1}. If there exists an interval I that contains a point in F_i and a point in F_{i+1}, then I will contain the point $q_j \in B_{i+1}$ just to the right of F_i, which is the reference point of the leftmost group-range G_j of the block B_{i+1}. This contradicts the existence of $I \in S$. Also, the size of I then has to be greater than one, which is impossible. □

Thus, the discriminating code for a free region F_i is disjoint from that of its neighboring free region F_{i+1}. So, we can process the free regions independently.

Processing of a free region: Let the neighboring group-ranges of a free region F_i be G_a and G_{a+1}, respectively. There are at most $\frac{4}{\epsilon}$ points lying between the reference points of G_a and G_{a+1}. Among these, several points of P to the right (resp. left) of the reference point of G_a (resp. G_{a+1}) are inside *block* B_i (resp. B_{i+1}). Thus, there are at most $\frac{4}{\epsilon}$ points in F_i. Let $S_{F_i} \subseteq S$ be a set such that the intervals in S_{F_i} cover at least one point of F_i. Note that, though we have deleted all the redundant intervals of S, there may exist several intervals in S whose one endpoint lies in a gap inside that free region, and their other endpoint lies in

distinct gaps of the neighboring block. In Figure 3.5, there are some blue intervals which are redundant with respect to the points $F_i \cap P$, but are non-redundant with respect to the whole point set P. However, the number of such intervals is at most $\frac{4}{\epsilon}$ due to the definition of (I_i^1, I_i^2) of the right-most group-range of the block B_i and left-most group-range of the block B_{i+1}.

Thus, we have $|S_{F_i}| = O(1/\epsilon^2)$. We consider all possible subsets of intervals of S_{F_i}, and test each of them for being a discriminating code for the points in F_i. Let $\mathcal{D}_i$ be all possible different discriminating codes of the points in F_i, with $|\mathcal{D}_i| = 2^{O(1/\epsilon^2)}$ in the worst case.

Processing of a block: Consider a block B_i; its neighboring free regions are F_i and F_{i+1}. Consider two discriminating codes $d \in \mathcal{D}_i$ and $d' \in \mathcal{D}_{i+1}$. We create a graph $G_i = (V_i, E_i)$ whose nodes V_i correspond to the gaps of B_i which are not discriminated by the intervals used in $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$. Each edge $e \in E_i$ corresponds to an interval in S that discriminates pairs of consecutive points corresponding to two different nodes (gaps) of V_i. Now, we can discriminate each non-discriminated pair of consecutive points in B_i by computing a minimum edge-cover of G_i in $O(|V_i|^2)$ time [MV80]. As mentioned earlier, all the points in B_i are covered. Thus, the discrimination process for the block B_i is over. We will use $\theta(d, d')$ to denote the size of a minimum edge-cover of B_i using $d \in \mathcal{D}_i$ and $d' \in \mathcal{D}_{i+1}$.

Computing a discriminating code for P: We now create a multipartite directed graph $H = (\mathcal{D}, \mathcal{F})$. Its i-th partite set corresponds to the discriminating codes in $\mathcal{D}_i$, and $\mathcal{D} = \cup_{i=0}^{l} \mathcal{D}_i$. Each node $d \in \mathcal{D}$ has its weight equal to the size of the discriminating code d. A directed edge $(d, d') \in \mathcal{F}$ connects two nodes d and d' of two adjacent partite sets, say $d \in \mathcal{D}_i$ and $d' \in \mathcal{D}_{i+1}$, and has its weight equal to $\theta(d, d')$. For every pair of partite sets $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$, we connect every pair of nodes (d, d'), $d \in \mathcal{D}_i$ and $d' \in \mathcal{D}_{i+1}$, where $i = 0, 1, \ldots, l-1$. Every node of $\mathcal{D}_0$ is connected to a node s with weight 0, and every node of $\mathcal{D}_l$ is connected to a node t with weight 0.

Lemma 3.1

The minimum weight s-t path in H is a lower bound on the size of the optimum discriminating code for (P, S), where the weight of a path is equal to the sum of costs of all the vertices and edges on that path.

Proof. Let Π be the minimum weight s-t path in the graph H, which corresponds to a set of intervals $S' \subseteq S$. To show, $|S'| \leq |S_{opt}|$, where $S_{opt} \subseteq S$ corresponds to the minimum size discriminating code. For a contradiction, let $|S'| > |S_{opt}|$. As S_{opt} is a discriminating code, the points of every *free region* F_i are discriminated by a subset, say $\delta_i \in S$. Since, we maintain all the discriminating codes in $\mathcal{D}_i$, surely the subset $\delta_i \in \mathcal{D}_i$. Let $b_i \subset S$ be the set of intervals that span the points of the block B_i. As S_{opt} is a discriminating code, the points in B_i are discriminated by the intervals in $b_i \cup \delta_i \cup \delta_{i+1}$. Thus, the set of intervals $\beta_i = b_i \setminus (\delta_i \cup \delta_{i+1})$ discriminate the pair of points of B_i that are not discriminated by $\delta_i \cup \delta_{i+1}$. Observe that, for every $i = 0, 1, \ldots, l$, we have $\delta_i \in \mathcal{D}_i$. Moreover, there exists a path Π_{opt} that connects $\delta_i, i = 0, 1, \ldots, l$, whose each edge (δ_i, δ_{i+1}) has cost equal to $|\beta_i|$. Thus, we have the contradiction that Π_{opt} is a path in H having cost less than that of Π. □

The set of intervals may not form a discriminating code for P, as the points in a block may not all be covered. However, the additional intervals $\{(I_i^1, I_i^2), i = 1, 2, \ldots, \lceil \frac{n\epsilon}{2} \rceil\}$ ensure that the optimum size of the discriminating code satisfies $|S_{opt}| \geq \lceil \frac{n+1}{2} \rceil$ due to the fact that we have $(n+1)$ gaps, and each interval in S covers exactly 2 gaps. This fact, along with Lemma 3.1 implies:

Lemma 3.2

$|SOL| \leq (1 + \epsilon)|S_{opt}|$.

Proof. By Lemma 3.1, $|S'| \leq |S_{opt}|$. The number of extra intervals to cover the *blocks* is $\frac{n\epsilon}{2}$. Again, $\frac{n}{2} \leq MEC(P) \leq |S_{opt}|$, where $MEC(P)$ is the size of minimum

edge-cover of the graph G created with the points in P and the intervals in S. Thus, $|SOL| = |S'| + \frac{n\epsilon}{2} \leq |S_{opt}| + \epsilon|S_{opt}| \leq (1+\epsilon)|S_{opt}|$. □

We now analyze the time complexity of the algorithm. Note that, the number of possible discriminating codes in a free region is $2^{O(1/\epsilon^2)}$. Thus, in the graph H, the number of edges between a pair of consecutive partite sets $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$ is $|\mathcal{D}_i| \times |\mathcal{D}_{i+1}| = 2^{O(1/\epsilon^2)}$. As the computation of the cost of an edge between the sets $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$ invokes the edge-cover algorithm of an undirected graph, it needs $O(|B_i|^2)$ time [MV80]. Thus, the total running time of the algorithm is $A+B$, where A is the total time of generating the edge costs, and B is the time for computing a shortest path of H. We have $A \leq \sum_{i=1}^{\lceil \frac{n\epsilon}{4} \rceil} 2^{O(1/\epsilon^2)} \times O(|B_i|^2)$. As the B_i's are mutually disjoint, we get $A = O(n^2 \times 2^{O(1/\epsilon^2)})$. Moreover, $B = O(|\mathcal{F}|) = O(\frac{n}{\epsilon} \times 2^{O(1/\epsilon^2)})$ [Tho99].

In order to reduce the space requirement of the algorithm, we generate partite sets of the multipartite graph H one by one, and compute the length of the shortest path from s up to each node of that set. Initially, the length of the path up to a node $d \in \mathcal{D}_0$ is $|d|$. While generating $\mathcal{D}_{i+1}$, the nodes in $\mathcal{D}_i$ are available along with the length of the shortest path $\chi(d)$ up to each node $d \in \mathcal{D}_i$ from s. Now, we execute the following steps:

Step 1: We generate the nodes of $\mathcal{D}_{i+1}$, and initialize their cost $\chi(.)$ with ∞.

Step 2: For each pair of nodes $(d, d'), d \in \mathcal{D}_i, d' \in \mathcal{D}_{i+1}$, do the following:

- Compute the edge cost $\theta(d, d')$, which is the size of the edge-cover of the block B_i using the discriminating codes d of the *free region* F_i and d' of the *free region* F_{i+1}. This needs $O(|B_i|^2)$ time using the matching algorithm of an undirected graph [MV80].
- Compute the length of the shortest path from s to d' using the edge (d, d'), which is $\chi^* = \chi(d) + \theta(d, d') + |d'|$.
- If the computed length is less than the existing value of $\chi(d')$, then update $\chi(d')$ with χ^*.

As the number of discriminating codes in each partite set is $2^{O(1/\epsilon^2)}$ in the worst case which are computed online while considering the $(i+1)$-th partite set, and each discriminating code is of length at most $O(\frac{1}{\epsilon})$, we have the following result.

Theorem 3.3

The Discrete-G-Min-Disc-Code problem in 1D for unit interval objects admits a PTAS: for every $\epsilon > 0$, there is a $(1+\epsilon)$-factor approximation algorithm with time complexity $2^{O(1/\epsilon^2)}n^2$ using $\frac{1}{\epsilon}2^{O(1/\epsilon^2)}n^2$ space.

Moreover, in this unit interval setting, we easily reduce an instance of Continuous-G-Min-Disc-Code problem to an instance of Discrete-G-Min-Disc-Code problem by first computing the $O(n^2)$ possible non-redundant unit intervals among the $n+1$ *gaps*. Thus:

Corollary 3.1

The Continuous-G-Min-Disc-Code problem in 1D for unit interval objects has a PTAS with the same approximation factor, time and space complexity as those for Discrete-G-Min-Disc-Code.

3.3 The G-Min-Disc-Code problem in 2D

Here, the point set $P = \{p_1, p_2, \ldots, p_n\}$ is given in $\mathbb{R}^2$, and the shape of allowed objects used for covering and discriminating the points of P are axis-parallel squares of equal size. We will use the term *unit square* to refer to these objects.

3.3.1 NP-completeness

In [GP19], it has been shown that Continuous-G-Min-Disc-Code for unit disks in 2D is NP-complete. They reduced the P_3-Partition-Grid problem, stated below, to Continuous-G-Min-Disc-Code for unit disks in 2D. We will modify

their reduction and apply it to Continuous-G-Min-Disc-Code for axis-parallel unit squares in 2D.

A *grid graph* is a graph whose vertices are positioned in $\mathbb{Z}^2$, and a pair of vertices are adjacent if they are at Euclidean distance 1 [vBBB+14].

Problem: P_3-Partition-Grid [vBBB+14]
Input: A grid graph G.
Output: A partition of the vertices of G into disjoint P_3-paths, where a P_3-path is a path with three vertices.

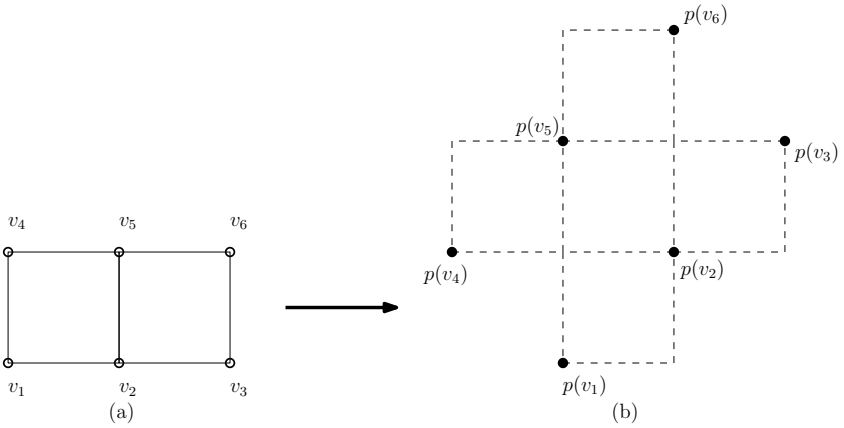


Figure 3.6: (a) A grid graph G. (b) Its corresponding geometric instance P_G, where the dashed axis-parallel unit squares are those covering two points each.

Given an instance G of P_3-Partition-Grid, we construct an instance P_G (a point set) for Continuous-G-Min-Disc-Code as follows. For every vertex v of G with coordinates (x, y), we create a point $p(v)$ with coordinates (x, y) and add it to P_G. The construction from [GP19] stops here, and we will now slightly change it. For each point $p(v)$ with coordinates (x, y), we replace it by a point with coordinates $(y - x, y + x)$, that is, we rotate the whole point set by an angle of $\pi/4$ and stretch it by a factor of $\sqrt{2}$ (See Figure 3.6 for an illustration).

Lemma 3.3

A P_3-partition for $G = (V, E)$ exists if and only if there exists a set of $\frac{2|V|}{3}$ axis-parallel unit squares discriminating the points in P_G.

Proof. ($\Longrightarrow$) The key idea is to notice that any axis-parallel unit square can contain at most two points of P_G, and if it contains two, then it contains two points corresponding to vertices of G joined by an edge (the center of the square is then placed at the middle-most position of the line segment joining the two points). Moreover, any two points corresponding to an edge of G can be covered by some axis-parallel unit square in that way. Three points corresponding to the three vertices of a P_3-path $v_1v_2v_3$ in G can be discriminated using two unit squares s and s', centered at the mid-points of the two segments joining $(p(v_1), p(v_2))$ and $(p(v_2), p(v_3))$, respectively. Now, $p(v_1)$ is covered by s only, $p(v_3)$ by s' only, and $p(v_2)$ by both. Thus, if a P_3-partition of G exists, we have our solution of size $\frac{2|V|}{3}$ to the Continuous-G-Min-Disc-Code problem.

($\Longleftarrow$) Conversely, assume that we have $\frac{2|V|}{3}$ axis-parallel unit squares that discriminate all points of P_G. Recall that every square can cover at most two points. For any square s covering two points $p(v_1)$, $p_(v_2)$, we necessarily have that v_1v_2 is an edge in G. Moreover, one of $p(v_1)$ and $p_(v_2)$ needs to be covered by a second square s' (so that the two points are discriminated). Thus, any solution needs at least $\frac{2|V|}{3}$ squares, and any solution of exactly this size will consist of disjoint sets of three points covered by two squares (one point covered by both squares, and the other two, by one of the squares each). These three points must correspond to three vertices of G forming a P_3. Thus, we obtain our P_3-partition of G, as claimed. □

Lemma 3.3 leads to the following result.

Theorem 3.4

CONTINUOUS-G-MIN-DISC-CODE and DISCRETE-G-MIN-DISC-CODE for axis-parallel unit squares in 2D are NP-complete.

Proof. The statement follows directly from Lemma 3.3 in the case of CONTINUOUS-G-MIN-DISC-CODE. Let S_G contain the set of all axis-parallel unit squares that cover two points of P_G. For DISCRETE-G-MIN-DISC-CODE, we can simply modify the reduction by creating the instance (P_G, S_G) from G. □

3.3.2 Approximation algorithms

We formulate an approximation algorithm by extending the ideas for the 1D case, described in Section 3.2.2. We will use the techniques of rounding some suitable Integer Linear Programmes (ILPs). Here, our goal is to choose a set Q of points in $\mathbb{R}^2$ of minimum cardinality such that (i) every point of P is covered by at least one axis-parallel unit square among those centered at the points in Q (*covering condition*) and (ii) for every pair of points $p_i, p_j \in P$ $(i \neq j)$, there exists at least one square in Q whose boundary intersects the interior of the line segment $[p_i, p_j]$ exactly once (*discrimination condition*).

We transform our CONTINUOUS-G-MIN-DIC-CODE problem into an equivalent problem of segment stabbing (which will be defined below). The segment stabbing problem can also be seen as a hitting set problem of a pair of shapes, called L-shapes. Each of these L-shape hitting set problems is further split into two hitting set problems of unit height rectangles (or unit width rectangles). A schematic representation of this process is shown in Figure 3.7.

We define the set of line segments $L(P) = \{[p_i, p_j] \text{ for all } p_i, p_j \in P, i \neq j\}$. Thus, the discrimination condition leads to the following problem.

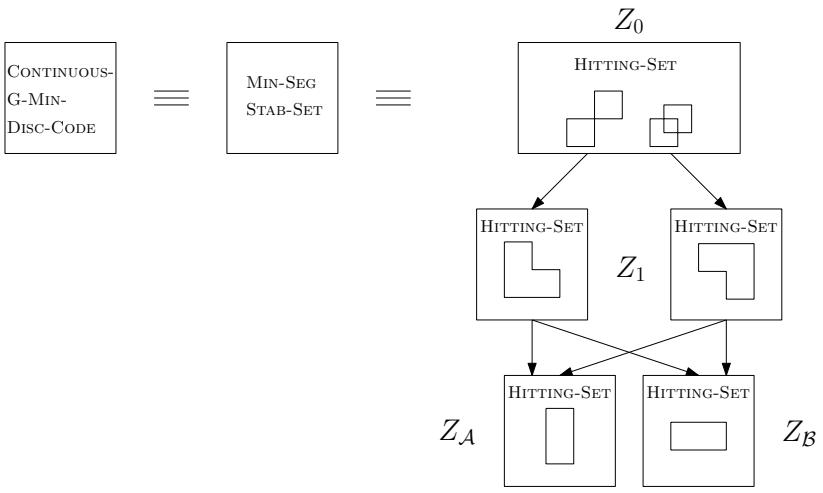


Figure 3.7: Schematic of the problem structure.

Problem: Minimum Segment-Stabbing Set (Min-Seg-Stab-Set)

Input: A set L of segments in 2D.

Output: A minimum-size set S of axis-parallel unit squares in 2D such that each segment is stabbed by some square of S.[a]

[a]**Note:** By *stabbing* a line segment ℓ by a unit square s, we mean that exactly one end-point of ℓ lies inside s.

In fact, Min-Seg-Stab-Set for the input segments $L(P)$ is equivalent to the Test Cover problem for P using axis-parallel unit squares as tests. As in the edge-cover formulation of Discrete-G-Min-Disc-Code problem in 1D (see Section 3.2.2), here also a feasible solution of Min-Seg-Stab-Set ensures the following:

Observation 3.4

Every feasible solution Φ of Min-Seg-Stab-Set (a) discriminates every point-pair in P, and (b) at most one point is not covered by any square in Φ.

In order to discriminate the two endpoints of a member $\ell = [a, b] \in L(P)$, we need to consider the two cases: $length(\ell) \geq 1$ and $length(\ell) < 1$, where $length(\ell)$

denotes the length of ℓ. In the former case, if a center is chosen in any one of the unit squares $D(a)$ and $D(b)$, the segment ℓ is stabbed, where $D(q)$ is the axis parallel unit square centered at a point q. However, more generally in the second case, to stab ℓ, we need to choose a center in the region $(D(a)\setminus D(b))\cup(D(b)\setminus D(a))$. In Figure 3.8 the shaded region is the feasible region for placing the center of the unit squares to stab a line segment in $L(P)$. We define a set of distinct objects $\mathcal{O}$ corresponding to the elements of $L(P)$, where each object corresponds to the feasible region of placing the center of a stabbing square of an element of $L(P)$. Thus, the MIN-SEG-STAB-SET problem reduces to a HITTING-SET problem, where the objective is to choose a minimum number of points in $\mathbb{R}^2$, such that each object in $\mathcal{O}$ contains at least one of those chosen points.

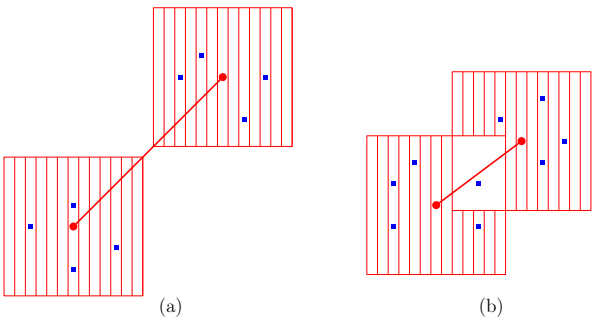


Figure 3.8: Object that needs to be hit corresponding to segment $\ell = [a, b]$, where (a) $length(\ell) \geq 1$ and (b) $length(\ell) < 1$.

The Hitting-Set problem. We use the technique followed in [ANPR19] to solve this problem. Consider the arrangement $\mathcal{A}$ of the objects in $\mathcal{O}$. Create a set Q of points by choosing one point in each cell of $\mathcal{A}$. Thus, the size of the set Q is polynomial in the size of the set P. A square centered at a point q inside a cell $A \in \mathcal{A}$ will stab all the segments whose corresponding objects have common intersection region A. For each point $q_\alpha \in Q$, we use an indicator variable x_α, and can write an integer linear programming (ILP) problem as follows.

$$
\begin{aligned}
Z_0 : \min & \sum_{\alpha=1}^{|Q|} x_\alpha, \\
\text{subject to } & \sigma_1(\ell) + \sigma_2(\ell) \geq 1 \text{ for each segment } \ell = [a,b] \in L(P), \\
\text{where } & \sigma_1(\ell) = \sum_{q_\alpha \in Q \cap (D(a) \setminus D(b))} x_\alpha, \\
& \sigma_2(\ell) = \sum_{q_\alpha \in Q \cap (D(b) \setminus D(a))} x_\alpha, \\
\text{and } & x_\alpha \in \{0,1\} \text{ for all points } q_\alpha \in Q.
\end{aligned}
$$

As the ILP problem is NP-hard [PS82], we relax the integrality condition of the variables x_α for all $q_\alpha \in Q$ from Z_0, and solve the corresponding LP problem in polynomial time.

$$
\begin{aligned}
\overline{Z}_0 : \min & \sum_{\alpha=1}^{|Q|} x_\alpha \\
\text{subject to } & \sigma_1(\ell) + \sigma_2(\ell) \geq 1 \ \forall \ \ell = [a,b] \in L(P), \\
\text{and } & \quad 0 \leq x_\alpha \leq 1 \ \forall \ q_\alpha \in Q.
\end{aligned}
$$

Observe that in the optimum solution $\overline{OPT}_0$ of $\overline{Z}_0$, for each constraint (corresponding to a segment $\ell \in L(P)$), at least one of $\sigma_1(\ell)$ or $\sigma_2(\ell)$ will be greater than or equal to $\frac{1}{2}$. We now define two sets, namely $\mathcal{O}_1$ and $\mathcal{O}_2$. If $\sigma_1(\ell) \geq \frac{1}{2}$ then we put ℓ in the set $\mathcal{O}_1$, and if $\sigma_2(\ell) \geq \frac{1}{2}$ then put ℓ in the set $\mathcal{O}_2$. In other words, we choose to hit the objects $(D(p_i) \setminus D(p_j))$ for all $\ell_{ij} = [p_i, p_j]$, $i < j$, if $\sigma_1(\ell_{ij}) \geq \frac{1}{2}$, and choose to hit the objects $(D(p_j) \setminus D(p_i))$ for all $\ell_{ij} = [p_i, p_j]$, $i < j$, if $\sigma_2(\ell_{ij}) \geq \frac{1}{2}$. It needs to be mentioned that, for a constraint corresponding to a point-pair $\ell = [a,b]$ both $\sigma_1(\ell) \geq \frac{1}{2}$ and $\sigma_2(\ell) \geq \frac{1}{2}$ may happen. In that case ℓ may be considered in any one the sets $\mathcal{O}_1$, $\mathcal{O}_2$ arbitrarily. We form a new ILP as follows:

$$
\begin{aligned}
Z_1 : \min & \sum_{\alpha=1}^{|Q|} x_\alpha \\
\text{subject to } & \sigma_1(\ell) \geq 1 \ \ \forall \ \ \ell \in \mathcal{O}_1, \\
& \sigma_2(\ell) \geq 1 \ \ \forall \ \ \ell \in \mathcal{O}_2, \\
\text{and } & \ \ x_\alpha \in \{0,1\} \ \forall \ q_\alpha \in Q.
\end{aligned}
$$

We use $\overline{Z}_1$ to denote the LP corresponding to the ILP Z_1, OPT_0 and OPT_1, the optimal solutions of Z_0 and Z_1 respectively, and $\overline{OPT}_0$ and $\overline{OPT}_1$ the optimal solutions of $\overline{Z}_0$ and $\overline{Z}_1$ respectively. Observe that $2\overline{OPT}_0$ produces a feasible solution to $\overline{Z}_1$. Thus,

$$\overline{OPT}_1 \ \ \leq 2\overline{OPT}_0 \ \ (\leq 2OPT_0). \tag{3.1}$$

However, as the values of the variables in $\overline{OPT}_1$ are fractional, it is not possible to generate a solution of Z_0 from $\overline{OPT}_1$. Observe the objects $\mathcal{O}_1 \cup \mathcal{O}_2$ considered in Z_1 are either a unit square, or a L-shaped object for which one of its length and its width is 1. Thus, our objective is to solve the L-HIT problem, stated below.

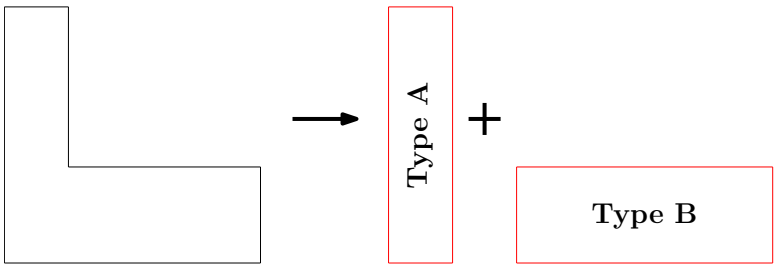


Figure 3.9: An L-shaped object, which is the union of a type A and a type B object.

The L-HIT problem. Here, given a set of L-shaped objects as defined above, and a set of points Q, we wish to choose a minimum-size set of points in Q to hit

all the L-shaped objects in $\mathcal{O}_1 \cup \mathcal{O}_2$.

We can view an L-shaped object as the union of two rectangles of type A and type B, where each type A rectangle has height 1 and width less than or *equal* to 1 and each type B rectangle has width 1 and height less than 1 (see Figure 3.9). (Unit squares are considered to be type A rectangles.)

While solving $\overline{Z}_1$, for each constraint (with respect to $\mathcal{O}_1$ and $\mathcal{O}_2$) any one or both of the following cases may happen: (a) the sum of variables whose corresponding points lie in a type A rectangle is $\geq \frac{1}{2}$, and (b) the sum of variables whose corresponding points lie in a type B rectangle is $\geq \frac{1}{2}$. We accumulate all the rectangles in the set $\mathcal{A}$ (resp. $\mathcal{B}$) where condition (a) (resp. condition (b)) is satisfied. The objective is to choose a minimum number of points in Q to hit all the rectangles in $\mathcal{A}$ and $\mathcal{B}$. We formulate two ILPs' $Z_{\mathcal{A}}$ and $Z_{\mathcal{B}}$ corresponding to two MIN-UHR-HIT-SET problems with the set of rectangles $\mathcal{A}$ and $\mathcal{B}$ respectively, as stated below.

Problem: MINIMUM UNIT HEIGHT RECTANGLE HITTING SET (MIN-UHR-HIT-SET)
Input: A set $\mathcal{R}$ of unit height rectangles in $\mathbb{R}^2$.
Output: A set of points that hits all the members of $\mathcal{R}$.

A PTAS for the MIN-UHR-HIT-SET problem is known [MR10]; however it cannot be used in Equation 6.2 since that does not guarantee any approximation factor for the optimum solution of the corresponding LP problem. However, the MIN-UHR-HIT-SET problem for a set of rectangles $\mathcal{R} = \mathcal{A}$ (resp. $\mathcal{B}$) can be formulated as an ILP $Z_{\mathcal{A}}$ (resp. $Z_{\mathcal{B}}$) as follows:

$$Z_{\mathcal{A}} : \min \sum_{\alpha=1}^{|Q|} x_\alpha, \quad \text{s. t.} \sum_{q_\alpha \in A_i \cap Q} x_\alpha \geq 1, \forall \text{ rectangle } A_i \in \mathcal{A}, \quad \text{and} \quad x_\alpha \in \{0,1\}, \forall\, \alpha \in Q.$$

$$Z_{\mathcal{B}} : \min \sum_{\alpha=1}^{|Q|} x_\alpha, \quad \text{s. t.} \sum_{q_\alpha \in B_i \cap Q} x_\alpha \geq 1, \forall \text{ rectangle } B_i \in \mathcal{B}, \quad \text{and} \quad x_\alpha \in \{0,1\}, \forall\, \alpha \in Q.$$

Denoting by $\overline{Z}_{\mathcal{A}}$ and $\overline{Z}_{\mathcal{B}}$ the LP version of $Z_{\mathcal{A}}$ and $Z_{\mathcal{B}}$, and $\overline{OPT}_{\mathcal{A}}$ and $\overline{OPT}_{\mathcal{B}}$ the optimum solutions for $\overline{Z}_{\mathcal{A}}$ and $\overline{Z}_{\mathcal{B}}$ respectively, we observe that $2\overline{OPT}_1$ gives a feasible solution to both $\overline{Z}_{\mathcal{A}}$ and $\overline{Z}_{\mathcal{B}}$ simultaneously. Note that, as the variables participating in $\overline{OPT}_{\mathcal{A}}$ and $\overline{OPT}_{\mathcal{B}}$ may not be disjoint, it is not possible to write $\overline{OPT}_{\mathcal{A}} + \overline{OPT}_{\mathcal{B}} = 2\overline{OPT}_1$. However, $\overline{Z}_{\mathcal{A}} \leq 2\overline{OPT}_1$ and $\overline{Z}_{\mathcal{B}} \leq 2\overline{OPT}_1$. Thus by Equation 6.2, we have

$$\overline{OPT}_{\mathcal{A}} + \overline{OPT}_{\mathcal{B}} \leq 4\overline{OPT}_1 \leq 8\overline{OPT}_0 \tag{3.2}$$

We apply the shifting strategy (see [HM85]) for solving the MIN-UHR-HIT-SET problem. Here $\mathbb{R}^2$ is split using x-axis parallel lines from a set $\mathcal{L} = \{\lambda_1, \lambda_2, \ldots\}$ such that λ_i and λ_{i+1} are at distance 1 of each other, and such that each rectangle is hit by one of the lines of $\mathcal{L}$. Thus, each rectangle in $\mathcal{A}$ is intersected by *exactly* one line of $\mathcal{L}$ (assuming that no rectangle in $\mathcal{A}$ is aligned with a line in $\mathcal{L}$). See Figure 3.10 for a visual representation.

Let $\mathcal{A}_{even}$ (resp. $\mathcal{A}_{odd}$) denote the set of rectangles in $\mathcal{A}$ that are intersected by even (resp. odd) numbered lines of $\mathcal{L}$. Now, denoting by $Z_{\mathcal{A}}$, $Z(\mathcal{A}_{even})$ and $Z(\mathcal{A}_{odd})$ the ILP of the hitting set problems corresponding to the set of rectangles $\mathcal{A}$, $\mathcal{A}_{even}$ and $\mathcal{A}_{odd}$ respectively, $OPT_{\mathcal{A}}$, $OPT(\mathcal{A}_{even})$ and $OPT(\mathcal{A}_{odd})$ as the optimum solutions of these problems, and $\overline{OPT}_{\mathcal{A}}$, $\overline{OPT}(\mathcal{A}_{even})$ and $\overline{OPT}(\mathcal{A}_{odd})$ as the optimum solutions of the corresponding LP problems, we have

$$\overline{OPT}(\mathcal{A}_{even}) \leq \overline{OPT}_{\mathcal{A}} \text{ and } \overline{OPT}(\mathcal{A}_{odd}) \leq \overline{OPT}_{\mathcal{A}}.$$

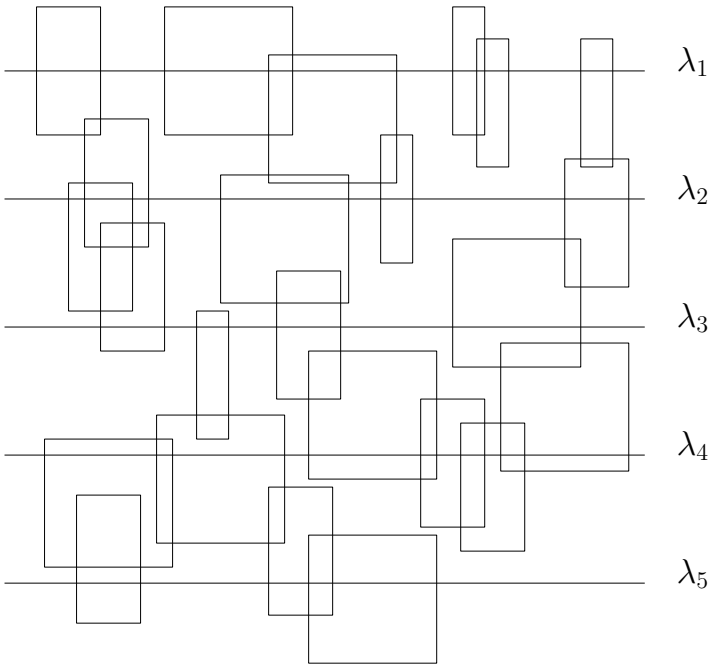


Figure 3.10: An instance of shifting strategy where λ_i's indicate the horizontal lines.

Now, combining these two inequalities, we have

$$\overline{OPT}(\mathcal{A}_{even}) + \overline{OPT}(\mathcal{A}_{odd}) \leq 2\overline{OPT}_{\mathcal{A}}. \tag{3.3}$$

Again, if $Z(\mathcal{A}_i)$ is the hitting set problem with the set of rectangles intersected by $\lambda_i \in \mathcal{L}$, $OPT(\mathcal{A}_i)$ and $\overline{OPT}(\mathcal{A}_i)$ are the optimum solutions for its ILP and LP versions, then we can show that $OPT(\mathcal{A}_i) = \overline{OPT}(\mathcal{A}_i)$, as follows.

In the arrangement of $\mathcal{A}_i$, each cell is adjacent to the horizontal line λ_i. Thus, the representative hitting point in $Q_{\mathcal{A}_i}$ of each cell in the ILP formulation $Z(\mathcal{A}_i)$ can be chosen on λ_i. Thus, the constraint matrix in the formulation of $Z(\mathcal{A}_i)$ will satisfy the *consecutive ones property*. It is a classic theorem that the matrix is totally uni-modular and $Z(\mathcal{A}_i)$ can be solved optimally by solving its corresponding LP $\overline{Z}(\mathcal{A}_i)$ [Sch03]. Thus, in the optimum solution of $\overline{Z}(\mathcal{A}_i)$ the variables will have value 0 or 1, and the corresponding points

can be used as the solution of the hitting set problem.

Thus, we have $\overline{OPT}(\mathcal{A}_{odd}) = OPT(\mathcal{A}_{odd}) = OPT(\mathcal{A}_1) + OPT(\mathcal{A}_3) + \ldots$. The reason is that for the problem $Z(\mathcal{A}_{odd})$ none of the rectangles in $\mathcal{A}_i$ overlap with any rectangle in $\mathcal{A}_j$, for all i, j odd and $i \neq j$. Thus, the hitting set problem for those instances can be solved independently. Similarly, we have $\overline{OPT}(\mathcal{A}_{even}) = OPT(\mathcal{A}_{even}) = OPT(\mathcal{A}_2) + OPT(\mathcal{A}_4) + \ldots$.

Equations 6.3, 6.4 and the subsequent discussions lead to the following. Considering the previously computed optimal solutions $SOL(Z_{\mathcal{A}})$ and $SOL(Z_{\mathcal{B}})$ for $Z_{\mathcal{A}}$ and $Z_{\mathcal{B}}$, which, by the previous discussions, together form a solution for Z_0, we obtain the following chain of inequalities.

$$\begin{aligned}
|SOL(Z_{\mathcal{A}})| + |SOL(Z_{\mathcal{B}})| &= |SOL(\mathcal{A}_{even})| + |SOL(\mathcal{A}_{odd})| + |SOL(\mathcal{B}_{even})| + |SOL(\mathcal{B}_{odd})| \\
&= \overline{OPT(\mathcal{A}_{even})} + \overline{OPT(\mathcal{A}_{odd})} + \overline{OPT(\mathcal{B}_{even})} + \overline{OPT(\mathcal{B}_{odd})} \\
&\leq 2 \times (\overline{OPT(\mathcal{A})} + \overline{OPT(\mathcal{B})}) \text{ by Equation 6.4} \\
&\leq 16 \times \overline{OPT_0} \text{ by Equation 6.3} \\
&\leq 16 \times OPT_0
\end{aligned}$$

Lemma 3.4

The aforesaid algorithm computes a 16-factor approximate solution for MIN-SEG-STAB-SET.

We accumulate the hitting set for type A rectangles corresponding to each horizontal line and the hitting set for type B rectangles corresponding to each vertical line in a set Q^*. By Observation 3.4, at most one point in P may not be covered by the squares centered at the points of Q^*. Thus, we may require at most one extra square to cover that uncovered point. Thus, we have the following.

Theorem 3.5

There exists a polynomial-time approximation algorithm for the CONTINUOUS-G-MIN-DISC-CODE problem for axis-parallel unit squares which can produce a solution of size at most $16OPT + 1$, where OPT is the size of an optimal solution.

3.3.3 Approximation algorithm for DISCRETE-G-MIN-DISC-CODE

In this section, we modify the algorithm of Section 3.3.2 for CONTINUOUS-G-MIN-DISC-CODE to solve DISCRETE-G-MIN-DISC-CODE. Recall that here, in addition to the set of points P (in $\mathbb{R}^2$), the set S of axis-parallel unit squares is also given in the input. As in Section 3.3.2, DISCRETE-G-MIN-DISC-CODE also reduces to the discrete version of the MIN-UHR-HIT-SET problem, whose objective is to hit a set $\mathcal{O}$ of unit/width height rectangles by choosing a minimum cardinality subset of a given set of points Q, where Q is the set of centers of the given set of squares S. Unlike the continuous version, the discrete version of the hitting set problem for a set of unit height rectangles intersected by a horizontal line cannot be solved in polynomial time, since the points to be used for hitting the rectangles are already specified. However, if we can design an α-factor approximation algorithm for the discrete version of the hitting set problem for a set of unit height rectangles intersected by a horizontal line, we can use that to get a 16α-factor approximation algorithm for DISCRETE-G-MIN-DISC-CODE.

Discrete hitting of rectangles stabbed by a horizontal line

Let us first solve a restricted version of the discrete MIN-UHR-HIT-SET problem, where the input is a set of axis-parallel unit-height rectangles R intersected by a horizontal line λ and a set of points Q (see Figure 3.11). The objective is to choose a minimum number of points from Q to hit all the rectangles in R. This problem

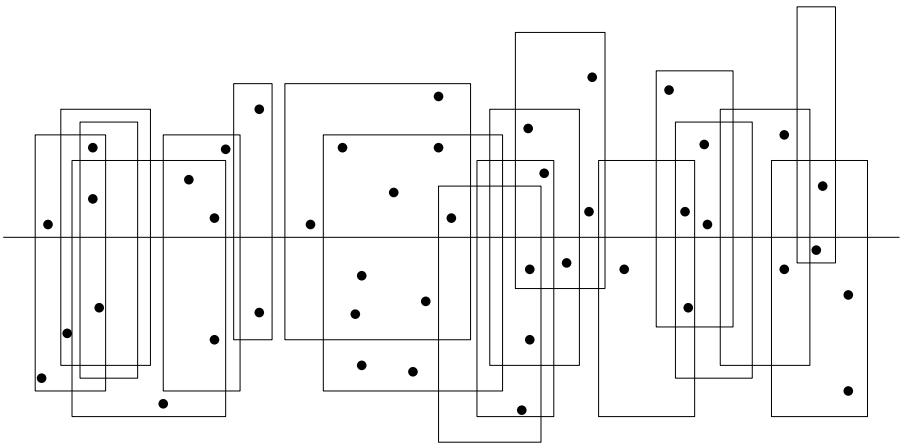


Figure 3.11: An instance of discrete hitting set of unit height rectangles stabbed by a horizontal line.

can be formulated as the following ILP.

$$\mathcal{U}_\lambda : \min \sum_{q_\alpha \in Q} x_\alpha$$
$$\text{Subject to } \sigma_1(r) + \sigma_2(r) \geq 1, \text{for all } r \in R,$$

where $\sigma_1(r)$ (resp. $\sigma_2(r)$) is the sum of the variables corresponding to the points above (resp. below) the line λ that lie inside the rectangle r. We will use OPT_λ to denote the optimum solution of this ILP.

On the basis of the LP relaxation of this ILP, we can partition the rectangles into two groups: R^a and R^b, such that R^a (resp. R^b) contains the rectangles whose solution in the LP relaxation satisfies $\sigma_1(r) \geq \sigma_2(r)$, and thus $\sigma_1(r) \geq \frac{1}{2}$ (resp. $\sigma_1(r) < \sigma_2(r)$ and thus $\sigma_2(r) > \frac{1}{2}$). The rectangles in R^a (resp. R^b) will thus be assumed to be hit by the points in Q^a (resp. Q^b) that lie above (resp. below) the line λ; $Q^a \cup Q^b = Q$, $Q^a \cap Q^b = \emptyset$.

Let $\mathcal{U}^a_\lambda$ and $\mathcal{U}^b_\lambda$ be the ILPs for the minimum hitting set problems for the rectangles in R^a and points in Q^a (R^b and Q^b, respectively). As opposed to the relation of $\overline{OPT}_\mathcal{A}$, $\overline{OPT}_\mathcal{B}$ and $\overline{OPT}_1$ in Equation 6.3, here we can say that if $\overline{OPT}^a_\lambda$ and

$\overline{OPT}^b_\lambda$ are optimum solutions of the LP relaxation of $\mathcal{U}^a_\lambda$ and $\mathcal{U}^b_\lambda$, respectively, then

$$\overline{OPT}^a_\lambda + \overline{OPT}^b_\lambda \leq 2\overline{OPT}_\lambda \tag{3.4}$$

due to the fact that Q^a and Q^b are disjoint point sets and $2\overline{OPT}_\lambda$ is a solution to both relaxations $\overline{\mathcal{U}}^a_\lambda$ and $\overline{\mathcal{U}}^b_\lambda$. Now, we concentrate on solving the ILP $\mathcal{U}^a_\lambda$. $\mathcal{U}^b_\lambda$ can be solved in a similar manner.

Approximation algorithm for solving $\mathcal{U}^a$

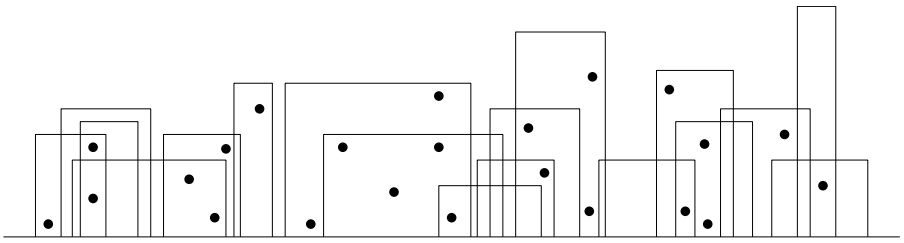


Figure 3.12: The instance where the rectangles above the horizontal line are considered.

Here, the rectangles in R^a are to be hit by the points in $Q^a \subseteq Q$ that lie above the line λ. Ignoring the portions of the rectangles in R^a below the line λ, the problem reduces to hitting a set of axis-parallel rectangles (R^a), whose one side is aligned with a horizontal line λ, using the input points of Q^a (see Figure 3.12). We solve this problem to compute OPT^a, the size of the optimum solution of this problem.

We compute the maximum independent set $\mathcal{I} = \{r_1, r_2, \ldots\}$ of the set of rectangles R^a such that **there does not exist any other maximum independent set $\mathcal{I}' = \{r'_1, r'_2, \ldots\}$ (of same size) where the span of a rectangle $r'_j \in \mathcal{I}'$ completely contains the span of a rectangle $r_i \in \mathcal{I}$ along the line λ**. The maximum independent set $\mathcal{I}$ can be computed in the same way we compute the maximum independent set of an interval graph where the horizontal range of each rectangle corresponds to an interval. It can be computed in $O(n \log n)$ time with the concept of interval scheduling [KT05].

$\Delta_{\mathcal{I}} = \{$lowest point of Q^a inside r_i for all the elements $r_i \in \mathcal{I}$ $\}$

is the minimum hitting set for the rectangles in $\mathcal{I}$. Such a point inside each r_i will always exist. Next, we consider the set of points

$\Delta'_{\mathcal{I}} = \{$lowest point of Q^a inside the strip χ_i bounded by the right side of r_i and left side of r_{i+1} for each pair of consecutive elements $r_i, r_{i+1} \in \mathcal{I}$, $i = 1, 2, \ldots, |\Delta_{\mathcal{I}}| - 1\}$.

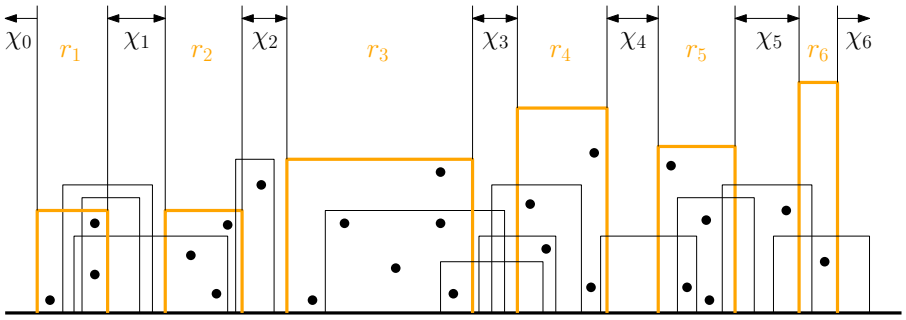


Figure 3.13: A maximum independent set of rectangles from R^a (shown in orange/thick lines) and the strips $\chi_i, i = 1, 2, \ldots, |\mathcal{I}| + 1$.

In Figure 3.13 the rectangles in the maximum independent set $\mathcal{I}$ are shown using orange color, and the strips χ_i are also shown.

Let $R' \subseteq R^a$ be the set of rectangles that are hit by $\Delta_{\mathcal{I}} \cup \Delta'_{\mathcal{I}}$. Below is a pictorial representation of the rectangles in R' (see Figure 3.14).

The remaining set of rectangles $R'' = R^a \setminus R'$ can be grouped into three exhaustive and mutually exclusive sets R_1, R_2 and R_3, where

R_1 $= \cup_{r_i \in \mathcal{I}} R_1^i$, where R_1^i is the set of non-hit rectangles in R^a that span from the interior of the strip χ_{i-1} up to the interior of the rectangle r_i.

R_2 $= \cup_{r_i \in \mathcal{I}} R_2^i$, where R_2^i is the set of non-hit rectangles in R^a that spans from the interior of the rectangle r_i up to the interior of the strip χ_i.

R_3 is the set of non-hit rectangles that overlap with the complete horizontal span of at least one rectangle in $\mathcal{I}$ or a strip.

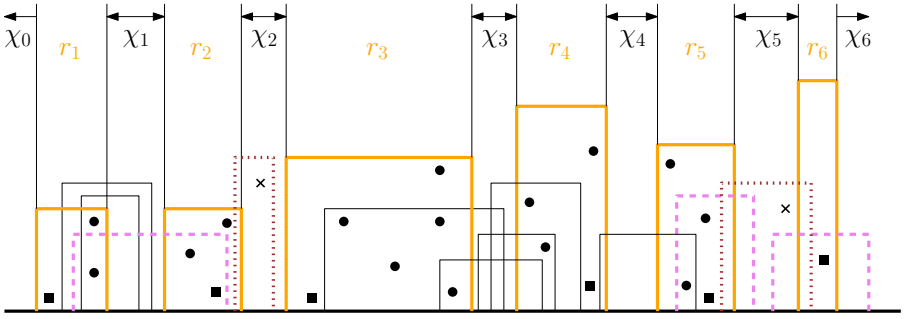


Figure 3.14: The points part of $\Delta_{\mathcal{I}}$ (shown as square points) and $\Delta'_{\mathcal{I}}$ (shown as cross points) and their corresponding rectangles in R' (shown by violet/dashed and brown/dotted lines respectively).

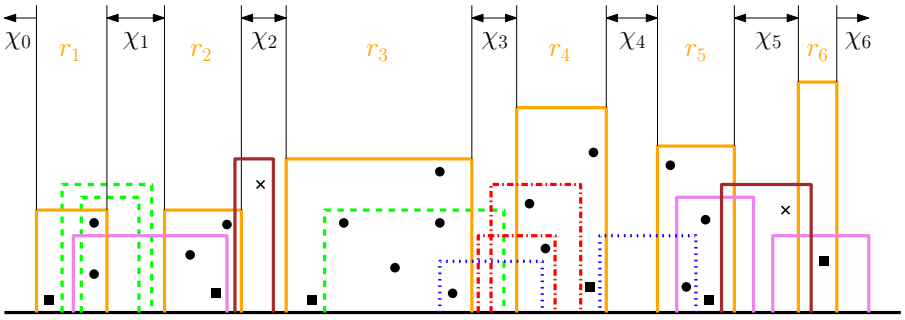


Figure 3.15: Demonstration of rectangles in R_1, R_2 and R_3 using dash dotted red line, dashed green line and dotted blue line respectively.

Note that, there exists no rectangle whose span along the line λ is included inside a strip (in that case it would have been included in $\mathcal{I}$) nor is included in a member of $\mathcal{I}$ (due to the restriction imposed in forming $\mathcal{I}$). In Figure 3.15, the rectangles in R_1, R_2 and R_3 are shown using red, green and blue colors, respectively.

Now, observe that in a feasible solution for R'', we have the following.

- Each rectangle $\rho \in R_1^i$ can be hit by a point of Q^a lying in the *right-part* of ρ inside r_i, or by a point of Q^a lying in the *left-part* of ρ inside the vertical strip χ_{i-1}.
- Similarly, a rectangle in $\rho \in R_2^i$ may be hit in its left-part (inside $r_i \in \mathcal{I}$) or in its right-part (inside the strip χ_i).

- Finally, consider the rectangles in R_3. If a rectangle in R_3 spans from the strip χ_{i-1} to the strip χ_j, and ρ is not hit by any point in $\Delta_{\mathcal{I}} \cup \Delta'_{\mathcal{I}}$, then the top boundary of ρ is below the chosen points in $r_i, \chi_i, r_{i+1}, \ldots, \chi_{j-1}, r_j$. Thus, it can only be hit by a point of $Q^a \cap \chi_{i-1}$ in its left part (the portion in the right side of the point chosen in $\chi_{i-1} \cap \Delta'_{\mathcal{I}}$) or by a point of $Q^a \cap \chi_j$ in its right-part (the portion in the left side of the point chosen in $\chi_j \cap \Delta'_{\mathcal{I}}$).

  Note that the left (resp. right) boundary of ρ may also lie inside of r_i and/or r_j. In that case, the left (resp. right) part of ρ lies inside r_i (resp. r_j).

  In a solution, a rectangle in R'' may be said to be a left-hit (resp. right-hit) rectangle if its left (resp. right) part is hit by a point in the solution.

Thus, one can decide whether a rectangle in $R'' = R_1 \cup R_2 \cup R_3$ is left-hit or right-hit by formulating an ILP with these rectangles as follows.

$$\mathcal{V} : \min \sum_{p_\alpha \in Q^a} x_\alpha$$
$$\text{Subject to } \sigma_{left}(\rho) + \sigma_{right}(\rho) \geq 1 \ \forall \ \rho \in R'',$$
$$x_\alpha \in \{0, 1\} \text{ for all } p_\alpha \in Q^a$$

where $\sigma_{left}(\rho)$ (resp. $\sigma_{right}(\rho)$) is the sum of the variables corresponding to the points in Q^a in the left-part (resp. right-part) of the rectangle ρ. The solution of its LP relaxation $\overline{\mathcal{V}}$ (see Section 3.3.2) partitions the set R'' into two subsets R_{left} (left-hit) and R_{right} (right-hit) such that for each $\rho \in R_{left}$, we have $\sigma_{left}(\rho) \geq \sigma_{right}(\rho)$ (thus, $\sigma_{left}(\rho) \geq \frac{1}{2}$) and for each $\rho \in R_{right}$, we have $\sigma_{left}(\rho) < \sigma_{right}(\rho)$ (thus, $\sigma_{right}(\rho) > \frac{1}{2}$).

Below, we describe the method of computing the optimum solution of the ILPs:

$$\mathcal{V}_{left} : \min \sum_{p_\alpha \in Q^a} x_\alpha$$
$$\text{Subject to } \sigma_{left}(\rho) \geq 1 \ \forall \ \rho \in R_{left},$$
$$x_\alpha \in \{0, 1\} \text{ for all } p_\alpha \in Q^a$$

$$\mathcal{V}_{right} : \min \sum_{p_\alpha \in Q^a} x_\alpha$$
$$\text{Subject to } \sigma_{right}(\rho) \geq 1 \ \forall \ \rho \in R_{right},$$
$$x_\alpha \in \{0,1\} \text{ for all } p_\alpha \in Q^a$$

Denoting by $\overline{OPT}_{\mathcal{V}_{left}}$, $\overline{OPT}_{\mathcal{V}_{right}}$, $\overline{OPT}_{\mathcal{V}}$ the optimal solutions of the LP versions of $\mathcal{V}_{left}$, $\mathcal{V}_{right}$ and $\mathcal{V}$, we have

$$\overline{OPT}_{\mathcal{V}_{left}} + \overline{OPT}_{\mathcal{V}_{right}} \leq 2\overline{OPT}_{\mathcal{V}} \tag{3.5}$$

Indeed, $2\overline{OPT}_{\mathcal{V}}$ is a solution for the LP versions of both $\mathcal{V}_{right}$ and $\mathcal{V}_{left}$. Moreover, the points involved in the solution of the LP version of $\mathcal{V}_{left}$ are distinct from those of the LP version of $\mathcal{V}_{right}$. Indeed, if for two rectangles in R'', the left-part of one and the right-part of the other were hit by a same point of Q^a, then together these two intervals would span an entire rectangle of $\mathcal{I}$ or a strip: but then one of them would have already been hit by $\Delta_{\mathcal{I}} \cup \Delta'_{\mathcal{I}}$, a contradiction.

As mentioned earlier, $|\Delta_{\mathcal{I}}| \leq \overline{OPT}^a_\lambda$ as $\Delta_{\mathcal{I}}$ satisfies a subset of constraints of $\mathcal{U}^a_\lambda$. Due to the same reason, $\overline{OPT}_{\mathcal{V}} \leq \overline{OPT}^a_\lambda$ since the rectangles in R'' are not hit by the points in $\Delta_{\mathcal{I}} \cup \Delta'_{\mathcal{I}}$. Moreover, the variables involved in $\Delta_{\mathcal{I}}$ and $\overline{OPT}_{\mathcal{V}}$ are different. Thus, using Equation 3.5, we have

$$|\Delta_{\mathcal{I}}| + |\Delta'_{\mathcal{I}}| + \overline{OPT}_{\mathcal{V}_{left}} + \overline{OPT}_{\mathcal{V}_{right}} \leq 4\overline{OPT}^a$$

We now show that $\overline{OPT}_{\mathcal{V}_{left}}$ (resp. $\overline{OPT}_{\mathcal{V}_{right}}$) are integer valued, and thus are in fact optimal solutions of the ILPs' $\mathcal{V}_{left}$ (resp. $\mathcal{V}_{right}$).

Computation of an optimal solution of $\mathcal{V}_{left}$

We will consider each vertical strip $\{S_0, S_1, S_2, \ldots, S_{2k+1}\} = \{\chi_0, r_1, \chi_1, r_2, \chi_2, \ldots, r_k, \chi_k\}$ in this order, where $\mathcal{I} = \{r_1, r_2, \ldots, r_k\}$. Consider a strip S_j, and let Γ_j be the portions of the members of R_{left} inside S_j; each member in Γ_j is to the right of the point in $\Delta \cup \Delta'$ chosen in that strip. Observe that the right side of the elements of Γ_j are aligned. The elements of Γ_j are arranged in order of their left-boundary. If an element $\rho \in \Gamma_j$ is completely contained in another element $\hat{\rho} \in \Gamma_j$, then $\hat{\rho}$ is deleted[3] from Γ_j. This pruning step of Γ_j can be made in a linear scan of the left boundaries of the elements of Γ_j in right-to-left order, and the remaining elements in Γ_j form a staircase. We can write an ILP for computing the minimum hitting set $\Delta_j \subseteq Q^a \cap S_j$ for Γ_j, where the variables correspond to the points in $Q^a \cap S_j$, and constraints correspond to the members in Γ_j. It can be shown that if the points in $Q^a \cap S_j$ are ordered and the rectangles in Γ_j are also ordered with respect to their left boundaries from right to left, then the incidence matrix becomes totally unimodular i.e. the 1's in each row of the constraint matrix are in consecutive columns. Thus, the LP solution of this problem becomes 0-1 valued. Finally, $\Delta_{left} = \cup_{j=1}^{2k+1} \Delta_j$ is the optimum solution $\overline{OPT}_{\mathcal{V}_{left}}$ for hitting the rectangles in R_{left}.

The same process is executed with the right-part of the rectangles in R_{right} to compute the optimal solution $\Delta_{right} = \overline{OPT}_{\mathcal{V}_{right}}$. Finally, we have

$$SOL^a = \Delta_{\mathcal{I}} \cup \Delta'_{\mathcal{I}} \cup \Delta_{left} \cup \Delta_{right},$$

Thus,

$$|SOL^a| \leq 4\overline{OPT}^a$$

The same process is executed to compute the subset $SOL^b \subset Q^b$ used for hitting the rectangles in R^b, and $|SOL^b| \leq 4\overline{OPT}^b$. Thus, $SOL_\lambda = SOL^a \cup SOL^b$, and hence $|SOL_\lambda| \leq 4(\overline{OPT}^a + \overline{OPT}^b)$. Now, using Equation 3.4, we have the following.

[3] as hitting ρ implies hitting $\hat{\rho}$

Lemma 3.5

For each line $\lambda \in \mathcal{L}$, we have $|SOL_\lambda| \leq 8 \times OPT_\lambda = 8\overline{OPT}_\lambda$.

Now, using Equations 6.3, 6.4 and Lemma 3.5, we have the following result:

Theorem 3.6

Discrete-G-Min-Disc-Code for axis-parallel unit squares in 2D has a polynomial-time algorithm that produces a solution of size at most $128OPT + 1$, where OPT is the size of an optimum solution.

3.4 Min-ID-Code for geometric intersection graphs

In this section, we will use techniques similar to those used in the previous sections and apply them to the setting of the graph problem Min-ID-Code, for the intersection graph of axis-parallel unit squares (unit square graphs). To the best of our knowledge, Min-ID-Code was not yet studied for unit square intersection graphs in the literature.

Here, the input is a set S of axis-parallel unit squares in 2D. In the graph $G = (V, E)$, the nodes in $V = \{v_1, \ldots, v_n\}$ correspond to the squares in S; an edge $e_{ij} = \{v_i, v_j\} \in E$ if the squares corresponding to v_i, v_j intersect.

Note that it is not mentioned in the literature whether Min-Id-Code on unit square graphs is NP-hard, however, the techniques from [MS09] used for unit disk graphs can be applied to prove it.

We can reformulate Min-ID-Code for unit square graphs in geometric terms: the objective is to compute a subset $S_{opt} \subseteq S$ of minimum cardinality such that each square in S intersects some square in S_{opt}, and for each pair of squares $s_i, s_j \in S$, there exists a square $\sigma \in S_{opt}$ such that ($\sigma \cap s_i \neq \emptyset$ and $\sigma \cap s_j = \emptyset$) or ($\sigma \cap s_i = \emptyset$ and $\sigma \cap s_j \neq \emptyset$). If we do only satisfy the discrimination constraint, then in order

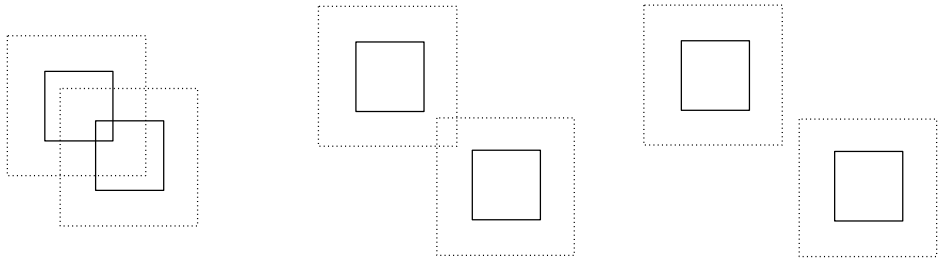


Figure 3.16: Possible intersection patterns of a pair of axis-parallel unit squares (full lines): the dotted square around each square corresponds to the locations where a square centered at this point will intersect the enclosed unit square.

to satisfy the domination constraint, we may need to include at most one more square from S in S_{opt}.

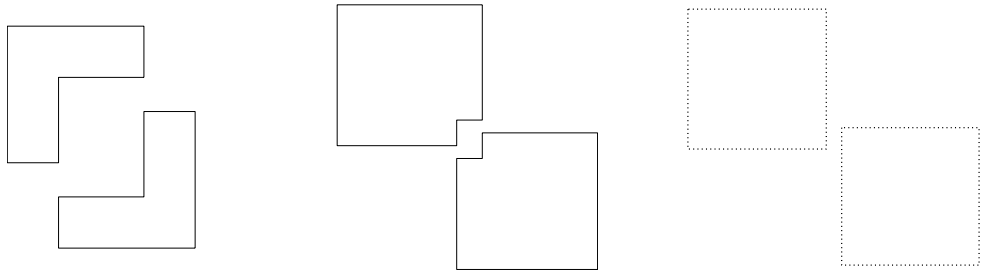


Figure 3.17: Feasible regions for placing the center of the square $s \in ID$ for discriminating $s_i, s_j \in S$: three possible situations.

Let $S' \subseteq S$ be an identifying code for the set of squares in S. A square $\sigma \in S'$ intersects a square $s_i \in S$ if the center of σ is placed inside the square δ centered at the center of s and the side-length of δ is twice the side-length of s (shown using dotted line around s_i in Figure 3.16). In order to satisfy the discrimination constraint among $s_i, s_j \in S$, the center of a square $\sigma \in S'$ must be placed inside $\delta_i \nabla \delta_j$, where ∇ is the symmetric difference operator, i.e., $(\delta_i \setminus \delta_j) \cup (\delta_j \setminus \delta_i)$. In Figure 3.16, different patterns of intersection of a pair $s_i, s_j \in S$ are depicted, along with their covering regions δ_i, δ_j. Thus, in order to satisfy the discrimination constraint (s_i, s_j), we need to place the center of a square $\sigma \in S'$ in the regions shown in Figure 3.17.

Thus, as in Section 3.3.2, we can solve Min-ID-Code for unit square graphs by solving a problem of hitting the feasible regions corresponding to each pair of

squares $s_i, s_j \in S$ using the centers of the squares in S. The objective will be to choose the minimum number of hitting squares from S. Thus, the same techniques as in Section 3.3.2 can be applied, and we obtain the following theorem.

Theorem 3.7

MIN-ID-CODE has a polynomial-time approximation algorithm for unit square graphs (if the unit square intersection model of the input graph is known) that produces a solution of size at most $128OPT + 1$, where OPT is the size of an optimal solution.

CHAPTER 4

Red-Blue Separation

Contents

4.1 Preleminaries

Before delving into the problem let us first discuss some definitions which will make the reading of the chapter easier. We first need to define some notations with respect to a graph G.

Order of a graph

Definition 4.1. By the *order* of a graph $G = (V, E)$ we mean the number of vertices in V which is usually denoted by n.

Neighborhood

Definition 4.2. In graph theory, an adjacent vertex of a vertex $v \in V$ is a vertex that is connected to v by an edge. The *open neighborhood* of a vertex v in a graph G is used to refer to sets of adjacent vertices of v where v itself is not included. The open neighborhood is denoted $N(v)$. The *closed neighborhood* of a vertex v in a graph G is used to refer to sets of adjacent vertices of v in which v itself is also included. The closed neighborhood is denoted $N[v] = N(v) \cup \{v\}$.

Split Graph

Definition 4.3. A graph G is known as a *split graph* if it can be partitioned into a clique and an independent set.

Figure 4.1 is a split graph where vertices v_1 and v_2 form an independent set and the vertices v_3 through v_6 form a clique.

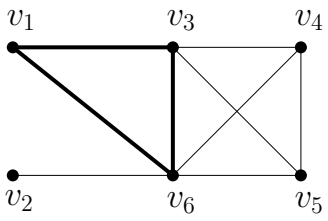


Figure 4.1: A split graph with a girth highlighted.

Girth of a graph

Definition 4.4. The *girth* of a graph is the length of a shortest cycle contained in the graph. See the highlighted portion of the graph in Figure 4.1.

Symmetric Difference

Definition 4.5. The symmetric difference $A \triangle B$ of two sets A and B is the set of elements which are in either of the sets A and B, but not in their intersection. Thus $A \triangle B = (A \setminus B) \cup (B \setminus A)$.

Since this chapter talks about the problem of Red-Blue Separation, let me first define what we mean by separation or separating code in a non-colored setup.

Separation in a graph

Definition 4.6. A *separating set* is a subset S of vertices of $G = (V, E)$ such that for each pair $u, v \in V$ of distinct vertices, we have $N[u] \cap S \neq N[v] \cap S$. The *separating code* of a vertex u is $N[u] \cap S$. The size of the separating set of the graph G is known as the *separation number* of G and is denoted by $\text{sep}(G)$.

4.2 Organization

In the graph setting, we are given a graph $G = (V, E)$ and a red-blue coloring function $c : V \to \{\text{red}, \text{blue}\}$ of the vertices of G; we want to select a (small) subset $S \subseteq V$ called *red-blue separating set*, such that for every red-blue pair $r, b \in V$ (r is of red color and b is of blue color), there is a vertex $v \in S$ such that the closed neighborhood of v contains exactly one of r and b, in other words $|N[v] \cap \{r, b\}| = 1$. Equivalently, $N[r] \cap S \neq N[b] \cap S$, where $N[x]$ denotes the closed neighborhood of vertex x; the set $N[x] \cap S$ is called the *code* of vertex x (with respect to S). Thus the code of each blue vertex is different from the codes of all the red vertices. Note that, the code of a pair of red (respectively blue) vertices may be same or different. The smallest size of a red-blue separating set of (G, c) is denoted by $\text{sep}_{\text{RB}}(G, c)$. In this chapter, a *twin* is a red-blue pair of vertices r, b such that $N[r] = N[b]$. If G contains a twin, then it cannot be separated. Thus, for simplicity, we will consider only *twin-free* graphs, that is, graphs where no two different colored vertices have the same closed neighborhood. We have the following associated computational problem.

Problem: RED-BLUE SEPARATION$((G, c), k)$

Input: A red-blue colored twin-free graph (G, c) and an integer k.

Output: Do we have $\text{sep}_{\text{RB}}(G, c) \leq k$?

It is also interesting to study the problem when the red-blue coloring is not part of the input. For a given graph G, we thus define the parameter $\text{max-sep}_{\text{RB}}(G)$ which denotes the largest size of a smallest red-blue separating set of (G, c) over all possible red-blue coloring c of G. Mathematically, $\text{max-sep}_{\text{RB}}(G) = \max_{c \in C}\{|S_c|\}$ where S_c is the separating set with respect to the color function c and $|C| = 2^n$. The associated decision problem is stated as follows.

Problem: MAX RED-BLUE SEPARATION(G, k)

Input: A twin-free graph G and an integer k.

Output: Do we have max-$\text{sep}_{\text{RB}}(G) \leq k$?

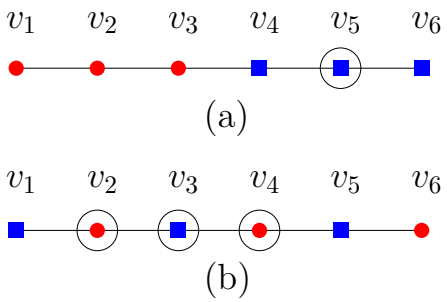


Figure 4.2: An example showing the difference between sep_{RB} and max-sep_{RB} in a path graph of 6 vertices.

In order to understand the difference between the two problems, let us look at Figure 4.2. In Figure 4.2(a), we are given a path graph of 6 vertices and a coloring c. If we choose the vertex v_5 in the separating set, then all the vertices in the path are separated. Thus, $\text{sep}_{\text{RB}}(G, c) = 1$. On the contrary in Figure 4.2(b), we are given just a path of 6 vertices. Over all possible colorings of this given path, if we consider the particular coloring shown in the figure, then the separating set includes the vertices v_2 through v_4 in order to separate all the vertices in the path. Hence, max-$\text{sep}_{\text{RB}}(G) = 3$.

In this chapter we show that RED-BLUE SEPARATION is NP-complete even for restricted graph classes such as planar bipartite sub-cubic graphs, in the setting where the two color classes have equal size. We also show that the problem is NP-hard to approximate within a factor of $(1 - \epsilon) \ln n$ for every $\epsilon > 0$, even for split graphs of order n having one color class of size 1. On the other hand, we show that RED-BLUE SEPARATION is always approximable in polynomial-time within a factor of $2 \ln n$ for any bi-colored graph. In contrast, for triangle-free graphs and for graphs of bounded maximum degree, RED-BLUE SEPARATION is solvable in polynomial time when the smallest color class is bounded by a constant (using algorithms that are in the parameterized class XP, with the size of the smallest

color class as parameter). However, on general graphs, the problem is $W[2]$-hard even when parameterized by the solution size plus the size of the smallest color class.

When the coloring is not specified, max-sep$_{\text{RB}}(G)$ is a parameter that is worth studying from a structural viewpoint. In particular, we study the possible values for max-sep$_{\text{RB}}(G)$. We show the existence of tight bounds on max-sep$_{\text{RB}}(G)$ in terms of the order n of the graph G, proving that it can range from $\lfloor \log_2 n \rfloor$ up to $n-1$ (both bounds are tight). For trees however we prove bounds involving the number of support vertices (i.e. which have a leaf neighbor), which imply that max-sep$_{\text{RB}}(G) \leq \frac{2n}{3}$. We also give bounds of max-sep$_{\text{RB}}(G)$ in terms of sep(G). We then show that the associated decision problem Max Red-Blue Separation is NP-hard, even for graphs of bounded maximum degree, but can be approximated in polynomial time within a factor of $O(\ln^2 n)$.

4.3 Complexity of Red-Blue Separation

We will prove some algorithmic results for Red-Blue Separation by reducing to or from the following problems.

Problem: Set Cover of $((U, \mathcal{S}), k)$
Input: A set of elements U, a family $\mathcal{S}$ of subsets of U and an integer k.
Output: Does there exist a cover $\mathcal{C} \subseteq \mathcal{S}$, with $|\mathcal{C}| \leq k$ such that $\bigcup_{C \in \mathcal{C}} C = U$?

Problem: Dominating Set
Input: A graph $G = (V, E)$ and an integer k.
Output: Does there exist a set $D \subseteq V$ of size k such that $\forall v \in V, N[v] \cap D \neq \emptyset$?

4.3.1 Hardness

Theorem 4.1

For a split graph of order n, Red-Blue Separation cannot be approximated within a factor of $(1-\epsilon)\cdot \ln n$ for any $\epsilon > 0$ even when the smaller color class has size 1 unless P = NP. Moreover, Red-Blue Separation is W[2]-hard when parameterized by the solution size together with the size of the smallest color class, even on split graphs.

Proof. For an instance $((U, \mathcal{S}), k)$ of Set-Cover, we construct in polynomial time an instance $((G, c), k)$ of Red-Blue Separation where G is a split graph and one color class has size 1. The statement will follow from the hardness of approximating Min Set Cover proved in [DS14], and from the fact that Set Cover is W[2]-hard when parameterized by the solution size [DF99].

Construction: We create the graph (G, c) by first creating vertices corresponding to all the elements in U and the sets in $\mathcal{S}$. Let V_U and $V_{\mathcal{S}}$ be the set of vertices corresponding to the elements in U and the subsets in $\mathcal{S}$ respectively. We connect a vertex $u_i \in V_U$ corresponding to an element $i \in U$ to a vertex $v_j \in V_{\mathcal{S}}$ corresponding to a set $S_j \in \mathcal{S}$ provided $i \in S_j$. We color all these vertices blue. We add two isolated blue vertices b and b'. We connect every pair of vertices in V_U. Also, we add a red vertex r and connect all vertices $u_i \in V_U$ to r. Now, note that the vertices $V_U \cup \{r\}$ form a clique whereas the vertices in $V_{\mathcal{S}}$ along with b and b' form an independent set[1]. Thus, our constructed graph G is a split graph. See Figure 4.3.

Now Claim 4.1, stated below, proves the result. □

[1]The reason for using two isolated vertices b and b' is to ensure that the red vertex r does not receive the empty code.

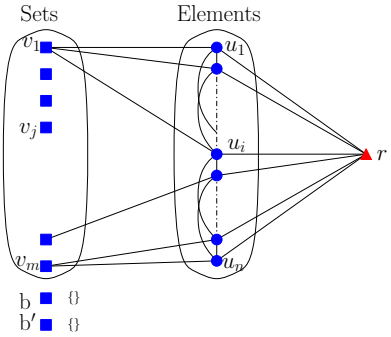


Figure 4.3: Proof of Theorem 4.1: reduction from Set Cover to Red-Blue Separation.

Claim 4.1

$\mathcal{S}$ has a set cover of size k if and only if G has a red-blue separating set of size at most $k+1$.

Proof. ($\implies$) Let $\mathcal{C}$ be a set cover of $(U, \mathcal{S})$ of size k. We construct a red-blue separating set S of (G, c) of size at most $k+1$ as follows. For each set S_j selected in the set cover $\mathcal{C}$, we choose the corresponding vertex v_j in S. Also, include the vertex r in S. Observe that some blue vertices[2] may have the empty code and the red vertex has itself as the code. Also, every vertex in V_U is dominated by some vertex/vertices in $V_{\mathcal{S}}$ and have some unique code different from r. Thus, every blue vertex has code different from the code of r. Therefore, S is a separating set of (G, c) of size at most $k+1$.

($\impliedby$) Conversely, consider a red-blue separating set S of (G, c) of size k'. Since the vertices in $V_U \cup \{r\}$ form a clique, choosing any vertex from V_U will not separate any two vertices of the clique. Let us assume that the red vertex $r \in V_U$ gets the empty code. Then, we have the two isolated blue vertices b and b', one of which also gets the empty code, which is not permissible. Thus, the red vertex has to be selected. But r, being part of the clique, has to be separated from the blue vertices in the set V_U. Thus, we have to choose vertices from the independent set to separate the blue vertices of the clique from r. So, we dominate the blue vertices

[2]corresponding to subsets that are not included in $\mathcal{C}$ and b, b'

of the clique by using the vertices in the independent set of V_S, which implies that the size of our set cover is at most $k' - 1 = k$. □

Theorem 4.2

RED-BLUE SEPARATION is NP-hard for bipartite planar sub-cubic graphs of girth at least 12 when the color classes have almost the same size.

Proof. We reduce from DOMINATING SET, which is NP-hard for bipartite planar sub-cubic graphs with girth at least 12 that contain some degree 2 vertices [ZZ95]. We reduce any instance (G, k) of DOMINATING SET to an instance $((H, c), k + 1)$ of RED-BLUE SEPARATION where the number of red and blue vertices in H differ by at most 2.

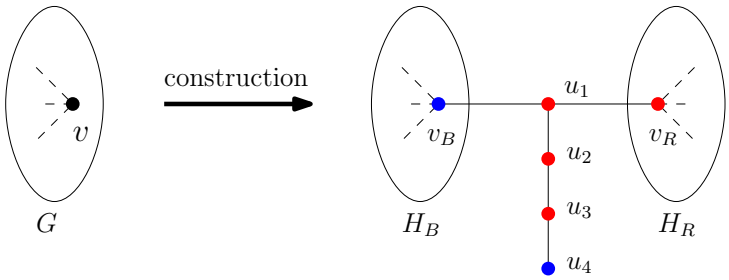


Figure 4.4: Reduction from an instance of DOMINATING SET to an instance of RED-BLUE SEPARATION.

Construction: We create two disjoint copies of $G = (V, E)$ namely $H_B = (V_B, E_B)$ and $H_R = (V_R, E_R)$ and color all vertices of H_B as blue and all vertices of H_R as red. Select an arbitrary vertex v of degree 2 in V (we may assume such a vertex exists in G by the reduction from the Dominating Set problem for 3-regular planar graphs of [ZZ95]) and look at its corresponding vertices $v_R \in V_R$ and $v_B \in V_B$. We connect v_R and v_B with the head of a path u_1, u_2, u_3, u_4 as shown in Figure 4.4. The tail of the path, i.e. the vertex u_4, is colored blue and the remaining three vertices u_1, u_2 and u_3 are colored red. The vertices in the graph H is $V_H = V_R \cup V_B \cup \{u_1, u_2, u_3, u_4\}$ and the edges of H are $E_H = E_B \cup E_R \cup \{(u_1, v_B), (u_1, v_R), (u_1, u_2), (u_2, u_3), (u_3, u_4)\}$. The coloring c is as described.

Now Claim 4.2 and Claim 4.3, stated below, proves the result. □

Claim 4.2

If G is a connected bipartite planar sub-cubic graph of girth at least g, then so is H.

Proof. The graph H is also bipartite as the vertices v_B and v_R can belong to the same partition of G where v belonged (both being replication of the vertex $v \in V$) along with u_2 and u_4, while the vertices u_1 and u_3 can belong to the other partition. □

Claim 4.3

The instance (G, k) is a YES-instance of Dominating Set if and only if $\text{sep}_{\text{RB}}(H, c) \leq k' = k + 1$.

Proof. ($\Longrightarrow$) Let D be a dominating set of G of size k. We construct a red-blue separating set S of (H, c) of size at most $k + 1$ as follows. For each vertex $v \in D$, include its corresponding vertex $v_R \in V_R$ in S. Also include in S, the vertex $u_2 \in V_H$. Observe that all blue vertices have the empty code and all red vertices have some non-empty code. Therefore S is a separating set of (H, c) of size at most $k + 1$.

($\Longleftarrow$) Consider a red-blue separating set S of (H, c) of size $k+1$. We are to show that there exists a dominating set of size k in G. Without loss of generality, let us assume that no red vertex of (H, c) gets the empty code. Then, the set S has to dominate all the red vertices in $V_R \cup \{u_1, u_2, u_3\}$ in order to ensure the separation of the red vertices from the blue vertices. Since u_3 can only be dominated by a vertex $x \in \{u_2, u_3, u_4\}$, the vertices in V_R are to be dominated by $S' = S \setminus \{x\}$. Now two cases need to be considered:

$S' \subseteq V_R$: Here, the set D, formed by choosing the vertices of V corresponding to the vertices of S' is a dominating set of G of size k.

$S' \not\subseteq V_R$: This implies $\exists\ y \in S'$ such that $y \notin V_R$. The only possible vertex which can be y is u_1, i.e., $u_1 \in S'$. That happens when neither v_R nor u_2 is in S. But if x is either u_3 or u_4 in the set S, x will not separate u_3 and u_4. So u_2 has to be in S to ensure that u_3 and u_4 are separated. But the sole choice of $u_2 \in S$ itself ensures the separation of the three red vertices $\{u_1, u_2, u_3\}$ from all the blue vertices. Hence, y is not necessary in S'.

Thus, $S' \subseteq V_R$, and it implies $|D| = k$. □

In the reduction of Theorem 4.2, the chosen graph could be any arbitrary graph G for which Dominating Set is known to be NP-hard. We could also create a graph H by simply taking two copies of the original graph G and obtain a coloring with two color classes of equal size and dominate just one color class. Thus, all the vertices of one color class, where the dominating set D is chosen in S, have non-empty code. All the vertices of the other color class receive empty code. Then, D is a separating set of H. But note that here H is a disconnected graph.

Theorem 4.3

Red-Blue Separation is NP-hard even when the input is a sub-cubic planar bipartite graph of girth at least 12.

Proof. We reduce from an instance (G, k) of the Dominating Set which is NP-hard when the input graph G is a sub-cubic planar bipartite graph of girth at least 12 [ZZ95].

Let (G, k) be an instance of the Dominating Set problem, where G is as mentioned above. We create an instance $((H, c'), k + 1)$ of Red-Blue Separation where c' is a coloring of H with the minimum color class being of size at most $k+1$ as follows (See Figure 4.5):

Construction: Without loss of generality let us assume the smaller sized color class to be red. For H we create a copy G_H of G with all its vertices colored

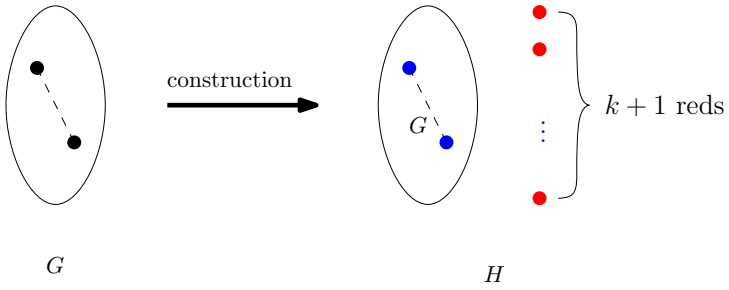


Figure 4.5: Construction from an instance of DOMINATING SET to an instance of RED-BLUE SEPARATION where size of the smaller color class is bounded.

blue and an independent set I of size $k+1$ all its vertices colored red.

Now Claim 4.4, stated below, proves the result. □

Claim 4.4

(G, k) is an YES-instance of DOMINATING SET if and only if $((H, c'), k)$ is a YES-instance of RED-BLUE SEPARATION.

Proof. ($\Longrightarrow$) Let D be a dominating set of G. We construct a red-blue separating set of (H, c') by selecting the corresponding vertices of D in G_H. Since D is a dominating set of G, all blue vertices receive a non-empty code. The red vertices on the other hand receive the empty code and we have a valid separating set.

($\Longleftarrow$) Let (H, c') have a separating set C of size at most k. Since there are $k+1$ independent red vertices, there will be at least one red vertex which receives the empty code. So, in order for C to be a valid separating set, all blue vertices must receive some non-empty code. This implies that $C \cap V(G_H)$ is a dominating set of G_H of size at most k, and which implies a dominating set of G of size at most k. □

4.3.2 Positive algorithmic results

We start with a reduction from $\text{sep}_{\text{RB}}(G, c)$ problem to Set Cover with an objective to suggest possible approximation algorithm(s) for the Red-Blue Separation problem.

Proposition 4.1

Red-Blue Separation has a polynomial time $(2 \ln n)$-factor approximation algorithm.

Proof. We reduce Red-Blue Separation to Set Cover. Let $((G, c), k)$ be an input instance of Red-Blue Separation. We reduce it to an instance $((U, \mathcal{S}), k)$ of Set Cover. For each red-blue vertex pair (r, b) in G create an element $u_{r,b}$ in U. For each vertex v in G create a set in $\mathcal{S}$ with all possible elements u_{r_i,b_j} in U such that v is in the closed neighborhood of exactly one of r_i and b_j in G. Observe that a set cover $\mathcal{C}$ of size k in $((U, \mathcal{S}), k)$ corresponds to a separating set S of size at most k of the graph G and vice versa (see Figure 4.6 for a visual representation).

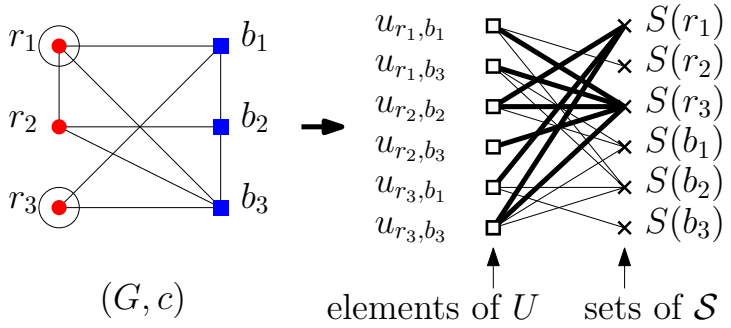


Figure 4.6: A reduction instance from Red-Blue Separation of (G, c) to Set Cover of $(U, \mathcal{S})$. The separating set in the colored graph (G, c) and the corresponding set cover in the set system $(U, \mathcal{S})$ has been highlighted.

The greedy algorithm for Set Cover has an approximation factor of $\ln |U| + 1$. Since, in our case $|U| \leq n^2/4$, the resulting approximation factor for Red-Blue Separation is at most $\ln(n^2/4) + 1 \leq 2 \ln n$. □

Proposition 4.2

Let (G, c) be a red-blue colored triangle-free and twin-free graph with R, B the two color classes. Then, $\text{sep}_{\text{RB}}(G, c) \leq 3 \min\{|R|, |B|\}$.

Proof. We construct a red-blue separating set S of (G, c) as follows. Without loss of generality, we assume $|R| \leq |B|$. First, we add all red vertices to S. It remains to separate every red vertex from its blue neighbors. If a red vertex $v \in R$ has at least two neighbors, we add (any) two such neighbors to S [3]. Since G is triangle-free, no blue neighbor of v is in the closed neighborhood of both these neighbors of v, and thus v is separated from all its neighbors. If v had only one neighbor w, and it was blue, then we separate w from v by adding one arbitrary neighbor of w (other than v) to S. Since G is triangle-free, v and w are separated. See Figure 4.7. Thus, we have built a red-blue separating set S of size at most $3|R|$.

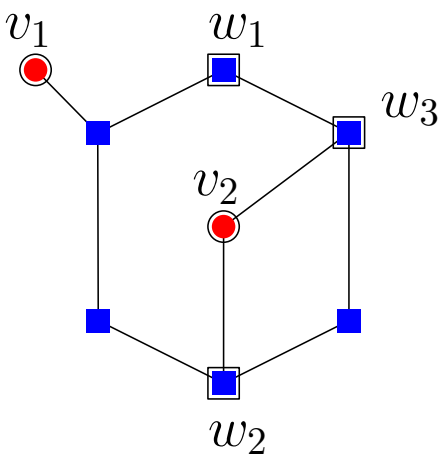


Figure 4.7: Illustration of Proposition 4.2: here v_1 has just one blue neighbor hence w_1 is added in S. v_2's neighbors w_2 and w_3 are also included in S.

□

[3] Note that as red vertices are already added in S, if a red vertex v has at least two red neighbors, then there is no need to add any more vertex in S as already two neighbors of v are already present in S. If v has one or zero red neighbors, then only this step needs to be executed.

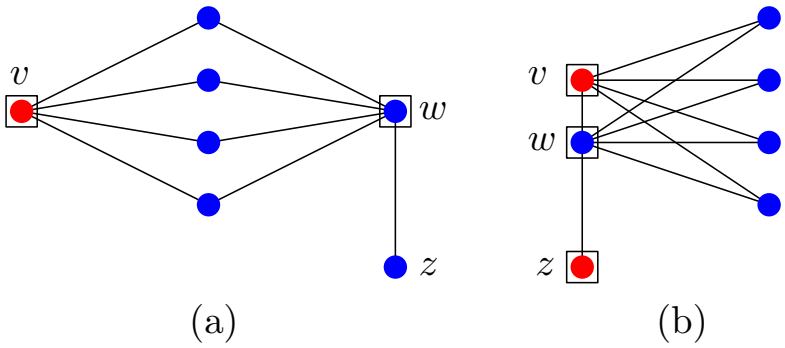


Figure 4.8: Illustration of Proposition 4.3: the two sub-cases of Case 1 when the vertices v and w are (a) not adjacent and (b) adjacent.

Proposition 4.3

Let (G, c) be a red-blue colored twin-free graph with maximum degree $\Delta \geq 3$. Then, $\text{sep}_{\text{RB}}(G, c) \leq \Delta \min\{|R|, |B|\}$.

Proof. We construct a red-blue separating set S of (G, c) as follows. Without loss of generality, let us assume $|R| \leq |B|$. For every red vertex $v \in R$, we need to consider the following two cases:

Case 1: If there is a blue vertex w whose closed neighborhood contains all neighbors of v (w could be a neighbor of v), we add both v and w to S. If v is adjacent to w, since they cannot be twins, there must be a vertex z that can separate v and w; we add z to S. Now, v is separated from every blue vertex in G (See Figure 4.8(a)).

Case 2: If such a vertex w does not exist, then we add all neighbors of v to S. Now again, v is separated from every vertex of G (See Figure 4.8(b)).

Thus, we have built a red-blue separating set S of size at most $\Delta|R|$. □

The previous propositions imply that Red-Blue Separation can be solved in XP time for the parameter "size of a smaller color class" on triangle-free graphs and on

graphs of bounded degree. This is in contrast with the fact that in general graphs, it remains NP-hard even when the smallest color class has size 1 (Theorem 4.1).

Theorem 4.4

Red-Blue Separation on red-blue colored graphs with color classes R and B can be solved in time $O(n^{3\min\{|R|,|B|\}})$ on triangle-free graphs, and in time $O(n^{\Delta\min\{|R|,|B|\}})$ on graphs of maximum degree Δ.

4.4 Extremal values and bounds for $\text{max-sep}_{\text{RB}}$

As defined earlier, we denote by $\text{sep}(G)$ the smallest size of a (non-colored) separating set of G, that is, a set that separates *all* pairs of vertices. In this section, we will throughout use the relation $\text{max-sep}_{\text{RB}}(G) \leq \text{sep}(G)$, which clearly holds for every twin-free graph G.

4.4.1 Lower bounds for general graphs

We can have a large twin-free colored graph with solution size 2 (for an example, in a large blue path with a single red vertex at any position, two vertices suffice). We show that in every twin-free graph, there is always a coloring that requires a large solution.

Theorem 4.5

There exists twin-free graph G of order $n \geq 1$ and $n \notin \{8, 9, 16, 17\}$, where $\text{max-sep}_{\text{RB}}(G) \geq \lfloor \log_2 n \rfloor$.

Proof. Let G be a twin-free graph of order n with $\text{max-sep}_{\text{RB}}(G) = k$. In a twin-free graph G, there are 2^n different red-blue colorings. Now $\text{max-sep}_{\text{RB}}(G) = k$ implies for each of these 2^n graphs $\text{sep}_{\text{RB}}(G, c) \leq k$. Consider the set of vertex

subsets of G which are separating sets of size k for some red-blue colorings of G. In G, there are $\binom{n}{k} \leq n^k$ sets of k vertices.

Consider such a separating set S of k vertices with some coloring function c. Let the set $I(S)$ be the subsets of S such that for each subset $S' \in I(S)$ for which there exists a vertex v of G with $N[v] \cap S = S'$. Let $i_S = |I(S)|$, the number of these subsets. Surely, we have $i_S \leq 2^{|S|} \leq 2^k$. As S is a separating set for (G, c), all vertices of G having the same intersection between their closed neighborhood and S must receive the same color by the coloring function c. Thus, there are at most $2^{i_S} \leq 2^{2^k}$ red-blue colorings of G for which S is a separating set. Overall, we thus have at most $\binom{n}{k}2^{2^k}$ possible red-blue coloring, implying $2^n \leq \binom{n}{k}2^{2^k} \leq n^k 2^{2^k}$, and thus $n \leq k \log_2 n + 2^k$.

We now claim that this implies that $k \geq \log_2(n - (\log_2 n)^2)$. Suppose to the contrary that this is not the case, then we would obtain:

$$\begin{aligned} n &< \log_2(n - (\log_2 n)^2) \log_2 n + n - (\log_2 n)^2 \\ n &< \log_2 n \log_2 n + n - (\log_2 n)^2 \\ n &< n \implies \text{a contradiction} \end{aligned}$$

Since k is an integer, we actually have $k \geq \lceil \log_2(n - (\log_2 n)^2) \rceil$. To conclude, one can check that whenever $n \geq 70$, we have $\lceil \log_2(n - (\log_2 n)^2) \rceil \geq \lfloor \log_2 n \rfloor$. Moreover, if we compute values for $2^n - \binom{n}{k}2^{2^k}$ when $1 \leq n \leq 69$ and $k = \lfloor \log_2 n \rfloor - 1$, then we observe that this is negative only when $n \in \{8, 9, 16, 17\}$. Thus, $\lfloor \log_2 n \rfloor$ is a lower bound for max-sep$_{\text{RB}}(G)$ as long as $n \notin \{8, 9, 16, 17\}$. □

Proposition 4.4

For any integer $k \geq 1$, if $n = 2^k$, there is a graph G of order n with max-sep$_{\text{RB}}(G) = k$.

Proof. We build G as follows. Let us take a set $S = \{v_1, \ldots, v_k\}$ of k vertices in G. Create another set T of vertices (disjoint from S). For every subset S' of S of size at least 2, we add a vertex $v_{S'}$ to T. We join $v_{S'}$ to all vertices of S' and also with the other vertices of T. Finally, we add an isolated vertex $v_\emptyset$ to G. Thus, the vertices of G are $\{v_\emptyset\} \cup S \cup T$. The edges in the undirected graph G can be classified into two types: for a subset $S' \subseteq S$, the vertex $v_{S'}$ is connected to (i) each $v_i \in S'$ and (ii) all vertices of type $v_{S''}$ where $|S''| \leq |S'|$ and $S'' \subset S$. Observe that there is no edge between the vertices of S. Thus, the graph G is a split graph since the set S forms an independent set and the set T induces a clique in G because of the edges of the second type.

To see that max-sep$_{\text{RB}}(G) \leq k$, notice that S is a separating set of G regardless of the coloring of the vertices of G. The reason is that every vertex of S receives a different code which is the id of that vertex. Since all possible subsets correspond to the vertices of T, thus all the vertices of G receive unique code.

If $k = 1$: $c(v_1) =$ red and $c(v_\emptyset) =$ blue shows that max-sep$_{\text{RB}}(G) \geq 1$.

If $k = 2$: $c(v_1) = c(v_2) =$ red and $c(v_{\{1,2\}}) = c(v_\emptyset) =$ blue. To separate $v_{\{1,2\}}$ from v_1, v_2 must belong to any separating set. Again, to separate $v_{\{1,2\}}$ from v_2, v_1 must belong to any separating set. Thus, we have max-sep$_{\text{RB}}(G) \geq 2$.

If $k \geq 3$: $c(v_S) =$ blue and the other vertices in G are red. For each subset S' of S with $|S'| = k-1$, in order to separate v_S from $v_{S'}$, any separating set needs to contain a vertex v_i where $\{v_i\} = S \setminus S'$. This shows that max-sep$_{\text{RB}}(G) \geq k$.

□

We next relate parameter max-sep$_{\text{RB}}$ to other graph parameters.

Theorem 4.6

Let G be a graph on n vertices. Then, $\text{sep}(G) \leq \min\{(\lceil \log_2 n \rceil \times \text{max-sep}_{\text{RB}}(G)), (\lceil \log_2(\Delta(G)+1) \rceil \times \text{max-sep}_{\text{RB}}(G) + \gamma(G))\}$, where $\gamma(G)$ is the domination number of G and $\Delta(G)$ is its maximum degree.

Proof. Let G be an arbitrary graph with n vertices, where $2^{k-1}+1 \leq n \leq 2^k$ for some integer k. We first prove that $\text{sep}(G) \leq \lceil \log_2 n \rceil \times \text{max-sep}_{\text{RB}}(G)$. Next, we prove that $\text{sep}(G) \leq \lceil \log_2(\Delta(G)+1) \rceil \times \text{max-sep}_{\text{RB}}(G) + \gamma(G)$. We denote each vertex by a different k-length binary word $x_1x_2\cdots x_k$ where each $x_i \in \{0,1\}$. Moreover, we give k different red-blue colorings $c_1, \ldots, c_k$ such that vertex $x_1x_2\cdots x_k$ is red in coloring c_i if and only if $x_i = 0$ and blue otherwise. For each i, let S_i be an optimal red-blue separating set of (G, c_i). We have $|S_i| \leq \text{max-sep}_{\text{RB}}(G)$ for each i. Let $S = \bigcup_{i=1}^{k} S_i$. Now, $|S| \leq k \cdot \text{max-sep}_{\text{RB}}(G) = \lceil \log_2 n \rceil \cdot \text{max-sep}_{\text{RB}}(G)$. We claim that S is a separating set of G. Assume to the contrary that for two vertices $x = x_1x_2\cdots x_k$ and $y = y_1y_2\cdots y_k$, $N[x] \cap S = N[y] \cap S$. As x and y are two distinct vertices, we have $y_i \neq x_i$ for some i. Thus, in coloring c_i, vertices x and y have different colors and hence, there is a vertex $z \in G$ such that $z \in N[y] \triangle N[x]$, a contradiction which proves the first bound.

Let S be an optimal red-blue separating set for such a coloring $c \in \{c_1, \ldots, c_k\}$ of the vertices in G and let D be a minimum-size dominating set in G; $S \cup D$ is also a red-blue separating set for the coloring c. Thus, $\text{sep}(G) \leq |S| + |D|$. At most $\Delta(G)+1$ vertices of G may have the same closed neighborhood in D. This situation arises when these $(\Delta(G)+1)$ vertices form a clique. Now we demonstrate the maximum possible size of S. As stated earlier, we may again choose $\lceil \log_2(\Delta(G)+1) \rceil$ different colorings, as explained earlier, and the corresponding optimal separating set for each of these colorings. Now we construct a red-blue separating set S by taking the union of the red-blue separating sets of all $\lceil \log_2(\Delta(G)+1) \rceil$ colorings. Since each of these sets has size at most $\text{max-sep}_{\text{RB}}(G)$, hence we have $|S| \leq \lceil \log_2(\Delta(G)+1) \rceil \times \text{max-sep}_{\text{RB}}(G)$. Thus, we have $\text{sep}(G) \leq |S| + |D| = \lceil \log_2(\Delta(G)+1) \rceil \times \text{max-sep}_{\text{RB}}(G) + \gamma(G)$. □

We do not know whether the previous bound from Theorem 4.6 is reached, but as seen next, there are graphs G such that $\text{sep}(G) = 2\,\text{max-sep}_{\text{RB}}(G)$.

Proposition 4.5

Let $G = K_{k_1,\ldots,k_t}$ be a complete t-partite graph for $t \geq 2$, $k_i \geq 5$ where k_i is an odd number for each i. Then $\text{sep}(G) = n - t$ and $\text{max-sep}_{\text{RB}}(G) = (n-t)/2$.

Proof. Let us name each of the t parts of G as $G_1, \ldots, G_t$ with vertex sets $V(G_i) = \{v_1^i, \ldots, v_{k_i}^i\}$. Observe that a set $S \subseteq V = \cup_{i=1}^{t} V(G_i)$ is a separating set in G if and only if $|S \cap V(G_i)| \geq k_i - 1$ for each $i = 1, 2, \ldots, t$. Indeed, if we have $v_j^i, v_h^i \notin S$, then $N[v_j^i] \cap S = N[v_h^i] \cap S$ since $N(v_j^i) = N(v_h^i)$ as the graph G is complete t-partite. Moreover, each vertex in S is separated from vertices not in S and each vertex in $V(G_i)$ is separated from vertices in $V(G_j)$. Thus, we have $\text{sep}(G) = n - t$.

Let c be such a coloring of G that attains $\text{sep}_{\text{RB}}(G, c) = \text{max-sep}_{\text{RB}}(G)$. We now form a red-blue separating set S' for coloring c as follows:

For each part G_i, we choose in S', the vertices of the color class whose count is lesser in that particular part (however, we have to choose at least two vertices in S' from each part). As the difference between the vertex sets of the two color classes in a part will be at least 1 (k_i being odd), nearly half the vertices from a part might be chosen in S'. Hence, observe that $|S'| \leq \sum_{i=1}^{t} \frac{k_i - 1}{2} = (n-t)/2$.

Now, we can see that S' is a red-blue separating set due to the fact that if $v_j^i \notin S'$ and $v_h^i \notin S'$, then v_j^i and v_h^i will have the same color as S' consists of all the vertices of the smaller color class from G_i and no vertex from the larger color class of G_i. □

4.4.2 Upper bound for general graphs

We will use the following classical theorem in combinatorics to show that we can always spare one vertex in the solution of Max Red-Blue Separation.

Theorem 4.7: Bondy's Theorem [Bon72]

Let V be a set of size n with a family $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$ of n distinct subsets of V. There is a subset X of V of size $(n-1)$ such that the sets $\mathcal{A}_1 \cap X, \mathcal{A}_2 \cap X, \mathcal{A}_3 \cap X, \ldots, \mathcal{A}_n \cap X$ are still distinct.

Corollary 4.1

Let be G be a twin-free graph on n vertices. Then we have $\text{max-sep}_{\text{RB}}(G) \leq \text{sep}(G) \leq n-1$.

Proof. Regardless of the coloring, by Theorem 4.7 we can always find a set of size $n-1$ that separates *all* pairs of vertices. □

This bound is tight for every even n for complements of *half-graphs* (stated below) which are studied in the context of identifying codes in [FGK+11].

Half-graph [Erd06]

Definition 4.7. For any integer $k \geq 1$, the *half-graph* H_k is the bipartite graph on vertex sets $\{v_1, \ldots, v_k\}$ and $\{w_1, \ldots, w_k\}$, with an edge between every pair (v_i, w_j) of vertices satisfying $i \leq j$.
Thus, the complement $\overline{H_k}$ of H_k consists of two cliques $\{v_1, \ldots, v_k\}$ and $\{w_1, \ldots, w_k\}$ and with an edge between every pair (v_i, w_j) of vertices satisfying $i > j$.

Proposition 4.6

For every $k \geq 1$, we have $\text{max-sep}_{\text{RB}}(\overline{H_k}) = 2k-1$.

Proof. The upper bound (i.e. max-sep$_{\text{RB}}(\overline{H_k}) \leq 2k-1$) follows from Corollary 4.1.

To prove the lower bound, consider the red-blue coloring c such that v_i is blue whenever i is odd, and red otherwise. If k is odd, w_i is red whenever i is odd, and blue otherwise. If k is even, w_i is blue whenever i is odd, and red otherwise.

For any integer i between 1 and $k-1$, v_i and v_{i+1} have different colors and can only be separated by w_i. Likewise, w_i and w_{i+1} have different colors and can only be separated by v_{i+1}. This shows that $\{w_1, \ldots, w_{k-1}\}$ and $\{v_2, \ldots, v_k\}$ must belong to any separating set of $(\overline{H_k}, c)$. Finally, consider w_1 and v_k. They also have different colors and can only be separated by either v_1 or w_k. This shows that we need at least $n-1$ vertices in any separating set. Thus, max-sep$_{\text{RB}}(\overline{H_k}) \geq 2k-1$ □

4.4.3 Upper bound for trees

We will now show that a much better upper bound holds for trees.

In a tree T, the vertices having degree 1 are called *leaves* and the set of leaves of T is denoted as $L(T)$. Vertices adjacent to leaves are called *support vertices*, and the set of support vertices of T is denoted by $S(T)$. We denote by $\ell(T) = |L(T)|$ and $s(T) = |S(T)|$. The set of support vertices with exactly i adjacent leaves is denoted $S_i(T)$ and the set of leaves adjacent to support vertices in $S_i(T)$ is denoted $L_i(T)$. Observe that $|L_1(T)| = |S_1(T)|$. Moreover, let $L_+(T) = L(T) \setminus L_1(T)$ and $S_+(T) = S(T) \setminus S_1(T)$. We denote the sizes of these sets $S_i(T), L_i(T), S_+(T)$ and $L_+(T)$ by $s_i(T), \ell_i(T), s_+(T)$ and $\ell_+(T)$ respectively.

Theorem 4.8

Let T be a tree on $n \geq 3$ vertices and let c be a coloring with exactly one red (or blue) vertex. We have sep$_{\text{RB}}(T, c) \leq 2$.

Proof. If T is a tree with two vertices and exactly one of them is red then it has to be a forest (otherwise it will not be twin-free) and sep$_{\text{RB}}(T, c) = 1$. So, we

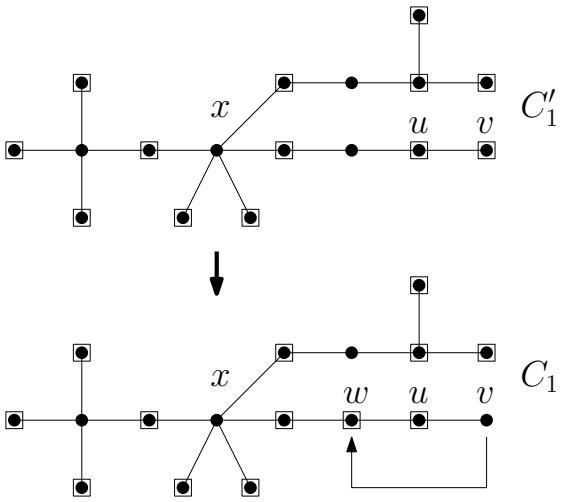


Figure 4.9: Construction of C_1 from C_1' where the highlighted elements represents members of the set.

assume that T is a tree with at least three vertices such that there is exactly one red vertex $v \in V(T)$.

Let us assume first that $v \notin L(T)$. Thus, v has at least two neighbors w and u. If we now include w and u in the separating set S, then v is the only vertex in T which has two adjacent vertices in S and hence, S is a red-blue separating set in T for coloring c.

On the other hand, if $v \in L(T)$, $u \in N(v)$ and $w \in N(u) \setminus \{v\}$, then $S = \{v, w\}$ is a red-blue separating set in T for coloring c. Thus, $\text{sep}_{\text{RB}}(T, c) \leq 2$. □

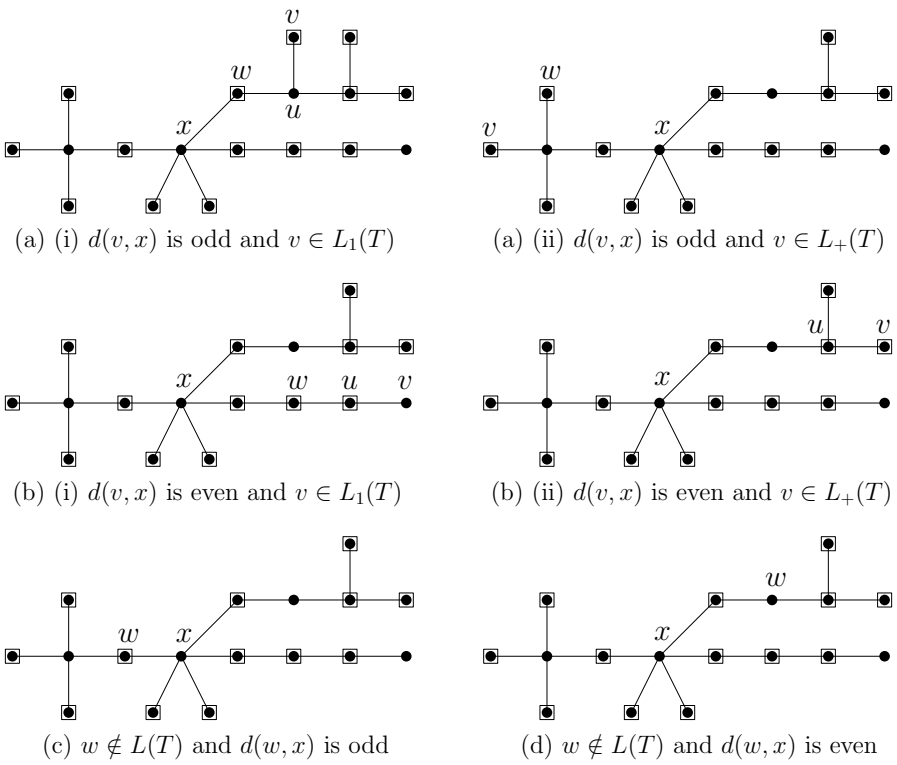
Theorem 4.8 says the lower bound on max-$\text{sep}_{\text{RB}}(T)$. Now we prove an upper bound on max-$\text{sep}_{\text{RB}}(T)$.

We build two separating sets C_1 and C_2 for T with $|C_1| \neq |C_2|$. We first create two sets C_1' and C_2' and obtain C_1 and C_2 from these sets respectively. We choose an arbitrary non-leaf vertex x.

Construction of C_1: We add to the first set C_1' every vertex at odd distance from x and every leaf of T. If there is a support vertex $u \in S_1(T) \cap C_1'$ and

an adjacent leaf $v \in L_1(T) \cap N(u)$, we create a separating set C_1 from C_1' which is obtained by replacing the leaf v with some vertex $w \in N(u) \setminus L(T)$. See Figure 4.9.

Construction of C_2: We add to the second set C_2' every vertex at even distance from x and every leaf of T. If there is a support vertex $u \in S_1(T) \cap C_2'$ and an adjacent leaf $v \in L_1(T) \cap N(u)$, we create a separating set C_2 from C_2' which is obtained by replacing the leaf v with some vertex $w \in N(u) \setminus L(T)$.



(a) (i) $d(v, x)$ is odd and $v \in L_1(T)$ (a) (ii) $d(v, x)$ is odd and $v \in L_+(T)$

(b) (i) $d(v, x)$ is even and $v \in L_1(T)$ (b) (ii) $d(v, x)$ is even and $v \in L_+(T)$

(c) $w \notin L(T)$ and $d(w, x)$ is odd (d) $w \notin L(T)$ and $d(w, x)$ is even

Figure 4.10: The cases of Claim 4.5.

Claim 4.5

Both C_1 and C_2 are separating sets with $|C_1| \neq |C_2|$.

Proof. Let us consider the set C_1. First, we show that each leaf $v \in L(T)$ is separated by C_1. Let $u \in N(v)$ be a support vertex adjacent to v. We know that

x is the chosen non-leaf vertex for creating C_1'. We show that v is separated from the other vertices of T through the following exhaustive cases. See Figure 4.10.

(a) If $d(x, v)$ is odd, then $v \in C_1$. Now we need to consider two cases depending on whether (i) $v \in L_1(T)$ and (ii) $v \in L_+(T)$. In the first case, the node $u \in S_1(T) \cap N(v)$ is at even distance from x and is not present in C_1. However, there exists a node $w \in N(u) \setminus \{v\}$ which is at odd distance from x and is present in C_1. In the second case, v has at least one other sibling w which is also at odd distance from x, and is present in C_1. Thus, in both cases $|N(u) \cap C_1| \geq 2$, where $u \in N(v)$. This indicates that the members in $N[u] \cap C_1 \neq N[v] \cap C_1$.

(b) If $d(v, x)$ is even, then (i) $v \in L_1(T)$, then $v \notin C_1$ by the rule of construction of C_1. However, $u \in N(v)$ and the member $w \in N(u) \setminus \{v\}$ is present in C_1 implying $|N[u] \cap C_1| \geq 2$ and (ii) $v \in L_+(T)$, then $v \in C_1$ and there exists at least two other members of $N(u)$ (they may or may not be leaves of T) in C_1, implying $|N[u] \cap C_1| \geq 3$.

Now, we show that each non-leaf $w \notin L(T)$ is separated by C_1.

(c) If $d(w, x)$ is odd, then $w \in C_1$. Moreover, each neighbor $u \in N(w)$ has even distance from x and is either a leaf which is separated from w or has at least two neighbors with odd distance from x and hence, $|C_1 \cap N(u)| \geq 2$ and u is separated from w.

(d) If $d(w, x)$ is even, then $w \notin C_1'$. Here at least one of the members in $N(w) \setminus \{u\}$ is in C_1. Thus w is separated from u.

Similarly it can be shown that the vertices in T are separated using the set C_2 as the separating set. Hence, the claim follows.

$|C_1| \neq |C_2|$ follows from the fact that the number of vertices at *odd* and *even* distance from $x \in T$ always differ. □

Let c be a coloring of T such that $\text{max-sep}_{\text{RB}}(T) = \text{sep}_{\text{RB}}(T, c)$. We assume that

$s(T) \geq 2$ which implies that T is not a star graph.

We now use C'_1 and C'_2 to create a red-blue separating set C. Let us denote by $NS_3(T)$ a smallest set of vertices in T such that for each vertex $v \in S_3(T)$ which has $N(v) \cap S_+(T) = \emptyset$, we have at least one vertex $u \in N(v) \setminus L(T)$ (say the predecessor of v in T) in $NS_3(T)$ (such a vertex will always exist as T is not a star).

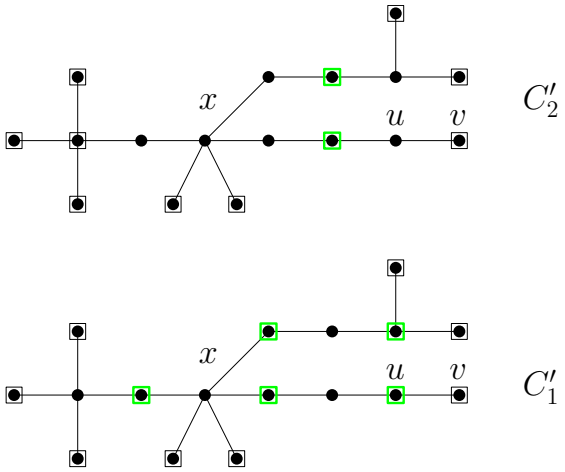


Figure 4.11: Comparison of C'_1 and C'_2 where the vertices highlighted in green belong to the set $V(T) \setminus (L(T) \cup S_+(T) \cup NS_3(T))$.

Let C'_a denote one of the the two sets C'_1 and C'_2 which contain lesser vertices from the set $V(T) \setminus (L(T) \cup S_+(T) \cup NS_3(T))$ (see Figure 4.11). In particular, it contains at most half of those vertices and we have $|C'_a \setminus (L(T) \cup S_+(T) \cup NS_3(T))| \leq (n - \ell(T) - s_+(T) - |NS_3(T)|)/2$. Now, we will construct a red-blue separating set C from C'_a. Initialize C by C'_a. First, for each support vertex $u \in S_+(T)$, remove from C every leaf $w \in L_+(T) \cap N(u)$ where w is in the larger color class with respect to coloring c among the vertices in $N(u) \cap L_+(T)$. Now, we add some more vertices to C as follows.

- For all $u \in S_i(T)$, $i \geq 4$, we add u and some leaves to C such that there are at least two vertices in $N(u) \cap C$. We have at most $|N[u] \cap L(T)|/2 + 1$ vertices from $(N[u] \cap L(T)) \cup \{u\}$ in the set C.

- For $i = 3$ i.e., $u \in S_3(T)$, we add to C either u or any vertex $v \in NS_3(T) \cap N(u)$, depending on which one does not already belong to C. Then, if all leaves in $N(u)$ have the same color and had been deleted from C at the beginning of the construction, we add one of them to C. After this, we have exactly three vertices in $C \cap N[u] \cap (L(T) \cup \{u\} \cup \{v\})$.

- Finally, for $i = 2$ i.e., $u \in S_2(T)$[4], if the two leaves adjacent to u have the same color and $u \notin C'_a$, we add u and one of its two leaves to C. If the two leaves have the same color and $u \in C'_a$, we add a non-leaf neighbor of u to C. If the leaves have different colors, one of them, say u', has the same color as u. We add u to C and to separate the leaves from u, we put u' in C. We have added at most two vertices to C in this case.

Thus, each time we added to C at most half the considered vertices in $N(u)$, and at most one additional vertex. After these changes, we might have to replace some vertices from $L_1(T)$ in C, similar to the way we have built C_a from C'_a.

Claim 4.6

$|C| \leq \frac{n+s(T)}{2}$.

Proof. As we know,

$$|C'_a \setminus (L(T) \cup S_+(T) \cup NS_3(T))| \leq (n - \ell(T) - s_+(T) - |NS_3(T)|)/2,$$

we get:

$$\begin{aligned} |C| &\leq \frac{n - \ell(T) - s_+(T) - |NS_3(T)|}{2} + \ell_1(T) + \frac{\ell_+(T) + |NS_3(T)|}{2} + s_+(T) \\ &= \frac{n + \ell_1(T) + s_+(T)}{2} = \frac{n + s(T)}{2}. \end{aligned}$$

□

[4]The set of support vertices with exactly 2 adjacent leaves is denoted $S_2(T)$

Claim 4.7

C is a red-blue separating set for the coloring c.

Proof. Since C_a is a separating set and $C_a \setminus C \subseteq L_+(T)$, if two vertices w, v are not separated by C, then they were separated by a leaf in $L_+(T)$ in C_a. Moreover, there exists a support vertex $u \in S_+(T)$ such that $v, w \in N[u]$. Recall that $S_+(T) \subseteq C$ and hence, $u \in C$. In the following, we go through all possibilities for w and v.

When $w, v \in L(T)$: If $w, v \in L(T)$, then they have the same color and do not need to be separated.

When $w, v \notin L(T)$ and $d(w, v) = 1$: Observe that either $w = u$ or $v = u$. Otherwise, we have triangle wuv. Assume that $w = u$. Notice that if any of the leaves in $N(w) \cap L(T)$ are in C, then w and v are separated. Since $u \in S_+(T)$ and how we construct C, we have $u \in S_2(T)$. The leaves adjacent to u have the same color, and $u \in C'_a$. Hence, $u \in C$. Since there are no cycles in T, $d(w, x)$ and $d(v, x)$ have different parities[5]. Thus, $v \notin C'_a$. As $v \notin L(T)$, there exists a vertex $b \in N(v) \setminus \{u\}$ and $d(w, x)$ has the same parity as $d(b, x)$. Thus, $b \in C'_a$. Since $v \notin C'_a$, we have $b \in C$ unless $v \in S_+(T)$, $b \in L_+(T)$ and we removed b from C'_a when we constructed C. However, in the same way as we concluded that $u \in S_2(T) \cap C'_a$, we observe that $v \in C'_a$, a contradiction. Thus, C separates v and w.

When one of w or v in $L(T)$: Let us say $v \in L(T)$ and since $v \in N(u)$, we have $v \in L_+(T)$. Assume first that $d(v, w) = 2$. Thus, $v, w \notin C$. However, then we again have $b \in C$ such that $b \in N(w) \setminus \{u\}$, either due to the parity of $d(x, b)$ or because b is a leaf. As the last case we have $u = w$ and $v \in L_+(T) \cap N(u)$. If $u \notin C'_a$, then, by parity, u has an adjacent non-leaf vertex in C since T is not a star. On the other hand, if $u \in C'_a$, then if $u \in S_2(T)$ and the two leaves have the same color, there is again a non-leaf neighbor in C. If $u \in S_2(T)$ and the two leaves have different colors, then

[5]Parity refers to being even or odd with respect to the distance in between the vertices.

there is a leaf of the same color, in $N(u)$, as u which is in C. If $u \in S_3(T)$, then u has a non-leaf neighbor in $NS_3(T) \cap C$ or in $S_+(T) \cap C$. If $u \in S_i(T)$ for $i \geq 4$, then u has at least two adjacent leaves which are in C. Hence, w and v are either separated or they have the same color.

□

Theorem 4.9

For any tree T of order $n \geq 5$, we have $\text{max-sep}_{\text{RB}}(T) \leq \frac{n+s(T)}{2}$.

Proof. Follows from Claim 4.5, Claim 4.6 and Claim 4.7. Observe that the claim holds when T is a star. Here, we select the leaves of the smaller color class, and at least two other leaves from the larger color class to handle all possible situations enlisted: (i) all vertices of same color, (ii) all leaves of same color and the root of different color, (iii) leaves are of different color and the color of the root is of the smaller color class and (iv) leaves are of different color and the color of the root is of the larger color class. In case (i) no vertex needs to be chosen in C. In case (ii) any two leaves in C will suffice. In case (iii) all the leaves of the smaller color class is enough. In case (iv) all the leaves of the smaller color class is enough if the size of the smaller color class is greater than one, otherwise one more leaf needs to be chosen in C which will belong to the larger color class. □

The upper bound of Theorem 4.9 is tight. Consider, for example, a path on eight vertices of alternating colors. Also, the trees presented in Proposition 4.7 (stated later in this chapter) are within a factor of 1/2 of this upper bound. In fact, the bound of the theorem is tight for all even paths except for P_6 (and P_2). In the following theorem, we offer another upper bound for trees which is useful when the number of support vertices is large.

Theorem 4.10

For any tree T of order $n \geq 5$, $\text{sep}(T) \leq n - s(T)$.

Proof. Let us choose for each support vertex $u \in S(T)$ exactly one adjacent leaf $v \in L(T)$ and say that these vertices form the set S'. Next, we form the separating set $S = V(T) \setminus S'$. Notice that $|S| = n - s(T)$. In the following, we show that S is a separating set in T.

Observe that if $v \notin S$, then v is a leaf and no support vertex has two adjacent leaves which do not belong to S. Thus, vertices which do not belong to S are pairwise separated. Since $S' \subseteq L(T)$, $T[S]$ is a connected induced subgraph of T due to the fact that $T[S]$ is missing only a few leaves of T. $V(T[S])$ forms a (non-optimal) separating set of $T[S]$. Moreover, as $n \geq 5$, we have $|N[w] \cap S| \geq 2$ for each vertex $w \in S$. Thus, vertices in S are separated from vertices which are not in S. Finally, any two vertices $w, w' \in S$ are separated since $|V(T[S])| \geq 3$; hence, each closed neighborhood is unique in $T[S]$. □

The following corollary is a direct consequence of Theorems 4.9 and 4.10. These two bounds will be equal when $s(T) = \frac{n}{3}$.

Corollary 4.2

For any tree T of order $n \geq 5$, we have $\text{max-sep}_{\text{RB}}(T) \leq \frac{2n}{3}$.

We next show that Corollary 4.2 (and Theorem 4.9) is not far from tight.

Proposition 4.7

For any $k \geq 1$, there is a tree T of order $n = 5k + 1$ with $\text{max-sep}_{\text{RB}}(T) = \frac{3(n-1)}{5} = \frac{n+s(T)-1}{2}$.

Proof. Consider the tree T formed by taking k disjoint copies $P^1, \ldots, P^k$ of a path of order 6 and identifying one endpoint of each path into one single vertex x. Intuitively we mean that we take k copies of P_6 which have one end in common. We consider the coloring that where x is colored red, and all other vertices are colored red-blue-red... following the bipartition of the tree. Let $v_1^i, \ldots, v_5^i$ be the vertices of P^i distinct from x, where x is adjacent to v_1^i. In order to separate v_5^i

from v_4^i, we need v_3^i in any red-blue separating set. To separate v_4^i from v_3^i, we need either v_2^i or v_5^i. To separate v_3^i from v_2^i, we need either v_1^i or v_4^i. Thus, we need at least three vertices (v_1^i, v_3^i, v_5^i separates all pairs of vertices) of P^i in any red-blue separating set, which shows that max-sep$_{\text{RB}}(T) \geq 3k$. Finally, since $s(T) = k$ and $n - 1 = 5k$, we have $3k = \frac{3(n-1)}{5} = \frac{n+s(T)-1}{2}$. □

4.5 Complexity of Max Red-Blue Separation

The problem Max Red-Blue Separation does not seem to be naturally in the class NP (it is in the second level of the polynomial hierarchy). Nevertheless, we show that it is NP-hard by reduction from a special version of 3-SAT [Tov84].

Problem: 3-SAT-2L

Input: A set of m clauses $C = \{c_1, \ldots, c_m\}$ each with at most three literals, over n Boolean variables $X = \{x_1, \ldots, x_n\}$, and each literal appears at most twice.
Output: Is there an assignment of X where each clause is satisfied?

Theorem 4.11

Max Red-Blue Separation is NP-hard even for graphs of maximum degree 12.

Proof. To show hardness we reduce from the 3-SAT-2L problem. Given an instance σ of 3-SAT-2L with m clauses and n variables, we create an instance (G, k) of Max Red-Blue Separation as follows.

We create a domination gadget and use it to construct the variable and clause gadgets required for this reduction. First let us explain the construction of a domination gadget and its properties. A *domination gadget* is a graph as shown in Figure 4.12(a). Here, we have 16 vertices, named $v_1, v_2, u_1, \ldots, u_4, p_1, \ldots, p_6, q_1, \ldots, q_4$.

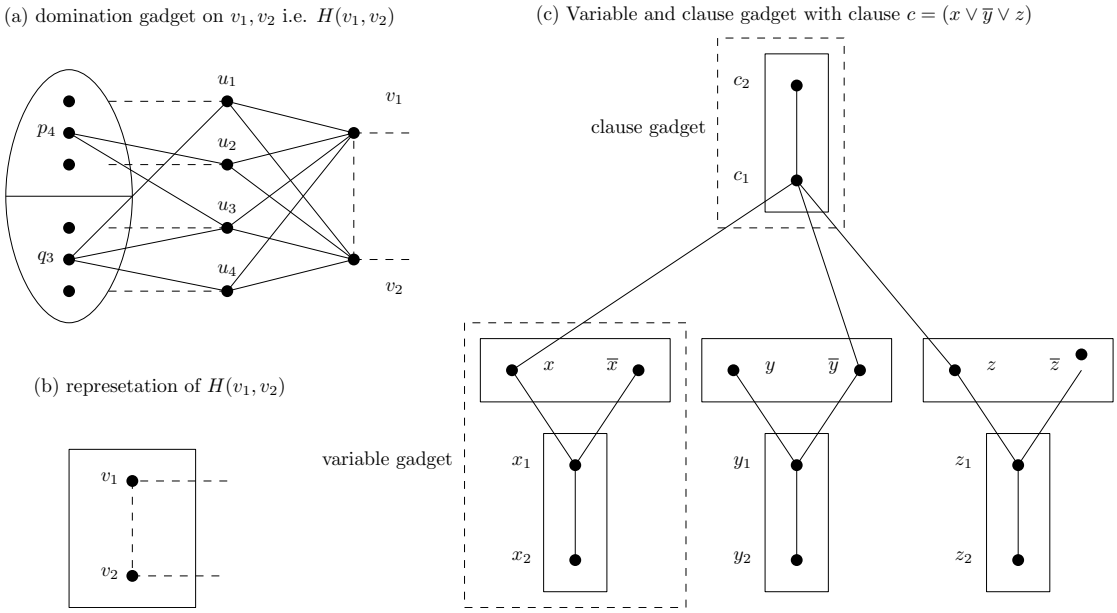


Figure 4.12: Gadgets used for the reduction from 3-SAT-2L to Max Red-Blue Separation.

The edges are drawn as follows. The vertices v_1 and v_2 may be connected to each other and/or to some other vertices which is represented by the dashed edges. Both v_1 and v_2 are also connected to the vertices u_1, u_2, u_3 and u_4 as shown in Figure 4.12(a) using solid lines. Next we have a clique K_{10} consisting of the vertices $\{p_1, \ldots, p_6, q_1, \ldots, q_4\}$. Every vertex p_i is connected to a unique pair of vertices from $\{u_1, u_2, u_3, u_4\}$ and every vertex q_j is connected to a unique triple of vertices from $\{u_1, u_2, u_3, u_4\}$. For example in the figure we have p_4 connected with the pair of vertices u_2 and u_3 (using solid edges) and q_3 connected with the triplet of vertices u_1, u_3 and u_4 (using solid edges). From now onwards this graph corresponding to a domination gadget will be referred to as $H(v_1, v_2)$ and represented as shown in Figure 4.12(b). $H(v_1, v_2)$ is a part of G and is connected with the rest of the graph G through the vertices v_1 and v_2.

The *variable gadget* for a variable x of the given 3-SAT-2L instance consists of the graph $H(x_1, x_2)$ and $H(x, \overline{x})$ with additional edges $(x_1, x_2), (x_1, x)$ and $(x_1, \overline{x})$. If x_1 and x_2 are colored differently, then either x or $\overline{x}$ needs to be in the red-blue separating set S. Selecting at least one of x or $\overline{x}$ also separates x and $\overline{x}$ themselves.

The *clause gadget* for a clause $c = (x \vee \overline{y} \vee z)$ is $H(c_1, c_2)$, where c_1 is connected to the vertices $x, \overline{y}$ and z from the variable gadgets $H(x_1, x_2)$, $H(y_1, y_2)$ and $H(z_1, z_2)$ and also to c_2. If c_1 and c_2 are colored differently, then S should contain at least one of $x, \overline{y}$ or z in order to separate them. The variable gadget and clause gadget are demonstrated in Figure 4.12(c).

In Figure 4.13, we illustrate the coloring of vertices in the various gadgets used in a 3-SAT-2L expression, using an example instance demonstrating the reduction from 3-SAT-2L to Max Red-Blue Separation. In this example we have the formula as $(\overline{w} \vee \overline{x} \vee y) \wedge (x \vee \overline{y} \vee z)$. Thus, we have two clauses $C_1 = \overline{w} \vee \overline{x} \vee y$ and $C_2 = x \vee \overline{y} \vee z$ and four variables w, x, y and z. Corresponding to each clause and each variable we have a clause gadget and a variable gadget respectively (as described earlier). The structure of the domination gadget attached with both the clause and variable gadgets has been shown separately to avoid making the Figure 4.13 too messy. We have shown one possible coloring in this instance to maximize the Red-Blue Separation. There are other possible colorings as well.

Say a satisfying assignment for the formula is when $w = 0, x = 1, y = 0$ and $z = 1$. Now according to the value of a variable v, either the vertex corresponding to v (if $v = 1$) or $\overline{v}$ (if $v = 0$) is included in the red-blue separating set S. The vertices u_1, u_2, u_3 and u_4 of all the domination gadgets are always required to be a part of the S. Thus in the instance shown below, along with the vertices of type u_i, the vertices $\overline{w}, x, \overline{y}$ and z are also a part of S, are squared in the figure.

We define a worst-coloring of G as a {red, blue} color assignment to the vertices of G for which $\text{sep}_{\text{RB}}(G, c) = \text{max-sep}_{\text{RB}}(G)$. We make the following claims that proves the theorem. □

Claim 4.8

For any worst-coloring c of G the optimal red-blue separating code of (G, c) will always contains the vertices u_1, u_2, u_3 and u_4.

Proof. Since the subgraph consisting of the vertices $\{p_i, i = 1, \ldots, 6\} \cup \{q_i, i =$

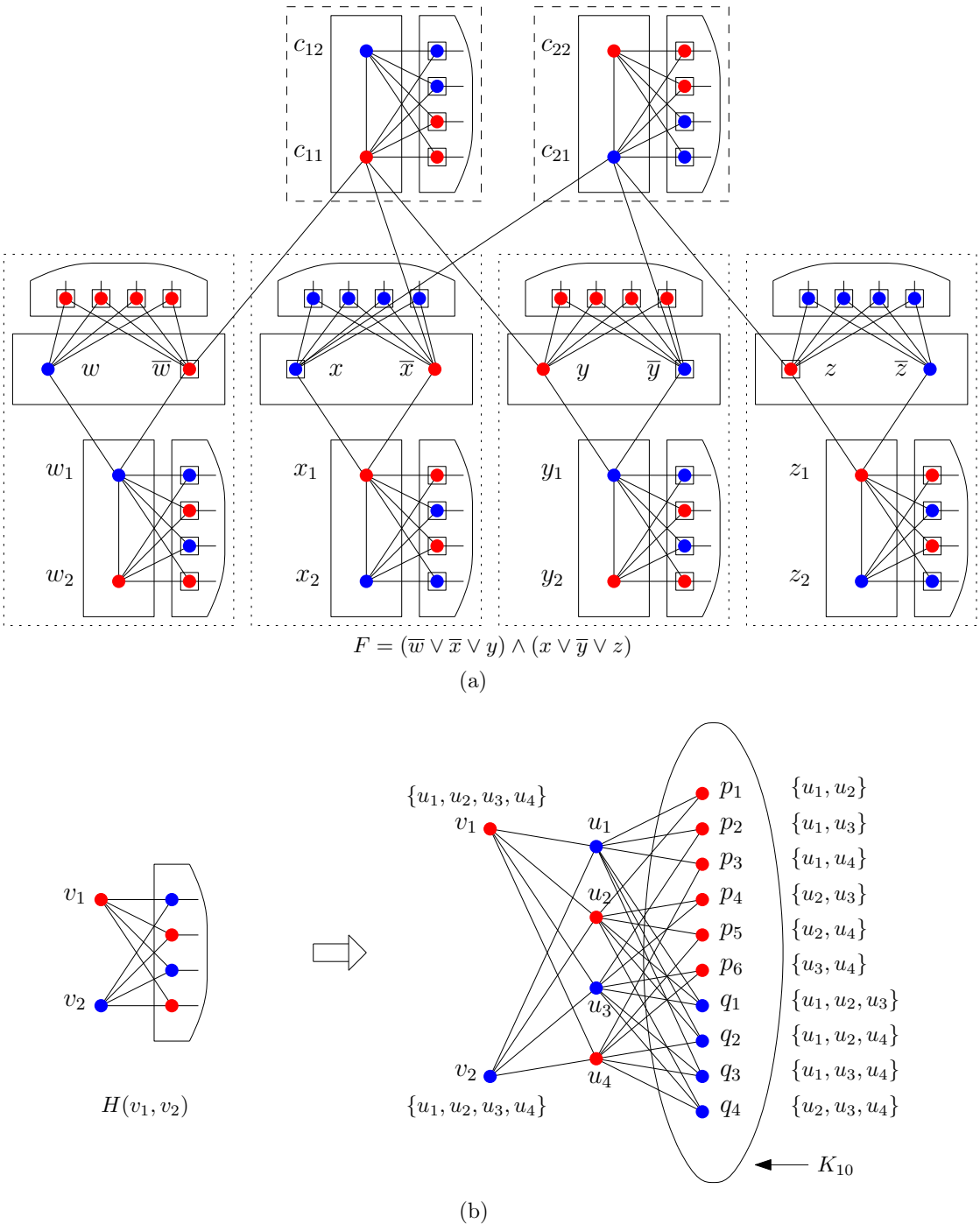


Figure 4.13: An example to show the reduction from 3-SAT-2L to MAX RED-BLUE SEPARATION where (a) Dotted rectangles are variable gadgets and dashed rectangles are clause gadgets and (b) Illustration of a domination gadget.

$1, \ldots, 4\}$ is a clique (denoted by K_{10}), its vertices can be separated amongst themselves only by taking the vertices in $\{u_1, u_2, u_3, u_4\}$ in the red-blue separating set, S as explained below. Now consider a red-blue coloring of a domination gadget $H(v_1, v_2)$ where all the p_i's are colored red and the q_j's colored blue. The coloring of the rest of the vertices are not relevant. Observe that for every vertex $u \in \{u_1, u_2, u_3, u_4\}$, there exists a p_i and a q_j such that $N[p_i] \triangle N[q_j] = \{u\}$. Since, p_i and q_j are colored differently, x must be chosen in the code in order to separate them. Therefore there exist red-blue colorings of G such that it is necessary to take all the vertices in $\{u_1, u_2, u_3, u_4\}$ in S of G with respect to those coloring. Thus, the claim follows. □

Claim 4.9

σ is satisfiable if and only if $\text{max-sep}_{\text{RB}}(G) \leq 4m + 9n$.

Proof. ($\Longrightarrow$) Let σ be satisfiable. Consider any worst-coloring $\hat{c}$ of G. For every variable x of the given 3-SAT-2L expression, if x_1 and x_2 have the same color in $\hat{c}$ then we change the color of x_2. We similarly change the color of c_2 in a clause gadget if the vertices c_1 and c_2 in G corresponding to that clause are of same color in $\hat{c}$. These two steps do not increase the size of S (since we had a worst-coloring), but since all the pair of vertices x_1 and x_2 in each variable gadget are now colored differently, therefore either x or $\overline{x}$ needs to be in S. And since σ is satisfiable, therefore, choosing the truth assignment of σ as in S along with the vertices u_1, u_2, u_3 and u_4 of each domination gadget separates all red-blue vertex pairs. If the number of variables is n and the number of clauses is m, then the number of domination gadgets is $m + 2n$. Therefore, the total size of S is $4m + 9n$ and our claim holds.

($\Longleftarrow$) For the reverse direction, assume that there is a coloring of G for which the size of S is $4m + 9n$. Again we change the coloring of x_2 and c_2 for all variables x and all clauses c as before. The size of S need not have to decrease for this change. Each variable has at most one of x or $\overline{x}$ chosen in S. Since $4m + 8n$ vertices are already chosen corresponding to the u_i's in all the domination gadgets, each

variable gadget can have exactly one of x or $\overline{x}$ chosen[6]. □

We can use Theorem 4.6 and a reduction to Set Cover to show the following.

Theorem 4.12

Max Red-Blue Separation can be approximated within a factor of $O((\ln n)^2)$ on graphs of order n in polynomial time.

Proof. Let A be a polynomial-time $(2\ln n + 1)$-approximation algorithm for the Separation problem the uncolored version of the graph G [GKM08]. For any graph G, let $S(G)$ denote the separating set returned by the algorithm A on the input graph G. Using Theorem 4.6, we have

$$|S(G)| \leq (2\ln n + 1) \cdot \mathrm{sep}(G) \leq (2\ln n + 1) \cdot \lceil \log_2 n \rceil \cdot \text{max-sep}_{\mathrm{RB}}(G).$$

Hence, algorithm A is a polynomial-time $O((\ln n)^2)$-approximation algorithm for Max Red-Blue Separation. □

[6]The size of the S shown in Figure 4.13 is $4m + 8n + n = 4 \times 2 + 8 \times 4 + 4 = 44$.

CHAPTER 5

Minimum Consistent Subset in Simple Graphs

Contents

5.1 Organization

In this chapter, we will study a graph-theoretic version of the minimum consistent subset (MCS) problem. Here, the distance between a pair of vertices u and v is the number of vertices in the shortest path from u to v, and will be referred to as hop-distance(u, v).

Problem: MINIMUM CONSISTENT SUBSET (MCS)

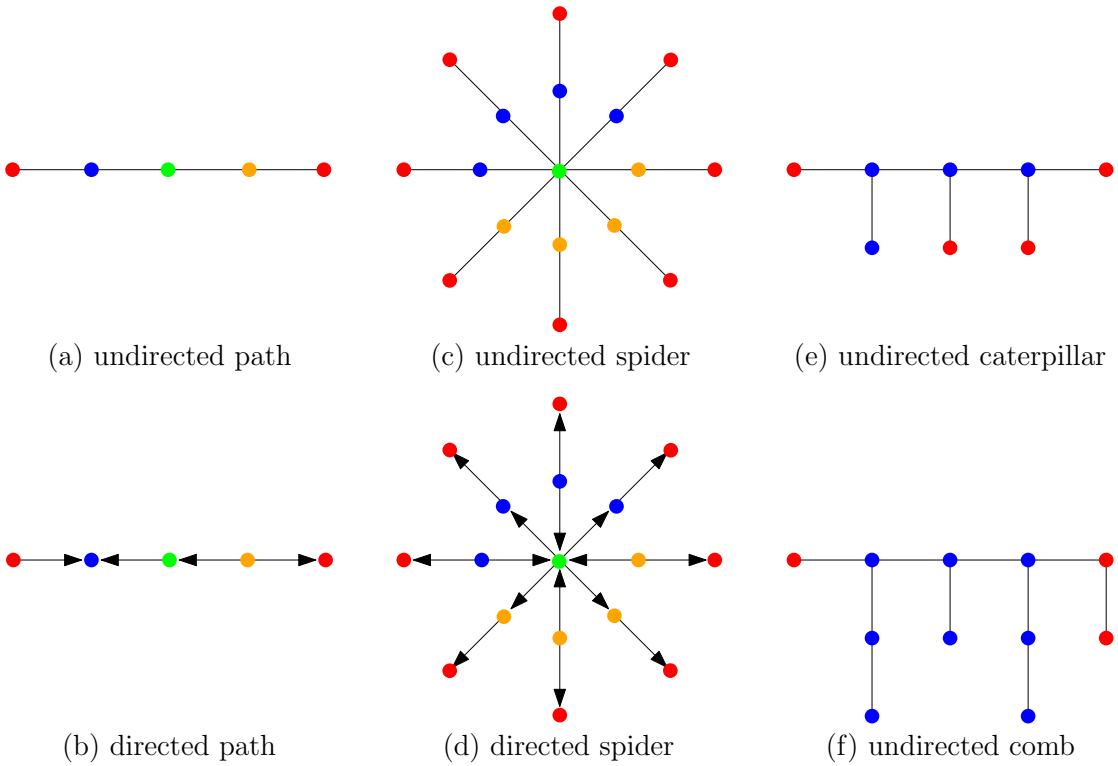
Input: A graph $G = (V, E)$ whose vertices are partitioned into k classes (i.e., colors), namely $V_1, V_2, \ldots, V_k$.

Output: Subsets $V_i' \subseteq V_i$, $i = 1, 2, \ldots, k$ such that for every member $v \in V$, if $v \in V_i$ then among its nearest neighbors in $\cup_{j=1}^{k} V_j'$ there is a vertex of V_i', and $\sum_{j=1}^{k} |V_k'|$ is minimum.

We first study the MCS problem for some simple graph classes, namely, (i) path, (ii) spider, (iii) caterpillar and (iv) comb. In the next chapter, we consider the problem of computing MCS for the more general problem where the given graph is a bi-colored undirected tree.

We introduce the concept of *run*, *gate* and *block* to partition the graph into subgraphs so that each subgraph can be handled independently with limited interactions with its 'neighboring' subgraphs. The non-trivial part of this approach using these structures is to carefully design techniques to handle the limited interaction. The basic idea is to use these structures from the given graph G to create a new graph H, called *overlay graph*. In doing so, we reduce the MCS problem on G to finding a shortest s-t path in this new graph H. The k-chromatic version for undirected paths and undirected spiders can be solved in $O(n)$ and $O(nm)$ time, respectively, where n is the number of vertices of that graph and m is the number of legs of the spider. We have also handled directed paths and directed spiders where the underlying undirected graph is a path or a spider respectively. The k-chromatic version for directed paths and directed spiders can be solved in

linear time. For the caterpillar graph and comb graph we can handle only the bi-chromatic and undirected version of the problem. The time complexity of these two algorithms are $O(n)$ and $O(n^2)$ respectively. In Figure 5.1 we show all the graphs handled in this chapter.



(a) undirected path (c) undirected spider (e) undirected caterpillar

(b) directed path (d) directed spider (f) undirected comb

Figure 5.1: The graphs considered in this chapter.

See Table 5.1 for a compilation of all the results presented in this chapter. In the rest of the chapter, we will use $\mathcal{C}$ to denote a MCS of the input graph $G = (V, E)$.

5.2 Path Graph

In a path graph or a linear graph $G = (V, E)$, the vertices in V are listed in the order $v_1, v_2, \ldots, v_n$, and each pair of consecutive vertices define an edge of the graph, i.e., $E = \{(v_i, v_{i+1}), i = 1, 2, \ldots, n-1\}$. Here each vertex has degree

Graphs		
k-chromatic Path	Undirected	$O(n)$ [Theorem 5.1]
	Directed	$O(n)$ [Theorem 5.2]
k-chromatic Spider	Undirected	$O(nm)$ [Corollary 5.1]
	Directed	$O(n)$ [Theorem 5.4]
bi-chromatic Caterpillar	Undirected	$O(n)$ [Theorem 5.5]
bi-chromatic Comb	Undirected	$O(n^2)$ [Theorem 5.6]

Table 5.1: Summary of results in the chapter.

2 excepting the two terminal vertices of the path which have degree 1. In a k-chromatic path graph, each vertex is assigned with one color in $\{c_1, c_2, \ldots, c_k\}$. We consider both the undirected and directed versions of the path graph.

5.2.1 Undirected Paths

We first handle the problem of computing the MCS of a path graph G where the edges of the graph are undirected.

5.1: Run

A run is a consecutive set of vertices of same color on the path (see Figure 5.2).

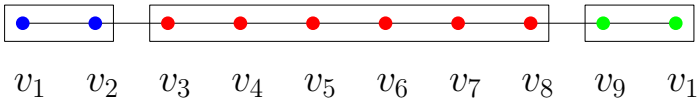


Figure 5.2: Runs in a path graph: each run is indicated by a black rectangle.

Lemma 5.1

In the minimum consistent subset of a path graph, each run will have at least one and at most two representatives. Moreover, exactly one vertex will be sufficient from the first and the last run.

Proof. The presence of at least one element from each run in $\mathcal{C}$ is obvious. It is also obvious that if more than two elements from a run R are chosen in $\mathcal{C}$ then all the chosen elements in that run excepting the first and last one can be dropped without violating the consistency, and thereby reducing the size of $\mathcal{C}$. By the same argument if more than one element from the first (resp. last) run is chosen in $\mathcal{C}$ then all those chosen elements excepting the rightmost in the first run (resp. leftmost in the last run) can also be dropped without violating the consistency. □

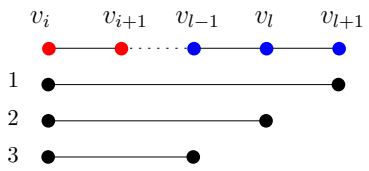


Figure 5.3: Valid Pairs: (v_i, v_{l-1}), (v_i, v_l) and (v_i, v_{l+1}).

Consider a pair of adjacent runs R_j and R_{j+1}, assume without loss of generality $|R_j| \leq |R_{j+1}|$. For each member $v_i \in R_j$, there exists at most three members, say $v_{l-1}, v_l, v_{l+1} \in R_{j+1}$ such that if v_i is included in $\mathcal{C}$ then any one of those three members of R_{j+1} must be included in $\mathcal{C}$ to satisfy the consistency property (with respect to hop-distance) of the boundary vertices of R_j and R_{j+1} that are adjacent to each other. Thus, $(v_i, v_{l+\theta})$ forms a *valid-pair*, where θ is chosen such that the number of red and blue vertices between v_i and v_l are same, and $\theta = -1, 0, 1$ (Figure 5.3).

We define the *overlay graph* $H = (V \cup \mathcal{D}, \mathcal{F})$ as follows. The vertices of H are the vertices of G, and r dummy vertices $\mathcal{D} = \{d_1, \ldots d_r\}$, where r is the number of runs. The edges in the set $\mathcal{F}$ are of two types. For each valid-pair, we put a directed *type-1* edge in $\mathcal{F}$, see Figure 5.4(a). For each vertex v_i in a run R_j we put two directed *type-2* edges (v_i, d_j) and (d_j, v_i) in $\mathcal{F}$, see Figure 5.4(b). The weight

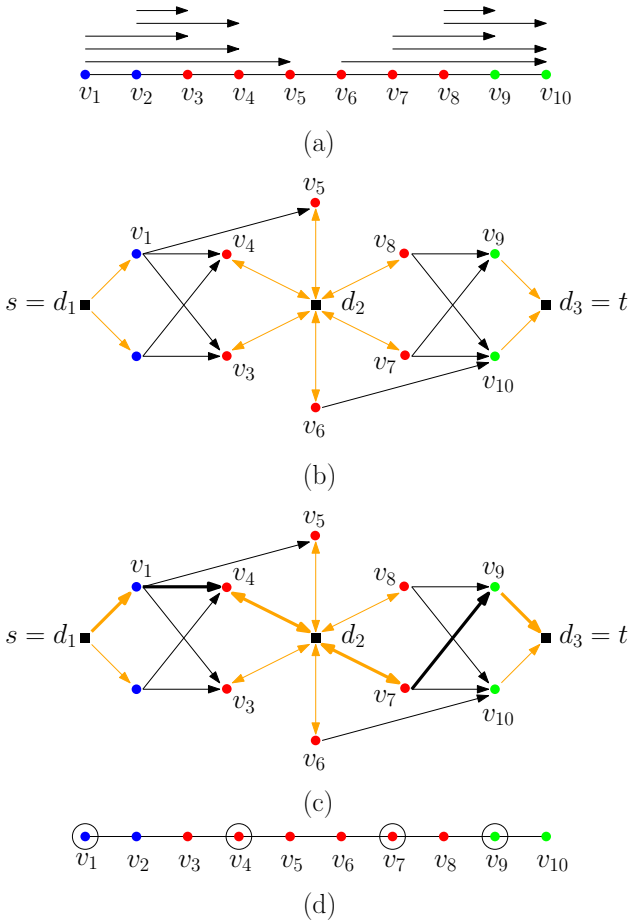


Figure 5.4: The graph H with type-1 and type-2 edges where (a) type-1 edges, (b) type-2 edges are in orange, (c) the shortest path is highlighted, and (d) the MCS $\mathcal{C}$ are circled in the path G.

of each *type-1* edge is 0. The *type-2* edges incident to $\mathcal{D} \setminus \{d_1, d_r\}$ have weight 1, and each *type-2* edge incident to d_1 and d_r has weight 0. For the complete demonstration of the graph H, see Figure 5.4(c). A *forward* s-t path is a path from s to t where the indices of the vertices along the s-t path appear in increasing order. Now we find the shortest forward s-t path with $s = d_1$ and $t = d_r$ in the graph H. Next we remove the d_j's from the obtained path to get the MCS of the given path graph G.

Theorem 5.1

The shortest s-t path of the *overlay graph* H gives the minimum consistent subset of the path G, and it executes in $O(n)$ time.

Proof. We first prove that any forward s-t path of the graph H constructed from the given path graph G gives a consistent subset of the graph G. Observe that at least one vertex from each run is present in the s-t path of the graph H. The reason is that the edges corresponding to the valid pairs are defined only between adjacent runs in forward direction, and each dummy vertex (say d_i) is bidirectionally connected with only the vertices of one run in the graph H. Each edge (v_i, v_j) of H between two consecutive runs in G justify the consistency of the vertices $\{v_i, v_{i+1}, \ldots, v_j\}$ of the graph G. As all the vertices in any path of H between a pair of vertices v_i, v_j in the same run are of same color, choice of those vertices on the path in the consistent subset does not destroy the consistency property of the not chosen vertices between v_i and v_j. Thus, any s-t path with $s = d_1$ and $t = d_r$ in H gives a consistent subset of the vertices in the path graph G.

Now, we will consider the nature of the minimum s-t path in H. Each forward move in the minimum s-t path from a vertex $v_i \in R_j$ either reaches a vertex $v_\ell \in R_{j+1}$ or to a vertex $v_m \in R_j$ through the dummy vertex d_j where $i < m$. If no dummy vertex in $\{d_2, \ldots d_{k-1}\}$ is visited in the s-t path then exactly one vertex is present from each run in the obtained s-t path of the graph H. However, the presence of every dummy vertex d_i in the shortest s-t path implies that two vertices of the corresponding run is present in the consistent subset obtained by that s-t path.

Also, the construction of the graph suggests that if a subset of vertices in $V \cup \mathcal{D}$ do not form a s-t path, then they cannot form a consistent subset. The minimality in the size of the consistent subset is justified from the choice of the shortest s-t path.

The number of *type-1* edges in the graph H is at most $3n$ since each vertex in a run G can participate in at most 3 *valid pairs* in its succeeding run. The number of *type-2* edges is $2n$ since each vertex in G is bidirectionally connected with the dummy vertex of its corresponding run. Thus we have $O(n)$ edges in H. In the special case of integer weights and directed connected graphs, Dijkstra's algorithm for shortest s-t path executes in $O(|E|)$ time ([Tho99]). Thus, the time complexity follows. □

5.2.2 Directed Paths

We are given a directed path graph $G = (V, E)$ where V is the set of vertices, where each vertex is assigned one of the colors in $\{c_1, c_2 \ldots, c_k\}$, and E is the set of edges where each edge $(v_i, v_{i+1}) \in E$ has a direction (i.e., $v_i \leftarrow v_{i+1}$ or $v_i \rightarrow v_{i+1}$) (see Figure 5.5). The MCS $\mathcal{C} \subseteq V$ for G is the subset $\mathcal{C} \subseteq V$ of minimum cardinality such that for a pair of consecutive vertices[1] $c, c' \in \mathcal{C}$ with $color(c) \neq color(c')$, we have $\forall v \in V$ between c and c' along the path with $color(v) = color(c)$ (resp, $color(v) = color(c')$), $distance(v, c) \leq distance(v, c')$ (resp. $distance(v, c') \leq distance(v, c)$)[2], and there is no vertex of color different from that of c and c' in the path segment from c to c' in G.

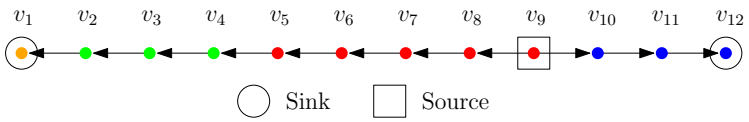


Figure 5.5: Illustration of a directed path with source and sink highlighted.

5.2: Source

A vertex in the directed graph G is said to be a **source** vertex, if only outgoing edge(s) are incident to it.

[1] By a pair of consecutive vertices in $\mathcal{C}$, we mean that in the directed path, while scanning the vertices from left to right, c is the vertex seen just before c' in $\mathcal{C}$.

[2] Here $distance(a, b)$ means the minimum number of edges in directed paths that connect a to b. If there is no directed path from a to b, we define $distance(a, b) = \infty$.

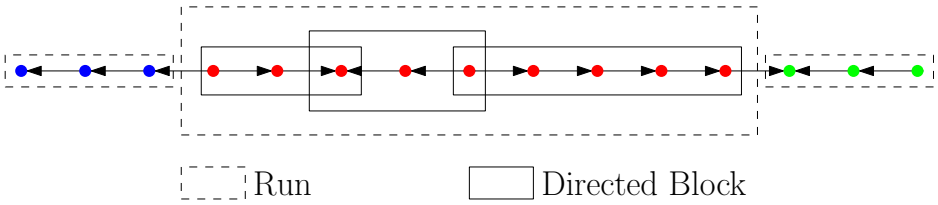


Figure 5.6: Illustration of run and directed block.

5.3: Sink

A vertex in the directed graph G is said to be a **sink** vertex, if only incoming edge(s) are incident to it.

Observation 5.1

All the sink vertices in G are in $\mathcal{C}$.

Proof. Follows from the fact that a sink vertex does not have any outgoing edge, and hence to make it consistent, it must thus be included in $\mathcal{C}$. □

Observation 5.2

Exactly one source vertex exists between a pair of consecutive sink vertices s_i and s_{i+1} in G. We use $\hat{s}_i$ to denote the source vertex between s_i and s_{i+1}.

As in the earlier section, a *run* is a set of consecutive vertices of same color along the path graph G.

A **directed block** in the graph G is a maximal directed path in a run (see Figure 5.6).

We partition the graph G using the sink vertices. In the left to right ordering of the vertices in G, every pair of consecutive sink vertices s_i and s_{i+1} defines a subgraph G_i. Thus, $G = G_0 \cup G_1 \cup \ldots \cup G_m$ where m is the total number of sinks in G. By Observation 5.1, other than the terminal vertices (s_i, s_{i+1}), the other members (if any) in the consistent subset of each subgraph G_i are disjoint. Let $\mathcal{C}_i$ denote the MCS of G_i.

Observation 5.3

The minimum consistent subset of each subgraph G_i can be computed independently and $\mathcal{C} = \mathcal{C}_0 \cup \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_m$ (See Figure 5.7).

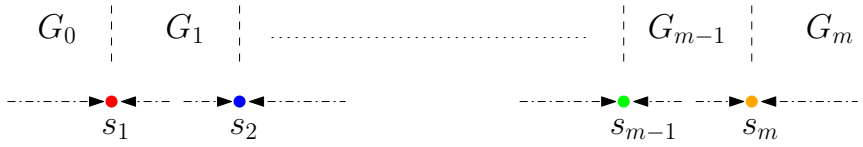


Figure 5.7: The graph G partitioned by the sinks.

We now describe the algorithm for computing the MCS $\mathcal{C}$ of a directed path (see Figure 5.8).

Lemma 5.2

If the number of runs in a subgraph G_i is m_i then $m_i \leq |\mathcal{C}_i| \leq m_i + 1$. The vertex at the end of each run of that subgraph must be present in $\mathcal{C}_i$.

Proof. Consider a partition G_i with m_i runs. Usually taking one vertex from each run suffices to make the partition consistent. However, sometimes the single source between two consecutive sinks might become inconsistent. In order to make the source consistent, one more vertex may have to be included in $\mathcal{C}$. We know that the sink vertices s_i must be included in $\mathcal{C}$ for $i = 1, 2, \ldots, m$. For $i = 0$, no extra vertex needs to be included in $\mathcal{C}_i$ as the path from $\hat{s}_0$ goes to s_1 only. So, we consider the case of each G_i where $0 < i \leq m$. Now, for every vertex v in between s_i and s_{i+1}, any one of the following three cases might occur:

$v \neq \hat{s}_i$ **and** $\exists$ **a path from** v **to** s_i : If the path is monochromatic, then s_i makes v and every vertex along the path $v \longrightarrow s_i$ consistent. Otherwise, the path changes color at v or some intermediate vertex v' on the path $v \longrightarrow s_i$. We have to include the last vertex of the same color as $color(v)$ along the said path to make v consistent.

$v \neq \hat{s}_i$ **and** $\exists$ **a path from** v **to** s_{i+1} : Same as above.

Algorithm 5.1: An MCS $\mathcal{C}$ of G

Input: A directed path graph $G = (V, E)$
Output: An MCS $\mathcal{C}$ for the graph G

```
1  //STEP 1;
2  Identify all m sinks in G;
3  C = C ∪ {s_1, s_2, ..., s_m};
4  //STEP 2;
5  for each partition G_i of G where i = {0, 1, ..., m} do
6  |  //STEP 2.1;
7  |  if G_i is monochromatic then
8  |  |  do nothing;
9  |  else
10 |  |  Traverse G_i;
11 |  |  //STEP 2.2;
12 |  |  Path from s_i to ŝ_i;
13 |  |  if new run encountered then
14 |  |  |  Put the first member in C;
15 |  |  end
16 |  |  //STEP 2.3;
17 |  |  Path from ŝ_i to s_{i+1};
18 |  |  if new run encountered then
19 |  |  |  Put the first member in C;
20 |  |  end
21 |  end
22 end
23 //STEP 3;
24 for each source ŝ_i where i = {0, 1, ..., m} do
25 |  if ŝ_i is not consistent then
26 |  |  C = C ∪ {ŝ_i};
27 |  end
28 end
29 return C;
```

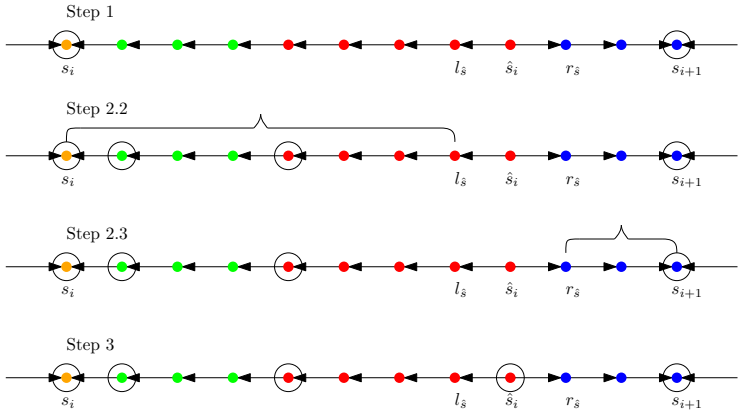


Figure 5.8: The step by step execution of the algorithm.

$v = \hat{s}_i$**:** Let v lie in the run B. If both s_i and s_{i+1} lie in B, then no further vertex needs to be included in $\mathcal{C}_i$. Otherwise, if s_i (resp. s_{i+1}) is outside B then the last vertex of B towards s_i (resp. s_{i+1}) needs to be included in $\mathcal{C}_i$. But the inclusion of this last vertex in $\mathcal{C}_i$ can make v inconsistent. So $\hat{s}_i$ has to be included in $\mathcal{C}_i$. Thus both the inequalities on the size of $\mathcal{C}_i$ follow. See Figure 5.9 for an illustration of different cases.

□

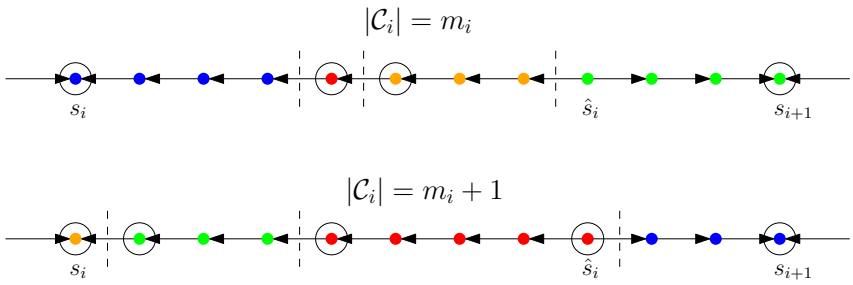


Figure 5.9: Illustration of $m_i \leq |\mathcal{C}_i| \leq m_i + 1$.

Theorem 5.2

Algorithm 6.1 correctly computes a minimum consistent subset of a directed path in linear time.

Proof. Identifying sources and sinks need $O(n)$ time by scanning the vertices of G from left to right. Processing each partition G_i also needs a total of n_i ($= |V_i|$) time for scanning from $\hat{s}_i$ to s_i and from $\hat{s}_i$ to s_{i+1}. As the partitions can be independently processed, the result follows. □

5.3 Spider Graph

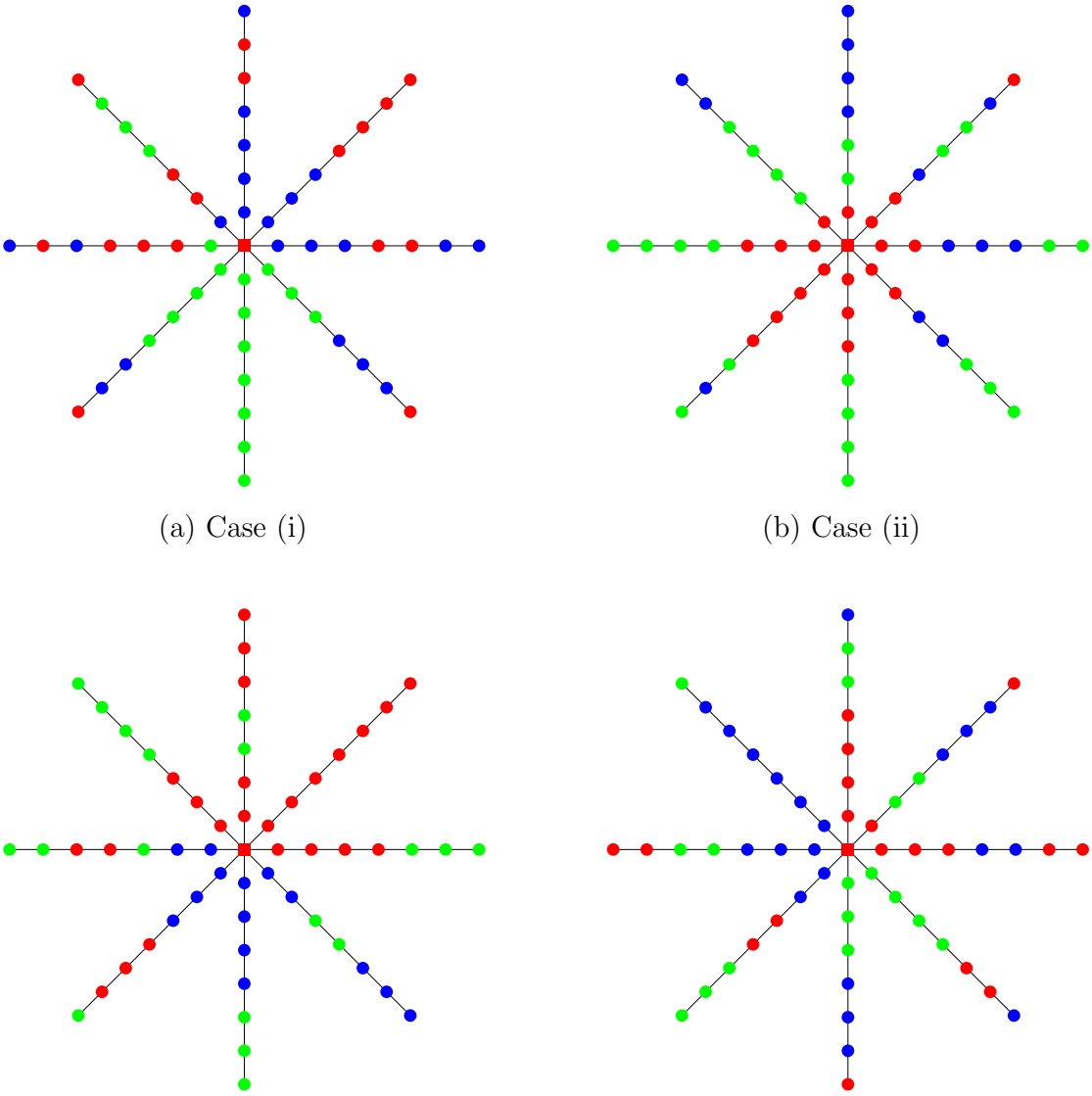
In a *spider graph* $G(V, E)$ is an undirected graph with $V = \{v\} \cup_{i=1}^{m} V_i$ where v is called the *head*, and each V_i is a path of length n_i whose one end is connected with v. Each vertex in V is assigned a color from the set $\{c_1, c_2, \ldots, c_k\}$. The set E is the union of edges of the m paths. $|V| = \sum_{i=1}^{m} |V_i| + 1$, and $|E| = \sum_{i=1}^{m} |V_i|$. We will refer each path V_i as a *leg* of the spider. In each V_i, the run attached to v is referred to as *first run* of that leg, and will be denoted as ρ_i. We will use $\mathcal{C}$ to denote the minimum cardinality consistent subset for the graph G. We will handle both undirected and directed spiders.

5.3.1 Undirected Spiders

For the sake of simplicity of the presentation, we shall first consider $k = 3$, and the colors used are *red*, *blue* and *green*. The algorithm can be easily extended for arbitrary integer k. Without loss of generality, assume that the color of the *head* vertex v is *red*. We will use $\mathcal{C}(u)$ to denote a minimum size consistent subset of G among all possible consistent subsets that contain the vertex u.

In order to compute the MCS $\mathcal{C}$, we need to consider the following four exhaustive situations (see Figure 5.10) depending on the color of the first run ρ_i of each leg V_i of G:

Case (i): ρ_i, $i = 1, 2, \ldots, m$ are all of colors different from $color(v) = red$,



(a) Case (i)

(b) Case (ii)

(c) Case (iii)

(d) Case (iv)

Figure 5.10: Tri-chromatic spiders.

Case (ii): ρ_i, $i = 1, 2, \ldots, m$ are all of color red; since it contains the head let us refer to this set of connected vertices as *head block*,

Case (iii): ρ_i, $i = 1, 2, \ldots, m$ are of 2 different colors with $color(\rho_i) = red$ for at least one leg V_i, $i = 1, 2, \ldots, m$, and

Case (iv): ρ_i, $i = 1, 2, \ldots, m$ are of 3 different colors; needless to say, here surely exists at least one leg V_i with $color(\rho_i) = red$.

Processing of Case (i): In this case, v must be included in $\mathcal{C}$. We compute a consistent subset $\mathcal{C}'$ as $\mathcal{C}' = (\cup_{i=1}^{m} \mathcal{C}_i) \bigcup \{v\}$, where $\mathcal{C}_i$ is the MCS of the path (leg) $V_i \cup \{v\}$ assuming that v is chosen in $\mathcal{C}_i$. This can be computed by slightly modifying the algorithm of Section 5.2.1 for undirected paths with $t = v$ assuming t has a color different from all the colors used in G. We have $\mathcal{C} = \cup_{i=1}^{m} \mathcal{C}_i$ and $|\mathcal{C}| = \sum_{i=1}^{m} |\mathcal{C}_i| - m + 1$ since v is included in each $\mathcal{C}_i$.

Preprocessing: For ease in the computation of cases (ii)-(iv), we execute the following preprocessing step. Let $\theta = \max_{i=1}^{m} |\rho_i|$. For each leg V_i, we create $V_i' = V_i \cup \rho$ where ρ is a path of vertices of size θ having color *red* ($= color(v)$). The elements of $\rho \cup \rho_i$ are numbered as follows: v is numbered as 0, starting from the next vertex of v, the elements of ρ_i are numbered as $1, 2, \ldots$ in order, and the elements of ρ are numbered as $-1, -2, \ldots$. We use $\mathcal{C}_i(w)$ to denote the size of the minimum consistent set of V_i' with $w \in \rho_i \cup \rho$ as a member of $\mathcal{C}_i(w)$. Note that, a single execution of the algorithm of Section 5.2.1 on V_i' computes $\mathcal{C}_i(w)$ for all $w \in \rho_i \cup \rho$.

We create a table T of size $m \times 2\theta$, and set $T[i, \alpha] = \mathcal{C}_i(w_\alpha)$ for $i = 1, 2, \ldots, m$ and $\alpha = -\theta, -\theta + 1, \ldots, 0, 1, 2, \ldots, |\rho_i|$ where w_α is the α-th element in $\rho_i \cup \rho$. The remaining entries will remain ∞. Assuming that the first θ columns in each row of T correspond to the members of ρ, the 0-th element in all the rows of the array T are aligned in a single column of T. This computation needs $O(nm)$ time due to the fact that (i) Each V_i can be handled independently, (ii) and each leg V_i can be of size $O(n)$ in the worst case due to the addition of ρ with it.

Processing of Case (ii): Here v and ρ_i, are of same color for all $i = 1, 2, \ldots, m$. Let $U = \bigcup_{i=1}^{m} \rho_i \cup \{v\}$. Lemma 5.3, stated below, describes an $O(nm)$ time algorithm for computing $\mathcal{C}$. For the demonstration, see Figure 5.11.

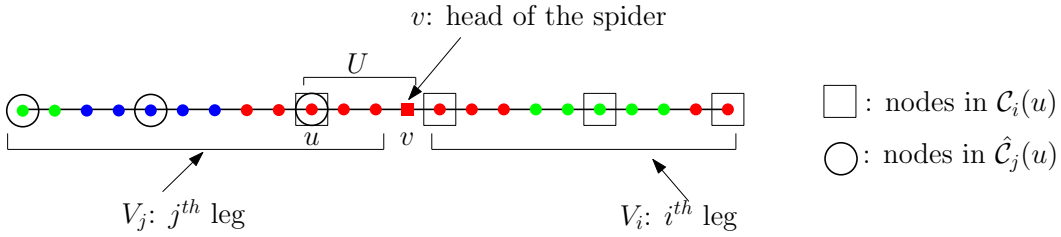


Figure 5.11: $\mathcal{C}_i(u)$: optimum solution for $V_i \cup U$, and $\hat{\mathcal{C}}_j(u)$: optimum solution for $V_j \setminus U \cup \{u\}$:

Observation 5.4

$\mathcal{C}$ must contain

(i) at least one member of the set $(\cup_{i=1}^{m} \rho_i) \bigcup \{v\}$, and

(ii) at least one member from each run, excepting its *first run*, in every leg of the spider.

From Observation 5.4, we know that $\mathcal{C}$ will always have at least one vertex from the head block. In order to compute the MCS of such a spider, we consider each vertex v' in the head block to be a member of $\mathcal{C}$ and compute the consistent subset of each leg, given $v' \in \mathcal{C}$. The following lemma states the computation using an array T of size $m \times 2\theta$.

Lemma 5.3

If u is the α-th element of ρ_j, then

(a) $\mathcal{C}_i(u) = T[i, -\alpha]$ for all $i \neq j$, and $\mathcal{C}_j(u) = T[j, \alpha]$,

(b) $\mathcal{C}(u) = \sum_{i=1}^{m} \mathcal{C}_i(u)$, and

(c) $\mathcal{C} = \min_{u \in U} \mathcal{C}(u)$.

Proof. Part (a) follows from the indexing of the array T; part (b) is true as the

legs can be processed independently and part(c) holds as atleast one node from the head block should be in the MCS. □

For processing Cases (iii) and (iv), we will require the definition of gates.

5.4: λ-*gate*

A λ-*gate* is defined as a λ-tuple $(u_1, u_2, \ldots, u_\lambda)$ $(\lambda \leq k)$ such that (see Figure 5.12):

- The colors of all u_is' in the tuple are distinct.
- u_is' appear in the first run of different legs of the spider,
- The head v is called the base of the gate.
- The hop-distances from the head v to all the vertices $\{u_1, u_2, \ldots u_\lambda\}$ are same.

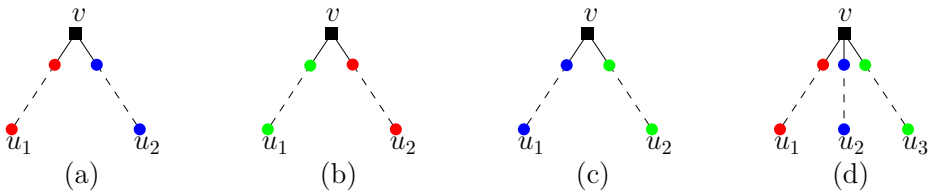


Figure 5.12: All possible gates of a trichromatic spider. Figure (a), (b) and (c) are the 2-gates of a trichromatic spider. Figure (d) is a 3-gate. The base of a gate is always the head v.

Processing of Case (iii): Here, v and the first run ρ_i for a few legs V_i, $i \in R \subset \{V_1, V_2, \ldots, V_m\}$ are colored *red*, and ρ_i for other legs $\{V_j, j \in B = \{V_1, V_2, \ldots, V_m\} \setminus R\}$ are colored *blue*. In order to handle this case, we introduce the concept of *gate* as follows:

In Case (iii), we have $\lambda = 2$, and in this section we will refer each 2-gate as a gate. Now, if such a 2-gate (u, w) is selected in $\mathcal{C}$, $u \in \rho_i$ and $w \in \rho_j$ and the color of u and w are *red* and *blue* respectively, then all the legs V_ℓ, $\ell \neq \{i, j\}$ and containing no green vertex, are consistently covered irrespective of the color (*red*, *blue*) of the

head vertex v of the spider. Thus, we need to consider the computation of the MCS of V_i, V_j, and all the legs $\Delta = \{V_\ell \mid V_\ell$ contains at least one *green* vertex$\}$. We will use R (resp. B) to denote the set of legs of the spider V such that $color(\rho_i) = red$ (resp. *blue*) for all $V_i \in R$ (resp. B).

We consider each pair of legs V_i, V_j, $i, j \in \{1, 2, \ldots, m\}$, $i \neq j$, and compute its MCS $\mathcal{C}_{ij}$, Finally, $\mathcal{C} = \min\{\mathcal{C}_{ij} | i, j \in \{1, 2, \ldots, m\}, i \neq j\}$.

Computation of $\mathcal{C}_{ij}$: Let us now denote $\theta = \min(|\rho_i|, |\rho_j|)$. Thus, each pair of vertices (u_α, w_α), $u_\alpha \in \rho_i$ and $w_\alpha \in \rho_j$ and $\alpha \leq \theta$ can form a gate. We consider each of the gates (u_α, w_α), $\alpha = 1, 2, \ldots \theta$, and compute the size of the MCS as follows, assuming u_α, w_α is in $\mathcal{C}_{ij}$. Finally, the one having the minimum size is $\mathcal{C}_{ij}$.

It needs to be mentioned that, we can consider the gates (u_α, w_α), $\alpha = 1, 2, \ldots, \theta$ at a time for computing $\mathcal{C}_{ij}$. The MCSs' for V_i (resp. V_j) for different u_α (resp. w_α) are available in $T[i, \alpha]$ (resp. $T[j, \alpha]$). The size of the MCS for each leg V_ℓ, having the presence of green vertices (i.e., $V_\ell \in \Delta$) are also available in $T[\ell, -\alpha]$ for different $\alpha = 1, 2, \ldots, \theta$. Thus, $\mathcal{C}_{ij} = \min_{\alpha=1}^{\theta}(T[i, \alpha] + T[j, \alpha] + \sum_{V_\ell \in \Delta} T[\ell, -\alpha])$, and $\mathcal{C} = \min\{\mathcal{C}_{ij} | i = 1, \ldots, m, j = 1, \ldots, m, i \neq j\}$.

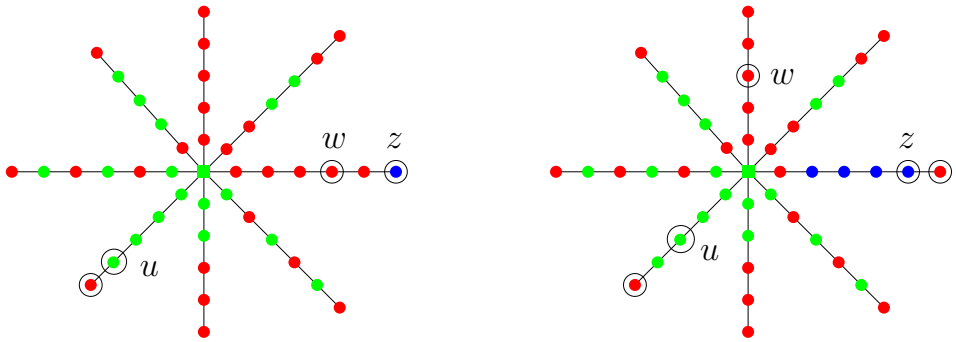
Lemma 5.4

The time complexity for processing this step is $O(nm)$.

Proof. Our computation proceeds as follows: Let $\theta^* = \max\{|\rho_i| \mid color(\rho_i) = color(v)\}$. We already have $T[\ell, \alpha]$, for all $V_\ell \in \Delta$, and $\alpha = -1, -2, \ldots, -\theta^*$. For each $\alpha = 1, 2, \ldots, \theta^*$ find $X[\alpha] = \min\{T[\ell, \alpha] \mid \ell \in R\}$ and $Y[\alpha] = \min\{T[\ell, \alpha] \mid \ell \in B\}$ by searching at most $O(m)$ elements of the array T, since $|R| + |B| = m - |\Delta|$. Finally compute $\mathcal{C} = \min_{\alpha=1}^{\theta^*}(X[\alpha] + Y[\alpha] + \sum_{V_\ell \in \Delta} T[\ell, -\alpha])$. Since $\theta^* = O(n)$ in the worst case and $|\Delta| < m$, the result follows. □

The following observation say that, there are situations where 2-gate exists, but considering that 2-gate sometimes leads to a non-optimal solution (see Figure

5.13).



(a) With 2-gate on the unique leg (b) Without 2-gate on the unique leg

Figure 5.13: Special trichromatic spiders for Case (iii).

Observation 5.5

If $|B| = 1$ then using a gate may lead to non-optimal result.

Proof. Consider a situation with $B = \{V_i\}$; $\delta = \max\{|\rho_j| \mid V_j \in R\}$, and $\delta < |\rho_i|$. For V_i, the optimal solution of the MCS (of size k) exists for a blue vertex $z \in \rho_i$ such that hop-distance$(v, z) > \delta$. The solution is perfectly admissible for G since for such a choice of $z \in \mathcal{C}$, the head vertex v can be made consistent with an appropriate choice $z' \in \rho_j$ (of color *red*) and hop-distance$(v, z') \leq \delta$. If we choose any 2-gate (u, w) with $w \in \rho_i$, then hop-distance$(v, w) \leq \delta$, and the size of the MCS of V_i with w in the solution is of size $\geq k+1$, creating a non-optimal solution. However,

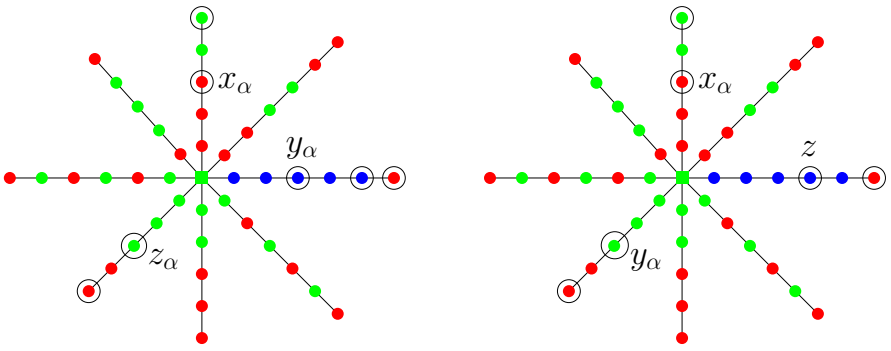
- if the optimal solution for V_i exists with $z \in \rho_i$ and hop-distance$(v, z) \leq \delta$, we can get optimum solution of G using a 2-gate even if V_i is the only leg with $color(\rho_i) = blue$.
- if there exists more than one leg with the *blue* color in their first run, i.e., $|B| > 1$, then this type of situation does not arise due to the fact that (i) if a gate is chosen only one leg in B will contribute in the optimum solution;

otherwise more than one leg from B will contribute in the solution, and (ii) as v is red, the size of optimum solution of each leg in B is ≥ 1.

Thus, the observation follows. □

Thus if $|B| = 1$, we compute the solution both with and without using 2-gate.

Processing of Case (iv): Here 3-gates exist (see Figure 5.14). For each gate $(x_\alpha, y_\alpha, z_\alpha)$ with $color(x_\alpha) = red$, $color(y_\alpha) = blue$ and $color(z_\alpha) = green$, we need to compute the MCS. Let $x_\alpha \in \rho_i$, $y_\alpha \in \rho_j$ and $z_\alpha \in \rho_\ell$. Now, for the choice of this gate in the solution, all the legs in $\{1, 2, \ldots, m\} \setminus \{i, j, \ell\}$ are covered. So, $\mathcal{C}[i, j, \ell] = T[i, \alpha] + T[j, \alpha] + T[\ell, \alpha]$. Finally $\mathcal{C} = \min\{\mathcal{C}[i, j, \ell] \mid i \in R, j \in B \text{ and } \ell \in \Delta\}$, where R, B and Δ are the legs in G with their first run *red*, *blue* and *green* respectively. Let $\theta^* = \max_{i=1}^{m} |\rho_i|$. In a linear scan in the array T, we can compute $A[\alpha] = \min_{i \in R} T[i, \alpha]$, $B[\alpha] = \min_{i \in B} T[i, \alpha]$ and $C[\alpha] = \min_{i \in \Gamma} T[i, \alpha]$ for all $\alpha = 1, 2, \ldots, \theta^*$. Thus, $\mathcal{C} = \min_{\alpha=1}^{\theta^*}(A[\alpha] + B[\alpha] + C[\alpha])$. Thus, this case can be solved in $O(n)$ time.



(a) With 3-gate (b) Without 3-gate

Figure 5.14: Special trichromatic spiders for Case (iv).

As in Observation 5.5, here also, we may have non-optimal solution using 3-gate. Thus, here (i) we need to consider all possible 2-gates if there exists exactly one leg with first run of *green* color, and also (ii) we need to consider all possible solution

without gate if there exists exactly one leg with first run of *green* color, and exactly one leg with first run of *blue* color.

Let the head vertex v is of color *red*, and the gate(s) with 3-tuple exists. In addition to consider the 3-gates, we also consider all possible 2-gates whose one elements are (*red*,*blue*). It can be shown that this case can also be solved as in Case (iii) in $O(nm)$ time. Thus, we have the following result:

Theorem 5.3

The proposed algorithm correctly computes the minimum consistent subset $\mathcal{C}$ of a tri-chromatic spider graph G in $O(nm)$ time, where n is the number of vertices, and m is the number of legs.

Proof. The correctness follows from the fact that the stated four cases are exhaustive. The processing of cases (i), (ii) and (iv) are straightforward. While processing case (iii) using gates, the exception, mentioned in Observation 5.5 is properly handled.

Time complexity in each of the four cases is analyzed while describing that case. □

When handling a k-chromatic spider, we will have $k + 1$ cases depending on the colors of the first runs of the m legs. For spiders following Case (i) and Case (ii), the computation of a MCS is exactly as explained earlier. But a k-chromatic spider with a λ-gate will have to be computed according to the value of λ. The procedure above has been explained in details for 2-gates and 3-gates. However, the similar idea follows for λ-gates where the value of λ is greater than 3. Thus we have the following corollary.

Corollary 5.1

The proposed algorithm computes the minimum consistent subset of a k-chromatic spider graph G in $O(nm)$ time, where n is the number of vertices in the graph G and m is the number of the legs of the spider.

5.3.2 Directed Spiders

In a directed spider graph $G = (V, E)$ (see Figure 5.15), the edges are all directed, but, the underlying undirected version of the graph is a spider. As earlier, assume that the number of legs is m, $|V| = n$, $|V_i| = n_i$, and each vertex is assigned one of the colors in $\{1, 2, \ldots, k\}$. Note that $V = \bigcup_{i=1}^{m} V_i \cup \{v\}$ where v is the head of the spider. The head v is connected with the first vertex of all the legs by an incoming or an outgoing edge. We also assume that there is no bidirectional edge between any pair of vertices in G. We use L_I (resp. L_O) to denote the set of legs having edge incident to the head v from that leg is incoming (resp. outgoing). We first introduce the concept of a *directed gate* as follows:

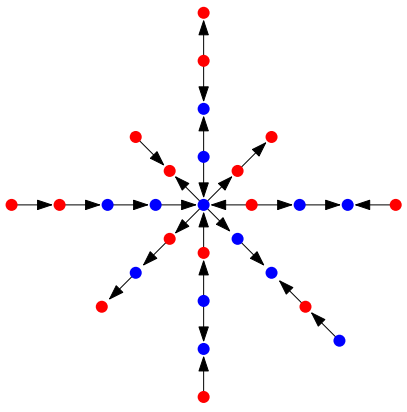


Figure 5.15: Directed Spider graph.

5.5: Directed Gate

A **directed gate** (see Figure 5.16) in a directed spider graph G is a λ-tuple $(u_1, \ldots, u_\lambda)$ (renaming the colors, we assume u_i has color i) such that:

- $\lambda \leq k$,
- v is assigned a color in $\{1, \ldots, \lambda\}$,
- vertices $(u_1, \ldots, u_\lambda)$ are in the first runs of different legs in L_O,
- $color(u_i) \neq color(u_j)$ $i, j \in \{1, \ldots, \lambda\}$, $i \neq j$
- there exists directed paths from v to each of the vertices $\{u_i,\ i = 1, \ldots, \lambda\}$ is of same length,
- all the vertices on the path $v \to u_i$ are of $color(u_i)$, $i = 1, \ldots, \lambda$.

Here, we will use the same terminologies as defined for directed path.

Observation 5.6

If $(u_1, \ldots, u_\lambda)$ forms a λ-gate, then in each leg V_i (with $u_i \in V_i$), the source vertex nearest to v is at least at a distance equal to the length of the path from v to u_i, for all $i = 1, 2, \ldots, \lambda$.

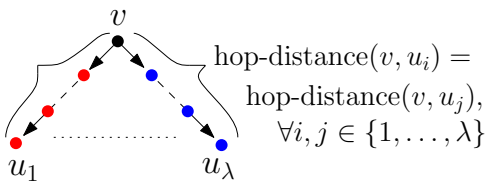


Figure 5.16: Gate in an directed spider.

Observation 5.7

If the λ-tuple of a gate $(u_1, \ldots, u_\lambda)$ is included in $\mathcal{C}$ then all the vertices from the source vertex closet to v in each leg of L_I that does not contain any vertex having color different from $\{c_1, c_2 \ldots, c_\lambda\}$ are covered by that tuple.

As an initial step, we choose the head vertex v in $\mathcal{C}$. Thus, every leg of the spider can be treated as an independent path, and its MCS is computed using the algorithm for a directed path (Section 5.3.2) with the constraint that $v \in \mathcal{C}$. The union of those solutions form an initial solution of $\mathcal{C}$. Next, we try to improve this solution (decreasing the size of $\mathcal{C}$, if possible) using gate (if any), i.e., without choosing v in $\mathcal{C}$. Here, we need to do an exhaustive case analysis.

Case 1: The head vertex v is a sink vertex

Case 2: The head vertex v is a source vertex.

Case 3: The head vertex v is neither a sink nor a source.

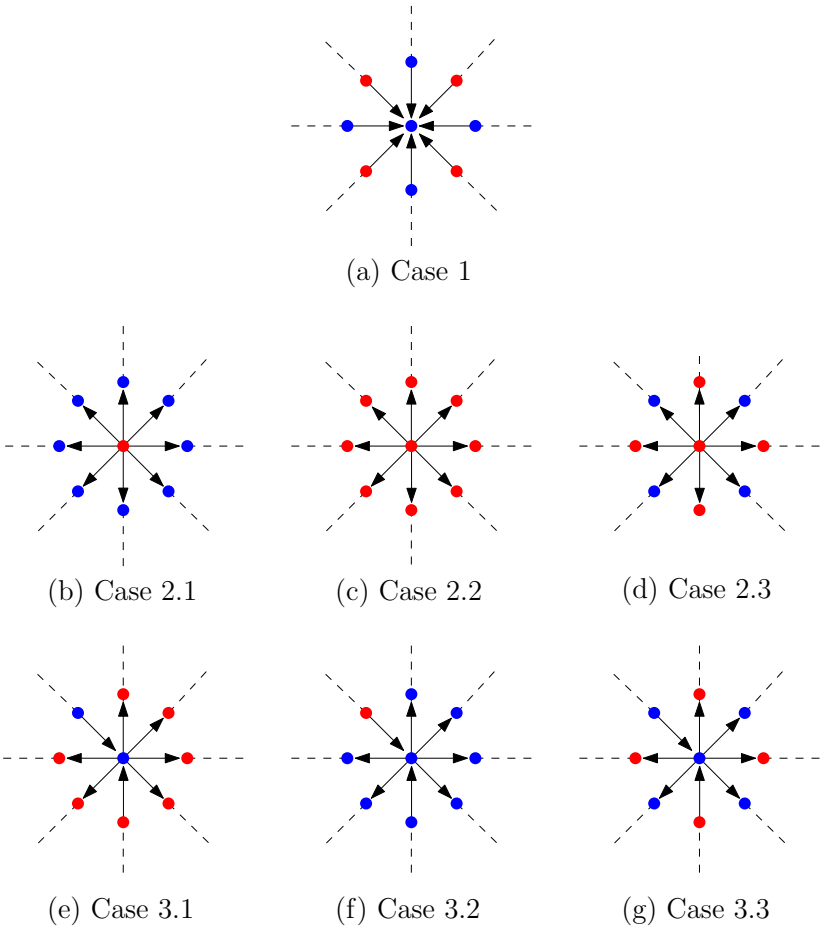
In Case 1, Observation 5.1 suggests that v should belong to the MCSs' (see Figure 5.17(a)). Thus here the solution of the initial step is the optimum solution.

In Case 2, we have the following sub-cases:

Case 2.1: there exists no leg having its first run of color same as that of v (see Figure 5.17(b)). Here, $v \in \mathcal{C}$. Hence this case can be considered as in Case 1.

Case 2.2: all the vertices adjacent to vertex v are of the same color as of v (see Figure 5.17(c)). Here, the head vertex will not contribute to the MCS. We compute the MCS of all the outgoing legs as paths and since all the adjacent vertices are of same color, the consistency of vertex v is guaranteed. Thus, the optimum $\mathcal{C}$ is the union of solutions of all the legs.

Case 2.3: The vertices adjacent to the vertex v are assigned different colors in $1 \leq \alpha \leq k$ (see Figure 5.17(d)). In this case, we find the MCS of all the outgoing legs excluding v as independent directed paths, and take their union as $\mathcal{C}'$. At the end, consistency of v is checked in $\mathcal{C}'$; if v is consistent then we are done; otherwise we include v in $\mathcal{C}'$. Next, if $|\mathcal{C}'| < |\mathcal{C}|$, we set $\mathcal{C} = \mathcal{C}'$.



(a) Case 1

(b) Case 2.1 (c) Case 2.2 (d) Case 2.3

(e) Case 3.1 (f) Case 3.2 (g) Case 3.3

Figure 5.17: Cases in an directed Spider graph.

Finally, to handle Case 3, let us classify the legs as L_I and L_O, where L_I (resp. L_O) is the set of legs whose edge incident to v is incoming (resp. outgoing). Let the colors of the vertices adjacent to v in the legs of L_O be $\{c_1, c_2, \ldots, c_\lambda\}$, where $\lambda \leq k$. If the color of v does not belong to $\{1, 2, \ldots, \lambda\}$ (see Figure 5.17(e)), then $v \in \mathcal{C}$ is to be chosen, and this case is considered earlier. Otherwise, we execute the following tasks:

Task 1: Compute the MCS of all the legs in L_O independently, and compute their union $\mathcal{C}'$.

Task 2: If $\lambda = 1$, i.e., all the outgoing edges from head are to vertices of same

color (see Figure 5.17(f)), then the consistency of v may be achieved by the first chosen vertex in any one of those legs.

Task 3: If $\lambda > 1$, i.e., outgoing and incoming edges at the vertex v are of more than one color (see Figure 5.17(g)). Let θ_i be the distance of the furthest vertex of color c_i from v in $V_i \in L_O$, and $\theta = \min_{i=1}^{\lambda} \theta_i$. For each j with $\theta_j > \theta$, we choose a leg with its first run of color c_j, and include a vertex u_j at a distance θ from v (see Figure 5.18). These additionally included vertices $\{u_1, u_2, \ldots, u_\lambda\}$ will form a gate. This gate will cover v, and a portion, say from vertex u_ℓ to v of each leg $V_\ell \in L_I$ such that there exists a directed path π_ℓ from u_ℓ to v where all the vertices are of color in the set $\{c_1, c_2, \ldots, c_\lambda\}$. Now, it remains to perform the following:

Task 4: In each leg $\ell \in L_I$, we need to solve the MCS problem for the subpath $V_\ell \setminus \pi_\ell$ independently. If $\mathcal{C}_\ell$ is the solution for the leg ℓ, then $\mathcal{C}'$ is updated to $\mathcal{C}' \cup_{\ell \in L_I} \mathcal{C}_\ell$. If $|\mathcal{C}'| < |\mathcal{C}|$ then $\mathcal{C} = \mathcal{C}'$ is assigned.

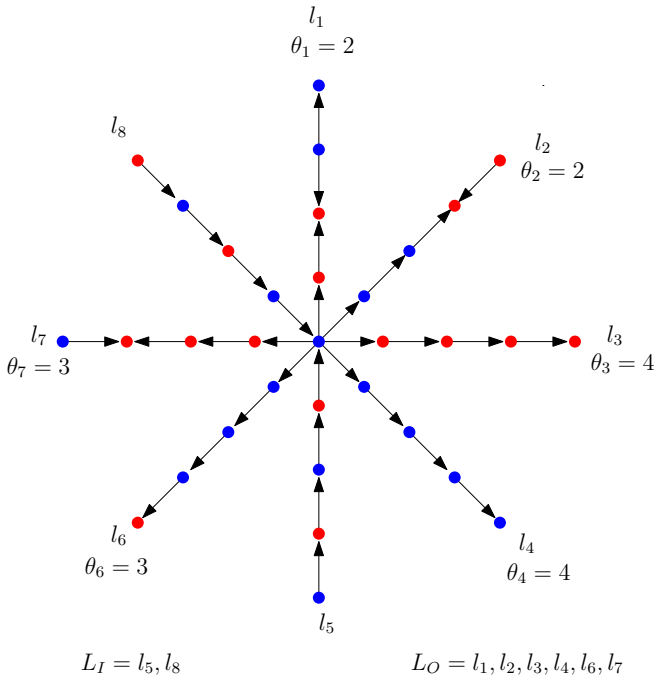


Figure 5.18: Illustration for θ_i's.

Note: It can be shown that unlike Observation 5.5 of the earlier subsection, here using a λ-gate will not produce non-optimal results. The reason is that in order to cover the first run of each leg $V_\ell \in L_I$ having its vertex incident to v of color c_i, we need to choose a vertex at distance at most θ^* from v among the legs $V_j \in L_O$ which is reachable from v and is of color c_i, in addition to the sink closest to v in V_j.

Theorem 5.4

The proposed algorithm for computing the minimum consistent subset of a directed spider is correct, and it needs $O(n)$ time, where $n = |V|$.

Proof. We have computed the solution both by including $v \in \mathcal{C}$ and by not including $v \in \mathcal{C}$ if at all possible, and reported the one having smaller size. While checking the possibility of getting a feasible solution by not including $v \in \mathcal{C}$, we have done an exhaustive case analysis.

The time complexity of the preprocessing phase is $O(n)$, where $n = \sum_{i=1}^{k} n_i$; n_i is the size of the i-th leg. Now, if v is not included in C , then for each leg a part of it needs to be solved independently. As solving a leg (or its portion) needs scanning the vertices in that path in order once, it needs time linear in the size of that leg. Unlike the undirected version of this problem, here we have only one choice of a gate. In each leg, the portion that is not covered by the gate is independent of the gate chosen. Thus, the uncovered portion can be solved independently as in a directed path. The total time complexity follows. □

5.4 Bi-chromatic Caterpillar Graph

A caterpillar $G = (V, E)$ is a tree in which every vertex is within distance 1 from a path in G, called the *skeleton*. The vertices in V that are not on the skeleton are termed as *dangling vertices*. Thus, $V = S \cup D$, where the vertices in S are on the skeleton, and D contains dangling vertices of all vertices in S, i.e., $D = \bigcup_{v \in S} D_v$,

where D_v is the set of vertices dangling at the vertex $v \in S$. Each dangling vertex in D_v is at distance 1 from a vertex $v \in S$. Also v can be called as the parent of the vertices in D_v.

In this section, we will consider the MCS problem for a bi-chromatic caterpillar where each vertex of V is colored by *red* or *blue*. The cases $|V| = 1$ or 2 can be solved trivially. If $|V| \geq 3$, then we assume that the first and the last vertex of the skeleton consist of at least one dangling vertex. If the first (resp. last) vertex v of the skeleton does not have any dangling vertex and the vertex adjacent to v is u, then we can consider vertex v as the dangling vertex of u.

Observation 5.8

If any vertex on the skeleton S has two dangling vertices p, q of opposite colors, then $\mathcal{C} = \{p, r\}$.

Proof. Let $q \in S$ be the common neighbor of p and r. Every vertex in $S \setminus \{p, r\}$ reaches p, r through the vertex q. Thus the result follows. □

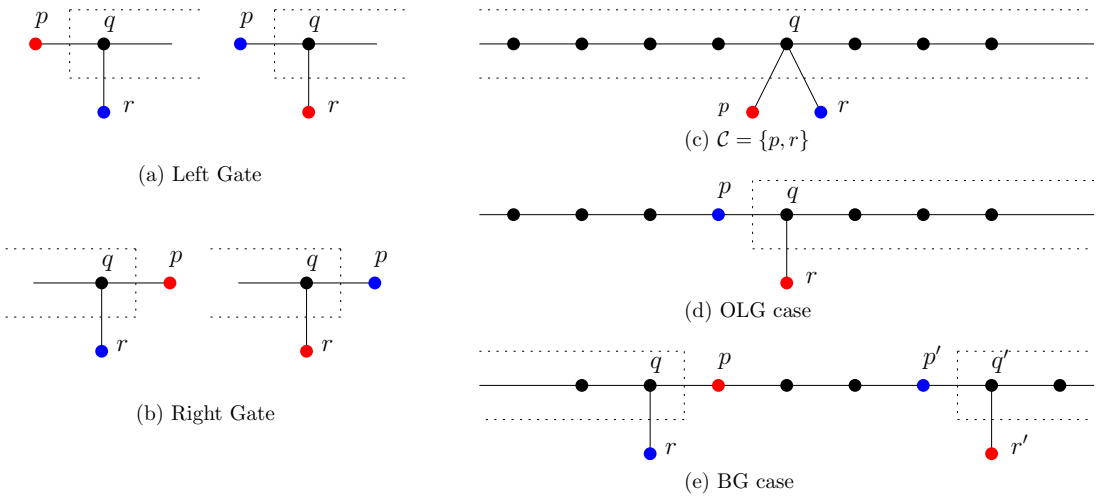


Figure 5.19: (a) Left gate. (b) Right gate. (c) Visualizing Observation 5.8. (d) A caterpillar with only left gate (OLG). (e) A caterpillar with both gates (BG). All dotted regions signify the part of the caterpillar covered by the gates.

So, we consider the cases where V does not satisfy Observation 5.8. In other words, if more than one dangling vertex is present at a vertex $v \in S$, then they are all of the same color. To simplify the exposition, we assume that the vertices in S are arranged on a horizontal line from left to right.

Consider two structures as shown in Figure 5.19 consisting of three vertices (p, q, r), where vertex $q \in S$ is of color either *red* or *blue*, and is attached with two vertices p and r of opposite colors. As we are considering instances that do not satisfy Observation 5.8, both p and r cannot be dangling at vertex q. Without loss of generality, let us assume that $p \in S$ and $r \in D_q$. We now define the concept of *gate* where vertex q is called the *base* of the gate. The two cases shown in Figure 5.19(a) and 5.19(b) are referred to as *left-gate* and *right-gate* respectively. The existence of a left-gate (resp. right-gate) (p, q, r) implies that by choosing $\{p, r\}$ in $\mathcal{C}$ all the vertices to the right (resp. left) of that gate, including the base vertex q, are covered[3], and we need to compute the MCS of the subgraph of G that is attached with $p \in S$ at its left (resp. right) side. Here, the following four situations need to be considered depending on the occurrence/non-occurrence of left and/or right-gate.

OLG: Only *left-gate*(s) is/are present in G. The leftmost left-gate is called LG.

ORG: Only *right-gate*(s) is/are present in G. The rightmost right-gate is called RG.

BG: Both *left-gate*(s) and *right-gate*(s) are present in G. Here LG and RG are defined as above.

NG: There is *no* gate in G.

In our algorithm, we consider each of the aforesaid four cases separately, and formulate the steps for solving them.

[3]By the term "a vertex $v \in V$ is covered by $\mathcal{C}$" we mean that the nearest vertex (or one of the nearest vertices) of the vertex v in $\mathcal{C}$ is of color(v).

5.4.1 Algorithm

The algorithm for computing the MCS of a bi-chromatic caterpillar has been discussed through the following four cases. Let us recall that, for a gate (p, q, r) in the caterpillar: vertices p and q are on the skeleton S and vertex r is dangling at vertex q.

Handling OLG

We find $LG = (p_{LG}, q_{LG}, r_{LG})$. Let S_R be the set of vertices in S that are to the right of q_{LG} (along with their dangling vertices), and S_L be the set of vertices to the left of p_{LG}.

Observation 5.9

If $\mathcal{C}$ contains $\{p_{LG}, r_{LG}\}$ and no vertex from S_R then

(a) all the vertices in $\{q_{LG}\} \cup S_R \bigcup (\cup_{u \in S_R} D_u)$, irrespective of their colors, are covered by $\mathcal{C}$; in addition,

(b) if $D_{p_{LG}} \neq \emptyset$ and if the color of all the dangling vertices in $D_{p_{LG}}$ are of $color(p_{LG})$, then the members of $D_{p_{LG}}$ are also covered; otherwise the vertices in $D(p_{LG})$ are to be included in $\mathcal{C}$ to make them consistent.

Proof. (a) Follows from the fact that for every vertex $v \in S_R$, its two nearest vertices in $\mathcal{C}$ are p_{LG} and r_{LG}, and both of them can be reached from v through q_{LG}. As p_{LG} and r_{LG} are of different colors, distance property of v is maintained irrespective of its color.

(b) The first part is trivial as we are choosing p_{LG} in $\mathcal{C}$. For the second part, we need to consider two cases:

- If the left neighbor v of p_{LG} in S is of color same as that of p_{LG}, then

(v, u, p_{LG}) is a left gate, where $u \in D_{p_{LG}}$. This contradicts the fact that $LG = (p, q, r)$ is the left-most left-gate.

- If the left neighbor v of p_{LG} in S is of color different from that of p_{LG}, then all the members of $D_{p_{LG}}$ need to be included in $\mathcal{C}$ as p_{LG} is included in $\mathcal{C}$.

□

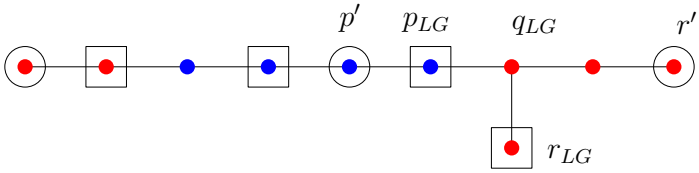
By Observation 5.9(b), if $D_{p_{LG}}$ are not covered, then all the members of $D_{p_{LG}}$ are to be included in $\mathcal{C}$. Now, as LG is the left-most left gate, there does not exist any gate in S_L. Thus, we solve the MCS problem for S_L as the NG case.

It might occur that even in the case where the caterpillar has only left gates, the left most left gate does not provide an optimal solution (see Figure 5.20). To handle such instances we have the following observation:

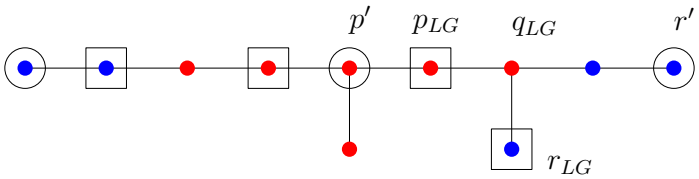
Observation 5.10

If all the vertices in S_R are of color(q_{LG}), then instead of including $\{p_{LG}, r_{LG}\}$ in $\mathcal{C}$ an appropriate pair $\{p', r'\}$ may be included in $\mathcal{C}$, where $p' \in S_L$ and $r' \in S_R$, to reduce the size of the MCS $\mathcal{C}$.

Proof. Since the color of the vertices in S_R is the same as r_{LG}, and q_{LG} (irrespective of its color) is not included in $\mathcal{C}$, we may include the right neighbor of q_{LG} (say $\hat{r}$) in $\mathcal{C}$ instead of r_{LG}, and $\mathcal{C}$ covers all the members in S_L. Now, $(p_{LG}, \hat{r})$ may further be replaced by (p', r') in $\mathcal{C}$ with color(p') = color(p_{LG}), where hop-distance(q_{LG}, p') = hop-distance(q_{LG}, r') and all the vertices in S from q_{LG} to p' are of color(p_{LG}). Also note that, as $LG = (p_{LG}, q_{LG}, r_{LG})$ is the leftmost left gate, the dangling vertices (if any) from q_{LG} to p' are of color(p_{LG}), and hence they will also be consistent for the presence of p' in $\mathcal{C}$ as their parents in S are consistent with p'. Figure 5.20 demonstrates the situation with both (a) color(q_{LG}) = color(r_{LG}) and (b) color$(q_{LG}) \neq$ color(r_{LG}). □



(a) $color(q_{LG}) = color(r_{LG})$



(b) $color(q_{LG}) \neq color(r_{LG})$

Figure 5.20: Demonstration of Observation 5.10. The squared vertices show the solution with LG, and the circled vertices show the optimal solution.

Thus, if Observation 5.10 is satisfied in the OLG case then $\mathcal{C}$ can be obtained by ignoring all the vertices to the right of q_{LG} (all are of color color(r_{LG})). After choosing appropriate vertices p' and r', the left of p' can have no more gates. Thus, we can process the modified problem instance as the NG case, explained later.

Handling ORG

We first find $RG = (p_{RG}, q_{RG}, r_{RG})$, the right-most right gate. Next, this case is handled analogously as the case OLG.

Handling BG

In a linear scan, we identify the base vertices of all the left gates and of all the right gates. Let $Q^L = \{q_0^L, q_1^L, \ldots\}$ be the base vertices of the left gates in left to right order, and $Q^R = \{q_0^R, q_1^R, \ldots\}$ be the base vertices of the right gates in right to left order. Note that, here $Q^L \neq \emptyset$ and $Q^R \neq \emptyset$ since it is neither a ORG case nor a OLG case. For each left gate (p, q, r) with $q \in Q^L$, if we choose (p, r) in

the set $\mathcal{C}$, all the vertices to the right of q are covered. Thus, it is beneficial to consider the left-most left-gate $LG = (p_{LG}, q_{LG}, r_{LG})$, where $q_{LG} = q_0^L$. In a similar argument, we also define the right-most right gate $RG = (p_{RG}, q_{RG}, r_{RG})$, where $q_{RG} = q_0^R$. Now, we may face two different scenarios: (i) LG is to the right of RG, and (ii) LG is to the left of RG.

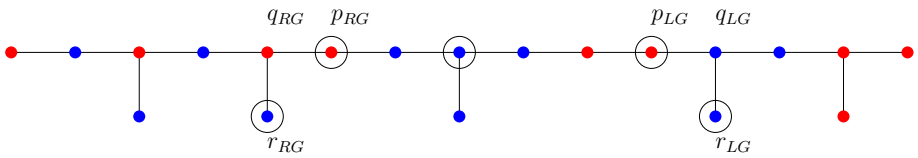
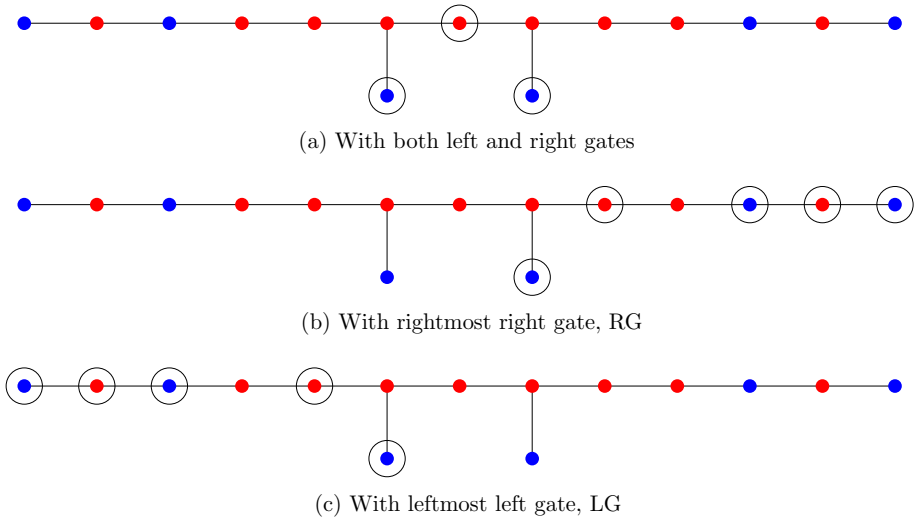


Figure 5.21: The covering region of **LG** and **RG** do not overlap

In Case (i), the portion to the left of q_{RG} that is covered by RG, and the portion to the right of q_{LG} covered by LG are disjoint, see Figure 5.21. We include the vertices $\{p_{LG}, r_{LG}, p_{RG}, q_{RG}\}$ in $\mathcal{C}$. As mentioned earlier, if the color of all the vertices in $D_{p_{LG}}$ (if any) are different from color(p_{LG}), then the members of $D_{p_{LG}}$ also need to be included in $\mathcal{C}$. Similarly, if the color of all the vertices in $D_{p_{RG}}$ is different from color(p_{RG}), then the members of $D_{p_{RG}}$ need to be included in $\mathcal{C}$. Here also Observation 5.10 may apply for LG or RG or both. Accordingly, the unsolved part S_{mid} (the portion between p_{RG} and p_{LG}) will be defined. We process S_{mid} as the NG case.

In Case (ii), the portions of S covered by LG and RG are overlapping. We separately consider the pair (p_{LG}, r_{LG}) and (p_{RG}, r_{RG}) for inclusion in $\mathcal{C}$ as was done in the OLG and ORG cases respectively. Hence, the portion of S to the right of q_{LG} or the left of q_{RG} will be covered depending on whether LG or RG is considered for inclusion in $\mathcal{C}$. If (p_{LG}, r_{LG}) (resp. (p_{RG}, r_{RG})) be considered for inclusion in $\mathcal{C}$ then S_L (resp. S_R) satisfies an NG case, and needs to be solved separately. In these two procedures, let the obtained solutions be $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. Here another situation may arise as shown in Figure 5.22, where both $\mathcal{C}_1$ and $\mathcal{C}_2$ produces non-optimal solution. Here, we need to execute another procedure as stated below: merge the base vertices of all the left-gates and all the right-gates in left to right order. Now, consider any pair of consecutive base vertices (q_i^R, q_j^L). Inclusion of (p_i^R, r_i^R) corresponding to the right-gate R_i and (p_j^L, r_j^L) corresponding to the



(a) With both left and right gates

(b) With rightmost right gate, RG

(c) With leftmost left gate, LG

Figure 5.22: A special case that arises if the covering region of **LG** and **RG** overlap

right-gate L_j in the MCS cover the vertices to the left of q_i^R and the vertices to the right of q_j^L. Moreover, the portion S' of S between the vertices p_i^R and p_j^L does not contain any gate. Thus, S' can be solved as an NG case. We consider all possible consecutive pairs of base vertices (q_i^R, q_j^L) of a left-gate and a right-gate. Compute an MCS taking union of $\{p_i^R, r_i^R, p_j^L, r_j^L\}$ and the MCS for the portion S' between q_i^R and q_j^L as a NG case. Let $\mathcal{C}_3$ be of minimum size among the solutions considering all possible pair of consecutive (right-gate, left-gate) pair. Finally, we set $\mathcal{C} = \min\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$.

Handling NG

We now describe the method of handling a part S' of S that does not contain any gate. It may happen that $S \equiv S'$. Similar to the concept of a *run* in Section 5.2, here we define a *block* as a connected component in S' of the same color (see Figure 5.23, where each block is highlighted by a box). Let us recall from Observation 5.8 that the dangling vertices (if any) attached to any vertex of S' are of same color. As S' does not contain any gate, we have the following observation.

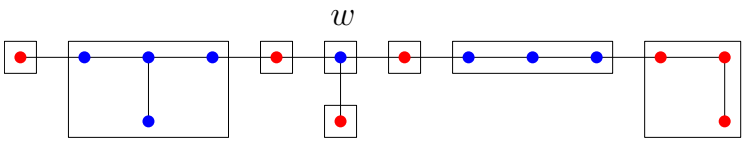


Figure 5.23: Caterpillar with the blocks highlighted.

Observation 5.11

(a) If a pair of adjacent vertices (u and v say) on the skeleton are of opposite color then each of them (u and v) can not have any dangling vertex of its own color.

(b) If a block on the skeleton has exactly one vertex (say w), then the dangling vertices (if any) of w are all of color different from that of w. We will name such a vertex w as *split-vertex* (see Figure 5.23).

Lemma 5.5

Each block in the NG scenario will have at least one and at most two representatives in $\mathcal{C}$.

Proof. If the vertices of S' form a path, i.e., there is no dangling vertex attached to any vertex in S', then by Lemma 5.1 the result follows. If the dangling vertices D_w ($|D_w| \geq 1$) of a vertex $w \in S'$ are of color(w) (i.e., w and D_w belong to the same block), then the member in $\mathcal{C}$ closest to w is also closest to all the members in D_w. Thus, here also by Lemma 5.1, the result follows. We only need to consider the case of a split vertices w, where the dangling vertex(s) D_w of vertex w is of opposite color. Thus, each member in the set $D_w \cup \{w\}$ is a separate block. We need to show that all the members in $D_w \cup \{w\}$ are in $\mathcal{C}$. If any member $v \in D_w$ is not in $\mathcal{C}$, then $w \notin \mathcal{C}$. Thus, we only need to show that $w \in \mathcal{C}$.

For a contradiction, let $w \notin \mathcal{C}$. As w is a split vertex, its neighbor(s) $u, u' \in S$ and also the members of D_w (if any) are of color different from color(w). If any member $v \in D_w$ belongs to $\mathcal{C}$, it can not be the closest member of w among the

members in $\mathcal{C}$. Thus, if $\beta \in \mathcal{C}$ is closest to w among the vertices in $\mathcal{C}$, then any one of u and u' is closer to β than to w. Thus, the consistency of w is violated with the members of $\mathcal{C}$. Thus, $w \in \mathcal{C}$, and this implies all the members in D_w are also in $\mathcal{C}$.

Using similar argument as in Lemma 5.1, it can be shown that more than two vertices in a block will never be included in $\mathcal{C}$. □

By Observation 5.11 and Lemma 5.5, each split vertex (along with its dangling vertices) is included in $\mathcal{C}$. Thus, S' is further divided using split-vertices. Let us now consider each of these unsolved parts separately as follows.

We define a graph H, whose vertices are those on the skeleton of that unsolved part $S'' \subseteq S'$ and at most one dangling vertex (if exists) of each vertex of S'' (by Observation 5.8). The dummy vertex for each block is also added; the edges, defined in H, are as in Section 5.2. If any (left/right) or both side(s) of S'' is (are) attached with a vertex in $\mathcal{C}$ (for handling a case with gate), then that vertex is chosen as s and/or t depending on the respective cases. If s (resp. t) is not defined for S'' then the dummy vertex of the first (resp. last) block will play the role of s (resp. t). Now, the vertices of S'' that appear on the shortest s-t path are included in $\mathcal{C}$.

5.4.2 Correctness and complexity

Theorem 5.5

The proposed algorithm for the caterpillar is correct, and produces optimum result in $O(n)$ time.

Proof. **Correctness:** The correctness trivially follows if Observation 5.8 is true for G, since every other vertex in S, and also its dangling vertices (if any) are equidistant from the only two opposite colored members of $\mathcal{C}$. Thus, we consider the sit-

uation where Observation 5.8 does not hold. Let us recall that $Q^L = \{q_0^L, q_1^L, \ldots\}$ were the base vertices of the left gates in left to right order, and $Q^R = \{q_0^R, q_1^R, \ldots\}$ were the base vertices of the right gates in right to left order. A further scan identifies the sets Q^L and Q^R, if any. We consider LG, RG and every (right-gate, left-gate) pair defined by (q_i^R, q_j^L) where $q_i^R \in Q^R$,$q_j^L \in Q^L$, and (q_i^R, q_j^L) are consecutive in $Q^L \cup Q^R$. For each pair, we considered the size of MCS by including vertices $(p_i^R, r_i^R, p_j^L, r_j^L)$ of the right- and left-gates corresponding to q_i^R and q_j^L respectively, the dangling vertices of p_i^R and p_j^L (provided their color do not match with that of p_i^R and p_j^L respectively), and solving the part between p_i^R and p_j^L as the NG case. Now, the correctness of the OLG, ORG and BG cases follow from the correctness of the NG case.

In the NG case, we further scan the skeleton to identify the split vertices, which, along with their dangling vertices are included in $\mathcal{C}$ by Lemma 5.5. This splits S into multiple unsolved parts[4]. Further using Lemma 5.5, we have a feasible solution for each block that is included in $\mathcal{C}$. Thus $\mathcal{C}$ is a feasible solution for S.

Optimality: In order to prove the optimality, we need to justify only the situation where gate(s) is/are present. If the covered side of the gate is bi-colored, then (p, r) is essential for covering all the vertices in the covered side. However, in the case where the covered side of a gate is uni-colored, q is ignored to convert it as a NG case. Finally, the optimality of the algorithm follows from the optimality of the shortest path of the created graph H in the NG case.

Time complexity: The time complexity follows from three scans of S; once each for identifying the members of Q^L and Q^R respectively, and once for scanning each NG case between a pair of consecutive (right-gate, left-gate) to identify the blocks. Next, for solving each NG case, we need to execute the algorithm of Section 5.2 for computing the MCS for a path which also needs time linear in the number of vertices. The time complexity of computing the shortest s-t path in the graph H is also linear in the number of edges in H. The time complexity follows from the fact that the NG cases we solved separately are all vertex disjoint. □

[4]If there is no split vertex, we have only one unsolved part.

5.5 Bi-chromatic Comb Graph

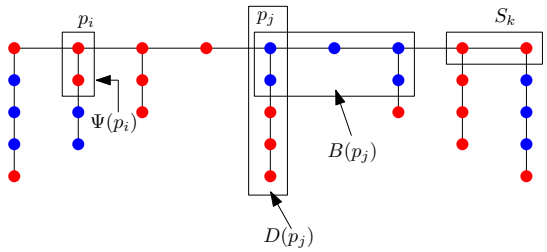


Figure 5.24: A comb graph.

A *comb graph* $G = (V, E)$ consists of a path S of m vertices, called *skeleton*, and at each vertex $p_i \in S$ a path $D(p_i)$ (called *leg*) is dangling. The size of $D(p_i)$ is n_i (≥ 1) that includes p_i also. Here, $V = \cup_{i=1}^{m} D(p_i)$ with $|V| = \sum_{i=1}^{m} n_i = n$. The edges in E are defined by the edges in the path S and the leg $D(p_i)$ at each skeleton vertex $p_i, i = 1, 2, \ldots, m$. Here also, we consider bicolored version of the problem where each vertex is assigned a color in {*red*, *blue*} (see Figure 5.24). The objective is to choose a MCS $\mathcal{C} \subseteq V$ for the graph G of minimum size.

We will use the following notations to describe our algorithm:

- A *run* is a maximal[5] monochromatic path on the skeleton. Assume that $S = S_1 \cup S_2 \cup \ldots \cup S_k$, where S_j is the j-th run of S; k is the number of runs in S.

- For a vertex $p_i \in S$, $\Psi(p_i)$ denotes the run in the path $D(p_i)$ attached to the vertex p_i.

- As in Section 5.4, here also we define a *block* as a connected set of vertices of same color. For an element $p_i \in S_j$, $B(p_i)$ is the block of vertices $S_j \bigcup \cup_{q \in S_j} \Psi(q)$, each of color$(p_i)$. Thus, all nodes of a run belong to the same block, and $B(p_i)$ is same for each p_i in the run S_j.

[5]extension of the path in any direction does not keep it monochromatic

- $L(p_i) \subset S$ is the subset of vertices in S that are to the left of $p_i \in S$. Similarly, $R(p_i) \subset S$ is the subset of vertices in S that are to the right of $p_i \in S$.

We will use the idea of Section 5.2 to formulate the problem as the shortest s-t path problem of an overlay graph $H = (U, F)$ whose vertices are $U = \cup_{i=0}^{m+1} \Psi(p_i)$, where $\Psi(p_0) = s$, and $\Psi(p_{m+1}) = t$, and $\cup_{i=1}^{m} \Psi(p_i)$ is the union of the run of all the legs attached to the skeleton. An edge $(q, r) \in F$, $q \in \Psi(p_\ell)$, $r \in \Psi(p_{\ell'})$ and q, r are not in the same leg, i.e., $\ell \neq \ell'$. The cost of the edge $w(q, r)$ is computed as follows.

For any two vertices p_ℓ and $p_{\ell'}$ any of the two scenarios may occur:

- p_ℓ and $p_{\ell'}$ belong to two adjacent runs: Here, we identify a pair of consecutive vertices $p_\theta, p_{\theta+1} \in S$ from the adjacent blocks with $\text{color}(p_\theta) \neq \text{color}(p_{\theta+1})$ such that p_θ (resp. $p_{\theta+1}$) is nearer to p_ℓ (resp. $p_{\ell'}$). Explained in details in Subsection 5.5.1 as type-1 edge.

- p_ℓ and $p_{\ell'}$ belong to the same block: Here, we identify a pair of consecutive vertices $p_\theta, p_{\theta+1} \in S$ from the same block hence with $\text{color}(p_\theta) = \text{color}(p_{\theta+1})$ such that p_θ (resp. $p_{\theta+1}$) is nearer to p_ℓ (resp. $p_{\ell'}$). Explained in details in Subsection 5.5.1 as type-2 edge.

Now, $w(q, r)$ is the sum of the sizes of the consistent subsets of (i) $D(p_\alpha)$ for all vertices p_α on the skeleton from p_ℓ to p_θ with q as the only vertex in the last run of the path $D(p_\alpha) \oplus \{p_\alpha, \ldots, q\}$, and (ii) $D(p_\beta)$ for all vertices p_β on the skeleton from $p_{\ell'}$ to $p_{\theta+1}$ with r as the only vertex in the last run of the path $D(p_\beta) \oplus \{p_\beta, \ldots, r\}$, where $\oplus$ is the concatenation operator.

Before describing the algorithm, we first state the following preprocessing phase.

5.5.1 Preprocessing and Algorithm:

Preprocessing:

For each vertex $q \in \cup_{i=1}^{m} \Psi(p_i)$, we create an array σ_q of size m as stated below. Let p_i and p_j belong to the same run, say S_α, of the skeleton, and $q \in \Psi(p_j)$. The i-th element of the array of σ_q contains the size of the *constrained* MCS of $D(p_i)$, denoted by $\mathcal{C}(D(p_i), q)$, with the constraint that only q ($\in B(p_i)$) from that particular run is in that consistent subset (see Figure 5.25).

Step 1: Generation of $\mathcal{C}(D(p_i), q)$ for all the vertices $q \in \cup_{i=1}^{m} \Psi(p_i)$.

- Let $p_i \in S_\alpha$. Compute $\mu = \max_{q \in B(p_i)} \text{hop-distance}(p_i, q)$, and let X be a chain of μ vertices of $color(p_i)$. Create a path $\Pi = D(p_i) \oplus X$. The first run of Π starts from the leaf vertex of $D(p_i)$ and its last run is $\Psi(p_i) \oplus X$.
- Create the overlay graph H for the path Π as in Section 5.2; the vertex s of H is connected to all the members in the first run of Π, and vertex t of H is connected to all the vertices in the last run of Π. Each vertex $q \in \Pi$ is attached with a weight field $w(q)$, initialized with ∞.
- We execute the algorithm of Section 5.2 on the path Π. After execution of the algorithm, the $w(q)$ field for each vertex q of the last run will contain the size of the MCS of $D(p_i)$ that contains the vertex q.
- Note that, as in Section 5.2, here also in the MCS of Π, the members present from both the first run and the last run are exactly one. Thus, for each vertex $q \in \Pi$, if the weight $w(q) = \infty$ then it implies that there does not exist any consistent subset of $D(p_i)$ with only the vertex q in the last run of Π.

Step 2: Assign the value of $\sigma_q(i)$ $(= \mathcal{C}(D(p_i), q))$ for each vertex $q \in \cup_{p_j \in B(p_i)} \Psi(p_j)$ with the w value of the θ-th elements of Π where $\theta = |\Psi(p_i)| + \text{hop-distance}(q, p_i)$.

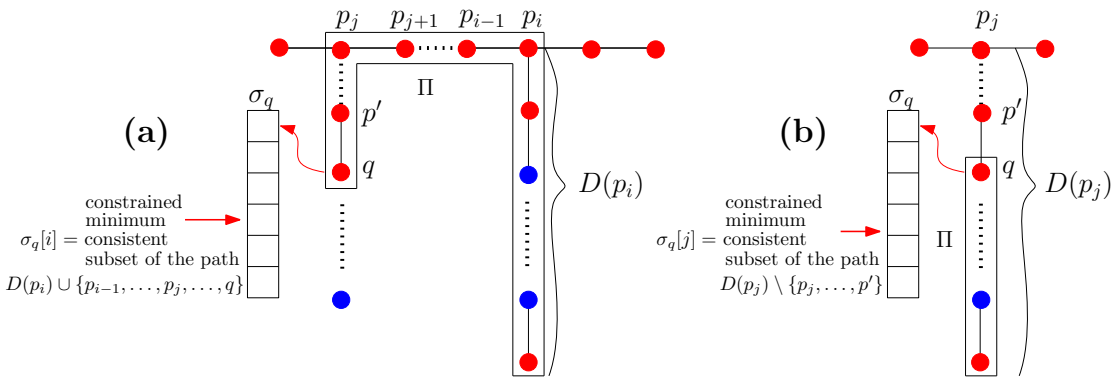


Figure 5.25: Data structure σ_q: (a) $\sigma_q(i)$ for $j \neq i$, (b) $\sigma_q(i)$ for $j = i$

Algorithm:

As in Section 5.4, here also we define three types of *gates*. Each gate is a tuple (q, p, r) of vertices in V, where $p \in S$ may be of any color; vertices q and r are of opposite colors. Unlike the case of caterpillar, here q, r may not always be adjacent to p. However, q and r are equidistant from p and all the vertices on the path from p to q (resp. p to r), excluding p, are of same color as that of q (resp. r).

Before defining the gates, we define $\Delta = \{q, r\} \cup \mathcal{C}(D(q), q) \cup \mathcal{C}(D(r), r)$ as a quantity which will be required for computing the minimum consistent set with any gate (q, p, r) such that $q, r \in \mathcal{C}$. The mathematical significance of Δ is that it consists of the constrained minimum consistent subset of $D(q)$ (resp. $D(r)$) given q (resp. r) is chosen in the MCS along with the vertices in the gate. Thus when a particular type of gate is chosen, Δ is an invariant for all the computations.

left-gate $\mathcal{L}_{q,p_i,r}$: It is a tuple (q, p_i, r), where $p_i \in S$, $r \in D(p_i)$, $q \in D(p_j)$, $p_j \in L(p_i)$ such that hop-distance(q, p_i) = hop-distance(p_i, r), and all the vertices on the path from q to p_{i-1} are of color(q), and all the vertices on the path segment $\Psi(p_i) \setminus \{p_i\}$ are of color(r) (see Figure 5.26((i)). Here p_i is referred to be the *base vertex* for this left-gate. If $\{q, r\}$ of this left-gate is included in $\mathcal{C}$, then all the vertices in $R(p_i)$ (right side of the vertex p_i on

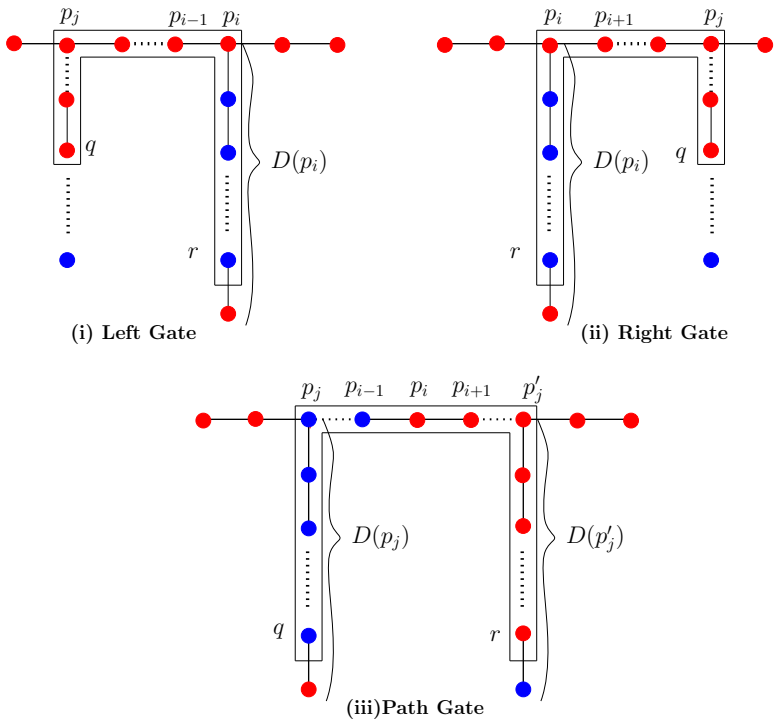


Figure 5.26: Gates in *comb graph*.

the skeleton) along with their dangling legs are covered. Thus, we have

$$\mathcal{C} = \Delta \bigcup (\cup_{p_\theta \in \{p_j, \ldots, p_{i-1}\}} \mathcal{C}(D_{p_\theta}, q)) \bigcup \mathcal{C}(L(p_j), q).$$

Note that, for all $p_\theta \in \{p_{i-1}, \ldots, p_j\}$, (D_{p_θ}, q) is already computed in pre-processing step and stored in σ_q. Moreover, $L(p_j)$ does not contain any left gate.

right-gate $\mathcal{R}_{q,p_i,r}$: It is a tuple (q, p_i, r), where $p_i \in S$, $r \in D(p_i)$, $q \in D(p_j)$, $p_j \in R(p_i)$ such that hop-distance(q, p_i) = hop-distance(p_i, r), and all the vertices on the path from q to p_{i+1} are of color(q) and all the vertices on the path segment $\Psi(p_i) \setminus \{p_i\}$ are of color(r) (see Figure 5.26((ii)). Here p_i is referred to be the *base vertex* for this right-gate. If $\{q, r\}$ of this right-gate is included in $\mathcal{C}$, then all the vertices in $L(p_i)$ (left side of the vertex p_i on

the skeleton) along with their dangling legs are covered. Thus, we have

$$\mathcal{C} = \Delta \bigcup (\cup_{p_\theta \in \{p_j, \ldots, p_{i+1}\}} \mathcal{C}(D_{p_\theta}, q)) \bigcup \mathcal{C}(R(p_j), q).$$

path-gate $\mathcal{P}_{q,p_i,r}$: It is a tuple (q, p_i, r), $q \in D(p_j)$, $r \in D(p_{j'})$; $p_j, p'_j \in S$ are respectively in the left and right sides of p_i and hop-distance(q, p_i) = hop-distance(p_i, r) (Figure 5.26((iii)). Here p_i is referred to be the *base vertex* for this path-gate. If $\{q, r\}$ of this path-gate is included in $\mathcal{C}$, then all the vertices in $D(p_i)$ (dangling at p_i) are covered. Thus we have L for the left uncovered part of p_i and R for the right uncovered part of p_i which will have to be combined with Δ to get the overall MCS.

$$L = \left(\cup_{\theta=j}^{i-1} \mathcal{C}(D(p_\theta), q)\right) \bigcup \left(\cup_{p_\theta \in L(p_j)} \mathcal{C}(D(p_\theta), q)\right)$$

$$R = \left(\cup_{\theta=i+1}^{j'} \mathcal{C}(D(p_\theta), r)\right) \bigcup \left(\cup_{p_\theta \in R(p_{j'})} \mathcal{C}(D(p_\theta), r)\right)$$

$$\mathcal{C} = \Delta \cup L \cup R$$

Arguing as in Section 5.4, here also if left-gates (resp. right-gates) are present in the problem instance, then the *base vertex* of the left-most (resp. right-most) left-gates (resp. right gates) needs to be considered. However, in both the cases, the q and r vertex need to be appropriately chosen to minimize the size of $\mathcal{C}$.

In a sequential scan from left to right, we can identify the *base vertex* p_ℓ of the left-most left-gate $\mathcal{L}_{q,p_\ell,r}$, and the *base vertex* p_ρ of the right-most right-gate $\mathcal{R}_{q,p_\rho,r}$ and the base vertex of all possible path gates. If left gate (resp. right gate) is not present, then we set $\ell = -\infty$ (resp. $\rho = \infty$). As in Section 5.4, here also, we need to consider the four situations, namely **OLG**, **ORG**, **BG** and **NG**, where **NG** case may include path gate(s).

Handling OLG and ORG Case

If there exists no right-gate, and has at least one left-gate, then we identify $\mathcal{L}_{q,p_\ell,r}$ as the left-most left-gate. We include $\{q, r\}$ in $\mathcal{C}$, and compute $\mathcal{C}$ as described for **left-gate**. Here, $\mathcal{C}(L(p_\ell), q)$ is processed as **NG** case. The **ORG** case is similarly handled for a right gate $\mathcal{R}_{q,p_\rho,r}$.

Handling BG Case

- if $\ell \leq \rho$, then we need to compute (i) the size of $\mathcal{C}$ considering only the left-most left-gate using the method for **OLG** case, and also (ii) the size of $\mathcal{C}$ considering only the right-most right-gate as in **ORG** case. The minimum of them is reported as the optimum $\mathcal{C}$.

- if $\ell > \rho$, then the $R(p_\ell)$ and the $L(p_\rho)$ part are already covered. We need to compute the consistent subset of the portion $L(p_\ell) \bigcap R(p_\rho)$ as the **NG** case.

  Note that, as in Section 5.4, here the special case will not arise due to the fact that we have considered all possible left-gates with p_ℓ as the base vertex, and all possible right-gates with p_ρ as the base vertex.

Handling NG Case

Now, we will explain the processing of the portion of S as the *NG* instance. From now onwards, we will use T for this portion of the given instance of the problem. We may have $T = S$. We create a multi-partite overlay graph $H = (U, F)$ with the vertices $U = U_0 \cup U_1 \cup U_2 \cup \ldots \cup U_m \cup U_{m+1}$ where U_i corresponds to $p_i \in S$, and its vertices correspond to the elements of $\Psi(p_i)$, $i = 1, 2, \ldots, m$; $U_0 = \{s\}$ and $U_{m+1} = \{t\}$. Let us remind that, if the last vertex of the MCS ($\mathcal{C}(D(p_i), q)$) of the leg $D(p_i)$ is $q \in \Psi(p_j)$ (p_i, p_j belongs to the same run of S) then $|\mathcal{C}(D(p_i), q)|$ is available in $\sigma_q(i)$.

Needless to say, there is no edge between any pair of vertices in U_i, $i = 1, \ldots, m$. For a pair of sets U_ℓ and $U_{\ell'}$, we put edge between every pair of vertices (q, r), where $q \in U_\ell$ and $r \in U_{\ell'}$. The edge weights are computed as follows:

Type-0 edge: The vertex $s \in U_0$ is connected with every vertex of $U_1 \cup \ldots U_{r_1}$, where $p_1, \ldots, p_{r_1} \in S_1$ (where r_1 is the length of S_1, the first run of S). If $q \in U_\ell$, where $\ell \leq r_1$, then the weight of the directed edge (s, q) is $\omega(s, q) = \sum_{i=1}^{\ell-1} \sigma_q(i) + \frac{1}{2}\sigma_q(\ell)$.

Type-1 edge: For a pair of vertices (q, r) where $q \in U_\ell$ and $r \in U_{\ell'}$ and the corresponding elements $p_\ell \in S_\alpha$ and $p_{\ell'} \in S_{\alpha+1}$ then the cost $\omega(q, r)$ of the *type-1* edge (q, r) is computed as follows (see Figure 5.27):

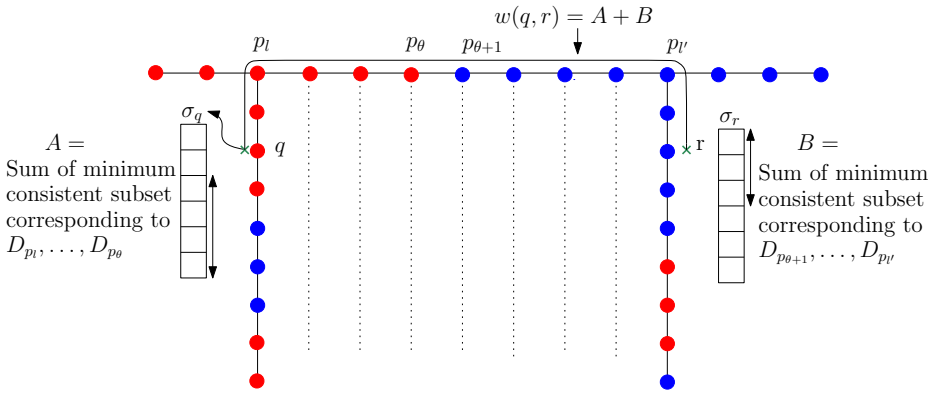


Figure 5.27: Type 1 edge.

- Let p_θ and $p_{\theta+1}$ be two consecutive elements in S that belong to S_α and $S_{\alpha+1}$ respectively. If the consistency condition for p_θ $(\theta - \ell \leq \ell' - \theta)$ and for $p_{\theta+1}$ $((\theta + 1) - \ell \leq \ell' - (\theta + 1))$ are satisfied then we set $\omega(q, r) = \frac{1}{2}\sigma_q(\ell) + \sum_{j=\ell+1}^{\alpha} \sigma_q(j) + \sum_{j=\alpha+1}^{\ell'-1} \sigma_r(j) + \frac{1}{2}\sigma_r(\ell')$;
- otherwise we set $\omega(q, r) = \infty$.

Type-2 edge: For a pair of vertices (q, r) where $q \in U_\ell$ and $r \in U_{\ell'}$ and both the corresponding elements $p_\ell, p_{\ell'} \in S_\alpha$ with $\lambda = \lceil \frac{\text{hop-distance}(q,r)}{2} \rceil$, then the cost $\omega(q, r)$ of the *type-2* edge (q, r) is computed as follows (see Figure 5.28):

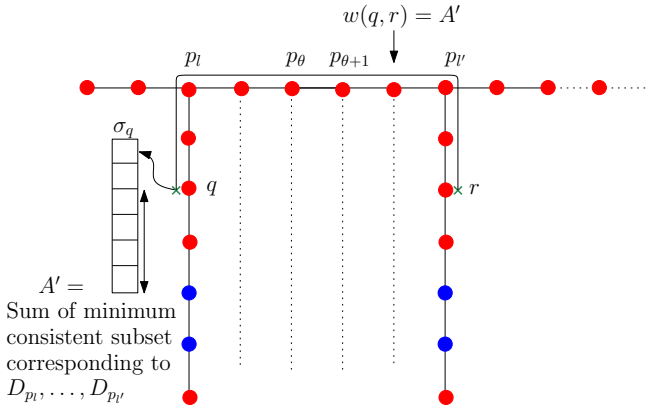


Figure 5.28: Type 2 edge.

- If $\lambda >$ max(hop-distance(q, p_ℓ), hop-distance$(r, p_{\ell'})$) then we can get $p_\theta \in S$ such that hop-distance$(q, p_\theta) = \lambda$. In such a case, for all elements in $\Psi(p_\ell), \ldots \Psi(p_\theta)$ the nearest element will be q, and for all elements $\Psi(p_{\theta+1}), \ldots \Psi(p_{\ell'})$ the nearest element will be r, and we set $\omega(q,r) = \frac{1}{2}\sigma_q(\ell) + \sum_{j=\ell+1}^{\lambda} \sigma_q(j) + \sum_{j=\lambda+1}^{\ell'-1} \sigma_r(j) + \frac{1}{2}\sigma_r(\ell')$;
- otherwise we set $\omega(q,r) = \infty$.

Type-0′ edge: $U_\beta \cup \ldots \cup U_m$ is connected with $t \in U_{m+1}$, where $p_\beta, \ldots, p_m \in S_k$ (last run of S). If $q \in U_\ell$, where $\beta \leq \ell \leq m$, then the weight of the directed edge (q,t) is $\omega(q,t) = \frac{1}{2}\sigma_r(\ell) + \sum_{i=\ell+1}^{m} \sigma_q(i)$.

Now, a shortest path from s to t in the overlay graph H will give the size of the MCS $\mathcal{C}$ for T. The following two notes are important for the correctness of the algorithm:

1 We need to specifically mention that the weight of an edge is equal to the sum of the size of the MCS of the legs whose corresponding elements in S are covered by that edge. Thus, for a valid edge (q,r) (i.e., $\lambda >$ max(hop-distance(q, p_ℓ), hop-distance$(r, p_{\ell'})$)), if any one of $\sigma_q(j)$ in the first sum or any one of $\sigma_r(j)$ in the second sum is ∞, then $\omega(q,r)$ is set to ∞.

2 In order to avoid duplicate counting, we have added 1/2 of the cost of each (one or two) terminal leg covered by an edge to the weight of that edge.

5.5.2 Correctness and complexity

Theorem 5.6

The aforesaid algorithm correctly computes the minimum consistent subset of a comb graph in $O(m(m+n))$ time, where $m = |S|$ (the size of the skeleton) and $n = |V|$ (the total number of vertices in the input graph G).

Proof. The correctness of the algorithm follows from the arguments in Sections 5.2 and 5.4, and the construction and weight assignment of the overlay graph H. The total weight of the edges in the shortest path of the graph H is exactly the size of the minimum consistent subset $\mathcal{C}$ of the graph G.

The worst case time complexity of the algorithm is dominated by the execution for the **NG** case, and is discussed below.

- In the graph H, a vertex $q \in \Psi(p_i)$, where $p_i \in S_\alpha$ is connected with at most 3 vertices of each $\Psi(p_j)$ (see Figure 5.3), $p_j \in S_{\alpha+1}$ using Type-1 edge, indicating that at most $O(\xi)$ Type-1 edges are incident at q, where ξ is the length of the largest run in S. By a similarly argument, it can be shown that at most $O(\xi)$ Type-2 edges are incident at q. The number of Type-0 and Type-0′ edges is at most $O(n)$. Thus, the total number of edges is $O(n\xi)$. The creation of every edge needs adding the size of the consistent subsets of different legs with sink vertex at the terminal vertices of that edge. This again needs $O(\xi)$ time. However, it is possible to compute the edge costs of all the edges in an amortized $O((m+n)\xi)$ time.
- The total number of edges in the graph is $O(n\xi)$. Dijkstra's shortest path algorithm needs $O(n\xi + n \log n)$ time.

As $\xi = O(m)$ in the worst case, the time complexity follows. □

CHAPTER 6

Minimum Consistent Subset in Trees

Contents

6.1 Organization

In this chapter, we will study the MCS problem for undirected bi-colored trees $\mathcal{T} = (V, E)$ whose vertices are assigned colors $\{red, blue\}$. We now describe the chapter section-wise. In Section 6.2 some of the preliminary concepts required to solve the minimum consistent problem have been discussed. In Section 6.3 we propose a polynomial time algorithm for the problem. Finally in Section 6.4, we propose a simple algorithm that produces a 2×1.386 factor approximation result. It establishes the connection of consistent subset problem for trees with the Steiner tree problem of a graph. We apply the best-known 1.386 factor approximation algorithm of an undirected weighted graph using LP-relaxation [BGRS10] to solve this problem[1]. In the rest of the chapter, we will use $\mathcal{C}$ to denote a minimum consistent subset of the input graph $\mathcal{T}$.

6.2 Preliminaries

We use $\mathcal{C} \subseteq V$ to denote a consistent subset of the tree $\mathcal{T} = (V, E)$ of minimum cardinality. We will use $u \longrightarrow v$ to denote a path between u and v in $\mathcal{T}$, and $dist(u, v)$ to denote the length of the path $u \longrightarrow v$ in $\mathcal{T}$.

Observation 6.1

If all the vertices in $\mathcal{T}$ are of the same color, then $\mathcal{C}$ consists of a single vertex (any vertex forms a MCS) of $\mathcal{T}$. If $\mathcal{T}$ can be arranged as a rooted tree with an appropriate vertex, say $\rho \in V$, as the root, such that the vertices in each alternate level of $\mathcal{T}$ are of a different color then $\mathcal{C} = V$.

Thus, we will consider the problem of computing $\mathcal{C}$ for the tree $\mathcal{T}$ where $\mathcal{T}$ does not satisfy Observation 6.1 and propose a polynomial-time algorithm. From now onwards, the term *covered* for a set of vertices implies that those vertices are

[1] Considering time complexity, it is not an efficient algorithm even though the quality of the solution is compromised.

consistently covered, i.e., the vertices satisfy the property of being consistent with some vertex in $\mathcal{C}$. We will use the following notations to describe our algorithm.

- A path $u \longrightarrow v$ of the same color is referred to as a *run*.
- Let v and x be two vertices in $\mathcal{T}$, and let v_x be the neighbor of v such that if we delete the edge (v, v_x) from $\mathcal{T}$, then $\mathcal{T}$ is split into two sub-trees, one (rooted at v_x) containing the vertex x, and the remaining part contains the vertex v. The sub-tree containing v will be referred to as T_v^{-x}. Similarly, for a triple of vertices v, x and y, $T_v^{-(x,y)}$ is obtained by deleting the sub-trees rooted at v_x and v_y by removing the edges (v, v_x) and (v, v_y).

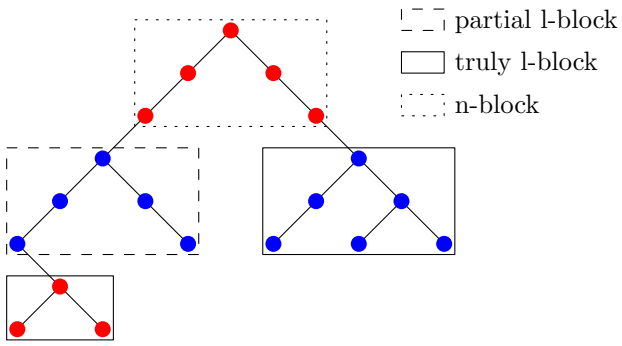


Figure 6.1: Illustration of n_block and ℓ_block.

6.1: Block

A block in a tree is defined as a connected set of vertices of the same color. We have two types of blocks, namely *leaf blocks* and *non-leaf blocks* (see Figure 6.1). A *leaf block* consists of at least one leaf vertex of $\mathcal{T}$, and is denoted as ℓ_block. An ℓ_block having exactly one vertex connected with a vertex of another color, through which this ℓ_block is connected with the rest of the tree, is called a *true* ℓ_block; if an ℓ_block has more than one vertex connected with vertices of other color, then it is called a *partial* ℓ_block. A *non-leaf block* does not contain any leaf vertex of $\mathcal{T}$, and is denoted as n_block. See Figure 6.1 for the demonstration.

6.2: Gate

A gate $\Gamma(u, w)$ is defined by a tuple (u, w), $u, w \in V$ such that (a) u and w are of different colors, (b) there exists *exactly* two runs of different colors on the path $u \longrightarrow w$ in $\mathcal{T}$, and (c) the difference in the number of vertices in these two runs is at most 1 (see Figure 6.2). If the number of vertices on the path $u \longrightarrow w$ is odd, then the middle-most vertex v on this path is referred to as the *anchor vertex* of $\Gamma(u, w)$. If the number of vertices on the path $u \longrightarrow w$ is even, then there is no anchor vertex in that gate. However, then the middle-most edge is referred to as the *anchor edge* of $\Gamma(u, w)$.

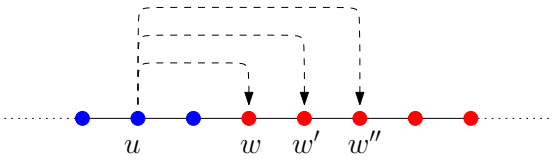


Figure 6.2: Illustration of gates $\Gamma(u, w)$, $\Gamma(u, w')$, $\Gamma(u, w'')$.

Observation 6.2

Every bi-colored tree $\mathcal{T}$ of size greater than three that does not satisfy Observation 6.1, has at least one gate.

Observation 6.3

Let the number of vertices in the path $u \longrightarrow w$ of a gate $\Gamma(u, w)$ is odd. Let v be the anchor vertex of $\Gamma(u, w)$, $N(v)$ be the set of neighbors of v, and for the pair of neighbors $v_u, v_w \in N(v)$, the subtrees T_{v_u} and T_{v_w}, obtained by deleting the edge (v, v_u) and (v, v_w) from $\mathcal{T}$, contains u (in T_{v_u}) and w (in T_{v_w}), respectively. Now, the inclusion of (u, w) in $\mathcal{C}$ causes all the vertices in $\mathcal{T} \setminus (T_{v_u} \cup T_{v_w})$ to be consistently covered (see Figure 6.3).

Proof. Follows from the fact that both the paths from every vertex in $T_v^{-(u,w)}$ to u and w passes through v and; u and w are equidistant from v. Thus, all of the vertices in $T_v^{-(u,w)}$ are covered by inclusion of (u, w) in $\mathcal{C}$. □

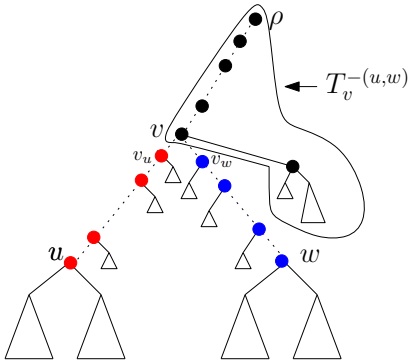


Figure 6.3: The covered tree $T_v^{-(u,w)} = \mathcal{T} \setminus (T_{v_u} \cup T_{v_w})$

6.3: Sibling Gate

In a rooted tree $\mathcal{T}$, a *gate* $\Gamma(u, w)$ will be referred to as a sibling gate if the number of vertices in the path $u \longrightarrow w$ is odd, and the anchor v of $\Gamma(u, w)$ is the predecessor of both u and w in $\mathcal{T}$. We denote this type of gate as $\Gamma_{sib}(u, v, w)$.

As mentioned earlier, if $\{u, w\}$ of $\Gamma_{sib}(u, v, w)$ is included in $\mathcal{C}$ then all the vertices of the tree $T_v^{-(u,w)}$ are consistently covered.

Observation 6.4

Let $\Gamma_{sib}(u, v, w)$ be a sibling gate in $\mathcal{T}$; v_u and v_w are two children of v such that the sub-trees rooted at v_u and v_w contain u and w, respectively. Now, if there exists a sibling gate $\Gamma_{sib}(\hat{u}, \hat{v}, \hat{w})$ in any of the sub-trees rooted at v_u and v_w (see Figure 6.4), then the set of vertices consistently covered by $\{u, w\}$ is a subset of vertices consistently covered by $\{\hat{u}, \hat{w}\}$.

Proof. If there exists a sibling gate $\Gamma_{sib}(\hat{u}, \hat{v}, \hat{w})$ in any of the sub-trees rooted at v_u and v_w, then the set of vertices consistently covered by $\{u, w\}$ is a subset of vertices consistently covered by $\{\hat{u}, \hat{w}\}$. In that case, the size of the consistent set of T_v cannot be smaller than that of $T_{\hat{v}}$, and hence there is no point in computing $\mathcal{C}(T_v)$. □

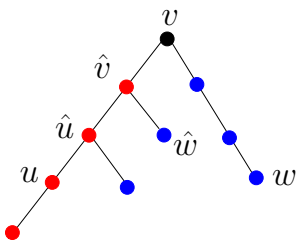


Figure 6.4: Nested sibling gate.

Observation 6.4 leads to the following definition for choosing a sibling gate for inclusion in $\mathcal{C}$.

6.4: Useful Sibling Gate

A sibling gate $\Gamma_{sib}(u, v, w)$ is said to be a useful sibling gate if the two subtrees of v containing the vertices u and w respectively do not contain any other sibling gates.

Assume that $\mathcal{T}$ doesn't satisfy Observation 6.1. Let us consider an MCS $\mathcal{C}$ of a tree $\mathcal{T}$. Let us consider a pair of bi-chromatic vertices (u, w) in $\mathcal{C}$ such that u is red and w is blue and among all such bi-chromatic pairs in $\mathcal{C}$ they have the minimum distance in $\mathcal{T}$. Observe that this pair of vertices forms a gate $\Gamma(u, w)$. There are two cases depending on whether the number of vertices in the path $u \longrightarrow w$ is even or odd.

First, consider the case it is odd. By Observation 6.3, the middlemost vertex v in the path $u \longrightarrow w$ is an anchor vertex. We orient the tree $\mathcal{T}$ such that its root is v. Let T_v denote the tree rooted at v. Now $\mathcal{T}$ has a minimum consistent subset $\mathcal{C}$ such that $\mathcal{C}$ contains a pair of vertices constituting a sibling gate with the root of T_v as the anchor.

Now consider the case that the number of vertices in the path $u \longrightarrow w$ is even. Here we have an anchor edge (see Definition 6.2). We introduce a *fictitious* vertex v on the anchor edge and root the tree $\mathcal{T}$ at this vertex (as if the tree is 'rooted at the anchor edge'). In this case, also, T_v has a minimum consistent subset $\mathcal{C}$ such that $\mathcal{C}$ contains a pair of vertices constituting a sibling gate with the root of T_v as

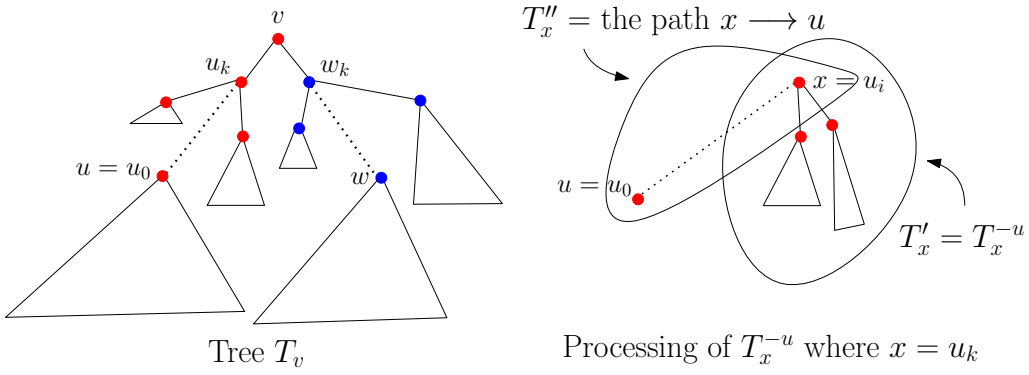


Figure 6.5: Processing of a useful sibling gate

the anchor. Thus we have,

Lemma 6.1

Assume that $\mathcal{T}$ doesn't satisfy Observation 6.1. There exists a vertex v (real or fictitious), such that if $\mathcal{T}$ is rooted at v, $\mathcal{T}$ has a minimum consistent subset $\mathcal{C}$ such that $\mathcal{C}$ contains a pair of vertices constituting a sibling gate with v as its anchor.

We now explain the main structure of our algorithm for the computation of MCS for a bi-colored tree $\mathcal{T}$. See Algorithm 6.1 for a pseudo-code. If $\mathcal{T}$ satisfies Observation 6.1, then the computation of MCS is straightforward. Otherwise, we compute MCS of all the rooted trees at the anchor vertices of the gates as stated in Lemma 6.1. Among all the rooted trees, the MCS of $\mathcal{T}$ will be the one that is of the smallest size.

Next, we briefly discuss each of the steps of Algorithm 6.1. Step 1 can be accomplished by rooting $\mathcal{T}$ at an arbitrary vertex and then checking whether conditions for Observation 6.1 are met. Step 2 requires rooting tree $\mathcal{T}$ at n vertices of $\mathcal{T}$ and $n-1$ fictitious vertices corresponding to each edge of $\mathcal{T}$. In Step 3, we need to check whether the root v of T_v is an anchor of a useful sibling gate. A simple procedure is outlined in Algorithm 6.2 for using in Step 3. Step 4 is discussed in detail in Section 6.3 where we compute the MCS of each rooted tree $\mathcal{C}(T_v)$. Finally, we show that Algorithm MCS($\mathcal{T}$) is correct and runs in polynomial time.

Algorithm 6.1: MCS($\mathcal{T}$)

Input: An un-rooted tree $\mathcal{T} = (V, E)$
Output: An MCS $\mathcal{C}$ for the tree $\mathcal{T}$

```
1  //STEP 1;
2  if T is monochromatic or T is alternating then
3  |  report MCS and return;
4  end
5  //STEP 2;
6  R = ∅;
7  for each vertex v of T do
8  |  root T at v, and let T_v be the resulting rooted tree;
9  |  R = R ∪ {T_v};
10 end
11 for each edge e of T do
12 |  Root T at a fictitious vertex v on e, let T_v be the rooted tree;
13 |  R = R ∪ {T_v};
14 end
15 //STEP 3;
16 for each tree T_v ∈ R do
17 |  if the root v is not an anchor of a useful sibling gate then
18 |  |  R = R \ {T_v};
19 |  end
20 end
21 //STEP 4;
22 for each tree T_v ∈ R do
23 |  Compute C(T_v) of T_v;
24 end
25 //STEP 5;
26 C = min_{T_v ∈ R} |C(T_v)|;
27 return C;
```

Algorithm 6.2: IsAnchorUseful(T_v)

Input: A rooted tree T_v

Output: Whether v is an anchor of a useful sibling gate

```
1  A = φ;
2  for each vertex x ∈ T_v do
3  |   x = unmarked;
4  end
5  Perform a post-order traversal on T_v and store it in P;
6  for each vertex p ∈ P do
7  |   if p is marked then
8  |   |   predecessor(p) = marked ;
9  |   end
10 |   if p is unmarked ∧ p has a pair of children of opposite colors then
11 |   |   A = A ∪ {p};
12 |   |   p = marked;
13 |   |   parent(p) = marked;
14 |   end
15 end
16 if any successor of v in A then
17 |   return NO;
18 else
19 |   return YES;
20 end
```

6.3 Computing MCS of a tree rooted at an anchor

In this section, we show how to compute minimum consistent subset $\mathcal{C}(T_v)$ for a tree T_v rooted at an anchor vertex v. Note that v is an anchor vertex (either a vertex of $\mathcal{T}$ or a fictitious vertex) corresponding to a sibling gate $\Gamma(u, v, w)$ (see Definition 6.3). Moreover, we can assume that $\Gamma(u, v, w)$ is a useful sibling gate (see Definition 6.4).

We first note that there may be several useful sibling gates that are anchored at the root v of T_v. Traverse each pair of bi-colored runs incident at v. Let $\Pi_{red} = \{u_1, u_2, ...\}$ and $\Pi_{blue} = \{w_1, w_2, ...\}$ be a pair of such runs, and $k = \min\{|\Pi_{red}|, |\Pi_{blue}|\}$. Let $u \in \Pi_{red}$ and $w \in \Pi_{blue}$ be two vertices having hop-distance k from v along the paths Π_{red} and Π_{blue}. Now, every pair of vertices (u_i, w_i) forms a useful sibling gate $\Gamma_{sib}(u_i, v, w_i)$, where $i = 1, 2, \ldots, k$, $u_i \in \Pi_{red}$, $w_i \in \Pi_{blue}$. Next we outline the computation of MCS with respect one of these useful sibling gates. Overall MCS of T_v is the minimum among the size of MCS for all these sibling gates.

The MCS $\mathcal{C}(T_v)$ for a useful sibling gate $\Gamma_{sib}(u, v, w)$ is computed using Equation 6.1, stated below.

$$\mathcal{C}(T_v) = \{u, w\} \bigcup \cup_{u_i \in U} \mathcal{C}_u(T_{u_i}^{-u}) \bigcup \cup_{w_i \in W} \mathcal{C}_w(T_{w_i}^{-w}), \tag{6.1}$$

where $U = \{u_0 = u, u_1, \ldots, u_k = v_u\}$, and $W = \{w_0 = w, w_1, \ldots, w_k = v_w\}$ (see Figure 6.5), and $\mathcal{C}_a(T_x^{-a})$ for a pair of vertices $a, x \in T_v$, where a and x belong to the same block, is defined as follows.

Observation 6.5

Let a and x be two vertices in a block. We use $\mathcal{C}_a(T_x^{-a})$ to denote the MCS of a subtree T_x^{-a}, assuming that the vertex x is consistently covered by the vertex $a \in \mathcal{C}_a(T_x^{-a})$ (see Figure 6.6(a)). In other words, there does not exist $y \in \mathcal{C}_a(T_x^{-a})$ with $color(y) = color(x) = color(a)$ such that $dist(x, y) < dist(x, a)$. Surely, in order to maintain the consistent covering of x, there does not exist any vertex $\zeta \in \mathcal{C}_a(T_x^{-a})$ of $color(\zeta) \neq color(a)$ (i.e., ζ in an adjacent block $z \longrightarrow z'$ of the block containing x) with $dist(x, \zeta) < dist(a, x)$.

Remark 1: There may not exist any such consistent set of the tree T_x^{-a}. This occurs when the run $\{z \longrightarrow z'\}$ of different color in T_x^{-a} closest to x satisfy $dist(x, z') < dist(x, a)$. Here, in order to consistently cover the vertices on the path $z \longrightarrow z'$ one needs to choose a vertex in the run $\{z \longrightarrow z'\}$ in $\mathcal{C}_a(T_x^{-a})$, which will make the vertex x inconsistent (see Figure 6.6(b)).

Remark 2: There may exist situation where the length of the run $\{x \longrightarrow x'\}$ (of $color(a)$) is greater than the length of its adjacent run $\{z \longrightarrow z'\}$ of other color (see Figure 6.6(c)). In such a case, one chooses another vertex $a' \in \{x \longrightarrow x'\}$ satisfying $dist(x, a') \geq dist(x, a)$, (i.e., maintaining x to cover by a) to include in $\mathcal{C}_a(T_x^{-a})$, such that a vertex $\zeta \in \{z \longrightarrow z'\}$ may be chosen to have a gate $\Gamma(a', \zeta)$, and the vertices in the path $z \longrightarrow z'$ be covered by including $\zeta \in \mathcal{C}_a(T_x^{-a})$.

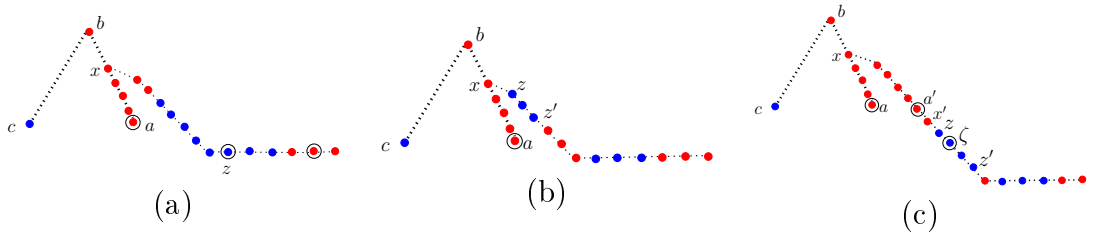


Figure 6.6: Demonstration of $\mathcal{C}_a(T_x^{-a})$: (a) $dist(x, z) > dist(a, x)$, (b) $dist(x, z') < dist(x, a)$, and (c) $dist(x, x') >> dist(z, z')$; we choose another vertex $a' \in x \longrightarrow x'$ (satisfying $dist(x, a') \geq dist(x, a)$) to have a next feasible gate $\Gamma(a, \zeta)$, $\zeta \in z \longrightarrow z'$.

Lemma 6.2

For a useful sibling gate $\Gamma_{sib}(u, v, w)$, the following holds

(a) Let $z, y \in U$, and $z \neq y$. The computation of the consistent subset $\mathcal{C}_u(T_z^{-u})$ does not affect the computation of $\mathcal{C}_u(T_y^{-u})$, and vice-versa.

(b) Similarly, if $z \in U$ and $y \in W$, then the computation of the consistent subset $\mathcal{C}_u(T_z^{-u})$ does not affect the computation of $\mathcal{C}_w(T_y^{-w})$, and vice-versa.

Proof. For part (a), let z be closer to u than y, and the color of u, z, y be *red*. The subtrees T_z^{-u} and T_y^{-u} are disjoint. Let a be the *blue* vertex that is closest to z in $\mathcal{C}_u(T_z^{-u}))$ and b be the *blue* vertex that is closest to y in $\mathcal{C}_u(T_y^{-u})$. We now consider the following situations independently:

Case 1: $\mathcal{C}_u(T_z^{-u})$ contains a *red* vertex a' that lies on the path $z \longrightarrow a$. Here, since z is covered by u, $dist(z, a') \geq dist(u, z)$. Observe that, there exists no vertex

$x \in T_y^{-u}$ which is only covered by the vertex a' since the path $x \longrightarrow a'$ passes through z, and $dist(z, a') \geq dist(u, z)$.

Case 2: $\mathcal{C}_u(T_z^{-u})$ does not contain any *red* vertex lying on the path $z \longrightarrow a$. Here the blue vertex $a \in T_z^{-u}$ does not violate the consistency of any *red* vertex x on the path $y \longrightarrow b$ in the tree T_y^{-u} since the path $x \longrightarrow a$ passes through the vertex z and the *red* vertices on the path $z \longrightarrow a$ are consistently covered.

Case 3: $\mathcal{C}_u(T_y^{-u})$ contains a *red* vertex b' lying on the path $y \longrightarrow b$. Using similar argument as in Case 1, it can be shown that there exists no vertex $x \in T_z^{-u}$ which is only covered by the vertex b'

Case 4: $\mathcal{C}_u(T_y^{-u})$ does not contain any *red* vertex lying on the path $y \longrightarrow b$. Using similar argument as in Case 2, it can be shown that the blue vertex $b \in T_y^{-u}$ does not violate the consistency of any *red* vertex x on the path $z \longrightarrow a$ in the tree T_z^{-u}.

For part (b), remember that the color of z and y are *red* and *blue*, respectively. The result follows from the fact that none of the (*red*) vertices on the path $v \longrightarrow z$ can be covered by some (*blue*) vertex in the tree $T_{v_w}^{-w}$ (See Observation 6.4, v_w is the child of v along the path that contains w), since these are closer to u than w, and every vertex $x \in \mathcal{C}_w(T_y^{-w})$ satisfies $dist(x, y) \geq dist(x, w)$. □

6.3.1 Computation of $\mathcal{C}(T_z)$

Recall Equation 6.1, and consider the processing of the uncovered sub-trees T_z rooted at every vertex z on the path $v \longrightarrow u$ of the sibling gate $\Gamma_{sib}(u, v, w)$. The processing of the uncovered sub-trees rooted at the vertices on the path $v \longrightarrow w$ is analogous. While processing the vertex z on the path $v \longrightarrow u$, we will consider $T_z = T'_z \cup T''_z$, where $T'_z = T_z^{-u}$, and T''_z = all the sub-trees rooted at the vertices of the path $z \longrightarrow u$. Note that the set $\mathcal{C}(T_v)$ contains u, and z is covered by the vertex u (see Figure 6.7). We know that T_z does not contain any sibling gate, as $\Gamma_{sib}(u, v, w)$ is a useful sibling gate. However, it may have ordinary *gate*s (see

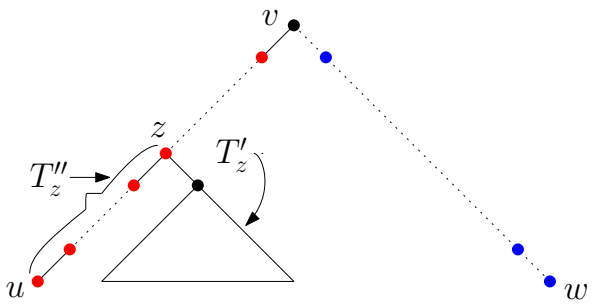


Figure 6.7: Computation of $\mathcal{C}(T_z)$: $T_z = T'_z \cup T''_z$

Definition 6.2). If Observation 6.1 holds for T'_z, then $\mathcal{C}(T_z)$ is easy to compute. Otherwise, the following characterizations are required to formulate the algorithm for computing $\mathcal{C}(T_z)$.

Lemma 6.3

$\mathcal{C}(T_z)$ contains at least one vertex that belongs to a leaf block of T_z.

Proof. Assume that there does not exist any vertex in $\mathcal{C}(T_z)$ that belongs to a leaf block of T_z. Consider an ℓ_block, say ℓ_1. As none of its members lies in $\mathcal{C}(T_z)$, ℓ_1 must belong to a subtree rooted at the anchor of a gate $\Gamma(u_1, w_1)$ so that the vertices of ℓ_1 are covered. Let $dist(z, w_1) > dist(z, u_1)$. Surely $w_1 \notin \ell_1$ due to the contradiction assumption that there does not exist any vertex in $\mathcal{C}(T_z)$ that belongs to a leaf block of T_z of the lemma. Now consider another ℓ_block, say ℓ_2, in the subtree rooted at w_1. It is again covered by one of the pair of vertices $(u_2, w_2) \in \mathcal{C}(T_z)$ of a gate $\Gamma(u_2, w_2)$ in the subtree T_{w_1}, where $dist(z, w_2) > dist(z, u_2)$. Needless to say, the depth of w_2 is greater than that of w_1 in the tree T_z. As the tree T_z is finite, proceeding similarly we will reach an ℓ_block, say ℓ_i, whose vertices are covered by $w_i \in \mathcal{C}(T_z)$, where w_i defines a gate $\Gamma(u_i, w_i)$ as well as $w_i \in \ell_i$. Thus the lemma follows. □

Lemma 6.4

The set $\mathcal{C}(T_z)$ contains a subset of vertices $\mathcal{C}'$ whose elements can be arranged in increasing order of their names, say $\{\chi_1, \chi_2, \ldots, \chi_m\}$, such that (i) χ_m is in a leaf block of T_z, (ii) at least one vertex from each block on the path from z to χ_m is present in $\mathcal{C}'$, and (iii) χ_i is the predecessor (not necessarily immediate predecessor) of χ_{i+1} in T_z for each $i = 1, 2, \ldots, m-1$.

Proof. Part (i) follows from Lemma 6.3. Part (ii) follows trivially since if there exists a block on the path from z to χ_m which has no representative in $\mathcal{C}'$, then the nearest neighbor of each vertex (of color, say red) in that block is of color blue, and hence it becomes inconsistent.

We prove part (iii) by contradiction. Assume that, $\chi \in \mathcal{C}(T_z)$ is the representative of a leaf block. Consider a sequence of vertices $\Psi = \{\psi_1, \psi_2, \ldots, \psi_{m'} = \chi\} \in \mathcal{C}'$ such that for each pair (ψ_j, ψ_{j+1}) either they are in the same block, or they are in the adjacent block, $j = 1, 2, \ldots, m'-1$. Suppose for a contradiction, (ψ_i, ψ_{i+1}) is the first pair observed in the sequence Ψ such that ψ_i is not the predecessor of ψ_{i+1}. Here, if ψ_i lies in a partial leaf block, the path from z to ψ_i satisfies the lemma. If ψ_i lies in a non-leaf block, say B, two cases may happen: (a) $color(\psi_{i+1}) = color(\psi_i)$, i.e., $\psi_i, \psi_{i+1} \in B$ (see Figure 6.8(a)), and (b) $color(\psi_{i+1}) \neq color(\psi_i)$ i.e $\psi_i \in B$ and $\psi_{i+1} \in B'$, where B and B' are adjacent blocks (see Figure 6.8(b)). In either case, as B is a non-leaf block, there exists another block B'' adjacent to B which can be reached from B using a gate $\Gamma(\psi', \phi')$ (where $dist(z, \psi_i) < dist(z, \psi')$) and ψ' is reachable from ψ_i using the successor links in $\mathcal{T}$. So instead of considering the path Ψ, we will consider the path $\Psi' = \{\psi_1, \ldots, \psi_i, \ldots \psi' \ldots \phi' \ldots\}$ (see the thick edges in Figure 6.8(a,b)). Proceeding in this way, we will reach a leaf block. Thus, the result follows. □

As demonstrated in Figure 6.6(c), there may exist multiple vertices of a block in $\mathcal{C}(T_z)$. Moreover, from the definition of the gate (of odd length), it is also clear that the representative of all the blocks may not have representatives in $\mathcal{C}(T_z)$. Thus, we consider each leaf τ of T_z and compute the size of the MCS assuming

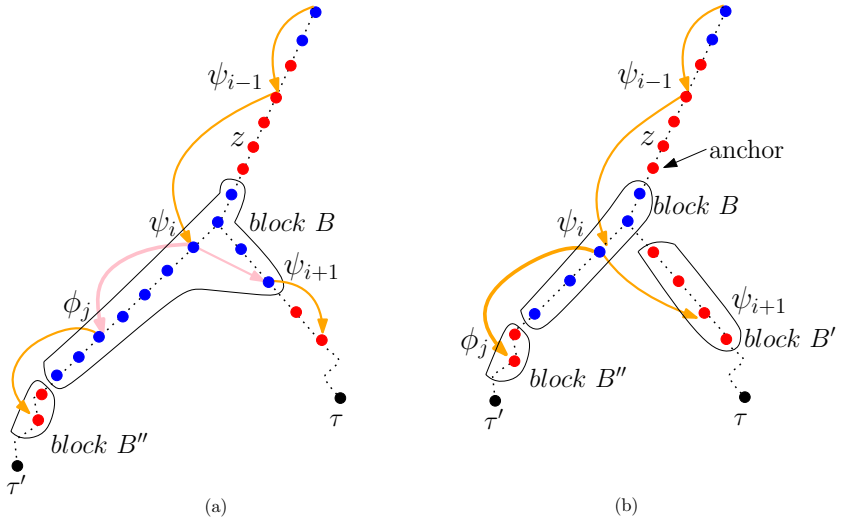


Figure 6.8: Illustration of part (iii) of Lemma 6.4

that the $\ell\_block\ \lambda$ containing τ has a representative in $\mathcal{C}(T_z)$.

Consider the path $\Pi = u \longrightarrow z \longrightarrow \tau$, where τ is a leaf of T_z; τ belongs to the leaf block λ. We formulate the problem as a shortest path problem in a directed weighted graph $G_\Pi = (X, E)$, called the *consistency graph*. Here X contains the vertices on the path Π and two vertices $\{u, t\}$, where u defines the sibling gate $\Gamma_{sib}(u, v, w)$ under process, and t is a dummy sink vertex, and z is a vertex on the path $u \longrightarrow v$. We assume that $u \in \mathcal{C}(T_z)$ and define the edge set $E = E_z^0 \cup E_z^1 \cup E_z^2 \cup E_z^3$, as follows (see Figure 6.9(a,b)).

E_z^0: It consists of consistency edges (colored orange) from u to the next run (of color different from that of u) in Π. We may have at most three such edges in E_z^0 depending on the size of that run. See Figures 6.9(a) and 6.9(b) for two different situations depending on the length of the run $\{u \longrightarrow z \longrightarrow z'\}$.

E_z^1: It consists of consistency edges (colored orange) between every pair of adjacent runs in Π. See Figures 6.10(a) and 6.10(b).

E_z^2: It consists of the edges (colored pink) of a complete graph among the vertices of each run on the path Π.

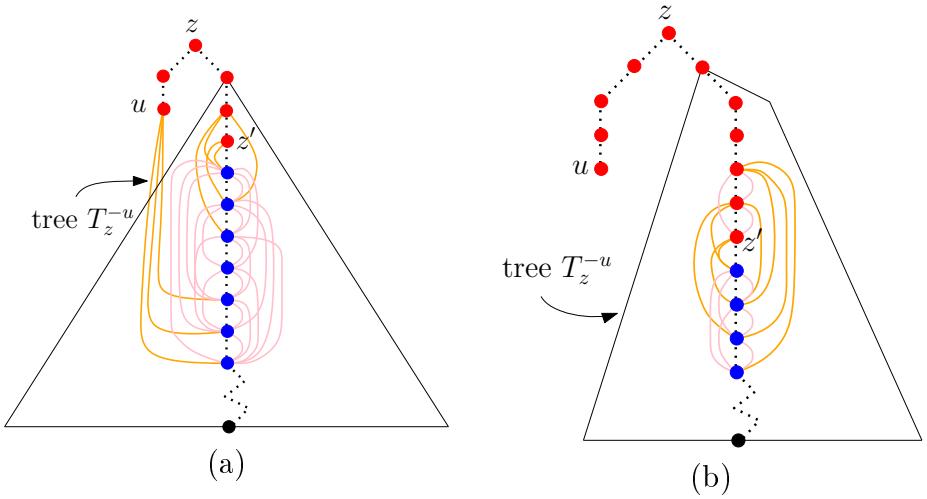


Figure 6.9: Demonstration of the graph G_Π for the path $\Pi = u \longrightarrow z \longrightarrow \tau$ for processing the tree T_z^{-u} – (a) where G_Π is connected, and (b) where G_Π is disconnected as the red run $\{z \longrightarrow z'\}$ is much longer than the next blue run so that there is no edge from u

E_z^3: The vertex t is connected with every vertex of the leaf block λ (these edges are not shown in Figure 6.7).

Now, we assign the weights of the esges in E. For an edge $\overrightarrow{ab} \in E_z^1$, if the number of vertices on the path $a \longrightarrow b$ is odd, then the subtree $T_c^{-(a,b)}$ rooted at the anchor c of $\Gamma(a,b)$ are consistently covered. Thus, the weight of the edge (a,b) is

$$w(a,b) = \sum_{x=a}^{c_a} |\mathcal{C}_a(T_x^{-a})| + \sum_{x=c_b}^{b'} |\mathcal{C}_c(T_x^{-b})|, \tag{6.2}$$

where b' is the neighbor of b on the path $a \longrightarrow b$, c_a and c_b are the neighbors of c along the path $a \longrightarrow c$ and $c \longrightarrow b$ respectively[2].

For an edge $\overrightarrow{ab} \in E_z^1$, if the number of vertices on the path $a \longrightarrow b$ is even, then

$$w(a,b) = \sum_{x=a}^{c_a} |\mathcal{C}_a(T_x^{-a})| + \sum_{x=c_b}^{b'} |\mathcal{C}_b(T_x^{-b})|, \tag{6.3}$$

[2]The subtree rooted at b will be considered when another edge from b to a successor vertex will be considered

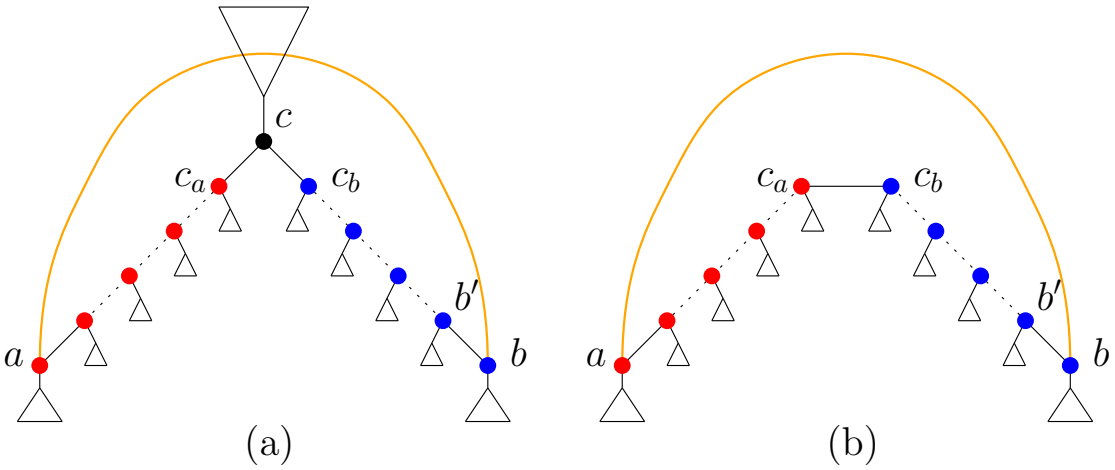


Figure 6.10: Demonstration of edges in E_z^1 where (a) the number of vertices on the path $a \longrightarrow b$ is odd, and (b) the number of vertices on the path $a \longrightarrow b$ is even.

where c_a and c_b are the pair of middle-most vertices along the path segment $a \longrightarrow b$. To avoid the confusion, we mention that $T_a^{-a} = T_a$.

The weight computation of the edges $(u, b) \in E_z^0$ are done with a minor change in the first sum in Equations 6.2 and 6.3; here the range of the first sum is from vertex z to c_a instead of u to c_a.

For an edge $(a, b) \in E_z^2$,

$$w(a,b) = \sum_{x=a}^{c_a} |\mathcal{C}_a(T_x^{-a})| + \sum_{x=c_b}^{b'} |\mathcal{C}_b(T_x^{-b})|. \tag{6.4}$$

Here, if the number of vertices on the path $a \longrightarrow b$ is odd then we assume $c_a = c$ and c_b is the immediate successor of c along the path $c \longrightarrow b$; and if it is even then c_a and c_b are as defined in Equation 6.3. Each edge $(a, t) \in E_z^3$ will have weight $w(a, t) = 0$. For pseudocode, see Algorithm 6.3.

The graph G_Π may not be connected as the vertex u may not be connected to a vertex $b \in X$ in the next run on the path Π. This situation happens when the path $u \longrightarrow c$ is much longer than the path $c \longrightarrow b$. In such a case the path Π (or

Algorithm 6.3: ConstructGraph(Π)

Input: A path $\Pi = u \longrightarrow z \longrightarrow \tau$
Output: A consistency graph $G_\Pi(X, E)$

```
Create a dummy vertex t;
X = {u, t} ∪ vertices of Π;
E_z^0 = Edges from u to the next run in Π;
E_z^1 = Edges between every pair of adjacent runs in Π.;
E_z^2 = Edges of a complete graph among the vertices of each run in Π.;
//weight of the edges in E_z^0, E_z^1, E_z^2 are calculated using eqns 6.2, 6.3, 6.4;
E_z^3 = Edges between t and every vertex of the leaf block λ;
//weight of the edges in E_z^3 is 0;
E = E_z^0 ∪ E_z^1 ∪ E_z^2 ∪ E_z^3;
return G_Π
```

the corresponding leaf τ) contributes ∞ in $\mathcal{C}(T_z)$. Thus, it remains to explain the computation of $\mathcal{C}_a(T_x^{-a})$, where a and x are in the same block on Π.

We use bottom-up dynamic programming technique to compute $\mathcal{C}(T_x)$ for all vertices $x \in \Pi$. Let $M(x)$ denote the members in the run containing the vertex x, and $m(x) = |M(x)|$. These vertices are named as $M(x) = \{b_1 = x, b_2, \ldots, b_{m(x)}\}$ in order. The vertex x is attached with an array A_x of size $M(x)$. For each element $b_i \in M(x)$, $A_x(b_i)$ contains $\mathcal{C}_{b_i}(T_x^{-b_i})$. While processing a vertex $x \in T_z$, we assume that the $A_x(b_i)$ parameters of all the vertices $b_i \in M(x)$ in the run containing x are available; otherwise we recurse. We initialize $A_x(b_i) = \infty$ for all $b_i \in M(x)$. Next, we consider every leaf vertex θ of the tree $T_x^{-b_i}$, and construct the graph G_Φ for the path $\Phi = \beta \longrightarrow b_i \longrightarrow \theta$, where β is the $m(x)$-th vertex in the run containing b_i. The edges in the graph G_Φ are similar to those in G_Π constructed while processing the vertex z on the path $\Pi = u \longrightarrow z$ as described earlier[3]. If for every vertex y on the path Φ the $A_y(.)$ values are available, then the costs of those edges can be computed using Equations 6.2 and 6.3 as described for z. Otherwise, this will lead to a further recursive call. Finally, $A_x(b_i)$ is updated by comparing the existing value of $A_x(b_i)$ and the shortest path cost of G_Φ. For pseudocode, see Algorithm 6.4.

[3]For vertex u defining the sibling gate $\Gamma_{sib}(u, v, w)$

Algorithm 6.4: ConsistentOf(T_z^{-u})

Input: The tree T_z^{-u}
Output: The minimum consistent subset $\mathcal{C}(T_z^{-u})$

```
1  //initialisation for the dynamic programming;
2  for each x ∈ T_z do
3  |  Create an array A_x of size |M(x)|;
4  |  for each b_i ∈ M(x) do
5  |  |  A_x(b_i) = ∞
6  |  end
7  |  for each b_i ∈ M(x) do
8  |  |  //β is the m(x)-th vertex in the run containing b_i;
9  |  |  for every leaf θ of T_x^{-b_i} do
10 |  |  |  ConstructGraph(Φ) // where Φ = β ⟶ b_i ⟶ θ;
11 |  |  |  for each y on Φ do
12 |  |  |  |  if A_y(b_i) ≠ ∞ then
13 |  |  |  |  |  Determine edge costs using eqns 6.2 and 6.3;
14 |  |  |  |  else
15 |  |  |  |  |  A_y(b_i) = ConsistentOf(T_x^{-b_i});
16 |  |  |  end
17 |  |  end
18 |  |  if A_x(b_i) > shortest path cost of G_Φ then
19 |  |  |  A_x(b_i) = shortest path cost of G_Φ
20 |  |  end
21 |  end
22 end
23 return C(T_z^{-u})
```

6.3.2 Analysis of Algorithm of MCS($\mathcal{T}$)

Theorem 6.1

Algorithm 6.1 correctly computes a minimum consistent subset of a bi-colored tree $\mathcal{T}$ on n vertices in $O(n^4)$ time.

Proof. From the property of sibling gates, it follows that the presence of $\{u, w\}$ in $\mathcal{C}(T_v)$ of any one sibling gate, say $\Gamma_{sib}(u, v, w)$, will consistently cover all the

vertices of $\mathcal{T} \setminus (T_{v_u} \cup T_{v_w})$. We have chosen the one of minimum size among all possible useful sibling gates anchored at v. Now, it remains to prove the correctness and minimality of computing $\mathcal{C}(T_v)$.

Again $\{u, w\}$ consistently covers the vertices on the paths $v \longrightarrow u$ and $v \longrightarrow w$. We added the MCS' $\mathcal{C}_u(T_x^{-u})$ for each $x \in \{v \longrightarrow u\}$ and $\mathcal{C}_w(T_y^{-w})$ for each $y \in \{v \longrightarrow w\}$. The computation of consistent subsets for the sub-trees rooted at the vertices on the path $v \longrightarrow u$ and $v \longrightarrow w$, under the condition that $u, w \in \mathcal{C}(T_v)$, can be done independently (see Lemma 6.2). Now, we prove $\mathcal{C}_u(T_x^{-u})$ is correctly computed. By Lemmas 6.3 and 6.4, there is a path Π from vertex x to a leaf of the tree T_x^{-u} such that the vertices on a path of the consistency graph G_Π are in the consistent subset $\mathcal{C}_u(T_x^{-u})$, and $\{u\} \cup \mathcal{C}_u(T_x^{-u})$ covers all the vertices on Π. The recursive argument of computing the MCS for the uncovered sub-trees of T_x justifies the correctness of computing the $\mathcal{C}(T_v)$. The minimality is ensured from the fact that for each leaf τ of T_x, we considered the path $\Pi = u \longrightarrow x \longrightarrow \tau$, and considered the shortest path of the graph G_Π, and have chosen the result for a leaf that produces the minimum cost.

Now, we will analyze the time complexity. Step 1 of Algorithm 6.1 can be implemented in $O(n)$ time. Step 2 requires $O(n^2)$ time as we are constructing $O(n)$ rooted trees. Step 3, for each tree T_v, can be implemented in $O(n)$ time using Algorithm 6.2. Now we analyze Step 4.

While processing a vertex z on the path $u \longrightarrow v$ of a sibling gate $\Gamma_{sib}(u, v, w)$, assuming that the array $A_x(.)$ of every vertex on $x \in T_v$ are available, the time of processing a path $\Pi = u \longrightarrow z \longrightarrow \tau$, where τ is a leaf of T_z^{-u}, needs computation of edge costs of G_Π and the computation of shortest path in G_Π. In the graph G_Π, the vertex u and each vertex of the path $z \longrightarrow \tau$ has at most three orange edges to its successor run in Π. Thus the total number of orange edges in G_Π is $O(m_\Pi)$, where m_Π is the length of Π. Moreover, the span of an edge (a_i, b_j) covers the span of another edge (a_{i+1}, b_{j-1}) (if it exists). Thus, the total time needed for computation of edge costs of all these pink edges in G_Π is $O(m_\Pi)$. However, we may have $O(m_\Pi^2)$ pink edges in G_Π. Processing each leaf vertex τ in T_z^{-u} incurs

$O(m_{\Pi}^2)$ time.

The shortest path computation of a directed graph needs time proportional to its number of edges. As the number of leaves of the tree T_z is $O(n_z)$ in the worst case, where n_z is the number of vertices in T_z, the total time of processing the vertex z is $O(n_z^3)$. Since the trees for the vertices along the path $u \longrightarrow z$ are disjoint, the time complexity of processing these vertices are additive. Again, as the vertices in the subtree rooted at the anchor of useful sibling gates in $\mathcal{T}$ are also disjoint, the total time for processing all the sibling gates is $O(n^3)$ in the worst case, provided the $A_x(.)$ values of every vertex on $x \in \mathcal{T}$ are available.

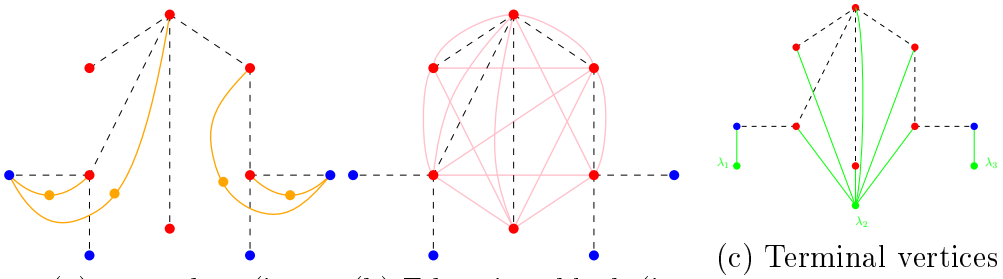
Now, we consider the computation of $A_x(.)$ values of every vertex $x \in T_v$. We consider the vertices in each level of the rooted tree T_v separately, and compute their $A_x(.)$ values. While processing the vertices in its predecessor level, we will use those without recomputing. Similar to the processing of the vertex $z \in \{u \longrightarrow v\}$ discussed earlier, the processing of every vertex $x \in T_z$ for the computation of their $A_x(.)$ values requires $O(n_x^3)$ time, where n_x is the number of vertices in T_x. The sub-trees rooted at the vertices in a particular level are disjoint, and the total computation time for the vertices in a level is additive. Thus, the overall time complexity of the algorithm is $O(n^3h)$, where h is the maximum number of levels among the sub-trees rooted at the anchor of all possible useful sibling gates in $\mathcal{T}$. Though we will consider the anchor of every gate as a sibling gate and do the above computation, this will not increase the time complexity since the result of a sibling gate once computed can be used later when it is needed. □

6.4 Approximation algorithm

In this section, we will demonstrate that computation of minimum consistent subset of an undirected tree $\mathcal{T} = (V, E)$ can be formulated as computing the minimum Steiner tree of an undirected weighted graph. Let us first give a brief idea about the Steiner tree of a graph. Let $G = (V, E)$ be an undirected graph with non-

negative edge weights c and let $S \subseteq V$ be a subset of vertices, called *terminals*. A Steiner tree is a tree in G that spans S. In the optimization problem associated with Steiner trees, the task is to find a minimum-weight Steiner tree.

As in Section 6.2, here also a *block* is a connected set of vertices of same color, the ℓ_block, n_block and *true* ℓ_block are as in Definition 6.1. Unlike Section 6.2, here we define only one type of gate as stated in Definition 6.2.



(a) gate edges (in orange color)

(b) Edges in a block (in pink color)

(c) Terminal vertices and edges (in green color)

Figure 6.11: Demonstration of vertices and edges in the graph H

6.4.1 Algorithm

We formulate the problem as the computation of a Steiner tree in the *consistency graph* $H = (V', E')$ constructed from the given tree $\mathcal{T}$. The graph H is an undirected weighted graph. Its vertices $V' = V \cup V_\ell \cup V_\eta$ and $E' = E_\ell \cup E_\eta$, where the subsets of the vertices in V' and the edges in E' are explained below.

V: the set of vertices in $\mathcal{T}$;

V_η and E_η: Needless to mention that the vertices in two adjacent runs on a path in the tree are of different colors. As in Section 5.2, in order to maintain consistency, we need to introduce two types of edges to cover the following two situations:

- Consider a path Π in $\mathcal{T}$. A vertex p of a run θ on a path Π may be connected by an edge with at most three vertices r, r' and r'' in its

adjacent run θ' on that path where the difference in the number of vertices of two different colors in the span of that edge is at most 1. We put three consistency edges of *type (i)* from p to r, r' and r''. Thus, the vertices adjacent to each type (i) consistency edge are of different colors.

- We may sometimes choose two vertex on a run in the set $\mathcal{C}$. Thus, every pair of vertices in a run is connected by a *type (ii)* consistency edge. Thus, the vertices adjacent to each type (ii) consistency edge are of same color.

We put a vertex $v_{(p,r)}$ (colored orange) on each type (i) edge (p, r), and replace that type (i) edge (p, r) by two orange edges $(p, v_{(p,r)})$ and $(r, v_{(p,r)})$ (see Figure 6.11(a)), each with weight equal to $\frac{1}{2}$. The consistency edges of type (ii) are colored *pink* (see Figure 6.11(b)), and the assigned weight is 1. The orange vertices are referred to as V_η, and the *orange* and *pink* edges are referred to an E_η.

V_ℓ and E_ℓ: We create a vertex v_λ corresponding to each ℓ*_block* λ of $\mathcal{T}$. We use V_ℓ to denote the set of vertices corresponding to the ℓ*_block*s in $\mathcal{T}$. The vertices in V_ℓ are indicated using green color. The edges from the vertices in V_ℓ to the vertex v_λ form the set E_ℓ, and are colored green (see Figure 6.11(c)). Below, we explain the method of generating the edges in E_ℓ.

Generation of E_ℓ

Let $(p, r) \in E_\eta$ be a consistency edge among the bi-colored vertices and the number of vertices k on the path from p to r is odd. As mentioned in Section 5.2, if we include p, r in the consistent subset $\mathcal{C}$, then all the vertices in each subtree rooted at the *anchor vertex* q at distance $\lceil \frac{k}{2} \rceil + 1$ from both p and r are *consistently covered*, excepting two subtrees rooted at q that contain p and r respectively. Thus, the covered subtrees need not contribute in the minimum consistent subset $\mathcal{C}$ that includes (p, r). So, we put edges from the orange vertex $v_{(p,r)}$ to the vertices in V_ℓ

corresponding to all the ℓ_*block*s in the subtrees rooted at q that are covered by (p, r). These edges are colored *green* with weight 0.

Observe that, *in a* true ℓ_*block* λ *if there exists a vertex in* $\mathcal{C}$*, all the vertices of that* ℓ_*block is covered.* Thus, from every vertex of the *true* ℓ_*block* λ, we add a *green* edge to the vertex v_λ of that ℓ_*block*. We assign weight 1 to these green edges.

Now, consider a *partial* ℓ_*block* of color *red* (say). There exists two types of run: (i) at least one run containing a *leaf* vertex, and (ii) at least one *non-leaf* run (i.e containing two extreme vertices of that run connected to *blue* vertices. Now, consider a situation where a vertex u of $\mathcal{T}$ in a partial ℓ_*block* (colored red) is reached by a Steiner tree $\mathcal{T}_{ST}$. Here one of the two situations may arise to reach in its adjacent (blue) run:

(i) from u (of color red) a *blue* vertex w in its adjacent run can be directly reached through an orange edge, and

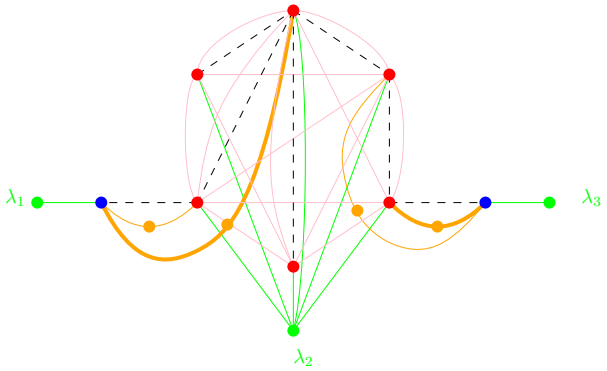
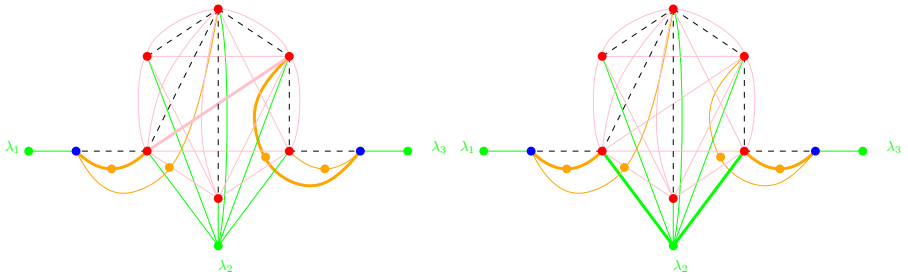


Figure 6.12: Demonstration of situation (i) that arise in a partial ℓ_block

(ii) from u (of color *red*) another *red* vertex u' in the same run is reached in the following two ways, and then from u' the *blue* vertex w is reached:

(ii-a) u' can be reached from u using either a pink edge or

(ii-b) u' can be reached from u through two green edges via the v_λ vertex of that ℓ_*block*.



(a) (b)

Figure 6.13: Demonstration of situations (ii-a) and (ii-b) that arise in a partial ℓ_block

Thus in both cases of situation (ii), the cost incurred to reach from u to w is 2. In situation (ii-b), if there exists a green edge (λ, u) in $\mathcal{T}_{ST}$, the other green edge (λ, u') can be replaced with a pink edge (u, u') as mentioned in (ii-a) without changing the tree property of $\mathcal{T}_{ST}$ and maintaining its cost (see Figure 6.13).

If there exists multiple green edges of a Steiner tree that are incident on v_λ vertex of a partial ℓ_*block* λ, then we retain one of them, say (v_1, v_λ) and replace every other green edges $\{(v_i, v_\lambda), i = 2, \ldots, k\}$ by a pink edge $\{(v_1, v_i), i = 2, \ldots, k\}$ in that partial ℓ_*block*, retaining the cost of the Steiner tree unchanged. The following things are important to mention.

- The terminal vertex v_λ of each ℓ_*block* λ in H are reached in $\mathcal{T}_{ST}$,
- some of the v_λ vertices that are reached from an orange vertex of some other block have incurred cost 0, and
- if a v_λ vertex is reached from a vertex of $\mathcal{T}$ in its corresponding ℓ_*block* it will incur cost 1. If M is the total cost of edges in $\mathcal{T}_{ST}$ due to the edges incident in the terminal vertices, then M is less than the number of terminal (λ) vertices present in $\mathcal{T}_{ST}$ since some of the terminal vertices are reached with cost 0 (see Figure 6.12), and some terminal vertices are reached with exactly one edge of cost 1 (see Figure 6.13(b)).

The Steiner tree obtained in this example is shown in Figure 6.14.

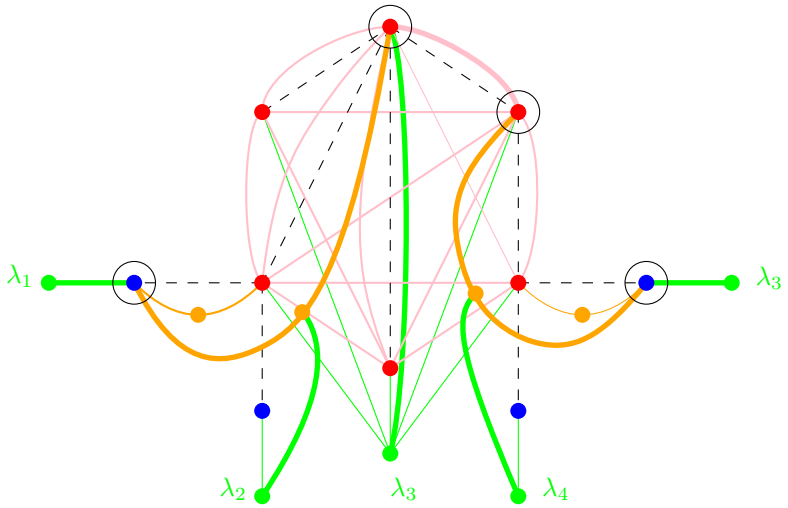


Figure 6.14: Obtained Steiner tree

6.4.2 Analysis

Lemma 6.5

The vertices of $\mathcal{T}$ that belong to a Steiner tree constitute a consistent subset of the vertices of $\mathcal{T}$, and vice-versa.

Proof. First part: Every orange edge (u, w) (where u and w are of different colors) makes all the vertices on the path from u to w in $\mathcal{T}$ consistent. Moreover, if the number of vertices from u to w is odd then the vertices in the subtree rooted at the middle-most vertex v on the path from u to w become consistent. Hence no vertex in that subtree is included in $\mathcal{T}_{ST}$ by our algorithm. Needless to mention, all the vertices in $\mathcal{T}$ on the path of every pink edge (u, w) in $\mathcal{T}_{ST}$ are consistent. For a green edge (u, v_λ) in a true ℓ_block, inclusion of v in $\mathcal{C}$ makes all the vertices of that ℓ_block consistent. For a green edge (u, v_λ) in a partial ℓ_block, all the vertices in its uni-colored branch are consistent for the inclusion of v in $\mathcal{C}$ as mentioned above, and in all the vertices of its bi-colored branch there will be an orange edge, say (u', w'), where $u', w' \in \mathcal{C}$ (u' may be equal to u). This make the vertices of color that of w' in its adjacent run consistent. The subtrees of the vertices on the

path $u \longrightarrow u' \longrightarrow w'$ (if exists) are all reached from u, u', w' using orange or pink edges. The subtree rooted at v' of the gate (u', w') (if exists) are all consistently covered for inclusion of $\{u', w'\}$ in $\mathcal{C}$.

Second part: Now to prove that every consistent subset of $\mathcal{T}$ corresponds to a Steiner tree ST in H. Consider a path Π in $\mathcal{T}$. Observe that, each pair of consecutive members of that consistent subset in the path Π are connected by an orange or a pink edge depending on whether their colors are different or same. Finally, for each leaf block containing at least one member of that consistent subset, we connect one member of them with the λ vertex of that leaf block. For each of the not-connected λ vertex trace the path until it reaches a vertex $v \in V$ (of $\mathcal{T}$) that is marked as *anchor*. Connect that λ vertex with the orange vertex on the orange edge connecting two vertices $u, w \in V$ of $\Gamma(u, w)$ whose anchor is v. Thus, all the terminal vertices of H are spanned by a tree ST which is an induced subgraph of the graph H. □

Lemma 6.5 says that the minimum consistent subset of $\mathcal{T}$ corresponds to the minimum size Steiner tree in the graph H. We can apply any heuristic algorithm to compute a Steiner tree of the graph H to get an aproximation result of the minimum consistent subset of $\mathcal{T}$.

Theorem 6.2

The approximation factor of our proposed algorithm for computing the minimum consistent subset of $\mathcal{T}$ is $2 \times \alpha$, where α is the approximation factor of the algorithm for computing Steiner tree of an weighted undirected graph with best known approximation factor in polynomial time.

Proof. By Lemma 6.5, the optimum consistent subset $\mathcal{C}_{opt}$ corresponds the vertices of $\mathcal{T}$ present in $\mathcal{T}_{ST}^{opt}$ (optimum Steiner tree), which is 1 more than the sum of edge costs of $\mathcal{T}_{ST}^{opt}$. Thus

$$|\mathcal{C}_{opt}| = cost(\mathcal{T}_{ST}^{opt}) + 1 - \#(\lambda_{leaf}^{opt})$$

, where λ_{leaf}^{opt} is the set of terminal (λ) vertices present in $\mathcal{T}_{ST}^{opt}$ that are reached

from a vertex of their corresponding leaf block[4], and $\#(\lambda_{leaf}^{opt})$ is the number of such vertices. Also, note that

$$\#(\lambda_{leaf}^{opt}) \leq |\mathcal{C}_{opt}|$$

, since every vertex in λ_{leaf}^{opt} is reached from a vertex of $\mathcal{T}$ that belongs to $\mathcal{C}_{opt}$.

Similarly, $|\mathcal{C}| = cost(\mathcal{T}_{ST}) + 1 + \#(\lambda_{leaf})$, where λ_{leaf} is the number of λ vertices in $\mathcal{T}_{ST}$ that are reached from the corresponding leaf block by our algorithm.

We already have $cost(\mathcal{T}_{ST}) \leq \alpha \times cost(\mathcal{T}_{ST}^{opt})$. As the number of vertices present in $\mathcal{T}_{ST}$ of the graph H is one more than the sum of edge costs of $\mathcal{T}_{ST}$, we have $|\mathcal{C}| \leq cost(\mathcal{T}_{ST}) + 1 \leq \alpha \times cost(\mathcal{T}_{ST}^{opt}) = \alpha \times (|\mathcal{C}_{opt}| + \#(\lambda_{leaf}^{opt}) \leq 2\alpha \times |\mathcal{C}_{opt}|$.

By Theorem 3 of [BGRS10], we know that the computation of $\mathcal{C}$ takes polynomial time too. □

[4]since only these terminal vertices are reached from a vertex of the corresponding leaf block with cost 1; other terminal vertices are reached with cost 0 from an orange vertex.

CHAPTER 7

Concluding Remarks

Finally, in this chapter, we summarize the main contributions of this thesis and state possible future directions of research.

Contents

7.1 Discrimination and Identification

We have seen that Discrete-G-Min-Disc-Code is NP-complete, even in 1D. This is in contrast with most covering problems and to Continuous-G-Min-Disc-Code, which are polynomial-time solvable in 1D [DBRDG17, GP19].

We also proposed a simple 2-factor approximation algorithm for the DISCRETE-G-MIN-DISC-CODE problem in 1D, and a PTAS for a special case where each interval in the set S is of unit length. It seems a challenging open problem to explore whether DISCRETE-G-MIN-DISC-CODE problem in 1D is polynomial time solvable for unit intervals. As noted in [GP19], this would be related to MIN-ID-CODE on *unit* interval graphs, which also remains unsolved [FMN+17]. In fact, it also seems to be unknown whether CONTINUOUS-G-MIN-DISC-CODE problem in 1D remains polynomial-time solvable with the restriction that each interval is of unit length. However our PTAS algorithm for DISCRETE-G-MIN-DISC-CODE problem with unit intervals in 1D also produces a PTAS for the continuous case. We also do not know whether a PTAS exists for the general 1D case.

In 2D, both CONTINUOUS-G-MIN-DISC-CODE and DISCRETE-G-MIN-DISC-CODE problems are NP-complete even when S must be a set of axis-parallel unit square objects. We propose polynomial-time approximation algorithms for both CONTINUOUS-G-MIN-DISC-CODE and DISCRETE-G-MIN-DISC-CODE using several steps of rounding using the LP-relaxation of integer programming. However, the approximation factors are very large; 16 for the continuous version and 128 for the discrete version. Improving the approximation factor is a challenging open problem. Can PTASes be obtained here?

Finally, we showed that a minor tailoring of the proposed algorithms for DISCRETE-G-MIN-DISC-CODE for unit square objects works for MIN-ID-CODE on unit square graphs, with the same approximation factors. However, the layout of the squares corresponding to the nodes of the graph is needed. It is worthy to see whether it is possible to design an approximation algorithm when only the graph is given[1]?

It is easy to observe that all the techniques for designing approximation algorithms for solving the aforesaid problems in 2D work for the case where the objects are fixed-size axis-parallel rectangles. It is an interesting open problem whether similar approximation algorithms exist, where the objects in S are unit disks, or arbitrary axis-parallel rectangles.

[1]Note that computing a geometric layout of square intersection graph is NP-hard [Bre96].

7.2 Red-Blue Separation

We have initiated the study of Red-Blue Separation and Max Red-Blue Separation on graphs, problems which seem natural given the interest that their geometric version has gathered, and the popularity of its "non-colored" variants Identifying Code on graphs or Test Cover on set systems.

When the coloring is part of the input, the solution size of Red-Blue Separation can be as small as 2, even for large instances; however, we have seen that this is not possible for Max Red-Blue Separation since $\text{max-sep}_{\text{RB}}(G) \geq \lfloor \log_2(n) \rfloor$ for any twin-free graphs of order n. $\text{max-sep}_{\text{RB}}(G)$ can be as large as $n-1$ in general graphs, yet, on trees, it is at most $2n/3$. We do not know whether the upper bound is tight, or whether the upper bound is $3n/5$, which would be best possible to hold. It would also be interesting to see if other interesting upper or lower bounds can be shown for other simple graph classes.

We have shown that $\text{sep}(G) \leq \lceil \log_2(n) \rceil \cdot \text{max-sep}_{\text{RB}}(G)$. Is it true that $\text{sep}(G) \leq c \cdot \text{max-sep}_{\text{RB}}(G)$, where c is some constant (independent of n)? As we have seen, $c = 2$ would be a tight bound in the sense that there exists instances which achieves this bound.

We have also shown that Max Red-Blue Separation is NP-hard, yet it does not naturally belong to NP. Is the problem actually hard for the second level of the polynomial hierarchy?

7.3 Consistency

We have seen that the MCS problem, in the undirected set-up, is solvable in linear time in k-chromatic paths. It is also solvable in quadratic time in k-chromatic spider graphs, bi-chromatic caterpillar graphs and bi-chromatic comb graphs. For the directed set-up, the problem is solvable in linear time in both k-chromatic

paths and bi-chromatic spiders. The problem remains open for k-chromatic and directed caterpillar graphs and comb graphs.

We have also presented a polynomial-time algorithm for computing the minimum consistent subset of bi-chromatic trees. We also present an easy to understand 2α-factor approximation algorithm for solving the minimum consistent subset problem for trees, where α is the best-known achievable approximation factor of the Steiner tree problem of an undirected weighted graph. According to the present literature $\alpha = 1.386$ [BGRS10], and it uses LP-rounding. Improving the approximation factor and running time (without using LP-rounding) may be an interesting direction to explore. The problem remains unsolved for outer-planar graphs. Note that the minimum consistent subset problem is NP-hard even for bi-chromatic planar graphs. The status of the problem with three or more colors is still unknown for trees.

Bibliography

[AAA+07] Hee-Kap Ahn, Helmut Alt, Tetsuo Asano, Sang Bae, Peter Brass, Otfried Cheong, Christian Knauer, Hyeon-Suk Na, Chan-Su Shin, and Alexander Wolff. Constructing optimal highways. *International Journal of Foundations of Computer Science*, 20(01):3–23, 2007.

[AB08] F. Angiulli and S. Basta. Optimal subset selection for classification through SAT encodings. In *IFIP International Conference on Artificial Intelligence in Theory and Practice*, pages 309–318, 2008.

[ACHL10] David Auger, Irène Charon, Olivier Hudry, and Antoine Lobstein. On the existence of a cycle of length at least 7 in a $(1,\leq 2)$-twin-free graph. *Discuss. Math. Graph Theory*, 30(4):591–609, 2010.

[ACHL13] David Auger, Irène Charon, Olivier Hudry, and Antoine Lobstein. Watching systems in graphs: an extension of identifying codes. *Discrete Applied Mathematics*, 161(12):1674–1685, 2013.

[ADBH+15] Esther M. Arkin, José Miguel Díaz-Báñez, Ferran Hurtado, Piyush Kumar, Joseph S.B. Mitchell, Belén Palop, Pablo Pérez-Lantero, Maria Saumell, and Rodrigo I. Silveira. Bichromatic 2-center of pairs of points. *Computational Geometry*, 48(2):94–107, 2015.

[AF07] F. Angiulli and G. Folino. Distributed nearest neighbor-based condensation of very large data sets. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1593–1606, 2007.

[AHM+00] Esther M. Arkin, F. Hurtado, J. S. B. Mitchell, C. Seara, and S. S. Skiena. Some separability problems in the plane. *Procceedings of the 16-th European Workshop on Computational Geometry*, pages 51–54, 2000.

[AHS+03] M. Abellanas, F. Hurtado, V. Sacristán, C. Icking, L. Ma, R. Klein, E. Langetepe, and B. Palop. Voronoi diagram for services neighboring a highway. *Information Processing Letters*, 86(5):283–288, 2003.

[Ang05] F. Angiulli. Fast condensed nearest neighbor rule. In *22nd International Conference on Machine learning*, pages 25–32, 2005.

[Ang07] F. Angiulli. Fast nearest neighbor condensation for large data sets classification. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1450–1464, 2007.

[ANPR19] Ankush Acharyya, Subhas C. Nandy, Supantha Pandit, and Sasanka Roy. Covering segments with unit squares. *Comput. Geom.*, 79:1–13, 2019.

[Aug08] David Auger. Induced paths in twin-free graphs. *The Electronic Journal of Combinatorics (E-JC)*, 15(17), 2008.

[BBC18] Sandip Banerjee, Sujoy Bhore, and Rajesh Chitnis. Algorithms and hardness results for nearest neighbor problems in bicolored point sets. In *LATIN 2018: Theoretical Informatics*, pages 80–93, 2018.

[BCC+19] Ahmad Biniaz, Sergio Cabello, Paz Carmi, Jean-Lou De Carufel, Anil Maheshwari, Saeed Mehrabi, and Michiel H. M. Smid. On the minimum consistent subset problem. In *Algorithms and Data Structures - 16th International Symposium, (WADS)*, pages 155–167, 2019.

[BCMPR20] Édouard Bonnet, Sergio Cabello, Bojan Mohar, and Hubert Pérez-Rosés. The inverse Voronoi problem in graphs I: hardness. *Algorithmica*, 82(10):3018–3040, 2020.

[BCMPR21] Édouard Bonnet, Sergio Cabello, Bojan Mohar, and Hubert Pérez-Rosés. The inverse Voronoi problem in graphs II: trees. *Algorithmica*, 83(5):1165–1200, 2021.

[BDNS19] Kaustav Basu, Sanjana Dey, Subhas C. Nandy, and Arunabha Sen. Sensor networks for structural health monitoring of critical infrastructures using identifying codes. *15th Int. Conf. on the Design of Reliable Communication Networks (DRCN)*, pages 43–50, 2019.

[Bel18] T. Bellitto. Separating codes and traffic monitoring. *Theoretical Computer Science*, 717:73–85, 2018.

[Ber01] N. Bertrand. *Combinatorial and algorithmic aspects of identifying codes in graphs*. PhD thesis, Université Bordeaux 1, France, June 2001.

[BFS19] Cristina Bazgan, Florent Foucaud, and Florian Sikora. Parameterized and approximation complexity of partial VC dimension. *Theor. Comput. Sci.*, 766:1–15, 2019.

[BGL19] Édouard Bonnet, Panos Giannopoulos, and Michael Lampis. On the parameterized complexity of red-blue points separation. *Journal of Computational Geometry*, 10(1):181–206, 2019.

[BGRS10] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvob, and Laura Sanita. An improved LP-based approximation for Steiner tree. In *42nd ACM Symposium on Theory of Computing*, pages 583–592, 2010.

[BLL+15] Nicolas Bousquet, Aurélie Lagoutte, Zhentao Li, Aline Parreau, and Stéphan Thomassé. Identifying codes in hereditary classes of graphs and VC-dimension. *SIAM J. Discrete Math.*, 29(4):2047–2064, 2015.

[Bon72] J. A. Bondy. Induced subsets. *Journal of Combinatorial Theory, Series B*, 12(2):201–202, 1972.

[BP88] Rajan Batta and Udatta S. Palekar. Mixed planar/network facility location problems. *Comput. Oper. Res.*, 15(1):61–67, 1988.

[Bre96] Heinz Breu. *Algorithmic Aspects of Constrained Unit Disk Graphs.* PhD thesis, University of British Columbia, Vancouver, Canada, 1996.

[BS07] Béla Bollobás and Alex Scott. On separating systems. *European Journal of Combinatorics*, 28(4):1068–1071, 2007.

[BS21] Kaustav Basu and Arunabha Sen. Identifying individuals associated with organized criminal networks: A social network analysis. *Social Networks*, 64:42–54, 2021.

[BU95] Ralph P. Boland and Jorge Urrutia. Separating collections of points in euclidean spaces. *Information Processing Letters*, 53(4):177–183, 1995.

[BZSG19] Kaustav Basu, Chenyang Zhou, Arunabha Sen, and Victoria Horan Goliber. A novel graph analytic approach to monitor terrorist networks. In *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/-SustainCom)*, pages 1159–1166, 2019.

[CCC+08] Emmanuel Charbit, Irène Charon, Gérard D. Cohen, Olivier Hudry, and Antoine Lobstein. Discriminating codes in bipartite graphs: Bounds, extremal cardinalities, complexity. *Advances in Mathematics of Communications*, 2(4):403–420, 2008.

[CCCH06] Emmanuel Charbit, Irène Charon, Gérard D. Cohen, and Olivier Hudry. Discriminating codes in bipartite graphs. *Electronic Notes in Discrete Mathematics*, 26:29–35, 2006.

[CCH+08] Jean Cardinal, Sébastien Collette, Ferran Hurtado, Stefan Langerman, and Belen Palop. Optimal location of transportation devices. *Computational Geometry*, 41(3):219–229, 2008.

[CCHL07] Irène Charon, Gerard Cohen, Olivier Hudry, and Antoine Lobstein. Links between discriminating and identifying codes in the binary hamming space. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 267–270, 2007.

[CCHL08] Irène Charon, Gérard D. Cohen, Olivier Hudry, and Antoine Lobstein. Discriminating codes in (bipartite) planar graphs. *Eur. J. Comb.*, 29(5):1353–1364, 2008.

[CDKW05] G. Cãlinescu, A. Dumitrescu, H. Karloff, and P.-J. Wan. Separating points by axis-parallel lines. *International Journal of Computational Geometry and Applications*, 15(6):575–590, 2005.

[CGJ+16] Robert Crowston, Gregory Z. Gutin, Mark Jones, Gabriele Muciaccia, and Anders Yeo. Parameterizations of test cover with bounded test sizes. *Algorithmica*, 74(1):367–384, 2016.

[CH67] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[CH98] Bogdan Chlebus and Nguyen Hoa. On finding optimal discretizations for two attributes. In *International Conference on Rough Sets and Current Trends in Computing (RSCTC)*, pages 537–544, 1998.

[CH15] Timothy M. Chan and Nan Hu. Geometric red–blue set cover for unit squares and related problems. *Computational Geometry*, 48(5):380–385, 2015.

[CHHL07] Irène Charon, Iiro Honkala, Olivier Hudry, and Antoine Lobstein. Structural properties of twin-free graphs. *Electr. J. Comb.*, 14(1), 2007.

[CHL03] Irène Charon, Olivier Hudry, and Antoine Lobstein. Minimizing the size of an identifying or locating-dominating code in a graph is NP-hard. *Theor. Comput. Sci.*, 290(3):2109–2120, 2003.

[CHL12] Iréne Charon, Olivier Hudry, and Antoine Lobstein. Extremal values for the maximum degree in a twin-free graph. *Ars Combinatoria*, 107:257–274, 2012.

[CKX10] Jianer Chen, Iyad A Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010.

[Cla94] K. Clarkson. More output-sensitive geometric algorithms (extended abstract). *35th Annual Symposium on Foundations of Computer Science*, pages 695–702, 1994.

[CN85] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.

[CSS87] Colbourn C.J., P.J. Slater, and L.K. Stewart. Locating dominating sets in series parallel networks. *Congr. Numer.*, 56(8):135–162, 1987.

[Das17] Tanmoy Das. Machine learning algorithms for image classification of hand digits and face recognition dataset. *International Research Journal of Engineering and Technology (IRJET)*, 4(12):640–649, 2017.

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, USA, 2008.

[dBHH+03] Koen M. J. de Bontridder, Bjarni V. Halldórsson, Magnús M. Halldórsson, A. J. Hurkens, Jan Karel Lenstra, R. Ravi, and Leen Stougie. Approximation algorithms for the test cover problem. *Math. Program.*, 98(1):477–491, 2003.

[DBKPLV13] José Díaz-Báñez, M. Korman, Pablo Pérez-Lantero, and Inmaculada Ventura. The 1-median and 1-highway problem. *European Journal of Operational Research*, 225(3):552–557, 2013.

[DBRDG17] Krupa R. Datta, Aniket Basu Roy, Minati De, and Sathish Govindarajan. Demand hitting and covering of intervals. In *Conference on Algorithms and Discrete Applied Mathematics (CALDAM)*, pages 267–280, 2017.

[DDSM76] S. R. Das, C. R. Datta, P. K. Srimani, and K. Mandal. Comments on "Derivation of minimal complete sets of test-input sequences using boolean differences". *IEEE Transactions on Computers*, 25(10):1053–1056, 1976.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer Verlag, 1999.

[DFNS20] Sanjana Dey, Florent Foucaud, Subhas C. Nandy, and Arunabha Sen. Discriminating Codes in Geometric Setups. In *31st International Symposium on Algorithms and Computation (ISAAC)*, volume 181 of *LIPIcs*, pages 24:1–24:16, 2020.

[DHMS01] Olivier Devillers, Ferran Hurtado, Mercè Mora, and Carlos Seara. Separating several point sets in the plane. In *13th Canadian Conference on Computational Geometry*, pages 81–84, 2001.

[DS14] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *46th Annual ACM Symposium on Theory of Computing*, pages 624–633, 2014.

[DX11] Hu Ding and Jinhui Xu. Solving the chromatic cone clustering problem via minimum spanning sphere. In *38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 773–784, 2011.

[EMP+82] Herbert Edelsbrunner, Hermann Maurer, F. Preparata, Arnold Rosenberg, E. Welzl, and D. Wood. Stabbing line segments. *BIT Numerical Mathematics*, 22(3):274–281, 1982.

[Erd06] Paul Erdős. Some combinatorial, geometric and set theoretic problems in measure theory. In *Measure Theory Oberwolfach 1983*, volume 1089, pages 321–327. 2006.

[FGK+11] Florent Foucaud, Eleonora Guerrini, Matja Kovše, Reza Naserasr, Aline Parreau, and Petru Valicov. Extremal graphs for the identifying code problem. *Eur. J. Comb.*, 32(4):628–638, 2011.

[FGN+13] Florent Foucaud, Sylvain Gravier, Reza Naserasr, Aline Parreau, and Petru Valicov. Identifying codes in line graphs. *Journal of Graph Theory*, 73(4):425–448, 2013.

[FMN+17] Florent Foucaud, George B. Mertzios, Reza Naserasr, Aline Parreau, and Petru Valicov. Identification, location-domination and metric dimension on interval and permutation graphs. II. algorithms and complexity. *Algorithmica*, 78(3):914–944, 2017.

[Fou12] Florent Foucaud. *Codes identifiants et codes localisateurs-dominateurs sur certains graphes.* PhD thesis, ENST, France, 2012.

[Fou15] Florent Foucaud. Decision and approximation complexity for identifying codes and locating-dominating sets in restricted graph classes. *J. Discrete Algorithms*, 31:48–68, 2015.

[Fre00] Robert Wilson Freimer. *Investigations in geometric subdivisions: linear shattering and cartographic map coloring.* PhD thesis, 2000.

[Gat72] G. Gates. The reduced nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 18(3):431–433, 1972.

[GEC+07] 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 *13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, page 310–319, 2007.

[GIK02] Daya Ram Gaur, Toshihide Ibaraki, and Ramesh Krishnamurti. Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem. *Journal of Algorithms*, 43(1):138–152, 2002.

[GJ02] Michael R. Garey and David S. Johnson. *Computers and intractability.* W. H. Freeman New York, 2002.

[GK79] K. Gowda and G. Krishna. The condensed nearest neighbor rule using the concept of mutual nearest neighborhood (corresp.). *IEEE Transactions on Information Theory*, 25(4):488–490, 1979.

[GKM08] Sylvain Gravier, Ralf Klasing, and Julien Moncel. Hardness results and approximation algorithms for identifying codes and locating-dominating codes in graphs. *Algorithmic Operations Research*, 3(1):43–50, 2008.

[GM07] Sylvain Gravier and Julien Moncel. On graphs having a $V \setminus \{x\}$ set as an identifying code. *Discrete Mathematics*, 307(3):432–434, 2007.

[GP19] Valentin Gledel and Aline Parreau. Identification of points using disks. *Discrete Math.*, 342:256–269, 2019.

[GT13] Dániel Gerbner and Géza Tóth. Separating families of convex sets. *Computational Geometry*, 46(9):1056–1058, 2013.

[Har68] P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.

[HHS98a] Teresa W. Haynes, Stephen Hedetniemi, and Peter Slater. *Domination in Graphs: Advanced Topics.* CRC Press, 1998.

[HHS98b] Teresa W. Haynes, Stephen Hedetniemi, and Peter Slater. *Fundamentals of Domination in Graphs.* CRC Press, 1998.

[HKSZ06] Teresa Haynes, Debra Knisley, Edith Seier, and Yue Zou. A quantitative analysis of secondary RNA structure using domination based parameters on trees. *BMC bioinformatics*, 7(1):1–11, 2006.

[HLT22] P. Hajnal, Zhihao Liu, and György Turán. Nearest neighbor representations of boolean functions. *Information and Computation (In Press)*, 2022.

[HM85] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32(1):130–136, 1985.

[HPJ20] Sariel Har-Peled and Mitchell Jones. On separating points by lines. *Discrete and Computational Geometry*, 63(3):705–730, 2020.

[HPV98] Michel Habib, Christophe Paul, and Laurent Vienot. A synthesis on partition refinement: A useful routine for strings, graphs, boolean matrices and automata. In *15th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, page 25–38, 1998.

[HY14] Michael A. Henning and Anders Yeo. Distinguishing-transversal in hypergraphs and identifying open codes in cubic graphs. *Graphs and Combinatorics*, 30(4):909–932, 2014.

[JL11] Ville Junnila and Tero Laihonen. Identification in Z2 using Euclidean balls. *Discrete Applied Mathematics*, 159(5):335–343, 2011.

[KCL98] Mark G. Karpovsky, Krishnendu Chakrabarty, and Lev B. Levitin. On a new class of codes for identifying vertices in graphs. *IEEE Trans. Information Theory*, 44(2):599–611, 1998.

[KE07] Jussi Kujala and Tapio Elomaa. Improved algorithms for univariate discretization of continuous features. In *11th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 188–199, 2007.

[KGG85] James M. Keller, Michael R. Gray, and James A. Givens. A fuzzy k-nearest neighbor algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(4):580–585, 1985.

[KKR18] Kamyar Khodamoradi, Ramesh Krishnamurti, and Bodhayan Roy. Consistent subset problem with two labels. In *4th International Conference on Algorithms and Discrete Applied Mathematics (CALDAM)*, pages 131–142, 2018.

[KMM+21] Stefan Kratsch, Tomáš Masařík, Irene Muzi, Marcin Pilipczuk, and Manuel Sorge. Optimal discretization is fixed-parameter tractable. In *32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1702–1719. 2021.

[Knu74] Donald E. Knuth. Postscript about NP-Hard problems. *SIGACT News*, 6(2):15–16, 1974.

[Knu97] Donald E. Knuth. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Third edition, 1997.

[Kog95] Alexander Kogan. On the essential test sets of discrete matrices. *Discrete Applied Mathematics*, 60(1):249–255, 1995.

[KPSV05] Jeong Kim, Oleg Pikhurko, Joel Spencer, and Oleg Verbitsky. How complex are random graphs in first order logic? *Random Structures and Algorithms*, 26(1-2):119–145, 2005.

[KSPSV02] Farinaz Koushanfar, Sasha Slijepcevic, Miodrag Potkonjak, and Alberto Sangiovanni-Vincentelli. Error-tolerant multi-modal sensor fusion. In *IEEE CAS Workshop on Wireless Communication and Networking*, pages 5–6, 2002.

[KT05] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., 2005.

[KT08] Matias Korman and Takeshi Tokuyama. Optimal insertion of a segment highway in a city metric. In *14th Annual International Conference on Computing and Combinatorics (COCOON)*, page 611–620, 2008.

[Lee04] D. T. Lee. Interval, segment, range, and priority search trees. In *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004.

[LHC20] Antoine Lobstein, Olivier Hudry, and Irène Charon. Locating-domination and identification. In *Topics in Domination in Graphs*, volume 4, pages 251–299. 2020.

[LHH+09] Antoine Lobstein, Olivier Hudry, Iiro Honkala, Irène Charon, and David Auger. Edge number, minimum degree, maximum independent set, radius and diameter in twin-free graphs. *Advances in Mathematics of Communications*, 3(1):97–114, 2009.

[Lob] Antoine Lobstein. Watching systems, identifying, locating-dominating and discriminating codes in graphs: a bibliography. *https://www.lri.fr/ lobstein/debutBIBidetlocdom.pdf*.

[LT08] Moshe Laifenfeld and Ari Trachtenberg. Identifying codes and covering problems. *IEEE Trans. Information Theory*, 54(9):3929–3950, 2008.

[LTCS09] Moshe Laifenfeld, Ari Trachtenberg, Reuven Cohen, and David Starobinski. Joint monitoring and routing in wireless sensor networks using robust identifying codes. *Mobile Network Applications*, 14(4):415–432, 2009.

[McC85] Edward M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985.

[MIH81] Shigeru Masuyama, Toshihide Ibaraki, and Toshiharu Hasegawa. The computational complexity of the m-center problems on the plane. *The Transactions of the Institute of Electronics and Communication Engineers of Japan*, E-64(2):57–64, 1981.

[MMS20] Neeldhara Misra, Harshil Mittal, and Aditi Sethia. Red-blue point separation for points on a circle. In *CCCG*, pages 266–272, 2020.

[MNP21] Raghunath Reddy Madireddy, Subhas C. Nandy, and Supantha Pandit. On the geometric red-blue set cover problem. In *15th International Conference and Workshops, WALCOM*, pages 129–141, 2021.

[Mon06] Julien Moncel. On graphs on n vertices having an identifying code of cardinality $\lfloor \log_2(n+1) \rfloor$. *Discrete Appl. Math.*, 154(14):2032–2039, 2006.

[MR10] Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete and Computational Geometry*, 44(4):883–895, 2010.

[MRT14] M. Milanič, R. Rizzi, and A.I. Tomescu. Set graphs. II. complexity of set graph recognition and similar problems. *Theoretical Computer Science*, 547:70–81, 2014.

[MS85] Bernard M. E. Moret and Henry D. Shapiro. On minimizing a set of tests. *SIAM Journal on Scientific and Statistical Computing*, 6(4):983–1003, 1985.

[MS09] Tobias Müller and Jean-Sébastien Sereni. Identifying and locating-dominating codes in (random) geometric networks. *Comb. Probab. Comput.*, 18(6):925–952, 2009.

[MV80] Silvio Micali and Vijay V. Vazirani. An O(sqrt(|V|) |E|) algorithm for finding maximum matching in general graphs. In *21st Symp. on Foundations of Computer Science*, pages 17–27, 1980.

[NAE+17] Adam B Noel, Abderrazak Abdaoui, Tarek Elfouly, Mohamed Hossam Ahmed, Ahmed Badawy, and Mohamed S Shehata. Structural health monitoring using wireless sensor networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 19(3):1403–1423, 2017.

[NAH02] Subhas C. Nandy, Tetsuo Asano, and Tomohiro Harayama. Shattering a set of objects in 2D. *Discrete Applied Mathematics*, 122(1):183–194, 2002.

[NF77] Patrenahalli M. Narendra and Keinosuke Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, C-26(9):917–922, 1977.

[Pat71] E.A. Patrick. Interactive pattern analysis and classification utilizing prior knowledge. *Pattern Recognition*, 3(1):53–71, 1971.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., 1982.

[Ré61] A Rényi. On random generating elements of a finite boolean algebra. *Acta Scientiarum Mathematicarum Szeged*, 22:75–81, 1961.

[RSTU04] Saikat Ray, David Starobinski, Ari Trachtenberg, and Rachanee Ungrangsi. Robust location detection with sensor networks. *IEEE J. Selected Areas Communications*, 22(6):1016–1025, 2004.

[RUP+03] Saikat Ray, Rachanee Ungrangsi, De Pellegrini, Ari Trachtenberg, and David Starobinski. Robust location detection in emergency sensor networks. In *22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 2, pages 1044–1053. IEEE, 2003.

[RWLI75] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour. An algorithm for a selective nearest neighbor decision rule (corresp.). *IEEE Transactions on Information Theory*, 21(6):665–669, 1975.

[Sch03] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

[SR84] P. J. Slater and D. F. Rall. On location domination numbers for certain classes of graphs. *Congressus Numerantium*, 45:97–106, 1984.

[SS10] Suk Jai Seo and Peter J Slater. Open neighborhood locating dominating sets. *Australas. J Comb.*, 46:109–120, 2010.

[Tho99] Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, 1999.

[Tou02] G. Toussaint. Open problems in geometric methods for instance-based learning. In *Japanese Conference on Discrete and Computational Geometry (JCDCG)*, pages 273–283, 2002.

[Tou05] G. Toussaint. Geometric proximity graphs for improving nearest neighbor methods in instance-based learning and data mining. *Int. J. Comput. Geom. Appl.*, 15(2):101–150, 2005.

[Tov84] Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

[Ull74] Julian R. Ullmann. A use of continuity in character recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-4(3):294–300, 1974.

[UTS04] Rachanee Ungrangsi, Ari Trachtenberg, and David Starobinski. An implementation of indoor location detection systems based on identifying codes. In *Intelligence in Communication Systems*, pages 175–189, 2004.

[vBBB+14] René van Bevern, Robert Bredereck, Laurent Bulteau, Jiehua Chen, Vincent Froese, Rolf Niedermeier, and Gerhard J. Woeginger. Star partitions of perfect graphs. In *International Colloqium on Automata, Languages, and Programming (ICALP)*, pages 174–185, 2014.

[VC15] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of Complexity*, pages 11–30. 2015.

[VL91] Jan Van Leeuwen. *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity.* MIT Press, 1991.

[Wil91] Gordon Wilfong. Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry (SCG)*, page 224–233, 1991.

[WL72] W. R. Willcox and S. P. Lapage. Automatic Construction of Diagnostic Tables. *The Computer Journal*, 15(3):263–267, 1972.

[WLH80] W. R. Willcox, S. P. Lapage, and B Holmes. A review of numerical methods in bacterial identification. *Antonie van Leeuwenhoek*, 46(3):233–299, 1980.

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, 2011.

[ZXLL04] Baihua Zheng, Jianliang Xu, Wang-Chien Lee, and D. T. Lee. Grid-partition index: a hybrid method for nearest-neighbor queries in wireless location-based services. *The VLDB Journal*, 15(1):21–39, 2004.

[ZZ95] Igor E. Zvervich and Vadim E. Zverovich. An induced subgraph characterization of domination perfect graphs. *Journal of Graph Theory*, 20(3):375–395, 1995.