Dahal, Ram Sewak

2017

Algorithms for weighted coloring problems

Department of Mathematics and Computer Science

ALGORITHMS FOR WEIGHTED COLORING PROBLEMS

RAM SEWAK DAHAL
Bachelor of Engineering In Information Technology, Pokhara University, 2008
Master of Engineering In Information and Communication Technology, Asian Institute of Technology, 2011

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

DOCTOR OF PHILOSOPHY

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

ALGORITHMS FOR WEIGHTED COLORING PROBLEMS

RAM SEWAK DAHAL

Date of Defence: April 24, 2017

Dr. Robert Benkoczi Supervisor	Associate Professor	PhD
Dr. Daya Ram Gaur Co-Supervisor	Professor	PhD
Dr. Shahadat Hossain Committee Member	Professor	PhD
Dr. Saurya Das Committee Member	Professor	PhD
Dr. Yash Aneja External Examiner	Professor	PhD
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	PhD

Dedication

To my family for all their support and sacrifices.

To Jayati for her unending support.

Abstract

In this thesis, we studied a generalization of vertex coloring problem (VCP). A classical VCP is an assignment of colors to the vertices of a given graph such that no two adjacent vertices receive the same color. The objective is to find a coloring with the minimum number of colors. In the first part of the thesis, we studied the weighted version of the problem, where vertices have non-negative weights. In a weighted vertex coloring problem (WVCP) the cost of each color depends on the weights of the vertices assigned to that color and equals the maximum of these weights. Furthermore, in WVCP, the adjacent vertices are assigned different colors, and the objective is to minimize the total cost of all the colors used. We studied WVCP and proposed an $O(n^2 \log n)$ time algorithm for binary trees. Additionally, we studied WVCP in cactus paths. We proposed sub-quadratic and quadratic time algorithms for cactus paths.

We studied a min-max regret version of the robust optimization where the weight of each vertex v is in the interval $[\underline{w}_v, \overline{w}_v]$. The objective of is to find a coloring that has the minimum regret value. We proposed a linear time algorithm for robust coloring on bipartite graphs with uniform upper bound and arbitrary lower bound weights on the vertices. We also gave an integer linear programming (ILP) for the robust weighted vertex coloring problem (RWVCP). We solved a relaxation of the ILP formulation using column generation. We also gave an algorithm based on the branch and price method. Lastly, we performed experiments to study the quality of our algorithms.

Acknowledgments

Firstly, I express my sincere gratitude to my advisor Dr. Robert Benkoczi and co-adviser Dr. Daya Gaur for their continuous support during my PhD study. Their patience, motivation, immense knowledge, and guidance helped me in all of my research work and writing this thesis. I could not have imagined having a better advisor and a mentor for my PhD. I am also grateful to the members of my committee for helping me overcoming numerous obstacles I have been facing throughout my study.

I thank my uncle, Kumar Dahal, for guiding me to the right direction. I am thankful to my fellow graduate students for their feedback, cooperation and of course, friendship. I am also grateful to my friends Amit, Mark, Arnab, Julia, Umair, Parijat, Nabi, Pampha and Sagar, for expecting nothing less than excellence from me. Last but not the least, I appreciate my family: grandparents, parents, brothers and sisters, Sangita and Pratima, for encouraging me throughout this wonderful journey.

Contents

Contents vi

List of Tables viii

List of Figures ix

1 Introduction 1
- 1.1 Graph Coloring Problem . . . 1
- 1.2 Motivation . . . 2
- 1.3 Contributions . . . 4
- 1.4 Organization of The Thesis . . . 5

2 Definitions and Notations 7
- 2.1 Basic Definitions . . . 7
  - 2.1.1 Vertex Coloring . . . 8
  - 2.1.2 Upper Bound on Chromatic Number . . . 8
  - 2.1.3 Applications . . . 9
- 2.2 Weighted Vertex Coloring Problem (WVCP) . . . 9
- 2.3 List Coloring . . . 11
- 2.4 Linear Programming (LP) . . . 12
  - 2.4.1 Column Generation . . . 15
  - 2.4.2 Example . . . 17
- 2.5 A Note on Notation . . . 20

3 Binary Trees 22
- 3.1 Preliminaries . . . 23
  - 3.1.1 Definition . . . 23
  - 3.1.2 Related Works . . . 25
- 3.2 Dynamic Programming . . . 28
- 3.3 Feasible weight sets . . . 29
- 3.4 An Exact Algorithm for Balanced Binary Trees . . . 32
  - 3.4.1 The dynamic programming algorithm (DP) . . . 32
  - 3.4.2 Feasible weight sets for leaf vertices . . . 33
  - 3.4.3 Recursive computation of feasible weight sets . . . 35
  - 3.4.4 Intersection and Union Operations . . . 37
  - 3.4.5 Analysis . . . 39
- 3.5 Example . . . 41
- 3.6 An Efficient Algorithm for Arbitrary Binary Trees . . . 45

3.6.1 Centroid Decomposition (CD) . . . 45
3.6.2 Spine Decomposition (SD) . . . 47
3.6.3 A WVCP algorithm for arbitrary binary trees - the general approach 49
3.6.4 Feasible Weight Sets . . . 49

4 Cactus Paths **54**
4.1 Preliminaries . . . 54
4.2 Weighted Vertex Coloring Problem (WVCP) on Cactus Paths . . . 56
4.2.1 Outline of the algorithm on cactus paths . . . 57
4.2.2 Brief overview of Kavitha *et al.* [51] algorithm on a path . . . 60
4.2.3 Running Time . . . 63
4.2.4 A Case When Boundary Vertices Are Independent . . . 63
4.2.5 Case when the maximum degree vertices are adjacent . . . 67
4.3 An Algorithm for Cycles . . . 71
4.3.1 Correctness . . . 72
4.3.2 Running Time . . . 74

5 Robust Weighted Vertex Coloring **75**
5.1 Robust Optimization . . . 75
5.1.1 Min-max Regret . . . 76
5.1.2 Problem definition . . . 76
5.1.3 Related Works . . . 77
5.2 A Path Graph With Uncertain Weights . . . 79
5.3 Mathematical Formulation of RWVCP . . . 84
5.3.1 Sub-problem . . . 87
5.3.2 Outline of RWVCP Algorithm . . . 88
5.3.3 Branch and Price . . . 89
5.4 Worst Case Scenario Given A Coloring . . . 89
5.4.1 Local Search . . . 93
5.5 Experimental Results . . . 94
5.5.1 Implementation Details . . . 95
5.5.2 Numerical Results . . . 96

6 Conclusion **101**
6.1 Future Study . . . 102
6.1.1 Bound on the number of colors . . . 102
6.1.2 Worst Case Scenario . . . 102
6.1.3 Lower Bound . . . 103

Bibliography **104**

List of Tables

2.1 Notations 21

5.1 Optimal Regret Calculation of the Path of Figure 5.1 82
5.2 Results on standard instances without using branch and price 98
5.3 Branch and price on library instances 98
5.4 Results on pathsusing branch and price 100

List of Figures

1.1 An example of the graph coloring problem where $\chi(G) = 2$ 2

3.1 Two coloring is not optimal. Greedy is not optimal. 27
3.2 Dynamic Programming example 29
3.3 The region in w_3 - w_4 space enclosing the feasible weight set. 30
3.4 List Coloring region when a color is fixed 31
3.5 Change in the boundary line when the weight of one color is fixed and other is decreased 31
3.6 Boundary line for the feasible weight set. 31
3.7 The staircase line is segment $[OP]$. 34
3.8 Base case for $S_3(v)$ 34
3.9 Base case for $S_4(v)$ 34
3.10 A subtree T_v and its children T_x and T_y 36
3.11 Combining the feasible weight sets $S_j(x)$ and $S_k(y)$. 36
3.12 Computing $S_1(v)$ from the nine temporary staircase lines $I_{jk}(v)$; only two temporary staircase lines are depicted. 36
3.13 Calculate the intersection region of two staircase functions 37
3.14 The optimal (w_3, w_4) pair returned by the boundary of $\mathcal{W}_2$. 41
3.15 An example of a binary tree 42
3.16 Base Case of vertices d and e 42
3.17 The staircase functions obtained by the combination of the fixed color 1 with other colors 43
3.18 The staircase functions obtained by the combination of the fixed color 2 with other colors 43
3.19 The staircase functions obtained by the combination of the fixed color 3 with other colors 43
3.20 The staircase functions obtained by the combination of the fixed color 4 with other colors 43
3.21 The union of staircase functions after the intersection when vertex c is fixed to color 1,2,3 and 4 44
3.22 The staircase functions when the vertex a is fixed to color 1 44
3.23 Final staircase function of a obtained by the union of all the staircase functions 45
3.24 Centroid decomposition of a tree 46
3.25 Spine Decomposition; the spine is represented by solid lines and the search tree is represented by thin lines [12]. 47
3.26 Spine Decomposition; x and y are leaves of the search tree 48
3.27 A leaf vertex in a decomposed tree 50

3.28 Vertex x and vertex y are the leaf the SD and v is the vertex of the binary search tree . . . 52
3.29 Vertex x is the leaf of SD and vertex y is any internal vertex in the binary search tree. T_1 represents the components hanging on the spine vertices . . 52
3.30 Vertex x and vertex y are internal vertices of the binary search tree and T_1 and T_2 are the components hanging of the spine vertices . . . 52

4.1 The odd and even vertices weighted cycle. The optimal WVCP on (a) is 16+4+2 = 22 and (b) is 16+3+2=21. . . . 55
4.2 A 4-colorable cactus path where weight 8 receives color 1, weight 4 receives color 2, weight 2 receives color 3, and weight 1 receives color 4 and the optimal cost is 15 . . . 55
4.3 Coloring the graph with Δ number of colors . . . 57
4.4 Two maximum degree vertices adjacent to each other . . . 57
4.5 A node representing a cycle or a path . . . 59
4.6 Two components of a cycle preprocessed with the algorithm [51] . . . 59
4.7 Boundary node are fixed to some colors . . . 64
4.8 A search tree on the top of a Cactus Path . . . 65
4.9 An internal nodes of the cactus path and it's associate components . . . 65
4.10 A Cactus Path with the maximum degree vertices adjacent to each other . . 67
4.11 Interchange of colors . . . 68
4.12 Feasible region before and after the introduction of w_4. . . . 71

5.1 A Weighted Path . . . 80
5.2 An outline of RWVCP algorithm . . . 94
5.3 Time comparison with and without branch and price . . . 98
5.4 Regret comparison: branch and price (BP) and worst case regret (UB) . . . 99
5.5 Regret comparison: fractional regret (r), branch and price (BP) and worst case formulation regret (UB) . . . 99

Chapter 1

Introduction

In this chapter of the thesis, we introduce the classical vertex coloring problem with its weighted generalization. In the subsequent sections, we offer the motivation to study this problem. An outline of the next chapters of the thesis is also provided.

1.1 Graph Coloring Problem

Graph coloring is a well known and extensively researched subject in the field of graph theory. It is defined as an assignment of colors to elements of graphs subject to certain constraints. It has many applications and conjectures which are studied by various mathematicians and computer scientists around the world. Graph coloring dates back to the middle of the nineteenth century, where the original problem was to color an administrative map of England by counties using only 4 colors [31], with the added constraints of assigning different colors to adjacent counties sharing a border. From there, the coloring problem was further generalized and expanded to a political map. This introduced the famous *four coloring problem*: is it possible to color any political map, satisfying the above constraints, using only four colors? The problem was solved by Appel and Haken [3] in 1976 using a computer program.

Two types of graph coloring problems have been considered in the literature: an edge coloring problem and a vertex coloring problem. An edge coloring problem assigns a color to each edge such that no two adjacent edges share the same color. Among these two graph coloring problems, the vertex coloring problem (VCP) has been extensively studied. In the

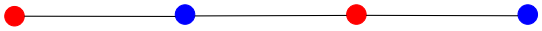


Figure 1.1: An example of the graph coloring problem where $\chi(G) = 2$

VCP, a graph $G = (V, E)$ is given, where V is the set of vertices, E is the set of edges, and the objective is to find the minimum number of colors for vertices such that no two adjacent vertices are assigned the same color. The minimum number of colors needed for a graph is also known as the chromatic number of the graph ($\chi(G)$). Computing $\chi(G)$ is one of the original NP-complete problems on Karp's list of the twenty-one problems.

We studied weighted version of the vertex coloring problem, where nodes have non-negative weights. In a weighted vertex coloring problem (WVCP) the cost of each color depends on the weights of the vertices assigned to that color and equals the maximum of these weights. Furthermore, in WVCP, the adjacent vertices are assigned different colors, and the objective is to minimize the total cost of all the colors used. The WVCP is also known to be NP-complete as the problem reduces to the determination of the chromatic number when all the weights are unitary.

Lastly, we also studied robust weighted vertex coloring problem (RWVCP). In the robust version, the uncertainty in the weights is represented using an interval, where the weight is chosen from the interval. No assumptions are made about the underlying probability distribution. A particular assignment of values to the weights from the intervals is called a scenario. The goal is to find a coloring X for which the difference between the cost of X under the worst-possible scenario and the cost of the optimal coloring for that scenario is minimized. Computing the optimal cost of a scenario requires solving the chromatic number problem, which is NP-complete.

1.2 Motivation

There are a number of motivations for this study of the problem. One of the applications of WVCP is on the scheduling of jobs with conflicts in a multiprocessor environment

[68]. When scheduling jobs onto a processor, the jobs accessing the same resource are not scheduled together. Furthermore, each job has a different processing time. The different processing times can be treated as weights and the objective of minimizing makespan (the time difference between start and the finish time of the jobs) can be modeled as WVCP. One such generalization is batch scheduling where an edge of the graph represents conflicts between pairs of jobs that cannot be processed in the same batch [39]. The problem of batch scheduling naturally occurs in distributed systems [39].

WVCP arises in the scheduling of data transmissions in a time division multiple access (TDMA) wireless network [59, 64]. One example of such TDMA technology is the Worldwide Interoperability for Microwave Access (WiMAX) standard which is responsible for a large portion of the data mobility services today. The WiMAX standard does not specify any channel allocation algorithms. This is to allow the most flexible and efficient use of resources possible. Moreover, the duration of the time slots in the WiMAX standard need not be uniform. Given a set of clients with different bandwidth requirements, a channel allocation scheme in a mesh network (see [2] for the definition of mesh network) using WiMAX technology seeks to group the transmissions in such a way that interference does not occur and the channel is used efficiently. Such a schedule can be obtained by solving the WVCP problem in the interference graph for the network. The vertex weights correspond to the bandwidth requirements for the devices participating in the communication. In particular, line, ring, and tree topologies are commonly used in telecommunication networks [52], hence our results on trees and cactus paths may be of interest in this domain as well.

The Matrix Decomposition Problem in Time Division Multiple Access Traffic Assignment can also be modeled as WVCP. The matrix decomposition problem arises in the context of satellite communication systems optimization. In this problem, a square $N \times N$ matrix T with m nonzero entries (traffic matrix) is decomposed into k matrices such that there is at most one non-zero element in each column and row, and each non-zero entry of T appears in one and only one matrix of the decomposition. The cost of the sub-matrix

is determined by the maximum of its non-zero elements. The objective is to minimize the total cost of the decomposition. This problem can be modeled as WVCP. The non-zero entries correspond to the vertices, and there is an edge when two entries are from the same row or the same column of T. A color in this graph corresponds to a sub-matrix and its weight is determined by the maximum weight of some vertex. Thus, a search for an optimal matrix decomposition reduces to WVCP. Ribeiro et al. [73] have provided an exact algorithm based on column generation and branch and bound (in the worst case these algorithms have exponential running time). Prais et al.[70] proposed a heuristic approach based on a Greedy Randomized Adaptive Search Procedure (GRASP). Another application of WVCP is the dynamic storage allocation problem also known as the interval coloring problem [15, 38].

1.3 Contributions

Throughout this thesis, we study two versions of the coloring problem: the WVCP where each vertex weight is deterministic, and robust weighted vertex coloring problem where each vertex weight form an interval. The objective in WVCP is to find a coloring with minimum weight. The objective in RWVCP is to find a coloring with minimum regret (see Chapter 5 for the definitions). Following are the two main studies in this thesis:

1. We studied the WVCP where the vertices have deterministic weights. For trees, the exact complexity of the weighted vertex coloring problem is still open [51, 68]. The problem most likely is not NP-complete as a sub-exponential algorithm is known [4]. Apart from from the algorithm by Kavitha et al.[51] on paths and the polynomial time approximation scheme (PTAS) [68] on trees (see Definition 3.10 of Chapter 3), no other results are known for trees. The only known method to solve WVCP in binary trees is based on the list coloring algorithm [54]. The list coloring technique solves the WVCP in binary trees in $O(n^4)$ time [54].

- In this thesis, we provided a $O(n^2 \log n)$ time algorithm for solving WVCP in binary trees (Chapter 3).
- We also provide a $O(n \log^2 n)$ time algorithm for solving WVCP in cactus paths (Chapter 4).

A portion of this study appears in [11].

2. We studied the robust vertex coloring problem (RWVCP) where the weight of each node v is in the interval $[\underline{w}_v, \overline{w}_v]$. Here, $\underline{w}_v$ is the lower bound weight and $\overline{w}_v$ is the upper bound weight. A uniform instance is where all the vertices have the same interval $[\underline{w}, \overline{w}]$ associated with them.

   - We propose a linear time algorithm for the robust coloring of bipartite graphs with uniform upper bound and arbitrary lower bound weights on the vertices.
   - We formulate the RWVCP problem as an integer programming (IP) formulation where the variables and the constraints are exponential in number. We gave a math-heuristic method based on column generation and local search.
   - We also describe a branch and price inexact method to obtain an integral coloring.
   - Lastly, we provide a quadratic programming formulation to determine a worst case scenario, given robust coloring.

   To the best of our knowledge, this is the first empirical study of RWVCP. A portion of this study appears in [10].

1.4 Organization of The Thesis

In Chapter 2 we define the various terms used in this thesis.

In Chapter 3 we study the WVCP problem on binary trees and gave a $O(n^2 \log n)$ algorithm. This is an improvement over the current best algorithm with running time $O(n^4)$. The main technique used here is prune and search with dynamic programming.

In Chapter 4 we further extend the algorithm of Kavitha et al. [51] to solve WVCP in cactus paths.

In Chapter 5 we provide a mathematical formulation for the min-max regret problem, RWVCP. In this chapter, we provide a heuristic based on column generation and local search. We gave a method based on branch and price to obtain an integral solution. We also show how to model the problem of determination of the worst case scenario as a quadratic program. We gave a linear time algorithm for RWVCP on a bipartite graph with uniform upper bound and the arbitrary lower bound weight on vertices.

We conclude in Chapter 6 with some open problems.

Chapter 2

Definitions and Notations

In this chapter, we define various terms that are used throughout the thesis. The definitions provided in this chapter can be found in the literature [27].

2.1 Basic Definitions

The graph $G = (V, E)$ in this thesis is simple and finite graph comprising a set V of vertices together with a set E of edges. We denote by $n = |V|$, the number of vertices, and $m = |E|$, the number of edges. When two vertices are the end points of the same edge, they are called *adjacent*. The *neighbors* of a vertex mean the vertices that are adjacent.

Definition 2.1 (Subgraph)**.** Let H be a graph with vertex set $V(H)$ and edge set $E(H)$, and let G be a graph with vertex set $V(G)$ and edge set $E(G)$. We say that H is a subgraph of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

Definition 2.2 (Induced Subgraph)**.** A subgraph H of G is called induced, if for any two vertices u, v in H, u and v are adjacent in H if and only if they are adjacent in G.

The neighborhood of a vertex $v \in V$ in a graph G is the induced subgraph of G consisting of all vertices adjacent to v. The degree $d_G(v) = d(v)$ of a vertex v of a graph G is the number of edges incident to the vertex. The maximum degree of G is $\Delta(G) = \max\{d(v) : v \in V\}$.

Definition 2.3 (Independent Set)**.** An independent set is a set of vertices I such that, no two vertices from I are adjacent.

A *path* is a graph $P = (V,E)$ of the form $V = \{v_0, v_1, ...v_n\}$ and $E = \{(v_0,v_1),(v_1,v_2),(v_2,v_3),...,(v_{n-1},v_n)\}$ where the vertices v_i are all distinct. The vertices v_0 and v_n are the end vertices, and vertices $v_1, v_2, ..., v_{n-1}$ are internal vertices. The path P is denoted by $(v_0, v_1, ..., v_n)$.

Definition 2.4 (Connected Graph)**.** A graph is defined as a connected graph if there is a path between every pair of vertices.

2.1.1 Vertex Coloring

A vertex coloring of graph G is a mapping $c : V(G) \to X$. The elements of X are colors; the set of vertices of one color form a color class α. If $|X| = k$, we say that c is k-coloring (often we use $X = \{1,2,...,k\}$). A coloring is *proper* or *feasible* if adjacent vertices have different colors. A graph is k-colorable if it has a feasible k-coloring. The *chromatic number* $\chi(G)$ is the least k such that G is k-colorable. In a feasible coloring, each color class forms an independent set. Hence, a k-coloring may also been seen as a partition of vertex set into k disjoint independent sets: $\alpha_i = \{v : c(v) = i\}$ where $1 \leq i \leq k$. A bipartite graph can be colored with 2-colors.

2.1.2 Upper Bound on Chromatic Number

The most widespread algorithm to obtain an upper bound on $\chi(G)$ is the greedy algorithm [61]. In a greedy coloring, we consider the vertices in a specific order $v_1, v_2, ..., v_n$. The algorithm assigns v_i, the smallest available color not used by v_i's neighbors in $v_1, v_2, ..., v_{i-1}$, and introduces a new color if needed.

Theorem 2.5 (Upper bound on $\chi(G)$ [14])**.** *Given a graph G, $\chi(G) \leq 1 + \Delta$ where Δ is the maximum degree in G.*

Proof. The above greedy coloring produces a proper coloring, since we are careful to avoid conflicts each time we color a new vertex. As each vertex has at most Δ neighbors, we will

always have at least one choice of a color that is less than $\Delta+1$; therefore, this creates a proper coloring of G that uses at most $\Delta+1$ colors. Hence, $\chi(G) \leq 1+\Delta$.

□

The bound of $\Delta+1$ is a weak upper bound on $\chi(G)$ and choosing the vertex ordering is an issue. We can use Brooks' Theorem [14] to determine whether the chromatic number is $\Delta+1$ or not. Indeed there is a vertex ordering relative to which the greedy algorithm yields an optimal coloring. But there are $n!$ possible orderings and it is difficult to find a good one. For any $k \geq 3$, it is NP-complete to decide if a graph on n vertices is k-colorable [36]. It is easy to decide if a graph is 1-colorable since such a graph has no edges. Determining whether a graph is 2-colorable graph can also be done in polynomial time using breadth-first search [19]. Lund and Yannakakis [58] have shown that it is NP-hard to approximate the chromatic number within n^{ε} for some positive constant ε unless $P = NP$. The best known approximation algorithm for a general graph is that of Karger *et al.* [47]. The current best approximation ratio for a 3-colorable graph is $n^{0.2111}$ [7].

Theorem 2.6 (Brooks' Theorem [14])**.** *For any connected undirected graph G with maximum degree Δ, $\chi(G) \leq \Delta$ unless G is a complete graph or an odd cycle, in which case $\chi(G) = \Delta+1$.*

2.1.3 Applications

The vertex coloring problem (VCP) has many applications in the field of engineering, including scheduling [55], timetabling [25], register allocation [16], frequency assignment [35] and in communication networks [77]. For a survey on vertex coloring problems see [61].

2.2 Weighted Vertex Coloring Problem (WVCP)

Let $G(V,E)$ be an undirected graph where V is the vertex set and E the edge set and each $v \in V$ has a positive weight w_v. Let X be a feasible coloring of G, and let $\alpha_1, \alpha_2 ..., \alpha_k$

be color classes. For any color class α of G, we define $w(\alpha) = max\{w_v | v \in \alpha\}$ to be the weight of a color class (or simply the weight of a color). The weighted vertex coloring problem, first introduced by Guan and Zhu [38], is to find a proper k-coloring of vertices of G so as to minimize the sum of the weights of the color class, i.e. find a coloring that minimizes $\sum_{i=1}^{k} w(\alpha_i)$ [68].

We first establish some properties in bipartite graphs. The following theorems help us to establish bounds on the number of colors needed in a bipartite graph and the distribution of weights of color classes in an optimal coloring. We first consider the claim by Pemmaraju et al. [68] on color class weight.

Lemma 2.7. *Let G be a bipartite graph. In any optimal weighted coloring , let $\{\alpha_1, \alpha_2, ..., \alpha_k\}$ be color classes of G with $w_i = w(\alpha_i)$ and $w_1 \geq w_2 ... \geq w_k$, we have that $w_i \geq \sum_{j=i+1}^{k} w_j$, $i = 1, 2, ...k-1$.*

Proof. By contradiction, suppose $w_i < \sum_{j=i+1}^{k} w_j$. We know that, bipartite graphs can be properly colored by two colors. The weight of this two coloring is at most $2w_i$ because the weight of each color class is at most w_i if w_i is the maximum weight in the graph. However, the optimal coloring has weight $\sum_{i}^{k} w_i$. Hence, $2w_i = w_i + w_i < w_i + \sum_{j=i+1}^{k} w_j = \sum_{j=i}^{k} w_j$, which is a contradiction. □

Next is a greedy algorithm proposed by Guan and Zhu [38] for the upper bound on the number of colors for weighted graphs.

Lemma 2.8. *Let G be the vertex weighted graph with the maximum degree Δ, then the number of colors needed in any optimal coloring of WVCP is $\Delta + 1$.*

Proof. Let $\{\alpha_1, \alpha_2, ..., \alpha_k\}$ be an optimal k-coloring of G. Let $w_i = w(\alpha_i)$ and w.l.o.g let $w_1 \geq w_2 ... \geq w_k$. To obtain a contradiction, suppose that $k > \Delta + 1$. For each vertex $v \in \alpha_k$, there is a color class $\alpha_i : i < k$ such that v has no neighbors in α_i. Re-color every vertex v in α_k with the color i, the color class to which v has no neighbors. This recoloring does not

increase the weight of the coloring. We now have a coloring with smaller number of colors with no larger weight. Iterate the argument till $k = \Delta + 1$. □

2.3 List Coloring

Another generalization of the vertex coloring problem is list coloring problem. In the list coloring problem, a possible list of colors is assigned to each vertex of a graph and every vertex can have a different list of colors, $L(v)$. Vertices can take colors only from the associated lists in any proper colorings.

Problem 1. (List Coloring Problem) [54]: Given a graph $G = (V,E)$ and a list of colors $L(v)$ for every vertex $v \in V$, does there exist an assignment of colors $f(v)$ for each vertex $v \in V$ in such a way that $f(v) \in L(v)$ and f is feasible coloring?

The job scheduling problem can be modeled as a list coloring problem. Here, the vertices represent the machines, and the lists represent possible jobs. Each job requires some non-sharable resource to run, and if two jobs require the same resource, they cannot run simultaneously and hence an edge [72]. A coloring partitions the jobs into independent sets. The set of jobs in each color class can run simultaneously.

List coloring is NP-complete even if all lists have length three [54]. Furthermore, Kratochvil et al. [54] proved that list coloring remains NP-complete even if: each color $x \in \bigcup_{v \in V} L(v)$ occurs in at most three sets $L(v)$, each vertex $v \in V$ has degree at most three, and G is a planar graph.

Let $G(V,E)$ be a vertex-weighted undirected graph where each $v \in V$ has associated a positive weight w_v. Given weights $w_1 \geq w_2 \geq ...w_k$, we can construct a list coloring problem as follows. The colors are $1,2,...,k$, each node v with weight $w_v \geq w_i$ cannot be assigned color i, all the other colors (such that $w_v \leq w_i$) are in the list $L(v)$. A feasible solution to the list coloring problem has cost $w_1 + w_2 + ... + w_k$. The problem now reduces to enumeration of the weights which can be done using dynamic programming and search. The ideas are from [68].

In the following sections, we explain an exact method to solve WVCP in a graph. The exact method is based on linear programming and branch and bound.

2.4 Linear Programming (LP)

The topic of linear programming is extensively studied in computer science and mathematics, and covered in many books [17, 67]. In this section, we provide standard definitions related to linear programming and refer to [17, 67, 22, 40] for proof of the statements. The simplex algorithm, which was created by Dantzig, is a popular method for solving linear programs [22].

Definition 2.9 (Objective Function)**.** An optimal function is the linear function representing cost to be maximized of minimized subject to the constraints.

Definition 2.10 (Constraints)**.** Constraints in a linear programming is a system of linear inequalities.

Definition 2.11 (Convex set [40])**.** A set $S \in \mathbb{R}^m$ is called a convex set if $\forall x, y \in S$ and $\lambda \in [0,1]$, $\lambda x + (1-\lambda)y \in S$.

Before we further discuss the simplex method, let us consider a linear program:

$$\min \quad c^T x \tag{2.1}$$

$$\text{subject to: } Ax \geq b, \tag{2.2}$$

$$x \geq 0 \tag{2.3}$$

where $x = (x_1, x_2, ..., x_n)$ are the variables, $c = (c_1, c_2, ..., c_n)$ are the coefficients of the objective function, A is a $m * n$ matrix, and $b = (b_1, b_2, ..., b_m)$ are constants. A feasible region is the set of all possible points that satisfy the problem's constraints 2.2 and 2.3. A linear program can be *feasible* or *infeasible* depending on whether a solution exists. The

values of x that satisfy all the constraints are the feasible values. A linear program is unbounded if, and only if, for every $M \in \mathbb{R}$ there exists a feasible solution x with $c^T x \geq M$. The values of x_i are the vertices or the extreme points of the polytope if the subset of column vectors A_i corresponding to non-zero entries of x are linearly independent [67]. Such an extreme point is also called a *basic feasible solution (BFS)*. An optimal solution to a linear program is the feasible solution with the largest objective function value (for a maximization problem) or the smallest objective function value (for a minimization problem). In the next paragraph, we introduce the simplex method which is a popular method for solving linear programs.

Let us again consider the linear program given in equations 2.1 to 2.2 and transform it into *standard form* by introducing a variable $s_j, j = 1,2,...m$ (also called *a slack* variable) for each constraint, which is:

$$\min \quad c^T x \tag{2.4}$$

$$\text{subject to: } Ax - s = b, \tag{2.5}$$

$$x \geq 0, s \geq 0 \tag{2.6}$$

A subset of column B of a constraint matrix A is called *basis*, if the matrix of columns corresponding to B, i.e. A_B, has linearly independent rows; in other words, A_B is non-singular. Similarly, a solution x is called a *basic* if there is basis B such that $x_j = 0$ for $j \neq B$ and $x_B = A_B^{-1} b$. If the solution x is also feasible, i.e. $A_B^{-1} b \geq 0$ in addition to basic, then it is called a *basic feasible solution*. In a basic solution, $(n-m)$ zero valued variables (n variables and m constraints) are *non-basic variables* and remaining m variables are *basic variables*.

The simplex method that works on the standard form, is an iterative algorithm that starts with a basic feasible solution and moves a neighboring basic feasible solution which

improves the objective function value. In each iteration, we need a variable that enters the basis and a variable that leaves the basis. There are numerous rules for the entering and leaving variables [17, 75]. In the next part of this section, we introduce the dual program.

Consider a linear program (we call it *primal*) of the following type:

$$\min \quad \sum_{j=1}^{n} c_j x_j \tag{2.7}$$

$$\text{subject to:} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \forall 1 \leq i \leq m \tag{2.8}$$

$$x_j \geq 0, \qquad \forall 1 \leq j \leq n \tag{2.9}$$

The dual program of the above linear program is given below:

$$\max \quad \sum_{i=1}^{m} b_i y_i \tag{2.10}$$

$$\text{subject to:} \quad \sum_{i=1}^{m} a_{ij} y_i \leq c_j, \forall 1 \leq j \leq n \tag{2.11}$$

$$y_i \geq 0, \qquad \forall 1 \leq i \leq m \tag{2.12}$$

The following weak duality theorem and strong duality theorem are very useful and the proof of these theorems can be found in the book by Vanderbei [75] :

Theorem 2.12 (Weak duality theorem)**.** *If* $(x_1, x_2, ..., x_n)$ *is feasible for the primal and* $(y_1, y_2, ..., y_n)$ *is feasible for dual, then* $\sum_i y_i b_i \leq \sum_j c_j x_j$.

Theorem 2.13 (Strong duality theorem)**.** *If the primal has an optimal solution* $(x_1^*, x_2^*, ..., x_n^*)$*, then the dual also has an optimal solution,* $(y_1^*, y_2^*, ..., y_n^*)$ *such that* $\sum_i y_i^* b_i = \sum_j c_j x_j^*$.

We assume the primal and the dual given by equations (2.7 - 2.9) and (2.10 - 2.12)

respectively.

2.4.1 Column Generation

The column generation method is another method to solve a linear programming (LP) with exponentially many variables by iteratively adding variables. This method is also known as Dantzig-Wolfe decomposition [21]. Column generation is based on the fact that in the simplex method, the solver does not need access to all the variables of the problem simultaneously. A solver begins with a particular subset of the constrained variables, called *the basis*, and then uses the reduced cost method to determine which other variables to use in order to improve the current solution [17]. For a survey of column generation see [57, 26]. We will use the matrix notation (for more detail, see Chvatal [17]).

Let the columns of A corresponding to the basic variables x_B and non-basic variables x_N be A_B and A_N, respectively. In standard form $Ax = b$, this equation can be written as:

$$A_B x_B + A_N x_N = b \tag{2.13}$$

which can further be written as:

$$A_B x_B = b - A_N x_N \tag{2.14}$$

Since the square matrix A_B is non-singular, we can multiply both sides of equation 2.14 by A_B^{-1} to obtain:

$$x_B = A_B^{-1} b - A_B^{-1} A_N x_N \tag{2.15}$$

Similarly, we can represent the cost vector c in terms of c_B and c_N for basic and non-basic variables respectively. The objective function then becomes $c_B x_B + c_N x_N$. Substituting for x_B in the objective function we get:

$$z = c_B(A_B^{-1}b - A_B^{-1}A_N x_N) + c_N x_N = c_B A_B^{-1}b + (c_N - c_B A_B^{-1}A_N)x_N \tag{2.16}$$

Replacing A_B by B as a basis matrix, the dictionary is:

$$x_B = B^{-1}b - B^{-1}A_N x_N \tag{2.17}$$

$$z = c_B B^{-1}b + (c_N - c_B B^{-1}A_N)x_N \tag{2.18}$$

From the above dictionary (2.17 - 2.18), it can be seen that the cost of non-basic variables is $c_N - c_B B^{-1}A_N$ where $c_B B^{-1}$ is the values of the dual variables. Thus, $c_N - c_B B^{-1}A_N = c_N - yA_N$, is the **reduced cost** for the non-basic variables. The expression of reduced cost (r) on column j of the dictionary is given by $r_j = c_j - yA_j$. For a minimization problem, the stopping criteria for the simplex algorithm is that the cost of all the reduced cost variables is non-negative. Whenever there is a column with **negative reduced cost**: the column is a candidate to enter the basis in the next step of the iteration. Thus, the main idea of column generation is to first consider the restricted LP with few variables in the primal basis called the **master problem**. After optimally solving the master problem, we will know the value of the dual variables. Then, we ask the question: *is there a column j not currently in the master problem, such that its reduced cost r_j in current basic feasible solution is negative?* The answer depends on the coefficients a and c since $r_j = c_j - \sum_{i=1}^{m} a_{ij}y_j$. If the answer is "NO", the current master problem is optimal. Furthermore, if the answer is "YES" we insert the column with negative reduced cost into the master problem and repeat the same process again. In algorithm 1, we list the basic steps of the column generation

procedure [21]. Note that in step 5, we can also find a column with most negative reduced cost.

Algorithm 1 Column Generation

1: Create an initial basic feasible solution
2: **while** True **do**
3: Solve the master problem
4: Solve the subproblem
5: Find the column with the most negative reduced cost and add it to the master problem
6: Exit the algorithm if there is no column with a negative reduced cost
7: **end while**

In the following Example 2.4.2, we describe a solution to WVCP using the column generation procedure.

2.4.2 Example

We describe how to solve WVCP using the column generation procedure. Let us begin with the formulation of VCP and extend it to WVCP. Mehrotra et al. [63] provide an exact method for solving VCP using the column generation approach and branch and bound which is known as branch and price. Each variable in their formulation represents an independent set in a graph. Since there are an exponential number of independent sets, during computation a large number of columns are generated. These columns are generated by solving a subproblem. Hence, the column generation method involves formulating the master problem as a set covering problem [76] and the column generation subproblem as the weighted independent set problem. The drawback to this technique is that it only solves small to moderate size instances exactly because of the difficult subproblem and the sophisticated branching rules. Malaguti et al. [60] extend this method and give a heuristic for solving larger instances. They give two branching procedures and integrate the exact method and a heuristic method to obtain better upper bounds on VCP. In this illustration, we use the formulation given by Malaguti et al. [59] for WVCP. The formulation is as follows: let w_h be the weight of a color class, h be a color, and x_{vh} be a binary variable that

represents whether a vertex v is assigned the color h or not.

$$\min \quad \sum_{h=1}^{n} z_h \tag{2.19}$$

$$\text{subject to: } z_h \geq w_v * x_{vh}, \qquad \forall v \in V, h = 1...n \tag{2.20}$$

$$\sum_{h=1}^{n} x_{vh} = 1, \qquad \forall v \in V \tag{2.21}$$

$$x_{uh} + x_{vh} \leq 1, \qquad (u,v) \in E, h = 1...n \tag{2.22}$$

$$x_{vh} \in \{0,1\}, z_h \geq 0 \tag{2.23}$$

The relaxed LP of above formulation is:

$$\min \quad \sum_{h=1}^{n} z_h \tag{2.24}$$

$$\text{subject to: } z_h \geq w_v * x_{vh}, \qquad \forall v \in V, h = 1...n \tag{2.25}$$

$$\sum_{h=1}^{n} x_{vh} = 1, \qquad \forall v \in V \tag{2.26}$$

$$x_{uh} + x_{vh} \leq 1, \ (u,v) \in E, h = 1...n \tag{2.27}$$

$$x_{vh} \geq 0, z_h \geq 0 \tag{2.28}$$

Constraints (2.25) ensure that the weight of the coloring equals the maximum weight of any vertex, constraints (2.26) force every vertex to be assigned exactly one color, and constraints (2.27) ensure that no adjacent vertices are assigned the same color. This formulation considers $h = \Delta + 1$ number of colors to begin with.

The drawback of this model (2.24 - 2.28) as noted in [59] is that the solution space contains many optimal solutions due to symmetries and the continuous relaxation of this model usually produces weak lower bound on the optimal solution value. To address these draw-

backs, we formulate WVCP as the set cover problem and we use column generation procedure to handle the exponential number of variables. Let X be the family of independent sets (color classes). For each color class $\alpha \in X$, we introduce a binary variable x_α taking value 1 if all the vertices of α receive the same color. Moreover, let $w_\alpha = \max\{w_v : v \in \alpha\}$. Following is a set cover based ILP for WVCP.

$$\min \quad \sum_{\alpha \in X} w_\alpha x_\alpha \tag{2.29}$$

$$\text{subject to:} \quad \sum_{\alpha \in X : v \in \alpha} x_\alpha \geq 1, \forall v \in V \tag{2.30}$$

$$x_\alpha \in \{0,1\} \tag{2.31}$$

$$\max \quad \sum_{v \in V} \pi_v \tag{2.32}$$

$$\text{subject to:} \quad \sum_{v : v \in \alpha} \pi_v \leq w_\alpha, \forall \alpha \in X \tag{2.33}$$

$$\pi_v \geq 0, \forall v \in V \tag{2.34}$$

Constraints 2.30 guarantee a proper coloring. Constraints 2.31 are the integrality constraint. We call the above formulation (2.29-2.31) the master problem. To solve the master problem, we consider a subfamily of all the independent sets of G. By solving the restricted relaxed master problem to optimality we obtain the dual values $\pi_v^*, v \in V$, of the dual variables associated with the constraints (2.30). The dual of the above relaxed formulation (2.29 - 2.31) is given in the formulation (2.32 - 2.34).

The variables (independent sets) to be added to the restricted master problem correspond to the violated dual constraints. We need to solve one subproblem for each w_v to find such a violated constraint. One method is to formulate a generic subproblem which find

a color class α^* for which (2.33) is violated, that is, a color class of total weight larger than w_v in $\mathcal{G} = (V_w, E_w)$, which is the subgraph of G induced by the subset of vertices $V_w = \{v : w_v \leq w_\alpha\}$. Thus, for each w_v, the subproblem can be represented by the following ILP, where binary variables $z_v : v \in V_w$, take the value 1 when vertex v belongs to α^* and 0 otherwise.

$$\max \quad \sum_{v \in V_w} \pi_v^* z_v \tag{2.35}$$

$$\text{subject to: } z_v + z_{v'} \leq 1, \forall (v, v') \in E_w \tag{2.36}$$

$$z_v \in \{0, 1\} \tag{2.37}$$

Constraints 2.36 specify that no two adjacent vertices receive the same color. ILP in (2.35 - 2.37) is the maximum weighted independent set problem (MWISP). Note that this problem is NP-hard for general graphs [37]. If the optimal solution of relaxed MWISP, restricted to vertices $v \in V_w$, has a value greater than w_v, then we have found an independent set with negative reduced cost. We add this negative reduced column to the restricted master problem and iterate. If there are no columns with negative reduced cost produced by the solution of relaxed MWISP then the relaxed restricted model (2.29 -2.31) is optimal. To solve several MWISPs to optimality in a sequence is a time-consuming task. However, we do not need to solve the problem to optimality when new columns with negative reduced cost is available. Optimality of the subproblems is required only to certify that such columns with negative reduced cost do not exist [60].

2.5 Notations

Throughout this thesis, the vertices of the graphs are associated with positive integral weights. Table 2.1 below explains the notation used throughout this thesis.

We use all the above definitions and concepts throughout this thesis. In the next chapter,

Table 2.1: Notations

Symbol	Description
G	Graph
T	Tree
V	Vertex Set
E	Edge Set
α	Color Class
$w(\alpha)$	Weight of Color Class
Δ	Maximum Degree of a Graph
χ	Chromatic Number
X	Coloring
w_v	Weight of Vertex v
$L(v)$	List of Colors for v
$S(v)$	Feasible Weight Set
Φ	Set of Coloring
s	A Scenario
R	Regret
Z	Minimum Regret
w_2	Weight of Color Class 2
w_3	Weight of Color Class 3
w_2^0	Global Lower Bound of Color Class 2
A	Constraint Matrix
A_B	Basis Matrix
A_N	Non-Basis Matrix
OPT	Optimal Solution
$\cap$	Intersection
$\cup$	Union

we describe WVCP on binary trees using the concept of list coloring and dynamic programming. Moreover, we use linear programming concept in the solution of a min-max regret optimization problem described in Chapter 5.

Chapter 3

Weighted Vertex Coloring in Binary Trees

In the last decade, weighted vertex coloring problem (WVCP) piqued the interest of an increasing number of researchers. Aside from its importance in practice, the problem has some special combinatorial characteristics. WVCP is a generalization of the vertex coloring problem but the presence of weights and the structure of the objective function poses new technical challenges. For example, WVCP is NP-hard even for graphs for which the chromatic number of a problem is easy, such as bipartite and chordal graphs.

In this chapter [1], we study WVCP in binary trees. First, we provide the related works on WVCP. Then we propose an algorithm with improved time complexity for balanced binary trees using the list coloring approach. In the entire chapter, we consider connected graph. However, the problem is still open for disconnected graphs. Although, we can optimally color the connected components of a disconnected graphs individually and merge the coloring to obtain a coloring of the whole graph, there always remains a possibility of finding different coloring of lesser weight by changing the color of a dominant vertex (a dominant vertex is a vertex with the maximum weight in a color class).

[1] This is joint work with Robert Benkoczi and Daya Ram Gaur, and appeared in [11], [20].

3.1 Preliminaries

3.1.1 Definition

Definition 3.1 (Weighted vertex coloring problem (WVCP) [38])**.** Given a vertex-weighted undirected graph G, find a feasible coloring X of G for which the weight of X is minimum.

When $w_v = 1$ for all $v \in V$, WVCP reduces to the proper vertex coloring problem (VCP). It follows that WVCP is strongly NP-hard [37].

Definition 3.2 (Acyclic Graph)**.** A graph that contains no cycles is defined as an acyclic graph.

A tree is an undirected graph that is both connected and acyclic. A tree T in which one vertex has been designated the root r is termed as a rooted tree. In a rooted tree, the parent of a vertex is the vertex connected to it on the path to the root and every vertex except the root has a unique parent. A child of a vertex v is a vertex of which v is the parent. A leaf vertex in a tree has degree 1. The height of a vertex v is the length of the longest path from v to a leaf.

Definition 3.3 (Binary Tree)**.** A binary tree is a tree in which each vertex has at most two children. These children are referred as the left child and the right child.

Definition 3.4 (Subtree)**.** A subtree of a tree T is a subgraph of T that is also a tree.

Definition 3.5 (Balanced Binary Tree)**.** For any vertex in a balanced binary tree, the height of its left subtree differs from the height of its right subtree by at most 1.

Tree Traversal

Tree traversal is the process of visiting each vertex in a tree data structure exactly once. There are three ways to traverse a tree:

1. Pre-order traversal: In this traversal method, the root is visited first, then the left subtree and finally the right subtree.

2. In-order traversal: In this traversal method, the left subtree is visited first, then the root and later the right subtree.

3. Post-order traversal: In this traversal method, the root is visited last. First, we traverse the left subtree, then the right subtree and finally the root vertex.

Definition 3.6 (Tree Decomposition [12])**.** A tree decomposition of tree T is a collection of subtrees $D(T)$ of T such that:

- $T \in D(T)$
- $\forall T_1, T_2 \in D(T)$, either T_1 and T_2 are disjoint or one is strictly contained in the other.

An element of $D(T)$ is called a component of the decomposition.

Definition 3.7 (Height of a decomposition [12])**.** The height of a decomposition $D(T)$ is the maximum cardinality of a subset of the component $C \subseteq D(T)$ whose elements strictly contain one another,

$$height(D(T)) = \max\{|C| : C = \{T_1, T_2, ..., T_k\}, T_1 \subset T_2 \subset ...T_k\}$$

Definition 3.8 (Tree Width)**.** The tree-width of a graph is defined as the minimum width of any tree decomposition.

Definition 3.9 (Partial k-trees)**.** A partial k-tree is a type of graph with tree width at most k.

Definition 3.10 (Polynomial-time approximation scheme (PTAS) [24])**.** Let Π be an NP-hard optimization problem with objective function f_Π. The algorithm A is an approximation scheme for Π if, on input (I, e), where I is an instance of Π and $e > 0$ is an error parameter, it outputs a solution s such that:

1. $f_\Pi(I, e) \leq (1 + \varepsilon) * OPT$ if Π is a minimization problem
2. $f_\Pi(I, e) \geq (1 - \varepsilon) * OPT$ if Π is a maximization problem

An approximation scheme A is called a polynomial time approximation scheme (PTAS) if, for each fixed $\varepsilon > 0$, its running time is bounded by a polynomial in the size of the instance I and exponential with respect to $\frac{1}{\varepsilon}$. A fully polynomial time approximation scheme (FPTAS) is a scheme where the running time of A is bounded polynomially in both the size of the instance I and by $\frac{1}{\varepsilon}$.

Definition 3.11 (Skinny Trees [51])**.** A tree in which the set of vertices of degree at least 3 forms an independent set.

Definition 3.12 (Hereditary Class)**.** A class $\mathcal{G}$ of graphs is hereditary, if for any $G \in \mathcal{G}$, every induced subgraph of G is also in $\mathcal{G}$.

Definition 3.13 (Planar Graph)**.** A graph is called planar if it can be drawn on the plane without any edge crossings.

Definition 3.14 (P_t Free Graph [42])**.** Graphs that do not contain paths of length $t-1$ as induced subgraph.

Definition 3.15 (Split Graph)**.** A split graph is a graph in which the vertices can be partitioned into a clique and an independent set.

3.1.2 Related Works

Guan and Zhu [38] ask the question whether it is possible to find an efficient algorithm for WVCP in a tree. They give a $O(n^{r+1})$ algorithm for a weighted graph with bounded tree width k and the number of colors r. There are a few partial answers including a PTAS in partial k-trees [68, 28] and a polynomial time algorithm in the path and skinny trees [51]. But there are no efficient algorithms for other classes of trees.

Pemmaraju et al. [68] show that WVCP in bipartite graphs is NP-hard. Additionally, they also give the $8/7$ approximation algorithm in bipartite graphs and a polynomial-time approximation scheme (PTAS) for trees (see Definition 3.10). The PTAS is based on the list coloring approach and the weight of a vertex depends upon a scaling function. The

feasibility of a coloring is checked by dynamic programming on trees which takes $O(n)$ time [68]. In the same paper, Pemmaraju et al. [68] also give a $4c$-approximation algorithm for a general graph where $c \in \mathbb{R}$. The algorithm orders the vertices in non-decreasing order of weights and performs a greedy coloring. The authors in [68] assume an existence of a c-approximation algorithm for coloring. Escoffier et al. [28] also gave a PTAS on partial k-trees and a constant factor approximation algorithm in hereditary classes (e.g. planar graphs, planar triangle free graphs, line graphs of bipartite graphs) of k-colorable graphs. These algorithms rely on enumeration and provide little intuition on approaches to design approximations for WVCP on other types of graphs.

For trees, the complexity of WVCP is still open, whereas, for path graphs, a polynomial time algorithm exists [51]. A path can be colored with at most three colors [51]. Hence, the polynomial time algorithm is as follows: fix the value of $w(\alpha_1) = \max_{v \in V}\{w_v\}$; find a lower bound on $w(\alpha_2)$; color all the vertices with weights greater than the lower bound of $w(\alpha_2)$ using color 1; find the minimum weight vertex between two successive vertices v_i and v_j colored with color 1 and color it with color 3. Next, sort the vertices colored 3 in non-decreasing order and iterate through all to find the smallest value for color 2. In summary, the algorithm divides a path into subpaths separated by vertices that are colored 3. The complete path is recreated by merging sub-paths one by one. At the time of merging the sub-paths, the value of 2^{nd} is updated. Finally, the minimum of $w(\alpha_1) + w(\alpha_2)$ is returned as an optimal solution. For skinny trees, two sets of weights are created: weights greater than the lower bound on color 2 (H), and the weights equal or smaller to the lower bound on color 2 (L). Again, iterating through all the values of L and transferring those weights into H gives various candidate solutions for $w(\alpha_2), w(\alpha_3)$ pairs. The optimal solution is the one with the minimum value of $\sum_{i=1}^{3} w(\alpha_i)$.

Recently, Araujo et al. [4] gave a $O(n^{\theta(\log n)})$ algorithm for trees based on enumerating the set of all weights of the colors on a feasible solution, where n is the number of vertices. This result shows that the problem of finding WVCP in trees cannot be reduced to other

NP-complete problem due to the presence of a sub-exponential algorithm [4].

de Werra et al. [24] proved that WVCP in planar graphs and P_8 free bipartite graphs are NP-complete. They also gave $O(n|w|\log n)$ time algorithm for P_5 free bipartite graph and PTAS for a split graph where $|w|$ is the number of distinct weights in the graph. In the same paper, they provide an $8/7$ approximation algorithm for WVCP in the bipartite graphs. Similarly, Hoang et al. [43] give an $O(n^3)$ time algorithm for $(P_5, \overline{P}_5)$ free graphs. Recently, Hsu and Chang [44] give another upper bound on the number of colors needed in an optimal coloring of WVCP. Their upper bound is based on the ratio of the weight of the heaviest vertex to the weight of the least heavy vertex.

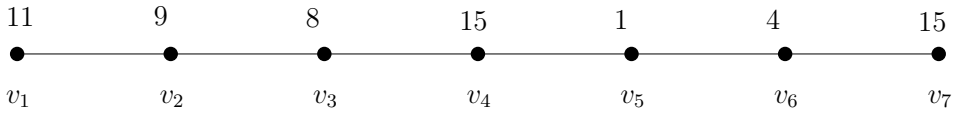


Figure 3.1: Two coloring is not optimal. Greedy is not optimal.

In the following sections, we restrict the maximum degree of a tree to three and propose an $O(n^2 \log n)$ time algorithm for WVCP in binary trees.

Overview of WVCP in Binary Trees

In binary trees, the maximum degree of a vertex is three, therefore at most four colors are needed in an optimal solution to the WVCP. This immediately gives an $O(n^4)$ algorithm for binary trees using the list coloring. Let $w_i = w(\alpha_i)$ represent the cost of color class i, we assume that $w_1 \geq w_2 \geq w_3 \geq w_4$. Certainly, fewer than four colors may be used in an optimal solution to the WVCP, in which case we set $w_4 = 0$, or even $w_3 = w_4 = 0$ as appropriate.

We enumerate all possible values for w_2, w_3 and w_4 for fix $w_1 = \max_{v \in V} w_v$. There are $O(n^3)$ such tuples. For each vertex $v \in V$, we assign a list of allowed colors based on the weight w_v of the vertex and the values w_i, as follows: $L(v) = \{i : w_v \leq w_i\}$. If the list coloring admits a feasible solution, then this solution is also a feasible WVCP coloring

with cost $w_1 + w_2 + w_3 + w_4$. Testing whether a list coloring problem is feasible in a tree graph can be done in $O(n)$ time [45]. Hence, this algorithm has $O(n^4)$ time complexity.

We can easily obtain an $O(n^3 \log n)$ algorithm for weighted binary trees if we enumerate values w_2 and w_3 and use binary search to find the smallest value of w_4 that admits a feasible list coloring.

Our approach is different. Rather than enumerating the values w_i with $1 \leq i \leq 4$ and test feasibility with a procedure for list coloring, we compute the set of values of w_i for which list coloring is feasible. We call this set of values the *feasible weight set*. In the next section, we characterize this set, and we show that its complexity is linear in the size of the tree.

3.2 Dynamic Programming

Before we proceed further, we describe a dynamic programming algorithm which we use throughout. The dynamic programming algorithm examines the previously solved subproblems and recursively combine their solutions to give the best solution (see Figure 3.2). For a given tree, the subproblems are defined over subtrees. Hence, this method is efficient for trees because the interaction between the connected subtree and rest of the tree occurs only through an edge.

One algorithm to solve an optimal VCP on a tree T rooted at v with a list of color $L(v)$ on each vertex, is to enumerate all $L(v)$ colors in each vertex and return the feasible coloring. However, the disadvantage of this approach is to compute the same feasible colorings at each subtree multiple times. We can reduce this repeated computation in each subtree by saving the feasible coloring at each subtree and use this information to color a parent. Moreover, we can apply this approach to weighted trees, where we compute the minimum weighted coloring at each subtree and use this information to compute the minimum weighted coloring of a parent vertex of the subtree. In the following section, we describe the process of computing feasible weight sets in each subtree.

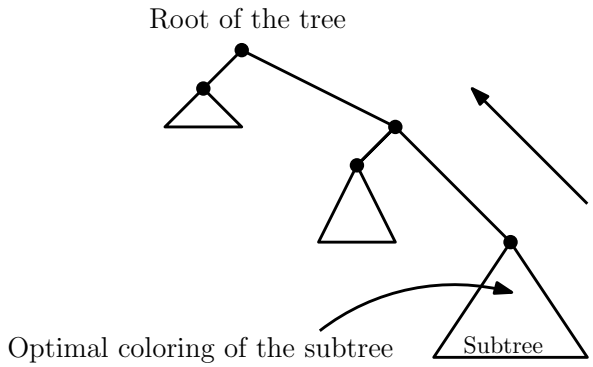


Figure 3.2: Dynamic Programming example

3.3 Feasible weight sets

Consider a vertex weighted balanced binary tree $T = (V, E, W)$ where $W : V \to \mathbb{N}$. Let $W(T)$ denote the set of weights. Let T_v be a subtree of T. We wish to represent the set of weight values $w_i : 1 \leq i \leq 4$ for subtree T_v, $\Delta \leq 3$ for which there is a feasible optimal coloring of T with color class cost w_i, and $w_1 \geq w_2 \geq w_3 \geq w_4$. Naturally, $w_1 = \max_{v \in V} w(v)$ is fixed. Additionally, we find a lower bound of w_2 (see Lemma 3.19) and fix a start value of w_2. For a fixed w_1, w_2, we are concerned with representing the set of values for w_3 and w_4. We represent $F(v, w_2)$ by points in the two dimensional space with coordinate axis w_3 and w_4 (see Figure 3.3). Let $F(v, w_2) = \{(w_3, w_4) \in \mathcal{W} : \mathcal{W} = W(T) \times W(T)$ and $\exists$ coloring with w_1, w_2, w_3, w_4 for $T_v\}$ be a feasible weight set for a fixed w_2. Because $w_3 \geq w_4$, the feasible set is contained in the upper triangular region. There are $O(n)$ possible choices for $w_3 \leq w_2$ and $w_4 \leq w_2$ and therefore, the size of $F(v, w_2)$ is $O(n^2)$. The important point here is that, although the size is $O(n^2)$ we can represent this set by a geometric construction with complexity $O(n)$.

We discuss few properties of $F(v, w_2)$ to help build our intuition. Fix a w_2. We first remark that the point $Q = (w_2, w_2) \in F(v, w_2)$. This is because there is a feasible coloring with $w_3 = w_4 = w_2$. In this case, there are two types of vertices in T: vertices v with $w_v > w_2$ which must be colored 1, and vertices with $w_v \leq w_2$ which can be colored with any color. If the choice of w_2 leads to a feasible solution, then the vertices colored 1 must

form an independent set. If we remove these vertices from T, we obtain a forest which can be colored with any two colors from the set of allowed colors $\{2,3,4\}$.

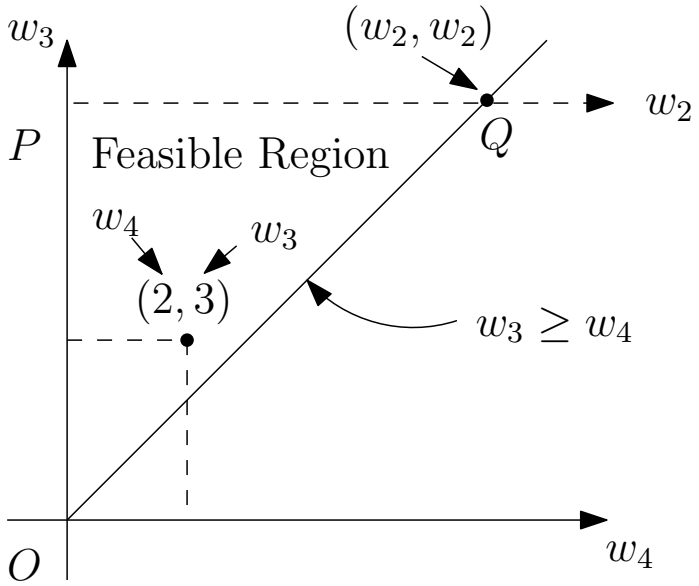


Figure 3.3: The region in w_3 - w_4 space enclosing the feasible weight set.

Another interesting point is the origin O in Figure 3.3. The origin is part of $F(v,w_2)$ if and only if there exists a feasible two coloring with color weights w_1 and w_2. Of course, this is not true for all values of w_2. Given the two observations above, $F(v,w_2)$ corresponds to a set of points inside $\triangle OPQ$ possibly separated from the origin by a polygonal line which we call the *boundary line of the feasible weight set* (Figure 3.6). We characterize this boundary line and claim that it has a complexity of $O(n)$. To do this, we first prove the following lemma.

Lemma 3.16. *Let $W(T)$ denote the set of weights. Let $A = (a,b)$ be a point that belongs to $F(v,w_2)$. Then any point $Z = (x,y)$ with $x \in W(T)$, $y \in W(T)$, $x \geq a$, and $y \geq b$ also belongs to $F(v,w_2)$.*

Proof. If the list coloring problem is feasible for point A ($w_3 = a$ and $w_4 = b$), then it must also be feasible for point Z since the list of allowed colors for the list coloring instance at point Z contains the lists of allowed colors from the instance at point A. □

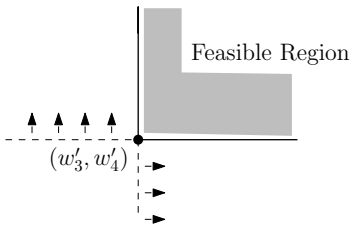


Figure 3.4: List Coloring region when a color is fixed

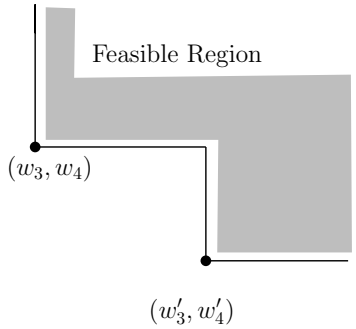


Figure 3.5: Change in the boundary line when the weight of one color is fixed and other is decreased

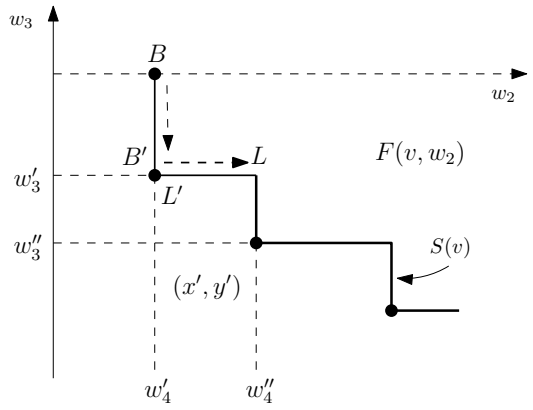


Figure 3.6: Boundary line for the feasible weight set.

Definition 3.17 ($x-y$ monotone polygonal line [71])**.** A polygonal line segment is a monotone, if there is a straight line L such that every line perpendicular to L intersects the line segment at most once. If L coincides with both x-axis and y-axis then the polygonal line segment is called $x-y$ monotone polygonal line.

For any fixed w_2, there is a smallest value of w_3 and a smallest value of $w_4(w_4')$. Consider a point B in Figure 3.6 with coordinate (w_3^B, w_4'). From Lemma 3.16, all points with $w_2 \geq w_3 \geq w_3^B$ and $w_4 = w_4'$ are feasible, so there is a point B' with minimum $w_3 = w_3'$ and $w_4 = w_4'$. Consider the points $A' = (x', y')$ such that the coloring is infeasible and $w_3 < x'$ and $w_4 < y'$. From Lemma 3.16, it follows that there is at most one such point for each vertical line with $w_4 \in W(T)$ and horizontal line with $w_3 \in W(T)$. Moreover, the feasible weight set is the union of the sets $Z(A') = \{(w_3, w_4) | w_3 \geq x', and, w_4 \geq y'\}$ (see Figures 3.4 -3.6). Hence the point (B', L') is the corner point of $x-y$ monotone polygonal line. For this reason, we also call the boundary line *the staircase line*. If the origin is part of the feasible

weight set, we consider a portion of the vertical line segment $[OP]$ to be on the boundary line (see Figure 3.3). The feasible weight set is the set of $O(n^2)$ points inside $\triangle OPQ$. We now claim the following corollary to Lemma 3.16 that defines our representation of the feasible weight set.

Corollary 3.18 (Feasible weight set)**.** *The feasible weight set for a fixed value of w_2 is uniquely determined by the boundary line (staircase) $S(v,w_2)$ (for simplicity $S(v)$) that starts at horizontal line $w_3 = w_2$ and ends at diagonal line $w_3 = w_4$.*

We first define a size as the number of corner points in a boundary line. A direct consequence of Corollary 3.18 is that the size of a boundary line is $O(n)$. This is because there are $O(n)$ vertices colored 3 and a boundary line contains at most one corner point for each w_3.

3.4 An Exact Algorithm for Balanced Binary Trees

3.4.1 The dynamic programming algorithm (DP)

We now describe our algorithm for solving WVCP on vertex weighted balanced binary trees. Consider binary tree T rooted at an arbitrary vertex r. For a vertex $v \in V$, let T_v represent the subtree rooted at v. Let $S_i(v)$ denote the set of corner points of the staircase line for the feasible weight set of tree T_v when v is colored with i. For a fixed w_2 we will compute the staircase line $S_i(v)$ of T_v for all $v \in V$ in a bottom up manner, under the restriction that vertex v receives color i for $1 \leq i \leq 4$.

Starting at the leaf vertices, we recursively compute the feasible weight sets represented by $S_i(v)$ until we obtain $S_i(r)$ for all $1 \leq i \leq 4$. We find the optimal solution to the WVCP by computing the cost $w_3 + w_4$ at each point in the feasible weight set $S_i(r)$ for all values of i and retain the point with the minimum value. This computation requires $O(n)$ time because the size of $S_i(r)$ is $O(n)$ and $1 \leq i \leq 4$. We then repeat the algorithm for the next value of w_2 and so on.

We can start our enumeration at the lowest value for w_2 and then we can increase w_2 until $w_1 = w_2$. The following Lemma proved in [51] provides a lower bound on w_2.

Lemma 3.19. *In any graph for every valid coloring, we must have* $w_2 \geq \max\{\min(w_u, w_v) : (u,v) \in E\}$.

Proof. By contradiction, suppose $w_2 < \max\{\min(w_u, w_v) : (u,v) \in E\}$. It means there is an edge $e \in E$ such that $w_v > w_2$ and $w_u > w_2$. As weights higher than w_2 are in color class 1, we have to color both u and v with color 1. This coloring violates the coloring constraint, therefore it is a contradiction. □

It is tempting to consider that this lower bound (is tight) for w_2 is the cost of color 2 in the optimal solution for WVCP. We describe an example that shows the lower bound weight of w_2 is not the optimal cost of color 2 in WVCP. Consider the path in Figure 3.1; in this example, we need at most three colors. If we use the 2 colors, then the weight of the coloring is $15 + 15 = 30$. We have $w_1 = 15$ and $w_2 \geq 9$. One option for 3-coloring is to take the minimum weight of w_2 and again determine the minimum weight of w_3 based on the fixed value of w_2 and w_1. This value for w_3 is the smallest value among the subset of vertices with weight smaller than w_2. In this example, when w_2 is 9 then all the vertices with weight greater than 9 are colored 1 so the cost is $15+9+8 = 32$. However, the optimal weight is $15 + 11 + 1 = 27$ with $w_1 = 15$, $w_2 = 11$ and $w_3 = 1$.

The pseudocode is described in Algorithm 2. In the following sections, we describe how the feasible weight set for leaf vertices is determined and how it is computed at internal vertices.

3.4.2 Feasible weight sets for leaf vertices

Let $v \in V$ be a leaf vertex. We have the following cases.

Case 1: $w_v > w_2$. Then v must be colored 1 and $S_i(v) = \emptyset$ for $i \geq 2$ and $S_1(v)$ corresponds to the points of the line segment $[OP]$ in Figure 3.7 (no restriction).

Algorithm 2 WVCP in a balanced Binary Tree

Require: Tree T, weight function w and root r
Ensure: A weighted coloring R of the vertices of T

Compute $w_1 = \max_v\{w_v\}$
Compute the lower bound on color 2 i.e w_2'
$R \leftarrow \emptyset$
for all $w_2 \in \{w_v : w_v \geq w_2'\}$ **do**
    for all $v \in V$ in post-order **do**
        if v is a leaf vertex **then**
            Compute the feasible weight sets $S_i(v)$ for $1 \leq i \leq 4$ using the base case.
        else
            Compute the feasible weight sets $S_i(v)$ for $1 \leq i \leq 4$ recursively given $S_i(x)$, $S_i(y)$ at the children x, y of v.
        end if
    end for
    Find the best solution R' in $S_i(r)$ for all values of i.
    Update the solution R if $w_2 + (\sum_i w_i : w_i \in R')$ is better.
end for
return $w_1 + (\sum_i w_i : w_i \in R)$

Case 2: $w_v \leq w_2$. Then $S_1(v)$ and $S_2(v)$ again correspond to the corner points of the line segment $[OP]$, no restriction. However, $S_3(v)$ corresponds to the corner points of staircase line in Figure 3.8 and $S_4(v)$ corresponds to the corner points of staircase line in Figure 3.9.

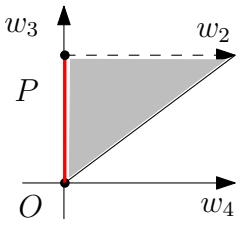


Figure 3.7: The staircase line is segment $[OP]$.

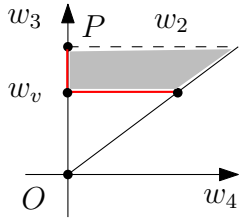


Figure 3.8: Base case for $S_3(v)$

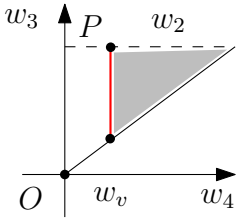


Figure 3.9: Base case for $S_4(v)$

3.4.3 Recursive computation of feasible weight sets

Let v be a subtree with children x and y. We now consider the computation of feasible weight sets represented by line $S_i(v) = S_j(x) \cap S_k(y)$ for all $j = 1,..,4, k = 1,..4$ available at the vertices x and y (refer to Section 3.4.4 for intersection operations). We have two cases depending on the value of parent vertex w_v.

Case 1: $w_v > w_2$. In this case vertex v must be colored 1 and $S_i(v) = \emptyset$ for all $i \geq 2$. $S_1(v)$ is computed as follows. We first combine the feasible weight sets $S_j(x)$ and $S_k(y)$ to obtain a temporary feasible set which we denote by $I_{jk}(v)$ (see Figure 3.11). We take the feasible weight points that are common to both $S_j(x)$ and $S_k(y)$ to obtain the feasible weight set at parent vertex v assuming colors j and k, both different from 1 at the children x and y. We perform the intersection operation because a feasible set of values for w_3 and w_4 is feasible for the combination of jk colors.

$$I_{jk}(v) = S_j(x) \cap S_k(y), j \neq 1, k \neq 1. \tag{3.1}$$

There are at most nine different feasible weight sets $I_{jk}(v)$ for all values of j and k. We are interested in the feasible sets with minimum cost, so we obtain the final feasible set $S_1(v)$ by computing the union of the nine staircase lines (see Figure 3.12). We can use the union operation because a feasible set of values for w_3 and w_4 is feasible for at least one combination of jk colors. The union operation explained in Section 3.4.4 is:

$$S_1(v) = \bigcup_{j \neq 1, k \neq 1} I_{jk}(v). \tag{3.2}$$

Case 2: $w_v \leq w_2$. This case is similar to Case 1, except that the feasible weight sets are additionally intersected by points in region greater than w_v based on color i. If $i = 3$ then the feasible weight sets of the children are intersected with points in the region $\{w_3 \geq w_v\}$ when computing the temporary feasible weight sets $I_{jk}(v)$. Similarly, if $i = 4$ then the feasible weight sets are intersected with points in the region $\{w_3 \geq w_v\}$ and $\{w_4 \geq w_v\}$

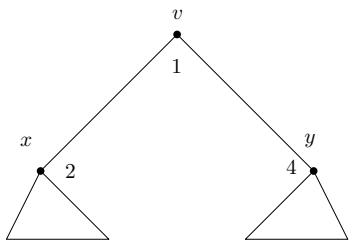


Figure 3.10: A subtree T_v and its children T_x and T_y

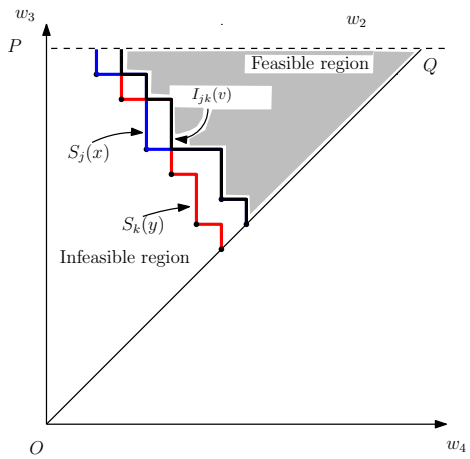


Figure 3.11: Combining the feasible weight sets $S_j(x)$ and $S_k(y)$.

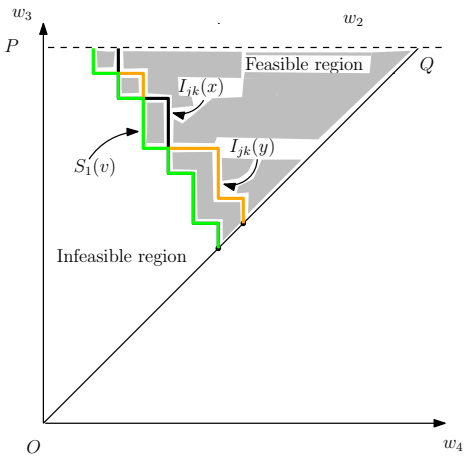


Figure 3.12: Computing $S_1(v)$ from the nine temporary staircase lines $I_{jk}(v)$; only two temporary staircase lines are depicted.

when calculating the temporary feasible weight sets $I_{jk}(v)$:

$$I_{jk}(v) = S_j(x) \cap S_k(y), j \neq 2, k \neq 2 \tag{3.3}$$

$$I_{jk}(v) = S_j(x) \cap S_k(y) \cap \text{points in the region} \{w_3 \geq w_v\}, j \neq 3, k \neq 3 \tag{3.4}$$

$$I_{jk}(v) = S_j(x) \cap S_k(y) \cap \text{points in the region} \{w_3 \geq w_v\} \cap$$
$$\text{points in the region} \{w_4 \geq w_v\}, j \neq 4, k \neq 4 \tag{3.5}$$

The process of calculating union for each $S_i(v) : i \geq 2$ is same as before.

3.4.4 Intersection and Union Operations

Let S_1 and S_2 are two staircase lines where S_1 has m and S_2 has l corner points. Let $S_1 = \{(w_3^i, w_4^i) : 1 \leq i \leq m\}$ and $S_2 = \{(w_3^j, w_4^j) : 1 \leq j \leq l\}$ are the corner points in S_1 and S_2 respectively. Similarly, let points $(w_2, w_4^{0_i})$ and $(w_2, w_4^{0_j})$ be the starting points of S_1 and S_2 respectively. Again, we suppose S_{12}^I be the corner points on a staircase line that represents the intersected region. To determine the corner points of S_{12}^I, we begin with the starting point $P = (w_2, \max\{w_4^{0_i}, w_4^{0_j}\})$, which is a point on the w_2 line (see Figure 3.13). After obtaining P, we begin a walk on the staircase S_1 or S_2 that contains P, as S_{12}^I. If both S_1 and S_2 contain P, S_{12}^I could be either S_1 or S_2. An outsider is a staircase line that doesn't include the point P. While calculating an intersection region, we always ignore the outsider because the boundary lines of each subtree are not feasible for both subtrees. Suppose S_2 contains P and we walk through the corner points of S_2 as S_{12}^I until the outsider intersects. Let S_1 be the outsider which contains a point $R = (w_3^i, w_4^i)$ and $Q = (w_3^j, w_4^j)$ be a point in S_2. Now, the conditions for the intersection of these two points are: $w_3^j \leq w_3^i$ and $w_4^j \geq w_4^i$. Hence, the intersection point is $U = (\min\{w_3^j, w_3^i\}, \max\{w_4^j, w_3^i\})$. If the intersection condition is satisfied then we reverse the role of the outsider and S_{12}^I. Thus, S_2 becomes the outsider and S_1 becomes S_{12}^I. We repeat this process until we reach the diagonal line ($w_3 = w_4$ line). Lastly, the point in the diagonal line which has the maximum value for w_4 is the end point of S_{12}^I. We provide the pseudocode for the intersection operation in Algorithm 3.

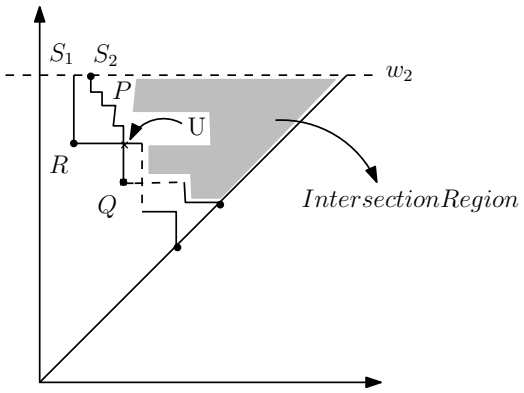


Figure 3.13: Calculate the intersection region of two staircase functions

Algorithm 3 Intersection between S_1 and S_2

Require: Staircases S_1, S_2 given by their corner points, l, m, $(w_2, w_4^{0_i})$ and $(w_2, w_4^{0_j})$
Ensure: Find the feasible weight set R that represent S_{12}^I, R contains corner points

Obtain the starting point $P = (w_2, \max\{w_4^{0_i}, w_4^{0_j}\})$ in S_1 and S_2
$R \leftarrow P$
if S_2 contains P **then**
    $S_{12}^I = S_2$
    S_1 is the outsider
else
    $S_{12}^I = S_1$
    S_2 is the outsider
end if
Consider next points (w_3^i, w_4^i) and (w_3^j, w_4^j) from S_1 and S_2 respectively
while $w_3^i \neq w_4^i$ and $w_3^j \neq w_4^j$ **do**
    if $w_3^j \leq w_3^i$ and $w_4^j \geq w_3^j$ **then**
        $R \leftarrow (\min\{w_3^j, w_3^i\}, \max\{w_4^j, w_3^i\})$
        if $S_{12}^I = S_1$ **then**
            $S_{12}^I = S_2$
            S_1 is outsider
        else
            $S_{12}^I = S_1$
            S_2 is outsider
        end if
    end if
end while
return R

Analysis: Each intersection points on staircase line can be charged to the vertex whose weight is responsible for the line segment in the staircase line. These vertices are obtained when we traverse the boundary of the feasible weight set from the horizontal w_2 line to the diagonal line only once. For each pair of corner points on the given staircase lines, if there is an intersect on vertical or horizontal with the boundary line then those intersection points are considered as corner points on the new staircase line. Algorithm 3 computes these new corner points in $O(n)$ time because we are traversing any one staircase line only once. Given the above discussion, we claim the following lemma.

Lemma 3.20. *An intersection operation between a pair of staircase lines given by their corner points requires $O(n)$ time.*

We use a similar approach to find the union of feasible weight sets. Suppose S_{12}^U is the staircase line representing the union operation. We consider all the points on the outsider to be in S_{12}^U and if intersection occurs between S_1 and S_2 we change the role of the outsider. The pseudocode for a union operation is in Algorithm 4.

3.4.5 Analysis

We know that the complexity of the boundary of the feasible weight sets is linear in the size of the tree they are defined on. From algorithms 3 and 4, we notice that the number of steps needed to compute union or intersection of feasible weight sets is proportional to the total complexity of the boundary lines of the feasible sets being merged. The lines are *x*-*y* monotone, and one can carefully traverse the two lines in the same direction and compute their union in an amortized constant time per point visited. As there are a constant number of feasible weight sets in a balanced binary tree at every vertex, we can obtain the union of all these feasible weight sets in $O(n)$ time. Given the above discussion, we claim the following lemma.

Lemma 3.21. *An union operation between a pair of staircase lines requires $O(n)$ time.*

Algorithm 4 Union between S_1 and S_2

Require: Staircases S_1, S_2 given by their corner points, l, m, $(w_2, w_4^{0_i})$ and $(w_2, w_4^{0_j})$
Ensure: Find the feasible weight set R that represent S_{12}^U, R contains corner points

Obtain the starting point $P = (w_2, \min\{w_4^{0_i}, w_4^{0_j}\})$ in S_1 and S_2
$R \leftarrow P$
if S_2 contains P **then**
    $S_{12}^U = S_2$
    S_2 is outsider
    $R \leftarrow S_2$
else
    $S_{12}^U = S_1$
    S_1 is outsider
    $R \leftarrow S_1$
end if
Consider next points (w_3^i, w_4^i) and (w_3^j, w_4^j) from S_1 and S_2 respectively
while $w_3^i \neq w_4^i$ and $w_3^j \neq w_4^j$ **do**
    if $w_3^j \leq w_3^i$ and $w_4^j \geq w_3^j$ **then**
        if S_1 is outsider **then**
            $S_{12}^U = S_1$
            $R \leftarrow S_1$
        else
            $S_{12}^U = S_2$
            $R \leftarrow S_2$
        end if
    end if
end while
return R

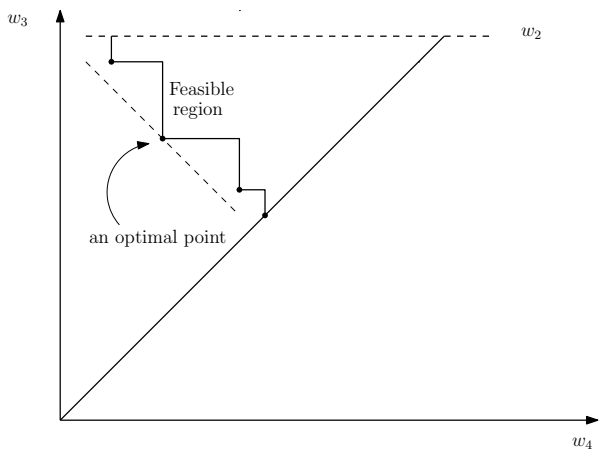


Figure 3.14: The optimal (w_3, w_4) pair returned by the boundary of $\mathcal{W}_2$.

Computing WVCP at the root r of a balanced binary tree: Starting at the leaf vertices, we recursively compute the feasible weight sets represented by $S_i(v)$ until we obtain $S_i(r)$ for all $1 \leq i \leq 4$. We find the optimal solution to the WVCP by computing the cost $w_3 + w_4$ at each corner point in the feasible weight set $S_i(r)$ for all values of i and retain the point with the minimum value (see Figure 3.14). This computation requires $O(n)$ time because the size of the boundary line is $O(n)$. We then repeat the algorithm for the next value of w_2 and so on. For each fixed w_2 we have the minimum for $w_3 + w_4$ and we consider the w_2 that gives us minimum value for $w_1 + w_2 + w_3 + w_4$. We can thus state the following theorem.

Theorem 3.22. *Algorithm 2 correctly computes the optimal solution to WVCP. The running time of algorithm 2 on balanced binary trees is $O(n^2 \log n)$.*

Proof. The correctness proof is standard. The intersection and the union operation take $O(n)$ time. There are $O(n)$ candidate values for w_2 and the height of a balanced binary tree is $O(\log n)$. For each fixed w_2, $O(n \log n)$ number of operations are required to calculate the minimum $w_3 + w_4$. Hence, the result follows immediately. □

3.5 Example

In this example, we illustrate algorithm 2. Let us consider the tree in Figure 3.15. Let d and e be the leaf vertices and a be the root vertex. The weight of color 1 is 4 and the lower

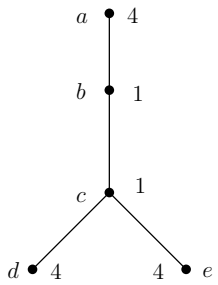


Figure 3.15: An example of a binary tree

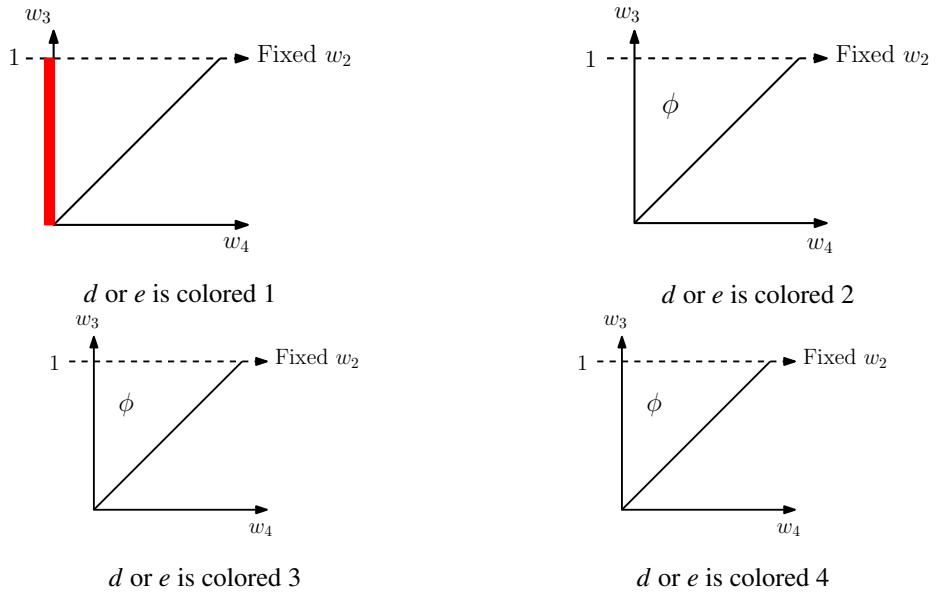


Figure 3.16: Base Case of vertices d and e

bound for the color 2 is 1. Hence, color 2 can take the weight 1 or 4. First, fix the weight of color 2 to 1. Since the weight of color 2 is fixed, the weight of color 3 and 4 can not be greater than the fixed weight of color 2.

The base case for vertices d and e is given in the Figure 3.16. The symbol ϕ in the figures is the empty staircase line. The staircase lines for the vertex d and e, when one is colored with color 2, 3 and 4 are ϕ because the weight of the vertex d or e is greater than w_2^0. So vertices d and e cannot be colored with color 2,3 and 4.

Figures 3.17, 3.18, 3.19 and 3.20 show the staircase functions when the leaf vertices takes different color combinations. Each figure is the staircase line obtained after the intersection of vertices d and e.

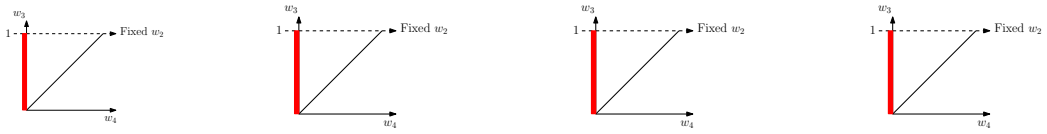


Figure 3.17: The staircase functions obtained by the combination of the fixed color 1 with other colors

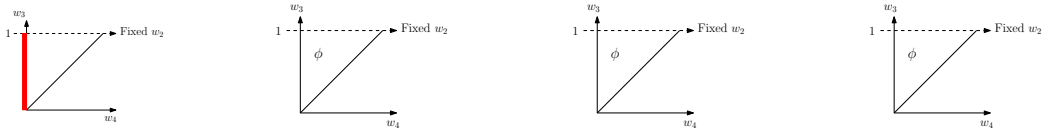


Figure 3.18: The staircase functions obtained by the combination of the fixed color 2 with other colors

Figure 3.19: The staircase functions obtained by the combination of the fixed color 3 with other colors

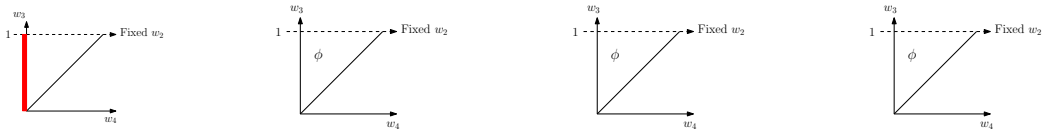
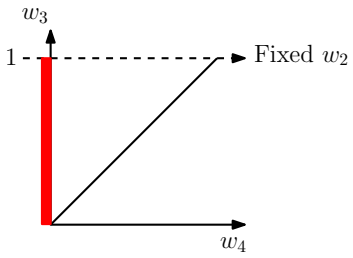


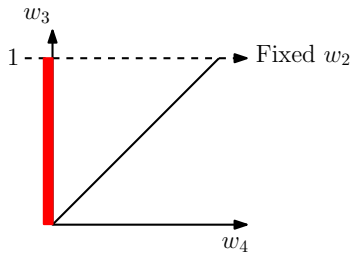
Figure 3.20: The staircase functions obtained by the combination of the fixed color 4 with other colors

w_v color 1 w_v color 2 w_v color 3 w_v color 4

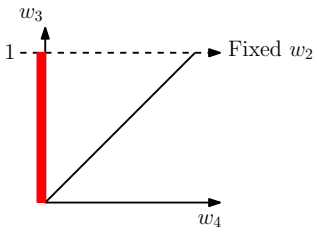
Figure 3.21: The union of staircase functions after the intersection when vertex c is fixed to color 1,2,3 and 4



Fixed color 1: child color 2



Fixed color 1: child color 3



Fixed color 1: child color 4

Figure 3.22: The staircase functions when the vertex a is fixed to color 1

Since the weight of c is equal to w_2^0, c can be colored with any color $1, 2, 3$ and 4. When vertex c is colored i, then the child cannot be colored with color i. Hence, for a fixed color i, all combinations except the combination containing i are present. For example, if c is colored 1 then the staircase for d and e are shown in Figures 3.17, 3.18, 3.19, 3.20 except the combination that has color 1, are present.

After obtaining all the 9 staircase functions for each color taken by vertex c, we perform the union operation to obtain the single staircase line for each fixed color. The staircase lines for each fixed color are shown in Figure 3.21. The staircase line for vertex b is same as the one for vertex c because c is the only children of b and the weight is equal.

The weight of the vertex a is greater than w_2^0. Hence, vertex a can only be colored 1. The staircase lines are given in the Figure 3.22. We take the union of these three staircase lines and the final staircase line is shown in Figure 3.23. The minimum values fpr $w_3 + w_4$

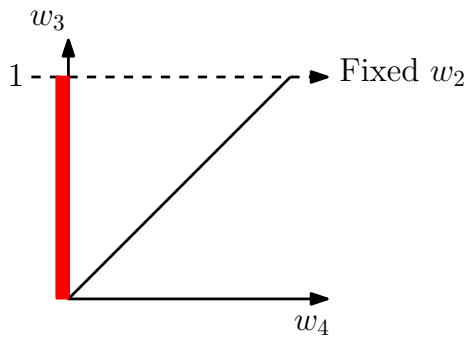


Figure 3.23: Final staircase function of a obtained by the union of all the staircase functions

for a fixed value of $w_2 = 1$ is shown in Figure 3.23 are 1 and 0. Therefore, the weight of the coloring for $w_2 = 1$ is $4+1+1+0 = 6$. We repeat the same process for $w_2 = 4$ and obtain the weight of the coloring.

3.6 An Efficient Algorithm for Arbitrary Binary Trees

The above analysis can be extended further to arbitrary binary trees if we use a special tree decomposition called the Spine Decomposition (SD) [12]. There are several tree decomposition mentioned in the literature and among them is a centroid decomposition [18].

3.6.1 Centroid Decomposition (CD)

Definition 3.23 (Centroid of a tree)**.** Let n be the total number of vertices in a given tree T. A centroid is a vertex v whose removal splits the given tree into a number of components T_i, where each of the component T_i contains no more than $n/2$ vertices.

The process to obtain a centroid of a tree begins by picking an arbitrary vertex as the root. Then, we use a depth first search strategy (DFS) to compute the size of each subtree. After finding the size of each subtree, we traverse from the root to the largest subtree until we reach a vertex v where no subtree has the size greater than $n/2$. This vertex v is the centroid of the tree. Further, we recursively decompose each of the new tree formed and attach their centroids as children to our root. Thus, a new centroid tree is formed from the given

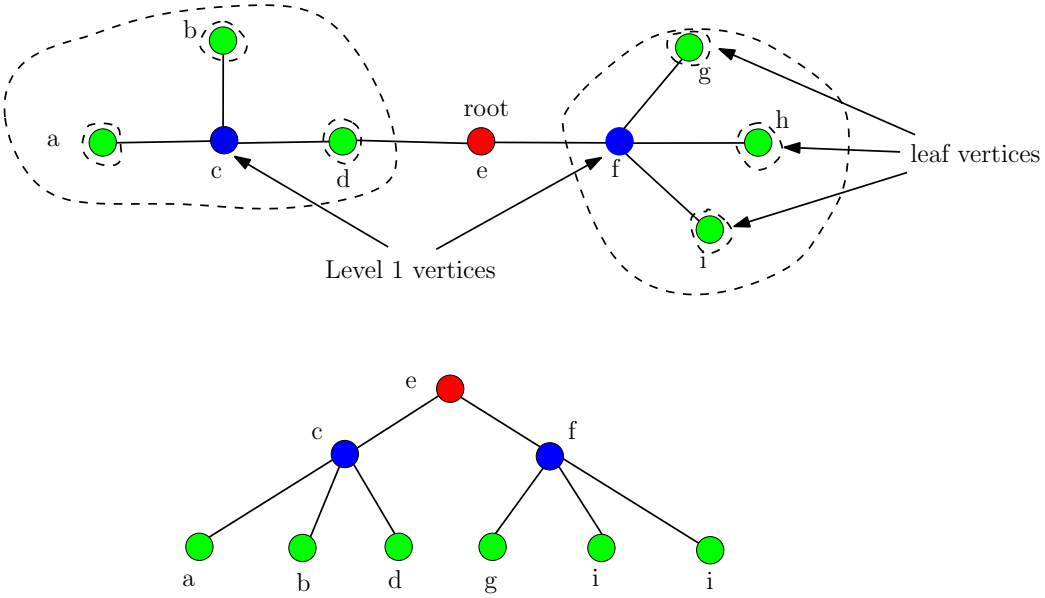


Figure 3.24: Centroid decomposition of a tree

tree T. Hence, the height of the centroid in the centroid decomposition is $O(\log n)$ [12]. Figure 3.24 shows an example of the centroid decomposition. Many tree based problems use centroid decomposition, e.g. nearest neighbor searching [8], p-center problem [62].

A major drawback of centroid decompositions is that centroids in successive subtrees are unrelated because we cannot control the representation of the path between two centroids. That means, we can infer no knowledge about the centroid of subtree T_i by knowing the centroid of subtree T_j if $T_i \subset T_j$. But in our WVCP we need control over such a path because of the interaction between coloring of vertices in a subtree and the outside world. In addition, we need to make sure that whatever we compute at some lower level in the decomposition remains valid at higher levels too. Hence, we use a spine decomposition instead of centroid decomposition. The spine decomposition is proposed by Benkoczi [12]. We use spine decomposition to obtain an algorithm with improved running time on binary trees.

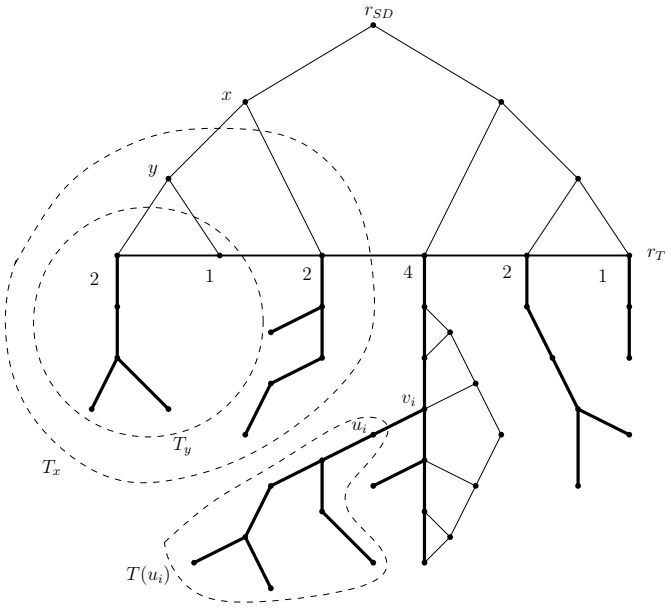


Figure 3.25: Spine Decomposition; the spine is represented by solid lines and the search tree is represented by thin lines [12].

3.6.2 Spine Decomposition (SD)

In the Spine Decomposition [12], we decompose the given rooted binary tree recursively into two or more subtrees called components. The decomposition is determined by a path computed in such a way that no component adjacent to this path is excessively large in size. This fact allows the decomposition to be balanced and this path is called the *spine*.

We build a *leaf vertex* in a balanced binary search tree on top of the spine. We construct the search tree in such a way that traversing the SD from the root of the search tree to an arbitrary tree vertex takes $O(\log n)$ steps regardless of the topology of the tree.

More precisely, consider a binary tree T rooted at a vertex r_T. We construct a path from r_T to a leaf such that the next vertex on this path has the most number of leaves associated with it. Furthermore, if $v_0 = r_T, v_1, v_2, ..., v_k$ are the vertices on the path, if $N_l(v)$ denotes the number of leaves hanging from v, then we have $N_l(v_{i+1}) \geq N_l(v_i)$. The path $v_0 = r_T, v_1, v_2, ..., v_k$ is the spine. Removal of the spine creates k disconnected components. Let $T(u_i)$ be one of those components rooted at u_i where u_i is adjacent to spine vertex v_i (in other words v_i is the parent spine vertex of u_i) and is not the spine vertex itself (see nodes

u_i and v_i in Figure 3.25). Let $\mu(v_i)$ be the number of vertices in $T(u_i)$. We create a binary search tree with vertices v_i as leaves, and we associate this tree with the spine. The root of the search tree is connected to the parent vertex of u_i. The search tree is constructed by considering the components with many leaves near to the root and is made balanced using the weight $\mu(v_i)$. Here, balance condition states that difference between the number of vertices between the two subtrees is at most 1. After creating the search trees for all the spines, we traverse the tree using only the search trees. The search tree traversal is as follows: start the traversal at root vertex r_{SD} of the balanced binary search tree then the search tree is traversed until leaf v_i on the spine is reached; the next search tree vertex is the root of the search tree constructed for $T(u_i)$. A path connecting any two search trees vertices is called a *super-path*. Let us denote the super-path by $\gamma(x,y)$ between the search tree vertices x and y. The properties of SD are given below by are from [12].

Theorem 3.24. *The length of any super-path $\gamma(x,y)$ in the SD is $O(\log n)$ where n is the number of vertices of the input tree T.*

Theorem 3.25. *The spine decomposition can be constructed in $O(n)$ time using $O(n)$ space.*

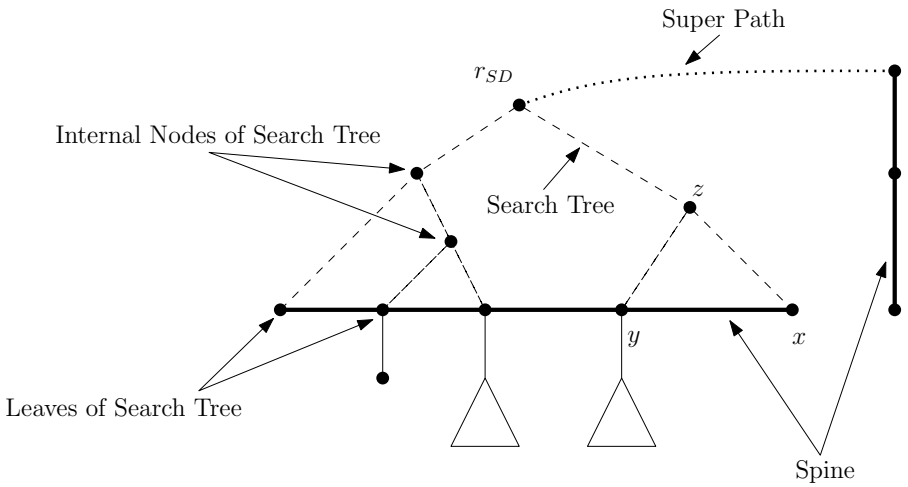


Figure 3.26: Spine Decomposition; x and y are leaves of the search tree

3.6.3 A WVCP algorithm for arbitrary binary trees - the general approach

In this section, we describe the dynamic program used to obtain the feasible weight sets on a binary tree using spine decomposition. The approach of the algorithm is virtually identical to the one for balanced binary trees. We compute feasible weight sets in a bottom-up fashion, and we find the optimal solution from the feasible weight sets at the root.

We provided pseudocode for solving the WVCP in a binary tree in Algorithm 5. In the following sections, we describe how the feasible weight sets for leaf vertices is determined and how it is recursively computed at internal nodes.

Algorithm 5 WVCP in a Binary Tree

Require: Tree T, weight w and root r
Ensure: An optimal weighted coloring R of the vertices of T

Compute $w_1 = \max_v\{w_v\}$
Compute the lower bound w_2' on color 2
Compute the spine decomposition of T
$R \leftarrow \emptyset$
for all $w_2 \in \{w_v : w_v \geq w_2'\}$ **do**
    for all $v \in V$ in post-order **do**
        if v is a leaf vertex **then**
            Compute the feasible weight sets $S_i(v)$ for $1 \leq i \leq 4$ using the base case.
        else
            Compute recursively the feasible weight sets $S_{ij}(v)$ for $1 \leq i, j \leq 4$ from two spine vertices x, y of v.
        end if
    end for
    Find the best solution R' in $S_{ij}(r)$ for all values of i, j.
    Update the solution R if $w_2 + (\sum_i w_i : w_i \in R')$ is better.
end for
return Return $w_1 + (\sum_i w_i : w_i \in R)$ is better.

3.6.4 Feasible Weight Sets

The feasible weight sets are associated with the internal vertices of the search trees of the SD (see Figure 3.26) where the vertex z is the internal vertex of the search tree and it is not the part of the original tree T). Additionally, these sets are determined by the tree vertices that are descendants of the internal search tree vertex (see Figure 3.26 where x and y are

the descendants of z and the feasible weight sets of z are obtained from the feasible weight sets of x and y). These subtrees have a particular structure, they attach to the tree T using at most two vertices and not one as in the case of balanced binary trees (see Figure 3.26). This means that the dynamic programming algorithm computes a feasible weight set, that corresponds to colors assigned, using at most *two* spine vertices (see Equation 3.8). This adjacency of spine vertices increases the number of feasible weight sets to at most 16 per node, which is larger but still constant. The recursive computation steps remain essentially the same as before except that there are more cases to consider.

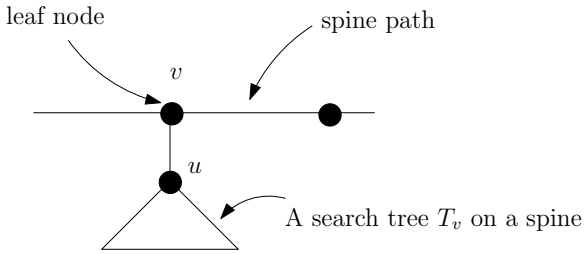


Figure 3.27: A leaf vertex in a decomposed tree

Base Case:

First, we describe the base case for computing the feasible weight sets at the leaf vertices (spine vertices) of balanced search tree. Let v be a leaf vertex and T_v be a search tree of a spine rooted at u (see Figure 3.27). As we can see in the Figure 3.27, u and v are connected with an edge. Suppose the feasible sets I_{jk} for u are recursively computed and given to us. We begin with fixing color $i : 1 \leq i \leq 4$ to v. Due to the coloring constraints, if we fix the color of v to i then we cannot color u with i. So, for each i on v, we first find the union of all feasible weight sets, i.e., $\bigcup_{j\neq i, k\neq i} I_{jk}$ and obtain feasible weights for v based on w_v. Let w_2 be the fixed value on color 2.

Case 1: $w_v > w_2$. If this is the case, v must be colored 1. Hence, $S_1(v) = \bigcup_{j\neq 1, k\neq 1} I_{jk}$ and $S_i(v) = \phi$ for all $i = 2, 3, 4$.

Case 2: $w_v \leq w_2$. If this the case, v can be colored with any of the four colors. Here,

$S_1(v) = \bigcup_{j\neq 1,k\neq 1} I_{jk}$ and $S_2(v) = \bigcup_{j\neq 2,k\neq 2} I_{jk}$. If $i = 3$ feasible weight sets are intersected with points in the region $\{w_3 \geq w_v\}$. Similarly, if $i = 4$, feasible weight sets are intersected by $w_3 \geq w_v$ and $w_4 \geq w_v$. In $i = 4$ we intersect feasible weight sets with the points in the region $\{w_3 \geq w_v\}$ because $w_3 \geq w_4$. Therefore,

$$S_3(v) = \{\bigcup_{j\neq 3,k\neq 3} I_{jk}\} \cap \text{points in the region} \{w_3 \geq w_v\}, \tag{3.6}$$

$$S_4(v) = \{\bigcup_{j\neq 4,k\neq 4} I_{jk}\} \cap \text{points in the region} \{w_3 \geq w_v\} \cap \text{points in the region} \{w_4 \geq w_v\} \tag{3.7}$$

Recursive Steps: In SD, leaf vertices are adjacent to each other (see Figure 3.28). As stated earlier, this adjacency increases the number of feasible weight sets to at most 16 per vertex in the binary search tree. We describe different cases needed to calculate the feasible weight sets at the internal vertices of the binary search tree in SD. The intersection and the union operations are same as in Section 3.4.4.

Case 1: As shown in Figure 3.28, we calculate the feasible weight sets of a parent vertex of the binary search tree that is connected to the leaf vertices x and y. We color v, 12 possible ways. Let $S_j(x) : j = 1,...,4$ represent the feasible weight sets for the leaf vertex x and $S_k(y) : k = 1,...,4$ represents the feasible weight sets for the leaf y.

$$S_{jk}(v) = S_j(x) \cap S_k(y), \forall j, \forall k, j \neq k \tag{3.8}$$

Case 2: This case is used to calculate the feasible weight sets of the parent vertex v whose immediate neighbor is a leaf vertex in SD and y is any intermediate vertex in the binary search tree (see Figure 3.29). Let $p = 1,...,4$ and $q = 1,...,4$:

$$S_{pq}(v) = \bigcup_{j\neq p} (S_p(x) \cap S_{jq}(y)) \tag{3.9}$$

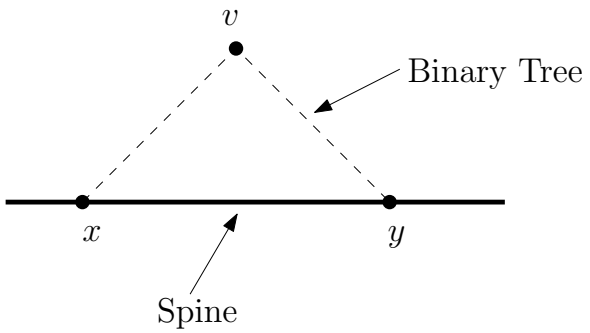


Figure 3.28: Vertex x and vertex y are the leaf the SD and v is the vertex of the binary search tree

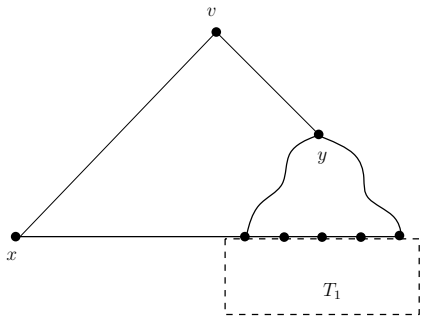


Figure 3.29: Vertex x is the leaf of SD and vertex y is any internal vertex in the binary search tree. T_1 represents the components hanging on the spine vertices

Case 3: We calculate the feasible weight sets of the root of the binary search tree when the feasible weight sets at the leaf vertices x and y are given (see Figure 3.30).

$$S_{pq}(v) = \bigcup_{j \neq k} (S_{pj}(x) \cap S_{kq}(y)) \tag{3.10}$$

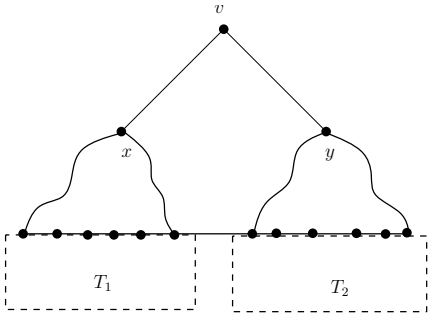


Figure 3.30: Vertex x and vertex y are internal vertices of the binary search tree and T_1 and T_2 are the components hanging of the spine vertices

Root r of binary trees: Starting from the leaf vertices of a spine, we recursively compute the feasible weight sets represented by $S_{ij}(v)$ on each internal node in the search tree for all $1 \leq i \leq 4$. Feasible weight sets at the root of the internal search tree represent the feasible weight set of a spine path. Hence, the feasible weight sets computed at the root of the internal search tree of one spine can be used to calculate the feasible weight set for another spine, using this super path. We recursively compute the feasible weight sets for all the spines, up to the root vertex of the search tree. We find the optimal solution to WVCP by computing the cost $w_3 + w_4$ at each corner point in the feasible weight set $S_{ij}(r)$ for all i and j values and retaining the point with the minimum value (see Figure 3.14). We then repeat the algorithm for the next value of w_2 and so on. For each fixed w_2 we have the minimum $w_3 + w_4$ and we know the minimum of $w_2 + w_3 + w_4$ and we choose the minimum of $w_2 + w_3 + w_4$ among all w_2.

Theorem 3.26. *The WVCP problem in binary trees can be solved optimally in time* $O(n^2 \log n)$.

Proof. We build a spine decomposition for a given tree in $O(n)$ time (from Theorem 3.25). We can perform the union and the intersection operations on the feasible weight sets in $O(n)$ time (see Section 3.4.4). The height of the decomposed tree is $O(\log n)$. Hence, for each w_2 we are performing $O(n \log n)$ operations to calculate feasible weight sets in the root of the search tree. Lastly, there are $O(n)$ choices for w_2. Therefore, the running time for WVCP in binary trees is $O(n^2 \log n)$. □

In the next chapter, we extend the concept of feasible weight sets for cactus graphs.

Chapter 4

Weighted Vertex Coloring in Cactus Paths

In this chapter [2], we extend our results on binary trees to cactus paths. Cactus graphs have many applications. For instance, we can model the combination of a bus network and a ring network as a cactus graph. These structures are frequently seen in Local Area Networks (LAN) [53]. Cactus graphs have also been studied in graph theory [41]. We begin with the definitions. Next, we exploit some properties of the cactus path to obtain the feasible weight sets.

4.1 Preliminaries

A cycle is a graph $C = (V,E)$ where $V = \{v_0, v_1, ...v_n\}$ and $E = \{(v_0,v_1),(v_1,v_2),(v_2,v_3),...,(v_{n-1},v_n),(v_n,v_0)\}$, the vertices v_i are all distinct. Each vertex $v \in V$ is associated with a positive weight w_v. If there are an even (odd) number of vertices, then we call the graph an even (odd) cycle (see Figure 4.1). We begin by giving the bound on the number of colors needed in any optimal solution to WVCP for cycles.

By using the result from Guan and Zhu [38], we can state that,

Lemma 4.1. *Every weighted even cycle graph C has an optimal WVCP using at most 3 colors and every weighted odd cycle graph C has an optimal WVCP using exactly 3 colors.*

Definition 4.2 (Cactus Graph)**.** A cactus graph $G = (V,E)$ is a connected graph in which any two cycles have at most one vertex in common. Likewise, every edge in a cactus graph

[2] This is joint work with Robert Benkoczi and Daya Ram Gaur, and appeared in [11].

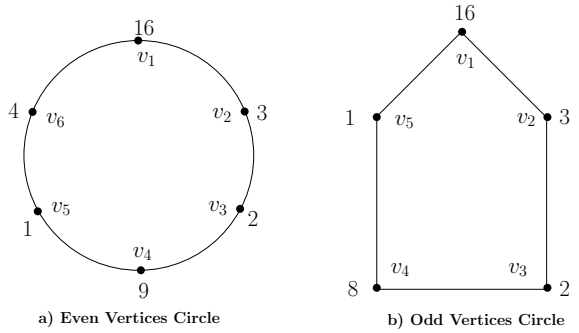


Figure 4.1: The odd and even vertices weighted cycle. The optimal WVCP on (a) is 16+4+2 = 22 and (b) is 16+3+2=21.

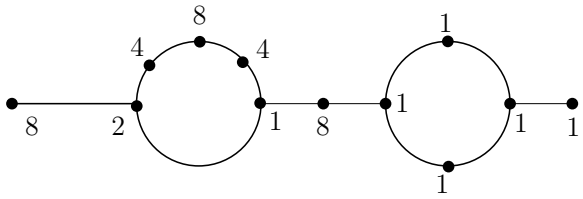


Figure 4.2: A 4-colorable cactus path where weight 8 receives color 1, weight 4 receives color 2, weight 2 receives color 3, and weight 1 receives color 4 and the optimal cost is 15

belongs to at most one cycle [23].

Definition 4.3 (Contraction of A Cycle)**.** A contraction of a cycle C in a cactus graph G is the replacement of C with a single vertex v such that the incident edges incident on C are now incident on v. The resulting graph has one cycle less.

Definition 4.4 (Cactus Path)**.** We call a cactus graph a cactus path, if contracting all of its cycles results in a path graph.

In this chapter, we consider *cactus paths* with the maximum degree three. Results of Guan and Zhu [38] imply that a cactus path with the maximum degree three may need at most 4 colors in any optimal coloring of WVCP. We consider Figure 4.2 to illustrates this above result. The cheapest three coloring has cost 16, whereas, the four coloring has cost 15.

4.2 Weighted Vertex Coloring Problem (WVCP) on Cactus Paths

We now establish several properties that characterize the optimal solution to the WVCP in cactus paths with maximum degree three. These features are essential in establishing the correctness of the proposed algorithm.

Lemma 4.5. *Let G be a graph with maximum degree Δ. If all of the maximum degree vertices of G have neighbors with degree $< \Delta$, then the optimal solution to WVCP uses at most Δ colors.*

Proof. Let $1,2,..,\Delta$ be the colors and let $w_1 \geq w_2 \geq ... \geq w_\Delta$ where w_i is the weight of the i^{th} color. Suppose we use color $\Delta+1$ and $w_\Delta \geq w_{\Delta+1}$. Consider any vertex u that is colored $\Delta+1$. If u has degree $\Delta-1$, then one color from set $\{1,2...,\Delta\}$ is not used by any of its neighbors. Thus, we can color u with this missing color without increasing the cost of coloring. We can apply this process repeatedly to re-color all the $\Delta-1$ degree vertices with colors from set $\{1,2...,\Delta\}$.

So, the only remaining vertices are of degree Δ which may be assigned color $\Delta+1$. Let v be such vertex (see Figure 4.3). All we need is that one color between 1 and Δ is not used by its neighbors. Let us assume that the neighbors of v are assigned all of the colors in the set $\{1,...\Delta\}$. Let u be the neighbor colored Δ. By the structure of G, we know that the degree of $u < \Delta$. One of the neighbors of u is v and there are at most $\Delta-2$ other neighbors of u; hence one of the colors from set $\{1,2..\Delta-1\}$ is missing from u's neighborhood. Thus, we can change the color of u from Δ to one of the missing colors from set $\{1,2,..\Delta-1\}$. The cost of the optimal solution doesn't increase after re-coloring u. We can repeat this process for all the neighbors of v that are colored with color Δ. Hence, Δ is eliminated from v's neighborhood, and v can be colored Δ. □

Corollary 4.6. *If the optimal solution to WVCP on a cactus path with the maximum degree three uses exactly four colors, then at least two vertices with the maximum degree are adjacent.*

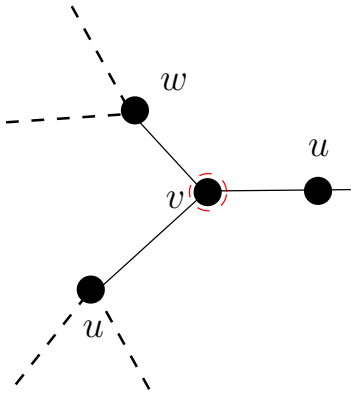


Figure 4.3: Coloring the graph with Δ number of colors

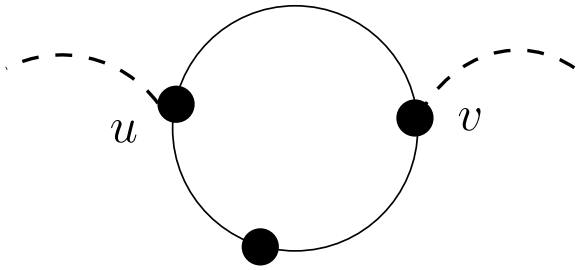


Figure 4.4: Two maximum degree vertices adjacent to each other

Proof. Suppose that no Δ degree vertices are independent. According to Lemma 4.5, this cactus path can be colored with 3 colors. Therefore, the optimal solution to WVCP on a cactus path with the maximum degree three uses exactly four colors if the maximum degree vertices are not independent. □

4.2.1 Outline of the algorithm on cactus paths

In this section, we give a brief description of the algorithm for cactus paths with maximum degree three. We begin with the definition of the boundary vertices and the component formed by the vertices of degree three in a cactus path.

Definition 4.7 (Boundary Vertex)**.** A boundary vertex is a vertex of degree greater three in a cactus path.

The removal of boundary vertices disconnects a cycle into components. This removal leaves components that are either paths or isolated vertices (see Figure 4.5). A component

is simply a path with the boundary vertices as the two end vertices and has maximum degree 2. When we take the union of components with its boundary vertex, we can get paths or at most two cycles. The approach used to calculate the feasible weight sets in a path can be applied to components. Furthermore, the union of the feasible weight sets of components gives the feasible weight sets for a cycle.

We begin by separating the cycles of a cactus path into two different components. We compute the feasible weight sets for each component using Algorithm described in Section 4.2.2 (see Figure 4.6). After obtaining the feasible weight sets for each component, we merge the feasible weight sets to get the feasible weight sets of the cycle (merging is calculating union and intersection operations described in Section 3.4.4 of Chapter 3. For simplicity, we associate a vertex with each cycle and path in the components of the cactus path (see Figure 4.5). Next, we build a balanced binary tree whose leaves are the nodes just created (see Figure 4.8). The internal nodes of the tree are associated with the feasible weight sets corresponding to the subgraph induced by the union of the components that are descendants of the internal tree node. As discussed in the previous sections, we consider at most three colors where w_i represents the cost of color i and $w_1 \geq w_2 \geq w_3$ for the case of a cactus path whose maximum degree vertices forms an independent set. We merge these feasible weight sets, in a bottom-up fashion to obtain the feasible weight sets of an internal node. At the root of the tree, we obtain the feasible weight sets for the entire cactus path. We recover the optimal (w_2, w_3) pair by processing this global weight set in the same way we did for the binary trees.

We extend the above process of computing feasible weight sets in a cactus path which requires at most four colors in the optimal coloring. Unlike the algorithm for trees, we fix the weight of the fourth color, and we represent feasible weight sets for each component in w_2- and w_3- axis in the 2D graph. We then consider a new value for the weight of the fourth color and compute an optimal $w_2 + w_3$ at the root of the search tree. It can be shown with amortized analysis that the total amount of work to update the feasible weight sets is

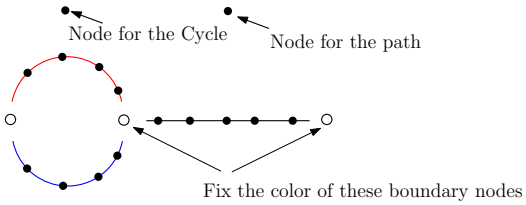


Figure 4.5: A node representing a cycle or a path

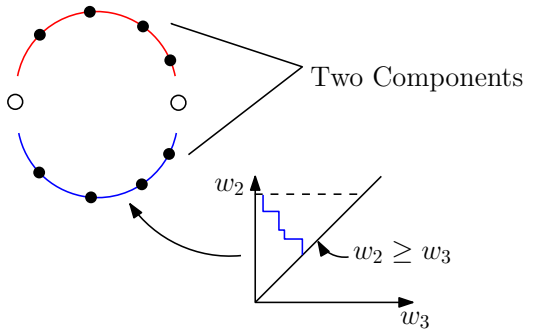


Figure 4.6: Two components of a cycle preprocessed with the algorithm [51]

$O(n \log n)$ if the cactus path is 3-colorable and $O(n^2 \log n)$ if the cactus path is 4-colorable (see Section 4.2.3). Before giving details, we describe an essential property of a cactus path.

Lemma 4.8. *Suppose* $w_{max} = \max_{v \in V}\{w_v\}$ *and* w_2 *be a lower bound on color class 2 in a cactus path. Suppose* $w_2 = w_{max}$. *If all the cycles in the cactus path are even cycles, then the two coloring is an optimal weighted coloring, and if any one of the cycles is odd cycle then a three coloring is the optimal weighted coloring.*

Proof. Let all the cycles be even. Let the number of colors be three and let $\{\alpha_1, \alpha_2, \alpha_3\}$ be an optimal coloring. Let $w_i = w(\alpha_i)$ and $w_1 \geq w_2, \geq w_3$. Suppose 3 colors are needed ina an optimal coloring. As we know from Corollary 4.6, $w_2 = w_{max}$ means there exist an edge $e \in E$ such that the adjacent nodes have the maximum weight. Let v and u be nodes of e. Among the 3 colors, let $v \in \alpha_1$ and $u \in \alpha_2$. Hence, the weight of the optimal 3-coloring is:

$$\begin{aligned} & w_{\alpha_1} + w_{\alpha_2} + w_{\alpha_3} \\ & = 2w_{max} + w_{\alpha_3}, \end{aligned} \tag{4.1}$$

An even cycle can be feasibly colored with two colors. Hence, if we use only two colors to color the cycle, the weight of the two coloring is:

$$w_{\alpha_1} + w_{\alpha_2} = 2w_{max} \tag{4.2}$$

The cost of coloring $\{\alpha_1, \alpha_2, \alpha_3\}$ is greater than the cost of the 2-coloring. Therefore, 2-coloring is the optimal coloring for a cactus path containing only even weighted cycles when $w_2 = w_{max}$.

Similarly, we examine the odd cycles. Since two of the adjacent vertices have w_{max} weights respectively, we can always color these two vertices by two colors. Hence, the weight of these two colors is $2w_{max}$. We can always select the smallest weight in each odd cycle as the weight of color 3 in each cycle. It can be shown that this 3-coloring is the optimal coloring for a cactus path containing odd cycles with $w_2^0 = w_{max}$, also 4-coloring has a smaller weight. □

In the following sections, we assume $w_2 < w_{max}$. We first describe a procedure to compute the feasible weight sets on paths using Kavitha *et al.* algorithm [51]. After calculating the feasible weight sets on a path, we use a similar process to calculate the feasible weight sets on the components obtained by removing the maximum degree vertices in the cactus path.

4.2.2 Brief overview of Kavitha *et al.* [51] algorithm on a path

The algorithm by Kavitha et al. [51] solves WVCP on a path graph. As the given graph is a path, at most three colors are needed in an optimal coloring. Let w_i be the weight of color i. The idea behind the algorithm is to enumerate the weight of 3 and obtain the minimum sum of $w_1 + w_2$ weights for each choice of color 3. We first fix the weight of color 1 to $w_{max} = \max_v\{w_v\}$, then we find the minimum weight for color 2 for each choice of w_3. The next question is, which vertices should be colored 3. We use the lower bound on color 2 (w_2) (see Lemma 3.19 of Chapter 3). All the vertices which have the weight

greater than w_2 are assigned color 1 and they form an independent set. Let us consider two successive vertices that are colored 1. If there is an odd number of vertices between two successive vertices that are colored 1 then we can color the intermediated vertices with only two colors. Hence, 2-coloring is the optimal coloring. However, if there are an even number of vertices between two successive color 1 vertices, then we need color 3. We choose vertices to color 3, as the minimum weight vertex between every successive pair of color 1 vertices. In summary, the algorithm divides a path into subpaths separated by color 3 vertices. The complete path is created by merging subpaths one by one. At the time of merging the subpaths, w_2 is updated along with w_3. Finally, minimum of $[w_1, w_2, w_3]$ is returned.

Let $\{z_1, ..., z_r\}$ be the set of color 3 vertices. These color 3 vertices decompose the path into the sequence $\rho_0, z_1, \rho_1, ..., \rho_{r-1}, z_r, \rho_r$, where each ρ_i is a sub-path of the original path, $z_v \in V$ is a color 3 vertex and $u_i \in \rho$ is a color 1 vertex. We provide pseudocode in Algorithm 6 for calculating feasible weight sets for a path. The following symbols are used in the description of the algorithm:

* α_i is the i^{th} color class
* w_2 is the lower bound on α_2
* w_2 is the weight of α_2
* w_3 is the weight of α_3
* u_i is the i^{th} color 1 vertex
* z_i is the i^{th} color 3 vertex
* ρ_i is the i^{th} sub-path of path P
* Ω is the set of odd-indexed vertices
* Σ is the set of even-indexed vertices

* $\omega_\rho = \max_{v \in \rho \cap \Omega} \{w(v)\}$ is the maximum weight among odd-indexed vertices in ρ
* $\sigma_\rho = \max_{v \in \rho \cap \Sigma} \{w(v)\}$ is the maximum weight among even-indexed vertices in ρ
* Q is a priority queue data structure
* R is a list data structure

Algorithm 6 Construct a feasible weight set for a path

Require: Path P, weight function w

Ensure: Feasible weight set of P

1: Calculate w_{max} such that $w(\alpha_1) = w_{max} = \max_{v \in V} w_v$
2: Find the lower bound on color 2 i.e w_2
3: Visit the path from left to right identifying color 1 nodes $u_0, ..., u_r$ s.t. $w(u_j) > w_2^0$
4: For each $i = 1, ..., r$ find a vertex z_i with minimum weight (in $u_{i-1}, ..., u_i$)
5: Decompose path into subpaths $\rho_0, z_1, \rho_1, ..., \rho_{r1}, z_r, \rho_r$ and compute ω_{ρ_i} and σ_{ρ_i} for each $i = 0, ..., r$.
6: $w_3^0 \leftarrow \max_{1 \le i \le r} w(z_i)$
7: $Q \leftarrow$ queue containing the vertices z_i in non-increasing order of $w(z_i)$
8: $R \leftarrow (w_2^0, w_3^0)$
9: **for all** $i = 1, ..., r$ **do**
10: $z \leftarrow$ dequeue first vertex from Q
11: let ρ' and ρ'' be the subpaths on each side of z in the current path decomposition
12: replace ρ', z, ρ'' with a new sub-path ρ in the path decomposition
13: compute ω_{ρ_i} and σ_{ρ_i} from $w(z), \sigma_{\rho_i'}, \omega_{\rho_i'}, \sigma_{\rho_i''}$ and $\omega_{\rho_i''}$
14: $w_2^i \leftarrow \max\{\min\{\sigma_\rho, \omega_\rho\}, w_2^{i-1}\}$
15: $w_3^i \leftarrow$ weight of the next z-vertex in Q
16: $R \leftarrow (w_2^i, w_3^i)$
17: **end for**
18: **return** R

4.2.3 Running Time

Theorem 4.9. *The running time of Algorithm 6 is $O(n \log n)$.*

Proof. The initial path decomposition and the first coloring $[w_{max}, w_2^0, w_3^0]$ can be computed in $O(n)$ time. We maintained this path decomposition in a linked list. We maintain a pointer to each color 3 vertex. We build a priority queue Q of each color 3 vertex which requires sorting. The sorting of color 3 vertices can be performed in $O(n \log n)$ using heap sort [32]. The merging of two adjacent subpaths requires $O(1)$ time. Each iteration removes one color 3 vertex from Q, so the running time to iterate all color 3 vertices is $O(r)$. We can update R by inserting a pair (w_2, w_3) in $O(1)$ time. Hence, our algorithm runs in $O(n \log n)$ time. □

4.2.4 A Case When Boundary Vertices Are Independent

In this section, we propose an algorithm for cactus paths when the boundary vertices of components are independent. Such cactus paths require at most three colors (from Lemma 4.5) in any optimal coloring.

Basis step

In this section, we use the Algorithm 6 with the restriction that the boundary vertices are colored with predetermined colors. Let x and y be two boundary vertices of a component(see Figure 4.7). We fix three color choices, for x and y and apply Algorithm 6 to a component to obtain the feasible weight sets. Moreover, there are at most 9 combinations to color x and y. Let us consider three different cases for x and y (see Figure 4.7).

Let w_2^0 be the lower bound on color 2.

Case 1: x is colored 1

First, we determine all the vertices $u \in V : w_u > w_2^0$ and color them 1. Let v be an adjacent vertex of x. If $w_v > w_2^0$, then v is colored with color 1. This coloring is infeasible. To remove this violation, we v as 2 and find a color 3 vertex between x and next vertex colored 1. While executing Step 13, we make sure that x is colored 1 and v is colored 2. We return

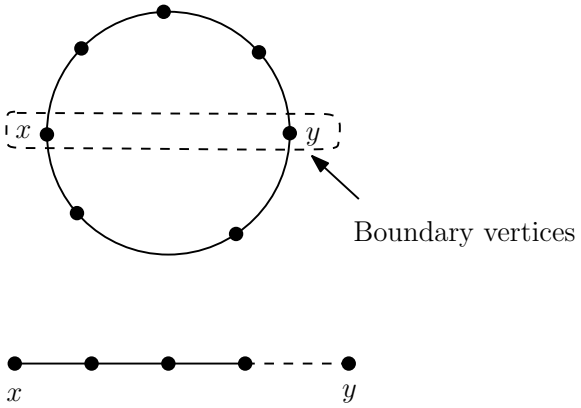


Figure 4.7: Boundary node are fixed to some colors

the minimum w_2 among odd-indexed or even-indexed vertices. However, if the adjacent vertex is not colored 1, we do not color v by 2. We simply find w_2 by executing Step 13 of this particular subpath by coloring x as 1.

Case 2: x is colored 2

If x is colored 2, the lower bound on color 2 should be $\max\{w_2^0, w_x\}$. We also make sure that the adjacent vertex of x is colored 1 while performing the 2-coloring of the subpath that contains x.

Case 3: x is colored 3

If x is colored 3, we make sure that the lower bound on color 2 is $\max\{w_2^0, w_x\}$. Furthermore, we compute w_2 for all choices of $w_3 < w_x$. The similar reasoning also applies for the vertex y when colored with color 1,2 and 3.

After obtaining the feasible weight sets of upper and a lower component of a cycle (see Figure 4.7), we intersect the feasible weight sets as described in Section 3.4.4 of Chapter 3 to obtain the feasible weight set of a cycle. We repeat this process for every cycle and path. Finally, we associate the feasible weight sets, of the corresponding components, with each leaf node in the balanced binary tree (see Figure 4.8).

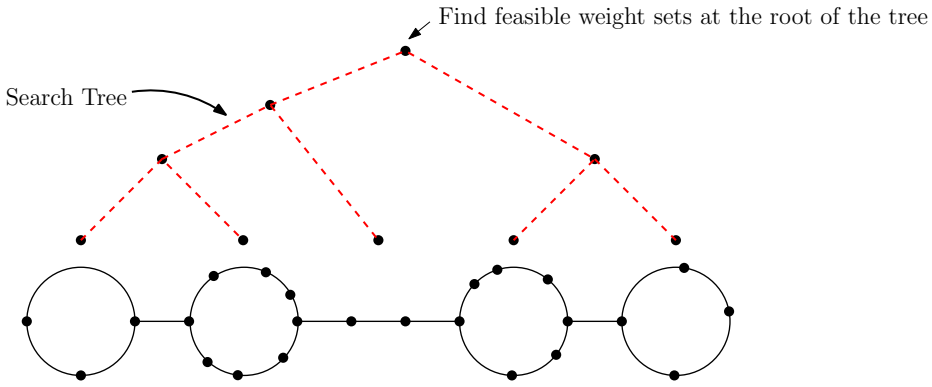


Figure 4.8: A search tree on the top of a Cactus Path

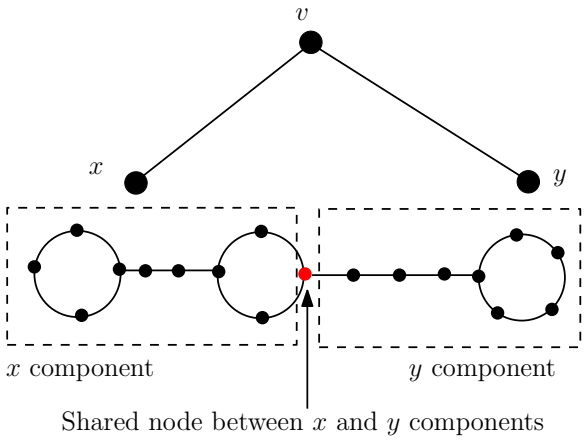


Figure 4.9: An internal nodes of the cactus path and it's associate components

Recursive step

Once we obtain the feasible weight sets for each leaf in the balanced binary tree, we proceed to obtain feasible weight sets of an internal node of the balanced binary tree (see Figure 4.9). We recursively compute the feasible weight sets of an internal vertex, of the balanced binary tree built on top of a cactus path, using the same intersection and union operations as described in Section 3.4.4 of Chapter 3. Let v be the root and x and y be the leaves of a subtree of the balanced binary tree (see Figure 4.9). Let the feasible weight sets of child vertices x and y be $S_{\alpha\phi}(x) : \alpha, \phi = \{1,2,3\}$ and $S_{\beta\phi}(y) : \beta = \{1,2,3\}$ respectively. The feasible weight set of v is calculated as:

$$S_{\alpha\beta}(v) = \bigcup_{\phi=\{1,2,3\}} (S_{\alpha\phi}(x) \cap S_{\phi\beta}(y)) \tag{4.3}$$

We use the recursive procedure to compute the feasible weight sets up to the root of the balanced binary tree. We recover the optimal (w_2, w_3) pair by processing each feasible weight set at the root of the tree in the same way we did for the binary trees. Pseudocode of an algorithm is given below:

Outline of the algorithm:

Algorithm 7 WVCP in a cactus path

1: Compute the components of the cactus path.
2: Compute the feasible weight set by traversing the balanced binary tree bottom-up.
3: Traverse the boundary of feasible weight sets to obtain the best solution. The best solution is the corner points having a minimum $w_2 + w_3$.
4: **Return** the best solution

Running Time:

Since the complexity of the boundary of the feasible weight sets is linear in the size of the corresponding cactus components, it can be shown that the entire computation takes $O(n \log n)$ time.

Theorem 4.10. *The WVCP problem in cactus paths, where all the maximum degree vertices are independent, can be solved exactly in time $O(n \log n)$.*

Proof. By Theorem 4.9, feasible weight sets of each component can be calculated in $O(n \log n)$. We note that the height of a balanced binary tree is $O(\log n)$ and there are a constant number of union and intersection operations at each internal node. However, each union/intersection operation takes time proportional to the size of the components covered by the internal node where the operations take place. Here, covered means the portion of cactus path represented by an internal node of the tree. Hence, the union/intersection operation takes $O(n)$ time since an internal node can cover at most $O(n)$ nodes and we perform these operations up to the root of the tree. The total time to compute an optimal solution to WVCP for this particular cactus path is $O(n \log n)$. □

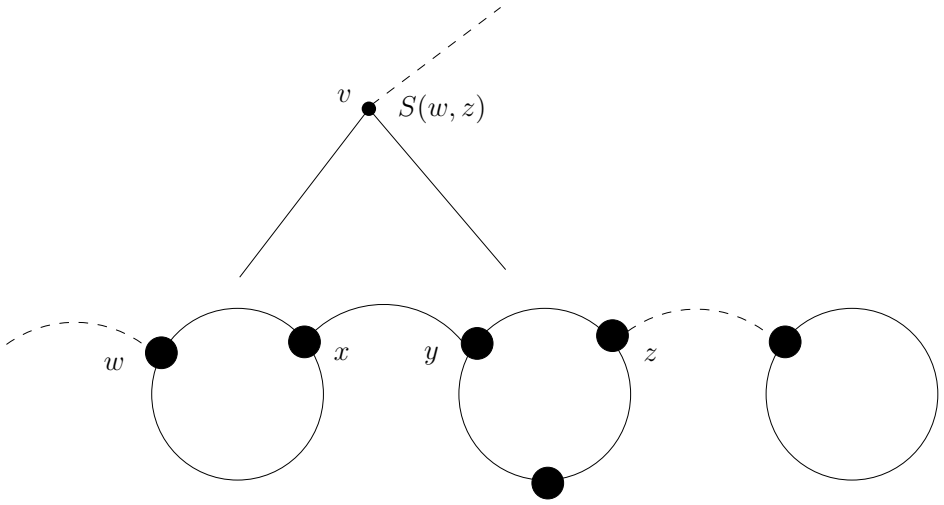


Figure 4.10: A Cactus Path with the maximum degree vertices adjacent to each other

4.2.5 Case when the maximum degree vertices are adjacent

In this section, we consider a cactus path where the maximum degree vertices are adjacent to each other (see Figure 4.10). This cactus path requires at most four colors in an optimal coloring. One way to find the optimal coloring on these cactus paths is to represent w_4 in $w_2 - w_3$ region. This representation is possible because all the other vertices other than the maximum degree vertices are of degree two. When there are vertices with degree two, optimal coloring can be obtained with at most three colors.

Observation 1: We can observe that the two neighbors of a color 3 vertex and a color 4 vertex are color 1 and color 2. Color 1 and color 2 vertices as neighbors of a $\Delta = 3$ degree 3 vertex forces color 3 to appear. This color 3 as well as color 1 and color 2 vertices as neighbors of another adjacent degree 3 vertex forces, color 4 to appear in the optimal solution (see Figure 4.2).

Lemma 4.11. *Let G be a cactus path. If an optimal solution uses four colors then from any pair of adjacent vertices of degree three, the vertex with the smallest weight receives color 4. If the weights are same, then any one of the vertices receives color 4.*

Proof. Consider vertices x and y such that $w_x > w_y$ (as shown in Figure 4.11). If $w_x = w_y$,

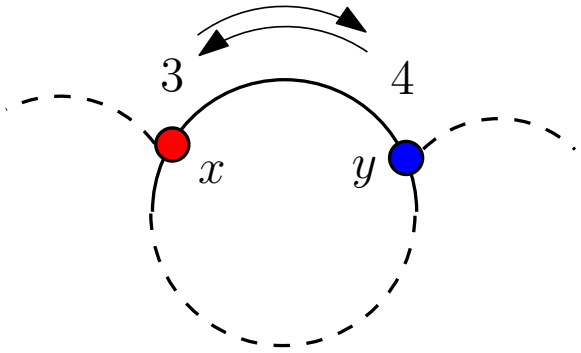


Figure 4.11: Interchange of colors

the weight of the optimal coloring remains the same for x and y. Let x be colored 4 and y be colored 3. By observation 1, the neighbors of x and y are of color 1 and 2. Since x and y have neighbors color 1 and 2, an interchange of colors between x and y does not violate the coloring constraint and also does not increase the cost of coloring. Hence, the lemma follows. □

We now discuss an algorithm with running time $O(n\log^2 n)$ and its limitations. We begin with a list of vertices to be colored 4. The vertices are introduced one by one to update a tree T obtained using Algorithm 7.

By Lemma 4.11, we can consider the minimum weight vertex among two maximum degree vertices of a cycle as a color 4 candidate vertex. Let x be the color 4 vertex, and the adjacent maximum degree vertex is the color 3 vertex. The idea is to color x 3, updating the existing feasible weight from leaf to the root node of the balanced binary tree by the introduction of the weight w_3. Although we color x 3, we record this weight as color 4 weight. It seems there is an infeasibility of coloring, but we let this coloring conflict happen because we are representing this weight in w_2- axis and w_3- axis 2D graph. This new coloring of x creates a new case where one of the boundary vertices are colored 3. We are adding this new combination as a feasible weight set in a leaf node of the balanced binary tree. The process of creating feasible weight sets for a component is the same as in Section 4.2.4. However, we can neglect color 4 vertex while running Algorithm 6 for finding the feasible weight set of a component.

Limitations: Let us again consider Figure 4.10. Let $S(w,x)$ be the feasible weight sets of a cycle with boundary vertices w and x, $S(y,z)$ be feasible weight sets of a cycle with boundary vertices y and z, $S_3(w,z)$ be feasible weight sets of v when three colors are used, and $S_4(w,z)$ be feasible weight sets of v when fourth color is used. The recursive computation of $S_3(w,z)$ is:

$$S_3(w,z) = \bigcup_{0 \leq i \leq 3, i \neq j} S_{ij}(w,x) \cap S_{ji}(y,z), 0 \leq j \leq 3 \tag{4.4}$$

Let x be colored 4. Then $S_4(w,z)$ is computed as:

$$S_4(w,z) = S_3(w,z) \cup \{S_{i4}(w,x) \cap S_{ji}(y,z)\}, \forall i, \forall j, i \neq j, j \neq 4 \tag{4.5}$$

By equation 4.5, an introduction of a color 4 vertex creates additional intersection and union computation on the existing feasible weight sets. Let us consider Figure 4.12. The first part of the figure shows the feasible weight set $S_3(w,x)$. However, after the introduction of a color 4 vertex, the intersection region is determined by $S_3(y,z)$. We can use binary search to compute the updated feasible weight set. That means we can obtain the updated feasible weight set in time $O(n \log n)$ using binary search. As in Figure 4.12, the complexity of computing an optimal point in a feasible weight set depends on the size of the feasible weight set of next child in the binary search tree. Hence, keeping all the previous feasible weight sets and applying binary search to obtain updated intersection and union points requires more space and time. Therefore, it is an open question to obtain the $O(n \log^2 n)$ time for solving WVCP in cactus paths.

To mitigate the above limitation of updating the existing feasible weight sets, we can create the feasible weight sets of the given tree from scratch for every w_4. We first find all the color 4 vertices. Then we create a balanced binary tree T. We obtain the minimum weight using three colors (where $w_4 = 0$) using Algorithm 7. We sort the color 4 vertices in non-decreasing order and introduce one by one. In this case, boundary vertices are

allowed to take color 4 hence, new combinations with color 4 are introduced. However, these combinations are constant in number. When we fix the weight of a vertex v to color 4, then all the vertices whose weight is lesser than w_4 may also be colored with color 4. Moreover, we can easily find feasible weight sets of a component with a boundary vertex having color 4. We can ignore this color 4 vertex and represent feasible weight sets in $w_2 - w_3$ region using Algorithm 6. We can ignore this weight because we have already considered the maximum color 4 weight in the solution. The pseudocode for WVCP in a cactus path is in Algorithm 8:

Algorithm 8 WVCP in a cactus path

1: Determine L i.e. the vertices that are candidates for the fourth color and sort them in non-decreasing order
2: Create a balanced binary tree T
3: Use Algorithm 7 on T
4: $R \leftarrow$ traverse the boundary of the feasible weight sets at the root of T and return the best solution 4.11.
5: **for all** $w_4 \in (\{0\} \cup L)$ **do**
6: Create feasible weight sets on T
7: Traverse the boundary of the feasible weight sets to obtain the best solution for w_4
8: Update $R \leftarrow$ best solution of w_4.
9: **end for**
10: **Return** the best solution

Running Time

We know that the intersection or union operation of the feasible weight sets takes $O(n)$ time. These operations are performed a constant number of times since the coloring combinations are constant in numbers. As there are $O(n)$ color 4 vertices and each update of w_4 is performed up to the root of the balanced binary tree which has the height of $O(\log n)$, the

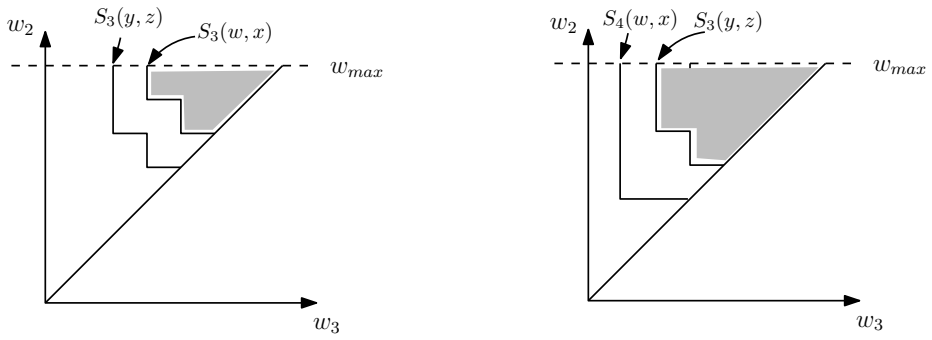


Figure 4.12: Feasible region before and after the introduction of w_4.

optimal WVCP in cactus path takes $O(n^2 \log n)$ time. Given the above discussion we can state the following theorem:

Theorem 4.12. *The WVCP problem in arbitrary cactus paths with maximum degree three can be solved exactly in time $O(n^2 \log n)$.*

4.3 An Algorithm for Cycles

To the best of our knowledge, this is the first study of WVCP on odd cycles. By Lemma 4.1, we need at most three colors to optimally color a weighted cycle. Moreover, a cycle resembles a path when we remove an edge. Thus, we can use Kavitha's et al. [51] algorithm (see Section 4.2.2) on weighted cycles. We modify Kavitha's algorithm slightly and propose an algorithm for a weighted cycle. The only modification to the algorithm [51] is that we terminate Algorithm [51] before it processes the last vertex colored 3 in an odd cycle because odd cycles require exactly three colors (see Corollary 4.1). Let $w_{max} = \max_{v \in V}$ be the weight of color class 1. We represent a coloring by $[w_{max}, w_2, w_3]$.

Let $[w_{max}, w_2^i, w_3^i]$ be a candidate coloring. We first calculate an initial coloring $[w_{max}, w_2^0, w_3^0]$. The lower bound on color 2, (w_2^0) is calculated as in Lemma 3.19 of Chapter 3. Similarly, w_3^0 is calculated as in Section 4.2.2. Our objective is to find the minimum of $w_2^i + w_3^i$. As in the path case, we decompose a cycle into subpaths ρ. Let $u_i : w(u) > w_2^0$ are vertices colored 1 and z_i are vertices colored 3. Additional symbols are defined in

Section 4.2.2. Pseudocode follows:

Algorithm 9 Algorithms for cycles

Require: Cycle C, weight function w

Ensure: An optimal weighted coloring on cycles

1: Calculate $w_{max} = \max_{v \in V} w_v$

2: Find the lower bound on color 2 $w_2^0 \geq \max\{\min(w_u, w_v) : (u,v) \in E\}$

3: For each $i = 1, ..., r$ find a vertex z_i with minimum weight (in $u_{i-1}, ..., u_i$)

4: Decompose a cycle into different subpaths $\rho_0, z_1, \rho_1, ..., \rho_{r1}, z_r, \rho_r$ by taking a reference vertex as a starting vertex, and compute ω_{ρ_i} and σ_{ρ_i} for each $i = 0, ..., r$.

5: $w_3^0 \leftarrow \max_{1 \leq i \leq r} w(z_i)$

6: $Q \leftarrow$ queue containing the vertices z_i in non-increasing order of $w(z_i)$

7: **for all** $i = 1, ..., r$ **do**

8: **if** Cycle is Odd **then**

9: End the loop when $i = r$.

10: **end if**

11: $z \leftarrow$ dequeue the first vertex from Q

12: let ρ' and ρ'' be the subpaths on each side of z in the current path decomposition

13: replace ρ', z, ρ'' with a new sub-path ρ in the path decomposition

14: compute ω_{ρ_i} and σ_{ρ_i} from $w(z), \sigma_{\rho_i'}, \omega_{\rho_i'}, \sigma_{\rho_i''}$ and $\omega_{\rho_i''}$

15: $w_2^i \leftarrow \max\{\min\{\sigma_\rho, \omega_\rho\}, w_2^{i-1}\}$

16: $w_3^i \leftarrow$ weight of the next z-vertex in Q

17: **end for**

18: **return** best coloring among the candidates $[w_{max}, w_2^i, w_3^i]$ for $i = 0, ..., r$

4.3.1 Correctness

Although the proof of the correctness of the algorithm in [51] are for paths, we show that it still holds true for cycles.

Lemma 4.13. *Every candidate triplet $[w_{max}, w_2^i, w_3^i]$ has a corresponding feasible coloring.*

Proof. We will prove this using induction.

1. If $i = 0$, the initial coloring is $[w_{max}, w_2^0, w_3^0]$. The initial coloring $[w_{max}, w_2^0, w_3^0]$ is a feasible coloring because we can always color all z vertices with color 3 and u vertices with color 1. The remaining vertices have weight at most w_2^0, so coloring these vertices with color 1 and color 2 gives a feasible coloring.

2. Suppose $i > 0$. The coloring induced by the path decomposition after the $i-1^{st}$ iteration is $[w_{max}, w_2^{i-1}, w_3^{i-1}]$ and is a feasible coloring. Let i^{th} iteration dequeues vertex z from the queue Q which has the weight w_3^{i-1}. The coloring at the end of i^{th} iteration differs from $i-1^{th}$ iteration by the nodes in ρ', z, ρ'', which are merged into a new sub-path ρ. Thus the weight of color 2 is the maximum of w_2^{i-1} and the weight of color 2 in an optimal 2-coloring of ρ, which is $\min\{\sigma_\rho, \omega_\rho\}$. This is also true of odd cycles because we did not dequeue the last color 3 vertex. Hence, we have a feasible coloring for $[w_{max}, w_2^i, w_3^i]$.

It follows by induction that every candidate triplet $[w_{max}, w_2^i, w_3^i]$ has a corresponding feasible coloring □

Lemma 4.14. *Let $i < r$ be such that $w_2^i < w_2^{i+1}$. For every feasible coloring $[w_{max}, w_2, w_3]$ if $w_2 < w_2^{i+1}$ then $w_3 \geq w_3^i$.*

Refer [51] for the proof of the above lemma.

Theorem 4.15. *Algorithm 9 outputs an optimal weighted vertex coloring in cycles.*

Proof. By Lemma 4.13, the coloring is a feasible coloring. To argue that the coloring is optimal, we show that there is a candidate $[w_{max}, w_2^i, w_3^i]$ at least as good for any feasible coloring $[w_{max}, w_2, w_3]$. If the optimal coloring of an even cycle is 2-coloring i.e. $w_2 \geq w_2^r$ then our algorithm consider the candidate $[w_{max}, w_2^r, w_3^r = 0]$ (at the last iteration the queue is

empty). This candidate output is as good as $[w_{max}, w_2^i, w_3^i]$. Otherwise, there exists $0 \leq i < r$ such that $w_2^i \leq w_2 \leq w_2^{i+1}$ (we know $w_2 \geq w_2^0$). Finally, from Lemma 4.14, if $w_2 < w_2^{i+1}$ then $w_3 \geq w_3^i$. Since algorithm outputs candidate solution $[w_{max}, w_2^i, w_3^i]$, the output is at least as good as $[w_{max}, w_2, w_3]$. Hence, our algorithm outputs an optimal solution. □

4.3.2 Running Time

The initial path decomposition and the first coloring $[w_{max}, w_2^0, w_3^0]$ can be computed in $O(n)$ time. We maintained this path decomposition in a circular linked list. We maintain a pointer for each color 3 vertex. We build a queue Q for each color 3 vertex which requires sorting. We can update the path decomposition of merging two adjacent subpaths in $O(1)$ time. Each iteration removes one color 3 vertex from Q, so the running time to iterate all color 3 vertices is $O(r)$. Hence, our algorithm runs in $O(n)$ time except the time to sort vertices. The sorting of color 3 vertices can be performed in $O(n \log n)$ using heap sort [32]. Given the above discussion, we can state the following theorem.

Theorem 4.16. *The WVCP problem in cycles can be solved exactly in time $O(n \log n)$.*

In the next chapter, we study the weighted vertex coloring problem when weights are uncertain.

Chapter 5

Robust Weighted Vertex Coloring

In this chapter [3], we explain the min-max regret solution to the weighted vertex coloring problem (WVCP). We call this problem the robust weighted vertex coloring problem (RWVCP).

5.1 Robust Optimization

Combinatorial optimization problems with uncertainty in the input parameters have been studied by an increasing number of researchers because of their importance in practice. In classical optimization problems, it is assumed that the parameters are known in advance. In reality, it is very difficult to obtain accurate data because of measurement or estimation errors. Even a small uncertainty in the data may make the nominal optimal solution completely meaningless [9]. Two ways of modeling uncertainty are currently popular: the stochastic approach and the worst case analysis approach (or robust approach). The goal of these algorithms is to provide algorithms for minimizing the cost in the presence of uncertainty. One approach provides a solution that has the smallest cost in expectation, the other minimizes the adverse effects, i.e. the difference between the cost of the solution returned and the optimal cost given a realization of the scenario minimized over all possible scenarios.

In the stochastic approach [13], it is assumed that we have information about the underlying probability distribution, and the objective is to find the solution with good expected

[3] This is joint work with Robert Benkoczi and Daya Gaur, and appeared in [10].

value. Various techniques are reviewed in [46] to handle this type of problem.

In the robust or the worst case approach, the uncertain parameters are from an interval with an unknown probability distribution. For these parameters, the values lie between known lower and upper bounds. Any particular assignment of values to the uncertain parameters in the allowed range is called a scenario. The goal is to find a "robust" solution". That is a solution X for which the difference between the cost of a solution X under the worst possible scenario and the cost of the optimal solution for that scenario is minimized. Computing the optimal cost of a scenario requires solving the deterministic version which can be difficult.

5.1.1 Min-max Regret

Min-max regret optimization deals with the uncertainty in the objective function at the time of solving the problem. The aim of the min-max regret is to find a feasible solution that would minimize the worst-case loss. This loss may occur in the objective function value because the solution is chosen before the actual realization of the objective function is known. In other words, min-max regret aims at constructing solutions having the best possible performance in the worst case [74].

5.1.2 Problem definition

Let $G = (V, E, w)$ be a weighted graph where $w : V \rightarrow \mathbb{N}$ be the weight function on the vertices. Weight of each vertex v, lies in the interval $[\underline{w}_v, \overline{w}_v]$. Let Γ be the Cartesian product of the uncertainty intervals $[\underline{w}_v, \overline{w}_v], v \in V$. An element $s \in \Gamma$ is called a *scenario*, where $w_v^s \geq 0$ is the weight of the vertex v under the scenario s. Let Φ be the set of all the feasible colorings for G. If $X \in \Phi$, then X is the partition of V into color classes $X = \{\alpha_1, \alpha_2, ..., \alpha_k\}$ such that α_i is an independent set (color class) in G. For any color class α, let $w^s(\alpha) = max\{w_v^s | v \in \alpha\}$. For a scenario s and a coloring X, the cost of the weighted coloring is:

$$F(s,X) = \sum_{\alpha \in X} \max_{v \in \alpha} \{w_v^s\} \tag{5.1}$$

Let $F^*(s)$ be the cost of the optimal solution under scenario $s \in \Gamma$, defined as:

$$F^*(s) = \min_{X \in \Phi} F(s,X) \tag{5.2}$$

Regret $R(s,X)$ is defined as the difference between the weight of the coloring X under scenario s and the optimal solution under scenario s:

$$R(s,X) = F(s,X) - F^*(s) \tag{5.3}$$

The maximum regret of a feasible coloring $X \in \Phi$ is given by:

$$Z(X) = \max_{s \in \Gamma} \{F(s,X) - F^*(s)\} \tag{5.4}$$

For given X, any scenario s which maximizes the right hand side of (5.4) is called the worst case scenario for X. Our objective is to find the coloring X among all the colorings in Φ which minimizes $Z(X)$,

$$\min_{X \in \Phi} Z(X) \tag{5.5}$$

5.1.3 Related Works

Kasperski et al. [49] in 2006 gave a 2-approximation algorithm for min-max regret optimization problems for problems with specific objective functions. Their approach approach can be applied to many interval versions of the basic combinatorial optimization problems, such as the minimum spanning tree problem [6] [5], the shortest path problem [66] and the assignment problem [48]. Their approach is based on the following fact: given a coloring X and elements e, *the worst case scenario* for X is the one where $e \in X$ have costs $\overline{w}_e$ and all the other elements have costs $\underline{w}_e$. Now, it is easy to determine a lower bound on the

regret. The approximation algorithm always considers the solution to the midpoint interval scenario. Additionally, Kasperski et al. [50] also gave a FPTAS (see Definition 3.10 of Chapter 3) for the same set of problems. The FPTAS assumes the existence of a pseudo-polynomial algorithm to find the regret of a given solution and a 2-approximation min-max regret algorithm.

Pereira et al. [69] studied the min-max regret for the set covering problem [76] with uncertain costs for every subset. They give an exact algorithm based on the mixed-integer programming (MILP) formulation. To solve the MILP, they initially consider a set of scenarios and apply Benders decomposition technique to obtain new constraints. The structure of the worst case scenario is the same as the worst case scenario in [49]. The algorithm is iterative and in every iteration, a constraint that violates the current solution is added to the formulation. Other faster heuristics are also discussed in the same paper. Montemanni et al. [65] provide a mathematical formulation for min-max regret for the robust traveling salesman problem. Here, an uncertainty parameter is the distance between any two vertices and this uncertainty is represented by an interval.

Definition 5.1 (Quadratic Assignment Problem (QAP) [56])**.** Given a set of n facilities and a set of n locations. For each pair of locations, a distance is specified and for each pair of facilities a weight or flow is specified (e.g., the amount of supplies transported between the two facilities). The QAP problem is to assign all facilities to different locations with the goal of minimizing the sum of the distances multiplied by the corresponding flows.

Recently, Feizollahi et al. [30] gave the MILP formulation for min-max regret for QAP where the material flows between the facilities are uncertain. Here, a worst case scenario is obtained by solving the deterministic QAP. Lastly, there are many research articles that study robust optimization, a few of them are reviewed in [29], [78]. A survey of robust optimization techniques is in [34] and [1].

Robust optimization is a very active area of research. One example of uncertainty is the scheduling of transmissions in wireless networks, where traffic burst is commonplace.

Robust optimization has found many applications in classical logistics, facility location, and finance [34]. However, the structure of RWVCP is very different from that of the majority of the robust problems studied in the literature. In the robust problems that we reviewed, [30], [49], [69], once a solution is fixed, finding a worst case scenario usually leads to an easier optimization problem. For RWVCP, finding a worst case scenario is intractable. Moreover, the objective function of RWVCP employs the *maximum* operator because the weight of a color class is determined by the maximum weight of a vertex. This operator disallows distribution of the cost of the objective function among the elements that make up a feasible solution. For this reason, the approach taken by Kasperski et al. [49] applied for a large class of problems does not apply to RWVCP. We propose a math-heuristic based on the column generation method and a local search heuristic for RWVCP. In the next section, our results.

5.2 A Path Graph With Uncertain Weights

Example: Consider a weighted path shown in Figure 5.1. Here, the weight of each vertex can take any integer value in $[1,5]$. One brute force algorithm to determine a color with minimum regret is to compute $Z(X)$, for each X. This requires computation of $5^4 * k^4$ entries where k is the number of colors. There are 5^4 scenarios and $O(k^4)$ different colorings for a choice of colors. In this example, we consider 2 and 3 as the choice of colors since a weighted path can be colored with at most three colors. In general, this approach is not practical because the number of scenarios is exponential in the number of vertices and for a choice of colors there are an exponential number of ways of coloring. We classify the scenarios into two types, those in which the weight of each vertex w_v is either $\underline{w}$ or $\overline{w}$, and those scenarios for which $\underline{w} < w < \overline{w}$ for some vertex v. We show that a scenario of the later type does not determine $Z(X)$ for any coloring X.

Lemma 5.2. *Let G be a graph. If s is a worst case scenario then $w_v = \underline{w}_v$ or $\overline{w}_v$ for each vertex v.*

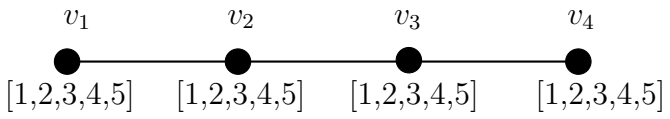


Figure 5.1: A Weighted Path

Proof. Consider a worst case scenario s, such that for some vertex v, $\underline{w}_v < w_v < \overline{w}_v$. Let $\overline{s_v}(\underline{s_v})$ be the scenario obtained by changing the weight of vertex v, $w_v = \overline{w}_v(w_v = \underline{w}_v)$ while keeping all the other weights the same. We will show that either $\underline{s_v}$ or $\overline{s_v}$ has the same regret as s, for any coloring X. Both $\underline{s_v}$ and $\overline{s_v}$ have one fewer vertex with value, $\underline{w}_v < w_v < \overline{w}_v$. Therefore this process can be iterated. Eventually we get a scenario in which for every vertex v, $w_v = \underline{w}_v$ or $w_v = \overline{w}_v$. By the definition of regret,

$$R(\underline{s_v},X) = F(\underline{s_v},X) - F^*(\underline{s_v}) = F(\underline{s_v},X) - \min_{X\in\phi} F(\underline{s_v},X) \tag{5.6}$$

$$R(\overline{s_v},X) = F(\overline{s_v},X) - F^*(\overline{s_v}) = F(\overline{s_v},X) - \min_{X\in\phi} F(\overline{s_v},X) \tag{5.7}$$

Also,

$$R(s,X) = F(s,X) - F^*(s) = F(s,X) - \min_{X\in\phi} F(s,X) \tag{5.8}$$

We consider two cases:

Case 1: Suppose vertex v determines the weight of some color class α in the X coloring of s. In this case, to obtain a contradiction assume $R(\overline{s_v},X) < R(s,X)$. That is

$$F(\overline{s_v},X) - \min_{X\in\phi} F(\overline{s_v},X) < F(s,X) - \min_{X\in\phi} F(s,X) \tag{5.9}$$

As vertex v determines the weight of some color class, $F(\overline{s_v},X) = F(s,X) + (\overline{w}_v - w_v)$, substituting this in equation (5.9) we get,

$$F(s,X) + (\overline{w}_v - w_v) - \min_{X \in \phi} F(\overline{s_v}, X) < F(s,X) - \min_{X \in \phi} F(s,X)$$

Further,

$$(\overline{w}_v - w_v) < \min_{X \in \phi} F(\overline{s_v}, X) - \min_{X \in \phi} F(s,X) \tag{5.10}$$

By changing the weight of one vertex by $\varepsilon > 0$, the maximum increase that we see in $F^*(s)$ for any scenario s, is ε. Therefore (5.10) is a contradiction.

Case 2: Suppose vertex v does not determine the cost of any color class α in coloring X. We compare equations (5.6) and (5.8). To obtain a contradiction assume $R(\underline{s_v}, X) < R(s,X)$.

$$F(\underline{s_v}, X) - \min_{X \in \phi} F(\underline{s_v}, X) < F(s,X) - \min_{X \in \phi} F(s,X) \tag{5.11}$$

As vertex v does not determine the weight of any color class, $F(\underline{s_v}, X) = F(s,X)$. Substituting this in equation (5.11) we get, $\min_{X \in \phi} F(\underline{s_v}, X) > \min_{X \in \phi} F(s,X)$. Once again decreasing the weight of a vertex, cannot increase the cost of $F^*(s)$ therefore we have a contradiction. □

Example: Let us number the vertices on the path 1,2,3,4. We denote the colors as R, G, B. By Lemma 5.2 there are 2^4 worst case scenarios. Each solution entry in Table 5.1 is of the form $F(s,X)/R(s,X)$. The last column is the cost of the optimal coloring of a scenario s, $F^*(s)$. The maximum regret over each column is shown in the second last row. The robust coloring is the coloring with the minimum value in the last row, $[RGRG]$ in this case.

Let us first consider paths when all the vertices have the same uncertainty interval $[\underline{w}, \overline{w}]$ associated with them.

Lemma 5.3. *If P is a path with the uniform uncertainty interval $[\underline{w}, \overline{w}]$ at each vertex then the 2-coloring is a robust solution.*

Table 5.1: Optimal Regret Calculation of the Path of Figure 5.1

Colorings / Senarios	RGRG	RGBR	RBGR	RGRB	$F^*(s)$
1111	2/0	3/1	3/1	3/1	2
1115	6/0	7/1	7/1	7/1	6
1151	6/0	7/1	7/1	7/1	6
1155	10/0	11/1	11/1	11/1	10
1511	6/0	7/1	7/1	7/1	6
1515	6/0	11/5	11/5	11/5	6
1551	10/0	11/1	11/1	11/1	10
1555	10/0	15/5	15/5	15/5	10
5111	6/0	7/1	7/1	7/1	6
5115	10/3	7/0	7/0	11/4	7
5151	6/0	11/5	11/5	7/1	6
5155	10/0	11/1	11/1	11/1	10
5511	10/0	11/1	11/1	11/1	10
5515	10/0	11/1	11/1	15/5	10
5551	10/0	15/5	15/5	11/1	10
5555	10/0	15/5	15/5	15/5	10
Z(X)	**3**	**5**	**5**	**5**	
$\min Z(X)$	***3***				

Proof. Consider the regret matrix M as in Table 5.1 where the rows corresponds to the scenarios and columns to the colorings. Entry $M(s,X)$ is $R(s,X)$. Let M_2 be the column corresponding to the unique 2-coloring of P. M_2 is not identically zero, otherwise 2-coloring is a robust solution. The maximum regret in column M_2 is obtained when 2-coloring is not optimal for some scenario s. The minimum cost of any 3-coloring of scenario s is at least $\overline{w}+2\underline{w}$. And the max 2-coloring cost is at most $2\overline{w}$. Therefore the maximum regret in M_2 is $2\overline{w}-(\overline{w}+2\underline{w}) \leq \overline{w}-2\underline{w}$. We will exhibit a scenario $(\overline{w}\ \overline{w}\ \overline{w}...\overline{w})$ for which any 3-coloring has a larger than $(\overline{w}-2\underline{w})$ regret. The regret for scenario $s' = (\overline{w}\ \overline{w}\ \overline{w}...\overline{w})$, is $3\overline{w}-2\overline{w} = \overline{w}$. Therefore the regret of any 3-coloring is greater than 2-coloring i.e. $\overline{w} > (\overline{w}-2\underline{w})$. This implies, 2-coloring is robust solution when a path has same uncertainty interval on each vertex. □

Lemma 5.4. *If P is a path with an interval* $[\underline{w}_v, \overline{w}]$ *on each vertex* $v \in V$ *then the 2-coloring*

is robust coloring (uniform upper bound, possibly different lower bounds).

Proof. The proof is similar to Lemma 5.3. Let $\{\alpha_1, \alpha_2, \alpha_3\}$ be three different color classes with $w_i = w(\alpha_i)$ and $w_1 \geq w_2 \geq w_3$. Assume that the vertices p and q determine the weight of α_2 and α_3 in the optimal coloring of s and $\underline{w}_p$ and $\underline{w}_q$ be their lower bounds respectively. The cost of any 3-coloring for scenario s is at least $\overline{w} + \underline{w}_p + \underline{w}_q$. We know that the regret value is not less than zero. Additionally, $w_i \geq \sum_{j=i+1}^{k} w_j : 1 \geq i \geq k$ for an optimal k-coloring in a bipartite graph [68]. Hence, $(\underline{w}_p + \underline{w}_q) \leq \overline{w}$, if $w_1 = \overline{w}$. The 2-coloring cost is at most $2\overline{w}$. Therefore, the maximum regret of 2-coloring is $2\overline{w} - (\overline{w} + \underline{w}_p + \underline{w}_q) \leq \overline{w} - (\underline{w}_p + \underline{w}_q)$. Once again for a scenario $(\overline{w}\ \overline{w}\ \overline{w}...\overline{w})$ any 3-coloring has regret larger than $\overline{w} - (\underline{w}_p + \underline{w}_q)$. The regret for scenario $(\overline{w}\ \overline{w}\ \overline{w}...\overline{w})$, is $3\overline{w} - 2\overline{w} = \overline{w}$. Therefore, the regret of any 3-coloring is greater than R_2 i.e. $\overline{w} > (\overline{w} - (\underline{w}_p + \underline{w}_q))$. Hence, 2-coloring is a robust solution for a path with uncertain interval $[\underline{w}_v, \overline{w}]$ associated with each vertex. □

Lemma 5.5. *If s is a worst case scenario and X is a coloring with k color classes then in each color class there is exactly one vertex v with $w_v = \overline{w}_v$ and for all the other vertices u, $w_u = \underline{w}_u$.*

Proof. Consider a worst case scenario s. Let X be the coloring with color classes α_1, α_2, ... , α_k. Let $w(\alpha_j)$ be the weight of j^{th} color class. Let w_v be the weight of a vertex v in s. By definition, $w(\alpha_j) = \max_v \{w_v : v \in \alpha_j\}$. Let v with weight $\overline{w}_v$ be the vertex that determines the weight of color class α_j. Let $\underline{s_v}$ be the scenario obtained by changing the weight of every vertex $v' \neq v : v' \in \alpha_j$ to $\underline{w}_{v'}$. By definition,

$$R(s, X) = F(s, X) - F^*(s) \tag{5.12}$$

$$R(\underline{s_v}, X) = F(\underline{s_v}, X) - F^*(\underline{s_v}) \tag{5.13}$$

Assume, for the sake of contradiction $R(\underline{s_v}, X) < R(s, X)$. In scenario $\underline{s_v}$, the vertex that determines the weight of a color class α_j is unchanged, so we have $F(s, X) = F(\underline{s_v}, X)$.

On the other hand, if we decrease the weight of a vertex, the optimal cost will decrease or remain the same. Hence $F^*(\underline{s_v}) \leq F^*(s)$. Thus,

$$\begin{aligned} R(\underline{s_v}, X) &= F(\underline{s_v}, X) - F^*(\underline{s_v}) \\ &= F(s, X) - F^*(\underline{s_v}) \\ &\geq F(s, X) - F^*(s) \\ &= R(s, X) \end{aligned} \tag{5.14}$$

a contradiction. □

The following theorem is now evident.

Theorem 5.6. *If P is a path, and each vertex has uncertainty in the interval $[\underline{w}_v, \overline{w}]$. Then robust coloring of P can be computed in $O(n)$ time.*

Proof. The weighted 2-coloring with minimum cost in a path can be easily computed in $O(n)$ using algorithm of Kavitha *et al.* [51]. Therefore, the proof follows from Lemma 5.4. □

Theorem 5.7. *If G is a bipartite graph, and each vertex has weight interval $[\underline{w}_v, \overline{w}]$ associated with it. Then 2-coloring is a robust coloring of G.*

Proof. Bipartite graph is 2-colorable. The optimal coloring of a scenario with all weights $\overline{w}$ is a 2-coloring because the coloring of a graph with uniform vertex weights is same as the classical vertex coloring. Therefore, in the proof of Lemma 5.4 any $k \geq 3$ coloring has larger regret. □

5.3 Mathematical Formulation of RWVCP

Our mathematical formulation of RWVCP is based on the WVCP model described in Section 2.4.1 of Chapter 2. This WVCP model is proposed by Malaguti et al. [60] and it

is based on the set covering formulation. The solution of WVCP is based on the column generation method proposed by Mehotra and Trick [63]. The integral solution to WVCP is obtained using branch and price method proposed by Furini and Malaguti [33].

Let X be the set of all the independent sets (color classes) of G, for each color class α introduce a binary variable x_α taking value 1 if the vertices of α receive the same color otherwise x_α is zero. Let $w_\alpha = \max_{v\in\alpha}\{w_v\}$ be the maximum weight of the vertices belonging to α. The mathematical formulation of WVCP based on the set cover formulation is as follows:

$$\min \quad \sum_{\alpha\in X} w_\alpha x_\alpha \tag{5.15}$$

$$\text{subject to:} \sum_{\alpha\in X: v\in\alpha} x_\alpha \geq 1, \forall v \in V \tag{5.16}$$

$$x_\alpha \in \{0,1\}, \qquad \alpha \in X \tag{5.17}$$

Constraints 5.16 specify proper coloring on the graph. Constraints 5.17 are the integrality constraints.

We proposed an ILP formulation for the RWVCP based on above set covering formulation. We describe a solution method for relaxed RWVCP formulation based on the column generation method. Let us describe the symbols used in the RWVCP formulation.

Let Γ be the set of worst case scenarios. An independent set α with $x_\alpha = 1$ corresponds to a color class in RWVCP. Let w_v^s represent the weight of vertex v under a scenario $s \in \Gamma$ and $F^*(s)$ represent the cost of the optimal solution to WVCP for a scenario s. Let r be a variable representing the regret of the solution encoded by variables x_α relative to the set of worst case scenarios in Γ. The mathematical formulation of RWVCP is:

$$\min_{r \in \mathbb{R}} \quad r \tag{5.18}$$

$$\text{subject to: } r \geq \sum_{\alpha \in X} x_\alpha \max_{v \in \alpha} \{w_v^s\} - F^*(s), \ \forall s \in \Gamma \tag{5.19}$$

$$\sum_{\alpha \in X : v \in \alpha} x_\alpha \geq 1, \qquad \forall v \in V \tag{5.20}$$

$$x_\alpha \in \{0, 1\} \tag{5.21}$$

The relaxed LP of the above formulation is:

$$\min_{r \in \mathbb{R}} \quad r \tag{5.22}$$

$$\text{subject to: } r \geq \sum_{\alpha \in X} x_\alpha \max_{v \in \alpha} \{w_v^s\} - F^*(s), \ \forall s \in \Gamma \tag{5.23}$$

$$\sum_{\alpha \in X : v \in \alpha} x_\alpha \geq 1, \qquad \forall v \in V \tag{5.24}$$

$$x_\alpha \geq 0 \tag{5.25}$$

Constraints 5.23 state that the regret of a scenario s under a fixed coloring is at most r. Constraint 5.24 generates proper coloring. Lastly, constraints 5.25 are the integrality constraints. We note that the cardinality of sets X and Γ is, in the worst case, exponential in the problem size.

Let $\mu_s, s \in \Gamma$ and $\pi_v, v \in V$ be the dual variables corresponding to constraints 5.23 and 5.24 respectively. The dual of the above formulation (5.22- 5.25) is:

$$\max \sum_{s\in\Gamma}(-\mu_s)F^*(s) + \sum_{v\in V}\pi_v \tag{5.26}$$

$$\text{subject to: } \sum_{s}\mu_s \leq 1, \tag{5.27}$$

$$\sum_{v\in\alpha}\pi_v - \sum_{s}\mu_s \max_{v\in\alpha}\{w_v^s\} \leq 0,\ \forall\alpha \in X \tag{5.28}$$

$$\pi_v, \mu_s \geq 0 \tag{5.29}$$

Then, the column generation sub-problem is to find an independent set α^* with minimum reduced cost. We describe two mathematical models for solving the sub-problem to find the column with the negative reduced cost.

5.3.1 Sub-problem

Method 1:

The formulation (5.30-5.32) is the maximum weighted independent set problem with a modified objective function. This problem is NP-hard in general graphs (see Gary and Johnson [37]).

$$\max \quad -\sum_{s\in\Gamma}\mu_s \max_{v:z_v=1}\{w_v^s\} + \sum_{v\in V}\pi_v z_v \tag{5.30}$$

$$\text{subject to: } z_v + z_{v'} \leq 1, \qquad \forall(v,v') \in E \tag{5.31}$$

$$z_v \in \{0,1\} \tag{5.32}$$

Constraints 5.31 specify no two adjacent vertices receive the same color. If the optimal solution to the above relaxed formulation (5.30 - 5.32) has a value smaller than 0, the independent set is added to the restricted master problem and we iterate. Otherwise, the restricted model of (5.22 -5.25) is optimal. The difficulty in solving this sub-problem is due

to the maximum operator in the objective function. In the worst case, we iterate all weights of each scenario to obtain the column with negative reduced cost. This is the reason, we describe the next sub-problem formulation.

Method 2:

The sub-problem formulation described above is linear in size because there are a linear number of weights associated with each vertex. However, we have to iterate through all the subgraphs related to each weight. Hence, we modify the objective function of Method 1 by representing $\max_{v:z_v=1}\{w_v^s\}$ as a variable y_s and added new constraints 5.35 such that the weight of an independent set is always determined by the maximum weighted vertex.

$$\min \quad \sum_{s\in\Gamma} \mu_s y_s - \sum_{v\in V} \pi_v z_v \tag{5.33}$$

$$\text{subject to: } z_v + z_{v'} \leq 1, \qquad \forall (v,v') \in E \tag{5.34}$$

$$w_v^s z_v \leq y_s, \qquad s \in \Gamma, v \in V \tag{5.35}$$

$$z_v \in \{0,1\}, y_s \geq 0 \tag{5.36}$$

In the next section, we describe our algorithm based on the above formulations.

5.3.2 Outline of RWVCP Algorithm

Initial Scenarios: We consider an initial scenario (s_{eo}) to start with: weight $\overline{w}_v$ on even indexed vertices and $\underline{w}_v$ on the odd indexed vertices.

Algorithm 10 RWVCP

1: Solve problem $F^*(s_{eo})$. Set $\Gamma^1 = \{s_{eo}\}$
2: Solve the LP relaxation of the *master problem* (5.22 - 5.25) using the column generation method. Let X^* be the optimal coloring and r^* be the optimal objective value .
3: Find a scenario Γ' by local search method such that constraints 5.23 is violated.
4: Add the scenario Γ' to the master problem and repeat step 2. Exit the algorithm if we cannot improve the regret.

5.3.3 Branch and Price

In each iteration of cutting plane, we convert a fractional solution to an integral solution similar to Mehrotra and Trick [63] and Furini and Malaguti [33] ideas due to originally by Zykov [79]. We first select a column with a maximum fractional value say, x_{α_i}, with $\alpha_i \in X$. We then select two vertices u and v, non-adjacent to each other and apply branching rules. We consider two branching rules. First branching rule assigns the same color to u and v. This rule is modeled by replacing u and v with a single vertex l, with $(l,o) \in E$ for every vertex o for which either $(u,o) \in E$ or $(v,o) \in E$. This branching can be written as a constraint $x_u = x_v$. The second branching rule assigns different colors to u and v. This rule is modeled by adding an edge between them. We represent this branching rule as a constraint $x_u + x_v \leq 1$. In case first case the graph has one less vertex, and in the second case, the graph has an additional edge.

Let X^* be an optimal fractional solution to the master problem. The selection of u and v is as follows: we choose u as the first vertex in a row i of $\alpha_1 \in X^*$ (α_1 has the maximum fractional value) whose upper bound weight is maximum among all the scenarios in the current solution. Then, we determine another column α_2 in current solution such that the fractional value of α_2 is same as the fractional value of α_1 and the row i is set to 1. If such column doesn't exist, we find the column with a fractional value smaller than α_1. Next, we find a vertex v in a row of α_2 which has upper bound lesser or equal to u.

5.4 Worst Case Scenario Given A Coloring

In this section, we discuss a formulation to compute the worst case scenario given a coloring. We describe the process of calculating the regret. Let us first discuss the compact formulation of WVCP given in [59]. Let z_h be the weight of a color class, h a color and x_{vh} a binary variable that represents whether a vertex v is assigned the color h or not. Following is an integer linear program.

$$\min \quad \sum_{h=1}^{n} z_h \tag{5.37}$$

$$\text{subject to: } z_h \geq w_v * x_{vh}, \qquad \forall v \in V, h = 1...n \tag{5.38}$$

$$\sum_{h=1}^{n} x_{vh} = 1, \qquad \forall v \in V \tag{5.39}$$

$$x_{uh} + x_{vh} \leq 1, \qquad (u,v) \in E, h = 1...n \tag{5.40}$$

$$x_{vh} \in \{0,1\}, z_h \geq 0 \tag{5.41}$$

Constraints 5.38 ensure that the weight of a color class is the maximum weight of any vertex in that color class. Constraints 5.39 ensure that every vertex is assigned exactly one color. Constraints 5.40 ensure that adjacent vertices are assigned a different color. This formulation considers $h = \Delta + 1$ number of colors.

Given a coloring X, let J be an independent set. Let c_j be the weight of j^{th} color class. Let $w_v = \underline{w}_v + y_v(\overline{w}_v - \underline{w}_v)$ for every $v \in V$ and y_v be a binary variable i.e. $y_v \in \{0,1\}$, where $\delta_v = (\overline{w}_v - \underline{w}_v)$. The quadratic formulation to compute the worst case scenario is:

$$\max \quad \sum_{j \in J} c_j - \sum_{h=1}^{n} z_h \tag{5.42}$$

$$\text{subject to: } \bigvee_{v \in j} (c_j \leq \underline{w}_v + y_v * \delta_v), \forall j \in J \tag{5.43}$$

$$z_h \geq (\underline{w}_v + y_v * \delta_v) * x_{hv}, \forall v \in V, h = 1...n \tag{5.44}$$

$$\sum_{h=1}^{n} x_{hv} = 1, \forall v \in V \tag{5.45}$$

$$x_{hu} + x_{hv} \leq 1, (u,v) \in E, h = 1...n \tag{5.46}$$

$$x_{hv} \in \{0,1\}, y_v \in \{0,1\}, z_h \geq 0, c_j \geq 0 \tag{5.47}$$

Note that constraints 5.43 force the weight on each vertex to be either $\overline{w}_v$ or $\underline{w}_v$ and

ensure that among all vertices in a color class, a maximum weighted vertex determines the weight of the color class in the given coloring. Constraints 5.44 ensure that the weight of a color class is the maximum weight of any vertex in that color class in the optimal coloring. All other constraints are as before.

The above mathematical formulation is quadratic in nature. One way to solve a quadratic program is to linearize it and solve it using LP method. In the following subsection, we describe the linearization method.

Linearization Method:

$$\max \quad \sum_{j \in J} c_j - \sum_{h=1}^{n} z_h \tag{5.48}$$

$$\text{subject to: } c_j \leq \sum_{v \in j} \beta_{jv}(\underline{w}_v + y_v * \delta_v), \forall j \in J \tag{5.49}$$

$$z_h \geq w_v * x_{hv}, \forall v \in V, h = 1...n \tag{5.50}$$

$$\sum_{h=1}^{n} x_{hv} = 1, \forall v \in V \tag{5.51}$$

$$x_{hu} + x_{hv} \leq 1, (u,v) \in E, h = 1...n \tag{5.52}$$

$$x_{hv} \in \{0,1\}, y_v \in \{0,1\}, z_h \geq 0, c_j \geq 0, \beta_v \in \{0,1\} \tag{5.53}$$

We introduce a convex hull β_{jv} in 5.43 to remove the disjunction between the constraints, so that a single vertex determine the maximum weight of the independent set. Due to the introduction of β_{jv} in constraints 5.43, constraints 5.49 again becomes quadratic. Hence, we have to linearize constraints 5.49 and 5.50.

We know that if x and y are two binary variables then the product $z = x * y$ can be replaced by the following three linear equations:

$$\begin{aligned} z \geq & x+y-1 \\ z \leq & y \\ z \leq & x \end{aligned} \tag{5.54}$$

Let $p_{hv} = y_v * x_{hv}$ and $q_{jv} = \beta_{jv} * y_v$. Thus, the above worst case quadratic programming can be linearized as:

$$\max \quad \sum_{j \in J} c_j - \sum_{h=1}^{n} z_h \tag{5.55}$$

$$\text{subject to: } c_j \leq \sum_{v \in j} (\beta_{jv} * \underline{w}_v + q_{jv} * \delta_v), \ \forall j \in J \tag{5.56}$$

$$y_v + \beta_{jv} - q_{jv} \leq 1, \ \forall v, \forall j \tag{5.57}$$

$$q_{jv} \leq \beta_{jv}, \ \forall v, \forall j \tag{5.58}$$

$$q_{jv} \leq y_v, \ \forall v, \forall j \tag{5.59}$$

$$\sum_{v \in j} \beta_{vj} = 1, \ \forall j \in J \tag{5.60}$$

$$z_h \geq \underline{w}_v * x_{hv} + p_{hv} * \Delta_v, \ \forall v \in V, h = 1...n \tag{5.61}$$

$$p_{hv} \geq y_v + x_{hv} - 1, \ \forall v \in V, h = 1...n \tag{5.62}$$

$$p_{hv} \leq y_v, \ \forall v \in V, h = 1...n \tag{5.63}$$

$$p_{hv} \leq x_{hv}, \ \forall v \in V, h = 1...n \tag{5.64}$$

$$\sum_{h=1}^{n} x_{hv} = 1, \ \forall v \in V \tag{5.65}$$

$$x_{hu} + x_{hv} \leq 1, \ (u,v) \in E, h = 1...n \tag{5.66}$$

$$y_v \in \{0,1\}, x_{hv} \in \{0,1\}, z_h \geq 0, p_{hv} \in \{0,1\}, c_j \geq 0, q_v \in \{0,1\}, \beta_v \in \{0,1\} \tag{5.67}$$

In our experiment, we relax all the variables except y_v because the value of y_v determines a worst case scenario. If we obtain the worst case scenario by taking $\overline{w}_v$ if $y_v = 1$ otherwise consider $\underline{w}_v$ if $y_v = 0$. Furthermore, we use the above upper bound formulation in our experiment to analyze the gap of our heuristic solution. In the next section, we describe a local search heuristic to obtain worst case scenarios.

5.4.1 Local Search

Our LP formulation (5.22 - 5.25) consists of an exponential number of constraints and an exponential number of variables. By using column generation, we can handle the exponential number of variables, but the problem still lies in determining constraints (worst case scenarios) for a given solution. We formulate LP (5.56- 5.64) to obtain a worst case scenario, but the presence of artificial constraints on the formulation makes the solution very fractional, hence the gap between the optimal regret and the regret returned by upper bound formulation increases. On the other hand, the dependency of the worst case formulation on the compact WVCP also restricts the use of the above formulation on larger instances. Therefore, we need a faster heuristic to obtain worst case scenarios for *the slave problem*. We choose local search heuristic which relies on neighborhood search for finding the worst case scenario.

Before we proceed further with the local search heuristic, we assume that the master problem is solved optimally. Then, to generate a new scenario s' at Step (3) of our algorithm 10 (see Figure 5.2), first we consider the constraint 5.23 corresponding to a scenario s^* that is tight (constraints $ax \leq b$ are tight if $ax = b$) for the optimal regret coloring X^*. Given s^*, if X^* is the current coloring with regret $Z(X^*)$ and OPT as the optimal weighted coloring of s^*, the new regret is obtained by $Z(X) = F(s', X^*) - OPT$. If $Z(X) > Z(X^*)$, then the new scenario s' is added as a constraint. Our goal is to use local search to find s'. We consider three local search operators for finding the new scenario s'.

1. Add: In this technique, we increase the weight of some vertex v of s^* to the upper

bound if it is at lower bound.

2. Delete: In this technique, we decrease the weight of some vertex v of s^* to the lower bound if it is at upper bound.

3. Swap: We swap the weight between a pair of vertices in scenario $s*$.

The overall steps of the algorithm are shown in Figure 5.2.

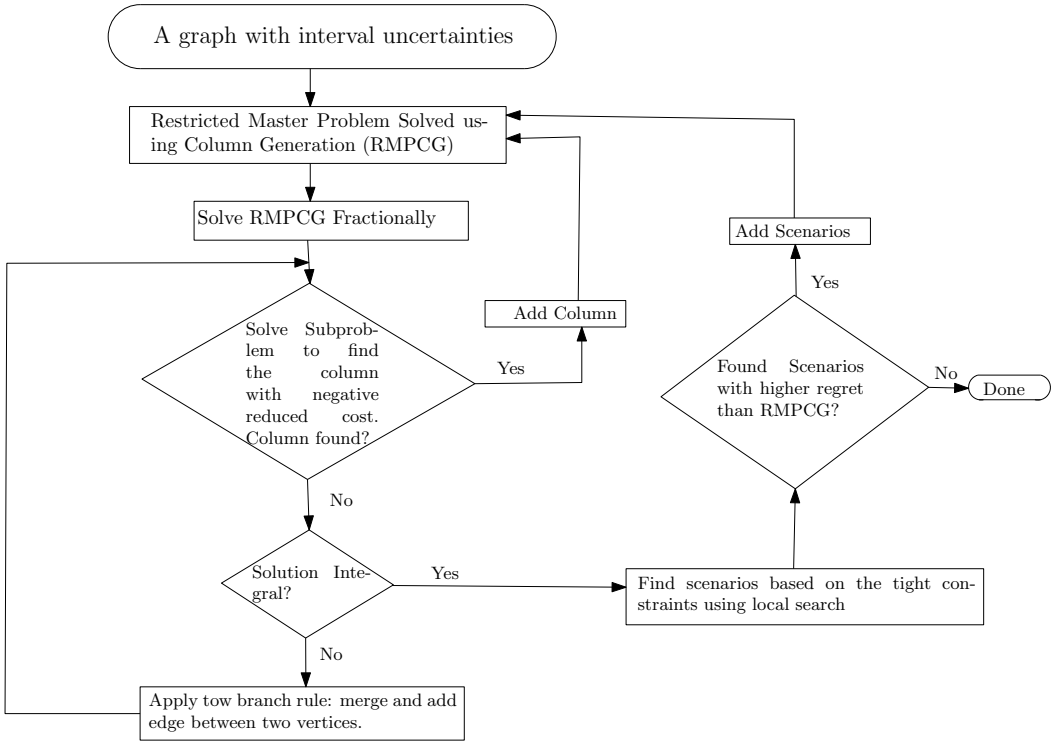


Figure 5.2: An outline of RWVCP algorithm

5.5 Experimental Results

As RWVCP has not been studied empirically, to the best of our knowledge no benchmark problems are available in the literature. For our empirical study, we have selected some VCP instances available in the DIMACS collections (http://mat.gsia.cmu.edu/COLOR02/). We also generate some instances on paths. The algorithms are coded in Octave 3.8.1. All experiments were conducted on a UNIX machine

with 8GB RAM and Intel i3 processor (1.9 GHZ, 3MB L3 cache). The algorithm has a time limit of 7,200 seconds for each iteration of column generation and 7,200 seconds for each iteration of the local search. If the regret value cannot be minimized or there are no new scenarios generated, we terminate the algorithm. Given an instance I of VCP, we only need to generate $\underline{w}_v$ and $\overline{w}_v$, these values are generated uniformly at random in [1, 25]. The instances follow:

a. Graph Coloring Instances

- Path-# : These are path instances with a symbol # representing the number of vertices in a path.
- queen: These are the graphs from Donald Knuth's Stanford GraphBase.
- myciel: These graphs are based on Mycielski transformation and are triangle free.
- Insertions: These are a generalization of Myciel graphs.
- FullIns: These are also a generalization of Myciel graphs.

b. Coloring with Fixed Sets

qwhdec and qwhopt are the instances of coloring with fixed sets.

c. Bandwidth and Multicoloring Instances

These are the Geometric graphs generated by Michael Trick. Points are generated in a 10,000 by 10,000 grid and are connected by an edge if they are close enough together. GEOM and R50 are the instances of this class.

5.5.1 Implementation Details

We first consider the restricted master problem with a single scenario where each vertex receives a distinct color. We solve this restricted master problem optimally using glpk solver. After calculating the optimal solution of the restricted master problem, we find the

dual solution and use the values of those dual variables in the sub-problem (5.33 -5.36) to generate the columns. These columns are repeatedly added to the restricted master problem until there are no negative reduced cost columns.

In the next step, a tight constraint (5.23) (a worst case scenario) with the maximum dual is used. A local search is then performed to find a new worst case scenario s'. We have implemented two techniques to obtain a new scenario s':

1. A worst case scenario s' is selected based on a fractional solution X. Here, we do not use the branch and price procedure.

2. A worst case scenario s' is selected based on an integral solution X, which is obtained using a branch and price procedure.

In each such iteration, we generate multiple scenarios using local search that satisfy $Z'(X) > Z(X)$. After obtaining these constraints, we add them to the restricted master problem. To speed up the column generation process, we reuse the previous columns in each iteration. After we obtain a robust coloring by solving the LP formulation (5.22-5.25), we use the worst case formulation (5.56 - 5.67) to determine the gap of the solution.

5.5.2 Numerical Results

In this section, we discuss the results on on paths and other test instances introduced earlier. The following symbols are used in the tables:

* **n** is the number of vertices in the graph,

* **r** is the current regret (which is also the lower bound on the regret),

* **BP** is the regret value obtained using the branch and price technique,

* **NC** is the number colors in fractional solution,

* **NCol** is the number of columns in the final solution,

* **Cut** is the number of cut (worst case scenarios),

* **t** is total time in seconds,

* **UB** is the upper bound regret value given by the worst case scenario formulation,

* **(UB -r)/UB** is the gap between the regret value returned by local search heuristic and the worst case scenario formulation.

In our solution, we generate scenarios using local search method, so the obtained bounds r, BP and UB are not valid bounds, but rather heuristic approximations.

In Table 5.2, we have presented the experimental results without the use of branch and price method. We see that the running time depends on the number of vertices and also on the number of worst case scenarios. Table 5.3 list the results when branch and price method is used. In Figure 5.3, we show the difference between the running time of the algorithm that uses branch price and the algorithm that does not use the branch and price technique. As we increase the number of vertices, the algorithm spent more time in the column generation and the generation of worst case scenarios. However, it takes less time to calculate the UB but the space requirements grow exponentially with an increase in the number of vertices.

We can also note that the regret values in Table 5.3 that use the branch and price method are better than regret value in Table 5.2. By the use of branch and price method, we obtain worst case scenarios (constraints) based on the integral solution. The difference between a number of colors in a fractional solution and integral coloring can be high. Hence, worst case scenarios obtained using a fractional solution may not be close to the worst case scenario for a robust coloring.

In Figure 5.4, we compare the regret values between branch and price and upper bound method on the DMACS library instances. Most of the time, branch and price technique provides better regret values compared to the worst case formulation using linearization. The introduction of artificial constraints due to linearize the quadratic programming increase the

Table 5.2: Results on standard instances without using branch and price

S.No.	Group	n	r	NC	Ncol	Bcut	t(sec)	UB	UB t(sec)	(UB-r)/UB
1	myciel3	11	20.1	16	33	41	12	74.1	0.128	0.73
2	GEOM20	20	47	79	441	877	11000	125	0.827	0.62
3	GEOM20a	20	58.2	42	215	194	406	135	0.9	0.57
4	myciel4	23	14.6	29	108	86	69.3	104	0.74	0.86
5	queen5_5	25	19.5	10	65	103	92.5	177	1.66	0.89
6	qwhdec.order5.holes10.1	25	44.4	36	171	211	635	161	2.81	0.72
7	GEOM30a	30	57.4	73	343	581	10900	191	2.23	0.7
8	1-Fullins_3	30	29.2	42	290	315	2100	86.2	6.85	0.66
9	queen6_6	36	49.9	52	180	475	6550	240	10.3	0.79
10	2-Insertions_3	37	10	52	390	453	11400	67.1	10.4	0.85

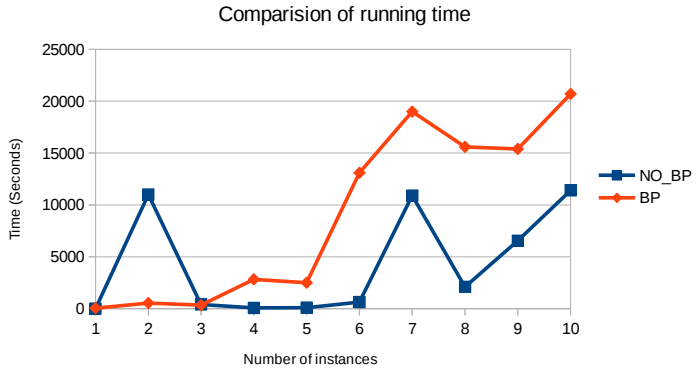


Figure 5.3: Time comparison with and without branch and price

gap. Figure 5.5 compares the fractional regret, the regret obtained by the branch and price method and the regret obtained by the linearize worst case scenario formulation. In most of the cases, the regret value based on branch and price is smaller than the regret value based on the quadratic formulation but larger than the fractional regret.

In Table 5.4, we present the experimental results on paths using a branch and price technique. Since paths are cycle free and the structure is simple, we are interested in examining

Table 5.3: Branch and price on library instances

S.No.	Group	n	BP	NC	Ncol	Bcut	t(sec)	UB	UB t(sec)
1	myciel3	11	86.5	14	23	33	49.1	75.9	0.123
2	GEOM20	20	54	22	92	80	538	110	0.521
3	GEOM20a	20	218	19	80	37	334	132	0.896
4	myciel4	23	111	23	86	78	2820	97.9	1.02
5	queen5_5	25	64	10	59	75	2500	177	1.21
6	qwhdec.order5.holes10.1	25	77	31	97	100	13100	182	649
7	GEOM30a	30	65	33	96	115	19000	165	304
8	1-Fullins_3	30	38	28	117	108	15600	63	2.46
9	queen6_6	36	58.7	49	105	103	15400	258	15.5
10	2-Insertions_3	37	28.9	49	215	93	20700	85	7.46

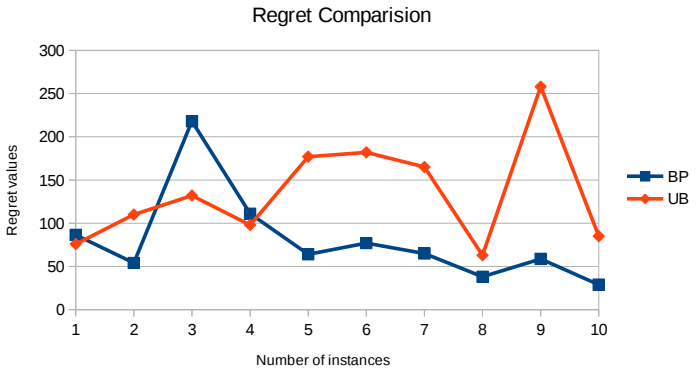


Figure 5.4: Regret comparison: branch and price (BP) and worst case regret (UB)

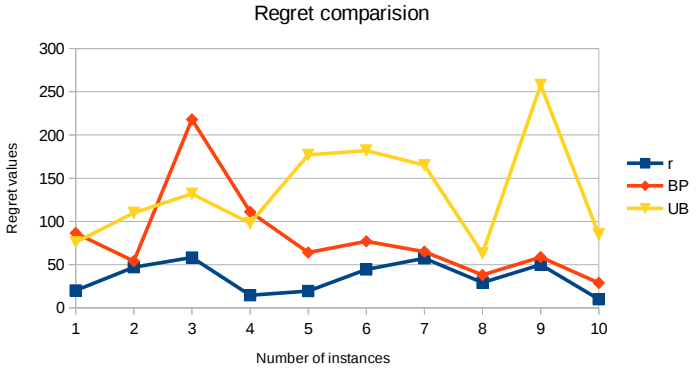


Figure 5.5: Regret comparison: fractional regret (r), branch and price (BP) and worst case formulation regret (UB)

Table 5.4: Results on pathsusing branch and price

S.No.	Group	n	BP	NC	Ncol	Bcut	t(sec)	UB	UB t(sec)	(UB-BP)/UB
1	Path-10	10	16	11	20	13	24.3	34.5	0.0467	0.54
2	Path-15	15	4	5	53	9	8.79	45	0.0824	0.91
3	Path-20	20	17	16	75	26	214	42.7	0.155	0.6
4	Path-25	25	13	8	119	48	1040	36.9	0.442	0.65
5	Path-30	30	5	11	244	33	790	57.3	0.53	0.91
6	Path-35	35	24	24	276	51	9410	56.1	0.996	0.57
7	Path-40	40	3	31	465	67	32900	55	1.44	0.95

the performance of our algorithm. The gap is very high for all the instances as for the standard library instances. We conclude that a better local search strategy to find the worst case scenario or a better linearization technique can only improve the results.

Chapter 6

Conclusion and Future Study

In this thesis, we studied the weight vertex coloring problem (WVCP) when the weights are deterministic and uncertain. In particular, we give algorithms with better time complexity for different classes of trees with deterministic weights. We gave an integer linear programming approach to minimize the maximum regret for the robust version. We gave the math-heuristic based on the ILP and local search. The following paragraphs summarize our contributions.

Deterministic Weight Case: We studied the trees and cactus paths. We improve the running time of the algorithm for binary trees from $O(n^4)$ to $O(n^2 \log n)$. Similarly, we studied cactus paths with the bounded degree of 3, and propose sub-quadratic and quadratic time algorithms. These results are obtained by a careful search over the region of all possible feasible weights using the ideas from previous research [51, 68, 38].

Uncertain Weight Case: We studied the weighted coloring problem when the weights are uncertain. We gave an ILP formulation for the robust weighted vertex coloring problem (RWVCP). We describe a solution procedure based on column generation and local search. We also provide an $O(n)$ time algorithm for the robust coloring of bipartite graphs with uniform upper bound and arbitrary lower bound uncertainties on vertices. We also describe a branch and price method to obtain an integral robust solution. We also provide a quadratic programming formulation to calculate the worst case scenario given a coloring.

6.1 Future Study

To the best of our knowledge, this is the first systematic study of the robust weighted coloring problem based on mathematical models. These results are interesting because the traditional approximation results based on the mid-point scenario do not apply to this problem. One of the key steps is to find a worst case scenario and we gave a quadratic programming formulation. We believe this formulation can be improved with a better linearization method. We also gave a local search heuristics to obtain the worst case scenario. These heuristics can be studied further. Following are other avenues for advancing this study.

6.1.1 Bound on the number of colors

It is an open problem to find a bound on the number of colors in an optimal robust coloring. We believe that $\Delta + 1$ is an upper bound on the number of colors needed in any optimal robust coloring. However, it is hard to prove this bound using the proof technique of moving a vertex from $k > \Delta + 1$ color class to other $k' \leq \Delta + 1$ color classes. A new coloring obtained after transferring a vertex may not give smaller regret value for all the scenarios.

6.1.2 Worst Case Scenario

For the problem considered in the past, finding a worst case scenario for any given solution was easy. In those problems, we obtain the worst case scenario by using $\overline{w}_v$, if v is present in the solution and setting all other elements u to $\underline{w}_u$. We cannot apply this simple strategy in RWVCP because all the elements are part of the solution and the cost of the solution is determined by elements with the maximum weight. As a result, it is hard to determine a specific vertex. We proposed a quadratic programming formulation to obtain the worst case scenario, but the bound obtained by this formulation is very weak compared to the optimal solution (see Figure 5.4 of Chapter 5). Hence, a better linearization technique or a better formulation for a given coloring is needed.

6.1.3 Lower Bound

Obtaining a good lower bound is also a difficult task. In the past, the difference between two solutions is used to obtain the lower bound on the regret. However, we cannot apply this approach directly. In our case, finding a worst case solution is a difficult problem. Better lower bounds will help us prove approximation ratios in future.

Bibliography

[1] Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European journal of operational research*, 197(2):427–438, 2009.

[2] Ian F Akyildiz and Xudong Wang. A survey on wireless mesh networks. *IEEE Communications magazine*, 43(9):S23–S30, 2005.

[3] Kenneth Appel and Wolfgang Haken. Every planar map is four colorable. *Bulletin of the American mathematical Society*, 82(5):711–712, 1976.

[4] Julio Araujo, Nicolas Nisse, and Stéphane Pérennes. Weighted coloring in trees. *SIAM Journal on Discrete Mathematics*, 28(4):2029–2041, 2014.

[5] Ionut Aron and Pascal Van Hentenryck. A constraint satisfaction approach to the robust spanning tree problem with interval data. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 18–25. Morgan Kaufmann Publishers Inc., 2002.

[6] Ionuţ D Aron and Pascal Van Hentenryck. On the complexity of the robust spanning tree problem with interval data. *Operations Research Letters*, 32(1):36–40, 2004.

[7] Sanjeev Arora and Eden Chlamtac. New approximation guarantee for chromatic number. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 215–224. ACM, 2006.

[8] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[9] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization*. Princeton University Press, 2009.

[10] Robert Benkoczi, Ram Dahal, and Daya Gaur. The robust weighted vertex coloring problem with uncertain data. In *Proc. 12th Workshop on Models and Algorithms for Planning and Scheduling Problems*, pages 68–70, 2015.

[11] Robert Benkoczi, Ram Dahal, and Daya Ram Gaur. Exact algorithms for weighted coloring in special classes of tree and cactus graphs. In *International Workshop on Combinatorial Algorithms*, pages 347–358. Springer, 2016.

[12] Robert Radu Benkoczi. *Cardinality constrained facility location problems in trees*. PhD thesis, Simon Fraser University, 2004.

[13] John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[14] R. L. Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37(2):194–197, 10 2008. doi:10.1017/S030500410002168X.

[15] Adam L Buchsbaum, Howard Karloff, Claire Kenyon, Nick Reingold, and Mikkel Thorup. Opt versus load in dynamic storage allocation. *SIAM Journal on Computing*, 33(3):632–646, 2004.

[16] Fred C Chow and John L Hennessy. The priority-based coloring approach to register allocation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(4):501–536, 1990.

[17] V. Chvatal. *Linear Programming*. Series of books in the mathematical sciences. W. H. Freeman, 1983. ISBN: 9780716715870.

[18] Richard Cole and Uzi Vishkin. The accelerated centroid decomposition technique for optimal parallel tree evaluation in logarithmic time. *Algorithmica*, 3(1-4):329–346, 1988.

[19] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[20] Ram Dahal. Exact algorithms for weighted coloring in special classes of tree and cactus graphs. Ottawa-Carleton Discrete Math Days, 2016.

[21] George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

[22] George Bernard Dantzig. *Linear programming and extensions*. Princeton university press, 1998.

[23] Kalyani Das. Cactus graphs and some algorithms. *arXiv preprint arXiv:1408.4005*, 2014.

[24] Dominique De Werra, Mare Demange, Bruno Escoffier, Jerome Monnot, and Vangelis Th Paschos. Weighted coloring on planar, bipartite and split graphs: complexity and improved approximation. In *Algorithms and Computation*, pages 896–907. Springer, 2005.

[25] Marc Demange, Dominique De Werra, Jérôme Monnot, and V Th Paschos. Time slot scheduling of compatible jobs. *Journal of Scheduling*, 10(2):111–127, 2007.

[26] Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.

[27] Reinhard Diestel. *Graph Theory*. Electronic library of mathematics. Springer, 2006. ISBN: 9783540261834.

[28] Bruno Escoffier, Jérôme Monnot, and Vangelis Th Paschos. Weighted coloring: further complexity and approximability results. *Information Processing Letters*, 97(3):98–103, 2006.

[29] Uriel Feige, Kamal Jain, Mohammad Mahdian, and Vahab Mirrokni. Robust combinatorial optimization with exponential scenarios. In *Integer programming and combinatorial optimization*, pages 439–453. Springer, 2007.

[30] Mohammad Javad Feizollahi and Igor Averbakh. The robust (minmax regret) quadratic assignment problem with interval flows. *INFORMS Journal on Computing*, 26(2):321–335, 2013.

[31] Piotr Formanowicz and Krzysztof Tanaś. A survey of graph coloring-its types, methods and applications. *Foundations of Computing and Decision Sciences*, 37(3):223–238, 2012.

[32] G. E. Forsythe. Algorithms. *Commun. ACM*, 7(6):347–349, June 1964. ISSN: 0001-0782.

[33] Fabio Furini and Enrico Malaguti. Exact weighted vertex coloring via branch-and-price. *Discrete Optimization*, 9(2):130 – 136, 2012. ISSN: 1572-5286.

[34] Virginie Gabrel, Cécile Murat, and Aurélie Thiele. Recent advances in robust optimization: An overview. *European Journal of Operational Research*, 235(3):471–483, 2014.

[35] Andreas Gamst. Some lower bounds for a class of frequency assignment problems. *Vehicular Technology, IEEE Transactions on*, 35(1):8–14, 1986.

[36] Michael R Garey and David S Johnson. Computers and intractability: a guide to the theory of np-completeness. 1979. *San Francisco, LA: Freeman*, 1979.

[37] Michael R Gary and David S Johnson. Computers and intractability a guide to the theory of np-completeness. 1979. *WH Freman and Co*, 1979.

[38] D.J. Guan and Zhu Xuding. A coloring problem for weighted graphs. *Information Processing Letters*, 61(2):77 – 81, 1997. ISSN: 0020-0190.

[39] Magnús M Halldórsson and Hadas Shachnai. Batch coloring flat graphs and thin. In *Algorithm Theory–SWAT 2008*, pages 198–209. Springer, 2008.

[40] Thomas Dueholm Hansen. *Worst-case Analysis of Strategy Iteration and the Simplex Method*. PhD thesis, Aarhus University, 2012.

[41] Frank Harary and George E Uhlenbeck. On the number of husimi trees i. *Proceedings of the National Academy of Sciences*, 39(4):315–322, 1953.

[42] Chính T Hoàng, Marcin Kamiński, Vadim Lozin, Joe Sawada, and Xiao Shu. A note on k-colorability of p 5-free graphs. In *International Symposium on Mathematical Foundations of Computer Science*, pages 387–394. Springer, 2008.

[43] Chính T Hoàng and D Adam Lazzarato. Polynomial-time algorithms for minimum weighted colorings of $(P_5, \overline{P_5})$-free graphs and similar graph classes. *Discrete Applied Mathematics*, 186:106–111, 2015.

[44] Hsiang-Chun Hsu and Gerard Jennhwa Chang. Max-coloring of vertex-weighted graphs. *Graphs and Combinatorics*, 32(1):191–198, 2016.

[45] Klaus Jansen and Petra Scheffler. Generalized coloring for tree-like graphs. *Discrete Applied Mathematics*, 75(2):135 – 155, 1997. ISSN: 0166-218X.

[46] Peter Kali and Stein W Wallace. *Stochastic programming*. Springer, 1994.

[47] David Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *Journal of the ACM (JACM)*, 45(2):246–265, 1998.

[48] A Kasperski and P Zielinski. Minimizing maximal regret in linear assignment problems with interval data. *Instytut Matematyki PWr., Wroclaw*, 2004.

[49] Adam Kasperski and Paweł Zieliński. An approximation algorithm for interval data minmax regret combinatorial optimization problems. *Information Processing Letters*, 97(5):177–180, 2006.

[50] Adam Kasperski and Paweł ZielińSki. On the existence of an fptas for minmax regret combinatorial optimization problems with interval data. *Operations Research Letters*, 35(4):525–532, 2007.

[51] Telikepalli Kavitha and Julián Mestre. Max-coloring paths: tight bounds and extensions. *Journal of Combinatorial Optimization*, 24(1):1–14, 2012.

[52] Nasreen Khan, Anita Pal, and Madhumangal Pal. Edge colouring of cactus graphs. *Adv. Model. Optim*, 11(4):407–421, 2009.

[53] WLG Koontz. Economic evaluation of loop feeder relief alternatives. *Bell System Technical Journal*, 59(3):277–293, 1980.

[54] Jan Kratochvil and Zsolt Tuza. Algorithmic complexity of list colorings. *Discrete Applied Mathematics*, 50(3):297–302, 1994.

[55] Frank Thomson Leighton. A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards*, 84(6):489–506, 1979.

[56] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European journal of operational research*, 176(2):657–690, 2007.

[57] Marco E Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[58] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994.

[59] Enrico Malaguti, Michele Monaci, and Paolo Toth. Models and heuristic algorithms for a weighted vertex coloring problem. *Journal of Heuristics*, 15(5):503–526, 2009.

[60] Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190, 2011.

[61] Enrico Malaguti and Paolo Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1):1–34, 2010.

[62] Nimrod Megiddo and Arie Tamir. New results on the complexity of p-centre problems. *SIAM Journal on Computing*, 12(4):751–758, 1983.

[63] Anuj Mehrotra and Michael A Trick. A column generation approach for graph coloring. *informs Journal on Computing*, 8(4):344–354, 1996.

[64] Arunesh Mishra, Suman Banerjee, and William Arbaugh. Weighted coloring based channel assignment for wlans. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(3):19–31, 2005.

[65] Roberto Montemanni, János Barta, Monaldo Mastrolilli, and Luca Maria Gambardella. The robust traveling salesman problem with interval data. *Transportation Science*, 41(3):366–381, 2007.

[66] Roberto Montemanni and Luca Maria Gambardella. An exact algorithm for the robust shortest path problem with interval data. *Computers & Operations Research*, 31(10):1667–1680, 2004.

[67] K.G. Murty. *Linear programming*. John Wiley & Sons, 1983. ISBN: 978-0-471-09725-9.

[68] Sriram V Pemmaraju and Rajiv Raman. Approximation algorithms for the max-coloring problem. In *Automata, languages and programming*, pages 1064–1075. Springer, 2005.

[69] Jordi Pereira and Igor Averbakh. The robust set covering problem with interval data. *Annals of Operations Research*, 207(1):217–235, 2013.

[70] Marcelo Prais and Celso C Ribeiro. Reactive grasp: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12(3):164–176, 2000.

[71] Franco P Preparata and Michael Shamos. *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[72] Rajiv Raman. *Chromatic scheduling*. PhD thesis, University of Iowa, 2007.

[73] Celso Carneiro Ribeiro, Michel Minoux, and Manoel Camillo Penna. An optimal column-generation-with-ranking algorithm for very large scale set partitioning problems in traffic assignment. *European Journal of Operational Research*, 41(2):232–239, 1989.

[74] Leonard J Savage. The theory of statistical decision. *Journal of the American Statistical association*, 46(253):55–67, 1951.

[75] R.J. Vanderbei. *Linear Programming: Foundations and Extensions*. International Series in Operations Research & Management Science. Springer US, 2013. ISBN: 9781475756623.

[76] V.V. Vazirani. *Approximation Algorithms*. Springer Berlin Heidelberg, 2013. ISBN: 9783662045657.

[77] Tai-Kuo Woo, Stanley YW Su, and Richard Newman-Wolfe. Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm. *Communications, IEEE Transactions on*, 39(12):1794–1801, 1991.

[78] Bo Zeng and Long Zhao. Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters*, 41(5):457–461, 2013.

[79] Alexander Aleksandrovich Zykov. On some properties of linear complexes. *Matematicheskii sbornik*, 66(2):163–188, 1949.