Phylogenetic decisiveness and no-rainbow hypergraph coloring

by

Ghazaleh Parvini

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR Of PHILOSOPHY

Major: Computer Science

Program of Study Committee:
David Fernández-Baca, Major Professor
Giora Slutzki
Pavan Aduri
Oliver Eulenstein
Ryan Martin

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2021

DEDICATION

To my amazing mother, Rouhangiz, who has always been my biggest inspiration;

To my awesome father, Shaahin, who always believed in me;

And to my beloved brother, Ramin, who made my childhood memorable.

TABLE OF CONTENTS

[1] **This chapter is joint work with Katherine Braught.**

LIST OF TABLES

Page

LIST OF FIGURES

Page

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my advisor, Professor David Fernández-Baca for giving me the opportunity and providing invaluable guidance throughout my PhD studies. It was a great honor and privilege to work and study under his supervision. Without his guidance and persistent help this dissertation would not have been possible.

I would like to express my sincere gratitude to my committee members, Professor Giora Slutzki, Professor Pavan Aduri, Professor Oliver Eulenstein and Professor Ryan Martin for serving on my advisory committee.

I owe a deep sense of gratitude to Professor Giora Slutzki and Professor Soma Chaudhuri. I was so lucky to have the opportunity to teach under their supervision. I am so thankful for their support, kindness and friendship.

I express my thankfulness to the chair of the Computer Science department, Professor Hridesh Rajan, for his valuable support during my doctoral studies.

I would like to acknowledge Dr. Uwe Schöning. The work I have done in chapter 5 is inspired by his amazing research on $K - SAT$ problem.

This acknowledgement would not be complete without mentioning my research colleague Katherine Brought. Most of the credit of practical results of chapter 6 goes to her.

Starting and continuing the graduate school was not possible without the encouragement and support of my friends. Dr. Amir Ghanbari, Dr. Farzaneh Derakhshan, Dr. Maryam Khademian, Dr. Banafsheh Dolatyar, Tarlaan Fathi, Alireza Ahmadi, Amirhossein Shirzad, Mehdi Kazemi, Motahareh Mounesan, and all my friends and colleagues in Ames. I am so lucky to have you in my life and I will never forget your kindness.

Last but surely not least I want to thank my parents, and my brother for their unconditional support and love.

ABSTRACT

Suppose we aim to build a phylogeny for a set of taxa X using information from a collection of loci, where each locus offers information for only a fraction of the taxa. The question is whether, based solely on the pattern of data availability, called a *taxon coverage pattern*, one can determine if the data suffices to construct a reliable phylogeny. The *decisiveness problem* expresses this question combinatorially. Informally, a taxon coverage pattern is *decisive* if the following holds for any binary phylogenetic tree T for X: the collection of phylogenetic trees obtained by restricting T to the subset of X covered by each locus uniquely determines T. The *decisiveness problem* is the problem of determining whether a given coverage pattern is decisive. Here we relate the decisiveness problem to a hypergraph coloring problem. We use this connection to (1) show that the decisiveness problem is co-NP complete, (2) obtain lower bounds on the amount of coverage needed to achieve decisiveness, (3) devise an exact algorithm for decisiveness, (4) develop problem reduction rules, and use them to obtain efficient algorithms for inputs with few loci, (5) apply local search and devise a deterministic and a randomized algorithm for decisiveness(6) devise Boolean satisfiability (SAT) and integer linear programming formulations (ILP) of decisiveness, which allow us to analyze data sets that arise in practice. For data sets that are not decisive, we use our SAT and ILP formulations to obtain decisive subsets of the data.

CHAPTER 1. INTRODUCTION

Missing data poses a challenge to assembling phylogenetic trees. The question we address here is how much data can we afford to miss without compromising accuracy. We focus on data sets assembled by concatenating data from many (sometimes thousands) of loci (13; 15; 28). Such data sets are used to construct phylogenetic trees by either (i) combining the data from all the loci into a single *supermatrix* that is then used as input to some standard phylogeny construction method (e.g., (10; 23)) or (ii) taking phylogenetic trees computed separately for each locus and combining them into a single *supertree* that summarizes their information (4; 21; 27). For various reasons, the *coverage density* of concatenated datasets — i.e., the ratio of the amount of available data to the maximum possible amount — is often much less than 1 (18). Reference (6) examines a wide range of phylogenetic analyses using concatenated data sets, and reports coverage densities ranging from 0.06 to 0.98, with the majority being under 0.5.

Low coverage density can give rise to ambiguity (19; 24; 29). In supertree analyses, ambiguity manifests itself in multiple supertrees that are equivalent with respect to the method upon which they are based. In super-matrix analyses, it is manifested in multiple topologically different, but co-optimal (in terms of parsimony or likelihood scores) trees. Note that high coverage density does not, by itself, guarantee lack of ambiguity. More important is the coverage pattern itself. The question is whether one can identify conditions under which a given coverage pattern guarantees a unique solution. Sanderson and Steel (26; 19) have proposed a formal approach to studying this question, which we explain next.

A *taxon coverage pattern* for a taxon set X is a collection of sets $\mathcal{S} = \{Y_1, Y_2, \ldots, Y_k\}$, where, for each $i \in \{1, 2, \ldots, k\}$, Y_i is a subset of X consisting of the taxa for which locus i provides information. $\mathcal{S}$ is *decisive* if it satisfies the following property: Let T and T' be two binary phylogenetic trees for X such that, for each $i \in \{1, 2, \ldots, k\}$, the restrictions of T and T' to Y_i are

isomorphic (restriction and isomorphism are defined in Section 2.2). Then, it must be the case that T and T' are isomorphic. The *decisiveness problem* is: Given a taxon coverage pattern $\mathcal{S}$, determine whether or not $\mathcal{S}$ is decisive.

Intuitively, if a taxon coverage pattern $\mathcal{S}$ is *not* decisive, we have ambiguity. That is, there are at least two trees that cannot be distinguished from each other by the subtrees obtained when these trees are restricted to the taxon sets in $\mathcal{S}$.

A necessary and sufficient condition for a coverage pattern to be decisive — the four-way partition property — is known (19; 26) (see also Section 2.2). However, as we show in Section 2.3, deciding whether this condition holds is co-NP complete. On the positive side, the *rooted* case, where at least one taxon for which every locus offers data, is known to be polynomially solvable, and software for handling this case is available (30). *Groves* (1; 7) are a related, but not identical, notion. For a discussion on the relationship between groves and decisiveness, see (18).

1.1 Overview

In **Chapter 2**, we define the *Decisiveness problem* and the *four-way partition theorem* precisely. We also investigate *hypergraphs* and *no-rainbow coloring problem* in hypergraphs. Finally we show the connection between decisiveness and no-rainbow coloring. We show for a given set X of taxa and a collection of sets $\mathcal{S} = \{Y_1, Y_2, \ldots, Y_k\}$ we can find a corresponding hypergraph H where each node is an element of X and each hyperedge is a set in $\mathcal{S}$. The coloring function assigns one color to each node. We call a coloring no-rainbow 4-coloring if there exists no hyperedge with at least 4 nodes with 4 different colors. The connection of two problems says, if there does not exist a surjective no-rainbow 4-coloring for H, then $\mathcal{S}$ is decisive.

In **Chapter 3**, we define lower bounds for the *no-rainbow coloring problem*. We use the fact that small number of hyperedges make it easier to find a no-rainbow coloring. We prove a tight lower bound on the number of hyperedges needed in the given hypergraph. We also prove a tight lower bound on the degree of each vertex by showing each vertex must be participated in sufficient number of hyperedges. We prove that any hypergraph with fewer hyperedges than the bound has

a no-rainbow coloring. Furthermore, any hypergraph with any node with smaller degree than the minimum degree bound has a no-rainbow coloring.

In **Chapter 4**, we present an exact algorithm for solving no-rainbow coloring problem. By the pigeonhole principle, we know that there must be one color which covers at most $\frac{n}{r}$ of the nodes. We start by fixing a subset of at most $\frac{n}{r}$ nodes with one color and decide about the uncolored vertices and the remaining colors. The exact algorithm in chapter 4 solves the decisiveness problem in $O(2.8^n)$. We then suggest *reduction rules* to find a *fixed-parameter* algorithm for the general decisiveness problem. We show if the number of columns are fixed, we can reduce the number of rows to 2^{k-1} and solve the problem in $O(2.8^{2^k})$ time.

In **Chapter 5** we apply *local search* to find deterministic and probabilistic algorithms for decisiveness and no-rainbow coloring problem. We use the idea of starting with a coloring candidate and apply a mixture of local search and other methods to solve the problem. We suggest using *covering codes* to do a better investigation. Finally we suggest a candidate that helps us to solve the problem using a *deterministic algorithm* in $O(2.28^n)$. Using the same candidate gives us a *randomized algorithm* with running time of $O(2^n)$.

Chapter 6 presents a *saisfiability formulation* and an *integer linear programming (ILP) formulation* of the decisiveness problem, along with some experimental results using these formulations. We also show that the ILP approach can be used to obtain subsets of taxa for which the given data is decisive.

In **Chapter 7** which is the final chapter of this research we present concluding remarks and suggest possible future works on this subject.

CHAPTER 2. DECISIVENESS AND NO-RAINBOW COLORINGS OF HYPERGRAPHS

2.1 Introduction

Constructing *supertrees* is the process of combining the information of many smaller overlapping phylogenetic trees to make a single supertree which presents all the needed information from the smaller trees. In this chapter we consider how much data from the set of phylogenetic trees is needed or is sufficient to make a single supertree. In the other words we argue if a given set of phylogenetic trees is decisive or not. We precisely define the *Decisiveness Problem* and present earlier results on this problem. We also show the connection between the *decisiveness problem* and the *no-rainbow coloring problem* in hypergraphs. The connection between the two problems helps us to view the problem from different perspectives and design more efficient algorithms.

2.2 Preliminaries

Throughout the rest of this paper, X denotes a set of taxa, n denotes $|X|$, and, for any positive integer q, $[q]$ denotes the set $\{1, 2, \ldots, q\}$.

2.2.1 Phylogenetic Trees

A *phylogenetic X-tree* (22; 25) is a tree T with leaf set X, where every internal vertex has degree at least three. Fig 2.1 shows two distinct binary phylogenetic X-trees for $X = \{1, 2, 3, 4, 5\}$(26). Biologists are often interested in *rooted* trees, where the root is considered as the origin of species and edges are viewed as being directed away from the root, indicating direction of evolution. Note, however, that most phylogeny construction methods produce unrooted trees.

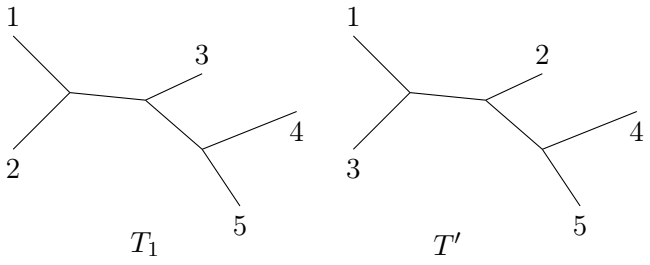

Figure 2.1 Two binary phylogenetic X-trees for $X = \{1, 2, 3, 4, 5\}$. If we take $Y = \{1, 2, 3, 4\}$ then $T|Y = 12|34$ and $T'|Y = 13|24$. (From (26).)

A *split* of taxon set X is a bipartition $A|B$ of X such that $A, B \neq \emptyset$. Let T be a phylogenetic X-tree. Each edge e in T defines a split $\sigma_T(e) = A|B$, where A and B are the subsets of X lying in each component of $T - e$. $\mathrm{Spl}(T)$ denotes the set $\{\sigma_e : e \in E(T)\}$. It is well-known that a phylogenetic X-tree T is completely determined by $\mathrm{Spl}(T)$ (22, Theorem 3.5.2). Two X-trees T and T' are *isomorphic* if $\mathrm{Spl}(T) = \mathrm{Spl}(T')$.

Let T be a phylogenetic X-tree, and suppose $Y \subseteq X$. The *restriction* of T to Y, denoted by $T|Y$, is the phylogenetic Y-tree where
$\mathrm{Spl}(T|Y) = \{A \cap Y | B \cap Y : A|B \in \mathrm{Spl}(T) \text{ and } A \cap Y, B \cap Y \neq \emptyset\}$. Equivalently, $T|Y$ is obtained from the minimal subtree of T that connects Y by suppressing all vertices of degree two that are not in Y.

2.2.2 Decisiveness

A taxon coverage pattern S for X is *phylogenetically decisive* if it satisfies the following property: If T and T' are binary phylogenetic X-trees, with $T|Y = T'|Y$ for all $Y \in S$, then $T = T'$. In other words, for any binary phylogenetic X-tree T, the collection $\{T|Y : Y \in S\}$ uniquely determines T (up to isomorphism). The *decisiveness problem* is the problem of determining whether a given coverage pattern is decisive.

Let Q_S denote the set of all quadruples from X that lie in at least one set in S. That is: $Q_S = \bigcup_{Y \in S} \binom{X}{4}$. A collection S of subsets of X satisfies the *four-way partition property* (for X) if,

for all partitions of X into four disjoint, nonempty sets A_1, A_2, A_3, A_4 (with $A_1 \cup A_2 \cup A_3 \cup A_4 = X$) there exists $a_i \in A_i$ for $i \in \{1, 2, 3, 4\}$ for which $\{a_1, a_2, a_3, a_4\} \in Q_S$.

Theorem 2.2.1. *(26) A taxon coverage pattern $\mathcal{S}$ for X is phylogenetically decisive if and only if S satisfies the four-way partition property for X.*

Corollary 2.2.2. *The decisiveness problem is in co-NP.*

Proof. A certificate for *non*-decisiveness is a partition of X into four disjoint, nonempty sets A_1, A_2, A_3, A_4, such that there is no quadruple $\{a_1, a_2, a_3, a_4\} \in Q_{\mathcal{S}}$ where $a_i \in A_i$ for each $i \in \{1, 2, 3, 4\}$. □

Note that Theorem 2.2.1 implies that a taxon coverage pattern $\mathcal{S}$ for X such that $X \in \mathcal{S}$ (that is, one set in $\mathcal{S}$ contains all the taxa) is trivially decisive.

Theorem 2.2.3. *(26) Let $\mathcal{S}$ be a taxon coverage pattern for X.*

(i) If $\mathcal{S}$ is decisive, then for every set $A \in \binom{X}{3}$, there exists a set $Y \in \mathcal{S}$ such that $A \subseteq Y$.

(ii) If $\bigcap_{Y \in S} Y \neq \emptyset$, then, $\mathcal{S}$ is decisive if and only if for every set $A \in \binom{X}{3}$, there exists a set $Y \in \mathcal{S}$ such that $A \subseteq Y$.

Part (ii) of Theorem 2.2.3 implies that decisiveness is polynomially solvable in the rooted case (25).

2.3 Hypergraphs, No-Rainbow Colorings, and Decisiveness

2.3.1 Hypergraphs

A *hypergraph* H is a pair $H = (X, E)$, where X is a set of elements called *nodes* or *vertices*, and E is a set of non-empty subsets of X called *hyperedges* or *edges* (2; 3). Two nodes $u, v \in V$ are *neighbors* if $\{u, v\} \subseteq e$, for some $e \in E$. The *degree* of a node v, denoted $d(v)$, is the number of edges that contain v. A hypergraph $H = (X, E)$ is *r-uniform*, for some integer $r > 0$, if each hyperedge of H contains exactly r nodes.

A *chain* in a hypergraph $H = (X, E)$ is an alternating sequence $v_1, e_1, v_2, \ldots, e_s, v_{s+1}$ of nodes and edges of H such that: (1) $v_1, \ldots, v_s$ are all distinct nodes of H, (2) $e_1, \ldots, e_s$ are all distinct edges of H, and (3) $\{v_j, v_{j+1}\} \in e_j$ for $j \in \{1, \ldots, s\}$. Two nodes $u, v \in X$ are *connected* in H, denoted $u \equiv v$, if there exists a chain in H that starts at u and ends at v. The relation $u \equiv v$ is an equivalence relation (2); the equivalence classes of this relation are called the *connected components* of H. H is *connected* if it has only one connected component; otherwise H is *disconnected*.

2.3.2 No-Rainbow Colorings and Decisiveness

Let $H = (X, E)$ be a hypergraph and r be a positive integer. An *r-coloring* of H is a mapping $c : X \to [r]$. For node $v \in X$, $c(v)$ is the *color* of v. Throughout this paper, r-colorings are assumed to be *surjective*; that is, for each $i \in [r]$, there is at least one node $v \in X$ such that $c(v) = i$. An edge $e \in E$ is a *rainbow edge* if, for each $i \in [r]$, there is at least one $v \in e$ such that $c(v) = i$. A *no-rainbow r-coloring* of H is a surjective r-coloring of H such that H has no rainbow edge.

Given an r-uniform hypergraph $H = (X, E)$, the *no-rainbow r-coloring problem* (r-NRC) asks whether H has a no-rainbow r-coloring (5). r-NRC is clearly in NP, but it is unknown whether the problem is NP-complete (5).

Let $\mathcal{S}$ be a taxon coverage pattern for X. We associate with $\mathcal{S}$ a hypergraph $H(\mathcal{S}) = (X, \mathcal{S})$, and we associate with $Q_{\mathcal{S}}$ a 4-uniform hypergraph $H(Q_{\mathcal{S}}) = (X, Q_{\mathcal{S}})$. The next result states that r-NRC is equivalent to the complement of the decisiveness problem.

Proposition 2.3.1. *Let $\mathcal{S}$ be a taxon coverage pattern. The following statements are equivalent.*

1. *$\mathcal{S}$ is not decisive.*
2. *$H(Q_{\mathcal{S}})$ admits a no-rainbow 4-coloring.*
3. *$H(\mathcal{S})$ admits a no-rainbow 4-coloring.*

Proof. (1) $\Leftrightarrow$ (2): By Theorem 2.2.1, it suffices to show that $\mathcal{S}$ fails to satisfy the 4-way partition property if and only if $H(Q_{\mathcal{S}})$ has a no-rainbow 4-coloring. $\mathcal{S}$ does not satisfy the 4-way partition property if and only if there exists a partition A_1, A_2, A_3, A_4 of X such that, for every $q \in Q_{\mathcal{S}}$,

there is an $i \in [4]$ such that $A_i \cap q = \emptyset$. This holds if and only if the coloring c, where $c(v) = i$ if and only if $v \in A_i$, is a no-rainbow 4-coloring of $H(Q_S)$.

(2) $\Leftrightarrow$ (3): It is clear that if c is a no-rainbow 4-coloring of $H(\mathcal{S})$, then c is a no-rainbow 4-coloring of $H(Q_S)$. We now argue that if c is a no-rainbow 4-coloring of $H(Q_{\mathcal{S}})$, then c is a no-rainbow 4-coloring of $H(\mathcal{S})$. Suppose, to the contrary, that there a rainbow edge $Y \in \mathcal{S}$. Let q be any 4-tuple $\{v_1, v_2, v_3, v_4\} \subseteq Y$ such that $c(v_i) = i$, for each $i \in [4]$. Then, q is a rainbow edge in $Q_{\mathcal{S}}$, a contradiction. □

Theorem 2.3.2. *The decisiveness problem is co-NP-complete.*

Proof. In Corollary 2.2.2 we established that decisiveness is in co-NP. Completeness follows from the fact that the no-rainbow 3-coloring problem is NP-complete (31). □

Proposition 2.3.3. *Let $H = (X, E)$ be a hypergraph and r be a positive integer.*

(i) If H has at least r connected components, then H admits a no-rainbow r-coloring.

(ii) If $r = 2$, then H admits a no-rainbow r-coloring if and only if H is disconnected.

Proof. (i) Suppose the connected components of H are $C_1, \ldots, C_q$, where $q \geq r$. For each $i \in \{1, \ldots, r-1\}$, assign color i to all nodes in C_i. For $i = \{r, \ldots, q\}$, assign color r to all nodes in C_i. Thus, no edge is rainbow-colored.

(ii) By part (i), if H is disconnected, it admits a no-rainbow 2-coloring. To prove the other direction, assume, for contradiction that H admits a no-rainbow 2-coloring but it is connected. Pick any two nodes u and v such that $c(u) = 1$ and $c(v) = 2$. Since H is connected, there is a (u, v)-chain in H. But this chain must contain an edge with nodes of two different colors; i.e., a rainbow edge.

□

Part (ii) of Proposition 2.3.3 implies the following.

Corollary 2.3.4. *2-NRC $\in P$.*

Lemma 2.3.5. *Let $H = (X, E)$ be an r-uniform hypergraph. Suppose that there exists a subset A of X such that $2 \leq |A| \leq r - 1$ and $A \not\subseteq e$ for any $e \in E$. Then, H has a no-rainbow r-coloring.*

Proof. Let c be the coloring where each of the nodes in A is assigned a distinct color from the set $[|A|]$ and the remaining nodes are assigned colors from the set $\{|A| + 1, \ldots, r\}$. Then, c is a no-rainbow r-coloring of H. □

CHAPTER 3. BOUNDS FOR FINDING A NO-RAINBOW COLORING

3.1 Introduction

Assume we have a hypergraph $H = (X, E)$ with n nodes and we want to use exactly r colors to color the nodes in H, while avoiding a rainbow coloring. When we have only a few number of hyperedges it is easier to avoid having a rainbow coloring since it is easier to find some nodes which never appear in the same hyperedge. Also if we have a node which does not participate in sufficient number of hyperedges, it is easier to find some nodes which never appear together in a hyperedge. Both of these observations lead to lower bounds for the no-rainbow colroing problem. In this chapter we present two lower bounds for the number of hyperedges in a given hypergraph H and for degree of each node. Both lower bounds are tight and we prove any hypergraph with fewer number of edges or with a node with smaller degree as mentioned lower bound has a no-rainbow coloring.

3.2 Lower bounds

The next result captures the intuition that a taxon coverage pattern $\mathcal{S}$ for X is unlikely to be decisive if it is not dense enough (i.e., if there is not enough overlap among the sets in $\mathcal{S}$) or if there is a taxon $x \in X$ that is included in few sets in $\mathcal{S}$. Let us define the *degree* of a taxon $x \in X$ in a coverage pattern S, denoted $d_S(x)$, as the degree of node x in $H(Q_{\mathcal{S}})$.

Theorem 3.2.1. *Let $\mathcal{S}$ be a taxon coverage pattern for X and let $n = |X|$, $n \geq 4$. Then, $\mathcal{S}$ is* not *decisive if either*

(i) $|Q_{\mathcal{S}}| < \binom{n-1}{3}$ *or*

(ii) $\min_{x \in X} d_{\mathcal{S}}(x) < \binom{n-2}{2}$.

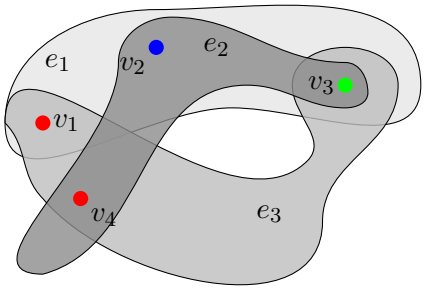

Figure 3.1 Example of non-existence of no-rainbow 3-coloring. If there were less number of edges or a node with any degree less than 2, then we could find a no-rainbow 3-coloring

Conditions (i) and (ii) do not suffice to guarantee non-decisiveness. Nevertheless, the bounds are tight. That is, there exist decisive taxon coverage patterns $\mathcal{S}$ for X such that either $|Q_{\mathcal{S}}| = \binom{n-1}{3}$ or $\min_{x \in X} d_{\mathcal{S}}(x) = \binom{n-2}{2}$.

By Proposition 2.3.1(ii), Theorem 3.2.1 is the special case of the following result for $r = 4$.

Theorem 3.2.2. *Let $H = (X, E)$ be an n-node r-uniform hypergraph with $n \geq r \geq 1$. Then H admits a no-rainbow r-coloring if either*

(i) $|E| < \binom{n-1}{r-1}$ *or*

(ii) $\min_{v \in X} d(v) < \binom{n-2}{r-2}$.

Conditions (i) and (ii) are not sufficient to guarantee no-rainbow r-colorability. Nevertheless, the bounds are tight. That is, there exist n-node r-uniform hypergraphs $H = (X, E)$ where either $|E| = \binom{n-1}{r-1}$ or $\min_{v \in X} d(v) < \binom{n-2}{r-2}$ such that every r-coloring of H has a rainbow edge.

Fig. 3.1 figure shows a 3-uniform hypergraph $H = (X, E)$ with 4 nodes and 3 hyperedges that illustrates Theorem 3.2.2. Here $|E| = 3 = \binom{3}{2} = \binom{n-1}{r-1}$ and $\min_{v \in X} d(v) = 2 = \binom{2}{1} = \binom{n-2}{r-2}$. We can verify by inspection that H does not admit a no-rainbow 3-coloring. However, if we remove any hyperedge or decrease the degree of any node, then every 3-coloring has a rainbow edge.

Proof of Theorem 3.2.2. Let us consider part (i) first.

For the base cases, consider $r = 1$ or $n = r$. Then, H has at most one hyperedge. If $|E| < 1 = \binom{n-1}{r-1}$, then H contains no hyperedges and H trivially admits a no-rainbow r-coloring. If $|E| = 1$, then any coloring of H that uses all r colors contains a rainbow edge.

Let us assume that for any i and j with $1 \leq i < n$ and $1 \leq j \leq r$, $\binom{i-1}{j}$ equals the minimum number of hyperedges an i-node, j-uniform hypergraph H that does not admit a no-rainbow r-coloring. We now prove the claim for $i = n$ and $j = r$.

Fix an arbitrary node $v \in X$. Any coloring c of H falls into one of two mutually disjoint classes, depending on whether

1. $c(v) \neq c(u)$ for any $u \in X \setminus \{v\}$ or

2. $c(v) = c(u)$ for some $u \in X \setminus \{v\}$.

For the colorings in class 1, we need hyperedges that contain node v, since in the absence of such hyperedges, any coloring is a no-rainbow coloring. Assume, without loss of generality, that $c(v) = r$. The question reduces to finding the number of hyperedges in an $(n-1)$-node $(r-1)$-uniform hypergraph (since v's color, r, is unique). The minimum number of hyperedges needed to avoid a no-rainbow $(r-1)$-coloring for an $(n-1)$-node hypergraph is $\binom{n-2}{r-2}$.

To find the minimum number of hyperedges needed to cover colorings of class 2, we ignore v, since v is assigned a color that is used by other nodes as well. The number of hyperedges needed for this class is $\binom{n-2}{r-1}$.

To obtain a lower bound, we add the lower obtained for the two disjoint classes of colorings. Thus, $\binom{n-2}{r-2} + \binom{n-2}{r-1} = \binom{n-1}{r-1}$.

Now, consider part (ii). Pick an arbitrary node $v \in X$. Consider the class of colorings of H in which $c(v) \neq c(u)$ for any $u \in X \setminus \{v\}$. As seen in the proof of part (i), H must have at least $\binom{n-2}{r-2}$ hyperedges that contain node v. The absence of any of these hyperedges leads to a no-rainbow r-coloring. □

Algorithm 3.1: Constructing a minimum no no-rainbow r-coloring

```
1  ConstructMin(n, r)
       Input: X = [n] and r colors
       Output: Set E where E = C(n-1, r-1) and H = (X, E) does not admit a no-rainbow coloring
2      E = ∅
3      if n < r then
4          E = ∅
5      end
6      else
7          if n = r then
8              E = E ∪ {{1, ··· , n}}
9          end
10         else
11             if r = 1 then
12                 E = E ∪ {{1}}
13             end
14             else
15                 E_1 = ConstructMin(n − 1, r)
16                 E_2 = ConstructMin(n − 1, r − 1)
17                 foreach e ∈ E_2 do
18                     e = e ∪ {n}
19                 end
20                 E = E_1 ∪ E_2
21             end
22         end
23     end
24     return E
```

3.3 Constructing a hypergraph with no no-rainbow coloring using minimum number of hyperedges

In 3.2 we showed a lower bound for the number of hyperedges for hypergraphs with no no-rainbow r-coloring. Here we use the same idea to construct a hypergraph with n nodes and $\binom{n-1}{r-1}$ hyperedges that does not admit a no-rainbow r-coloring for the given r.

Algorithm 3.1 shows how we can construct a set of hyperedges E, where $|E| = \binom{n-1}{r-1}$ and $H = (X, E)$, where $X = \{1, \cdots, n\}$ for which every surjective coloring has a rainbow edge.

Theorem 3.3.1. *Algorithm 3.1 constructs a set E where:*

1. $|E| = \binom{n-1}{r-1}$
2. *$H = (X, E)$ does not admit any no-rainbow r-coloring.*

Proof. First we prove 1. The proof is by induction on n and r. When $r = 1$, the algorithm returns $E = \{\{1\}\}$. Since $|E| = \binom{n-1}{r-1} = \binom{n-1}{0} = 1$ the algorithm gives the correct answer. Also when $n = r$ we have $E = \{\{1, \cdots, n\}\}$ and since $|E| = \binom{n-1}{r-1} = 1$ the algorithm gives the correct answer. Let us assume for any hypergraph with i nodes where $r \leq i < n$ and j colors where $1 \leq j \leq r$ the algorithm works correctly. Now we want to show the algorithm gives the correct answer for n nodes and r colors. Based on hypothesis $|E_1| = \binom{n-2}{r-1}$ and $E_2 = \binom{n-2}{r-2}$. Since $E = E_1 \cup E_2$, we know $|E| = \binom{n-2}{r-1} + \binom{n-2}{r-2} = \binom{n-1}{r-1}$ and so the statement is proved.

The proof of part 2 is easy since we followed the logic we used in the Theorem 3.2.1 in our construction. The base case is obvious for the cases $n = r$ and $r = 1$ since in these cases any edge is a rainbow. We assume for any hypergraph with i nodes where $r \leq i < n$ and j colors where $1 \leq j \leq r$ the algorithm finds E where there is no no-rainbow r-coloring. Now we want to prove the algorithm is correct when there are n nodes and r colors. Following the proof of Theorem 3.2.1 we consider two sets of hyperedges; the ones which cover node n and the ones which does not cover n. The hyperedges that cover n need to have $r - 1$ other nodes from the remaining $n - 1$ nodes, which can be found by `ConstructMin`$(n-1, r-1)$ and adding n to each of these sets since we know they definitely cover n. The hyperedges that does not cover n must have r nodes from $n - 1$ nodes since n is not part of our coverage. We can ignore n and find these edges by `ConstructMin`$(n-1, r)$. The union of these two sets gives us E.

□

Corollary 3.3.2. *Algorithm 3.1 constructs a minimum size hyperedge set E for H such that there does not exist any no-rainbow coloring for H. It takes polynomial time to generate each set.*

Proof. By Theorem 3.2.1 any hypergraph with $|E| < \binom{n-1}{r-1}$ has a no-rainbow coloring. In Theorem 3.3.1 we showed there is no no-rainbow coloring $H = (X, E)$ with the constructed E where

$|E| = \binom{n-1}{r-1}$. So we can claim the suggested E is a minimum size set of hyperedges such that $H = (X, E)$ does not admit a no-rainbow coloring. □

Observe that the base case of Algorithm 3.1 when $r = 1$ we construct an edge with node 1 while any edge with any node results in a graph which does not have any no-rainbow r-coloring. By changing this edge we may get other Es which satisfies our conditions. Thus the set of hyperedges E s.t $|E| = \binom{n-1}{r-1}$ and $H = (X, E)$ does not admit a no-rainbow r-coloring is not unique.

Theorem 3.3.3. *We can generate all the set E for $H = (X, E)$ such that $|E| = \binom{n-1}{r-1}$ and H does not have any no-rainbow r-coloring.*

Proof. We can change the base case of Algorithm 3.1 by applying a for loop to generate all the possible edges where $r = 1$. □

Example 3.3.4. *For a hypergraph H with $n = 5$ nodes and $r = 3$ colors if there are less than $\binom{4}{2} = 6$ 3-uniform hyperedges such that there exists a no-rainbow coloring for H.*

We apply Algorithm 3.1 to construct E. Since $n \neq r$ or $r \neq 1$, the algorithm calls itself to construct E_1 =`ConstructMin(4, 3)` *and E_2 =*`ConstructMin(4, 2)` *and add 5 to each set of E_2.*

To construct E_1 we need to call `ConstructMin(3, 3)` *which is equal to $\{\{1, 2, 3\}\}$ and* `ConstructMin(3, 2)` *and add the element 4 to each set of the latter one, so we have:*

$$\{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}\}$$

For E_2 we need to call `ConstructMin(3, 2)` *and* `ConstructMin(3, 1)` *and add 4 to each set in the latter one, so we have: $\{\{1, 2, 5\}, \{1, 3, 5\}, \{1, 4, 5\}\}$. Finally we have*

$$E = \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{1, 2, 5\}, \{1, 3, 5\}, \{1, 4, 5\}\}$$

CHAPTER 4. EXACT AND FIXED-PARAMETER TRACTABLE ALGORITHMS

4.1 Abstract

In this chapter we present an exact algorithm and one fixed-parameter tractable algorithm to solve no-rainbow r-coloring problem. The exact algorithm suggested in 4.2.1 uses the idea of fixing a subset of nodes with one color and continuing the process for the uncolored nodes. In section 4.3 we suggest a fixed-parameter tractable algorithm for the decisiveness problem. A *reduction rule* for the decisiveness problem is a rule that replaces an instance $\mathcal{S}$ of the problem by a smaller instance $\widetilde{\mathcal{S}}$ such that $\mathcal{S}$ is decisive if and only if $\widetilde{\mathcal{S}}$ is.

4.2 Introduction

The naïve way to use Theorem 2.2.1 to test whether a coverage pattern $\mathcal{S}$ is decisive is to enumerate all partitions of X into four non-empty sets A_1, A_2, A_3, A_4 and verify that there is a set $Y \in \mathcal{S}$ that intersects each A_i. Equivalently, by Proposition 2.3.1, we can enumerate all surjective colorings of $H(S)$ and check if each of these colorings yields a rainbow edge. In either approach, the number of options to consider is given by a Stirling number of the second kind, namely $\left\{ {n \atop 4} \right\} \sim \frac{4^n}{4!}$ (9). The next result is a substantial improvement over the naïve approach.

Theorem 4.2.1. *Let $\mathcal{S}$ be a taxon coverage pattern for a taxon set X. Then, there is an algorithm that, in $O^*(2.8^n)$ time[1] determines whether or not $\mathcal{S}$ is decisive.*

We prove Theorem 4.2.1 by showing that there exists an algorithm that, given a 4-uniform hypergraph $H = (X, E)$, determines if H has a no-rainbow 4-coloring in time $O^*(2.8^n)$. Before proving this, we consider the problem of determining whether a 3-uniform hypergraph has a

[1] *The O^*-notation is a variant of O-notation that ignores polynomial factors (8).*

no-rainbow 3- coloring. In addition to serving as an introduction to the ideas behind the no-rainbow 4-coloring algorithm, the no-rainbow 3- coloring algorithm is needed in the FPT algorithm for decisiveness described in Section 4.3.

4.2.1 No-Rainbow 3-Colorings

We prove the following.

Lemma 4.2.2. *There exists an algorithm that, given a 3-uniform hypergraph $H = (X, E)$ determines if H has a no-rainbow 3-coloring in time $O^*(1.7^n)$.*

Proof. We claim that algorithm `Find3NRC` (Algorithm 4.1) solves 3-NRC in $O^*(1.89^n)$ time. `Find3NRC` relies on the observation that if H has a no-rainbow 3-coloring c, then there must exist a subset $A \subseteq X$ where $|A| \leq \lfloor \frac{n}{3} \rfloor$, such that all nodes in A have the same color, which is different from the colors used for $X \setminus A$. `Find3NRC` tries all possible choices of A and, without loss of generality, assigns $c(v) = 1$, for all $v \in A$. We are now left with the problem of determining whether we can assign colors 2 and 3 to the nodes in $X \setminus (A \cup B)$ to obtain a no-rainbow 3-coloring for H.

Let c be the current coloring of H. For each $e \in E$ and each $i \in [3]$, $m_e^c(i)$ denotes the number of nodes $v \in e$ such that $c(v) = i$. Consider the situation after `Find3NRC` assigns colors 1 to the nodes in A. There are two cases, both of which can be handled in polynomial time.

1. *There is no $e \in E$ such that $m_e^c(1) = 1$.* Then, if we partition the nodes of $X \setminus A$, arbitrarily into subsets B and C and assign $c(v) = 2$ for each $v \in B$ and $c(v) = 4$ for each $v \in C$, we obtain a no-rainbow 3-coloring of H.

2. *There exists $e \in E$ such that $m_e^c(1) = 1$.* Let e be any such edge. Then e must contain exactly two uncolored nodes, x and y. To avoid e becoming a rainbow edge, we must set $c(x) = c(y) \neq 1$. Without loss of generality, make $c(x) = c(y) = 2$. Next, as long as there exists any hyperedge e such that $m_e^c(i) = 1$ for each $i \in [2]$, the (unique) uncolored node x in

e must be assigned $c(x) = 2$, because setting $c(x) = 3$ would make e a rainbow hyperedge. Once no such hyperedges remain, we have two possibilities:

(a) *X does not contain uncolored nodes.* Then, there does not exist a no-rainbow 3-coloring, given the current choice of A.

(b) *X contains uncolored nodes.* Then, there is no $e \in E$ such that $m_e^c(i) = 1$ for each $i \in [2]$. Thus, if we set $c(u) = 3$ for each uncolored node u, we obtain a no-rainbow 3-coloring for H.

The total number of sets A considered throughout the execution of `FindNRC` is at most $\sum_{i=1}^{\lfloor \frac{n}{3} \rfloor} \binom{n}{i} = O(2^{h(1/3)n})$, where $h(\delta) = -\delta \log_2(\delta) - (1-\delta)\log_2(1-\delta)$ is the binary entropy function (20). Thus, the number of sets considered is $O(1.89^n)$. The time spent per set A is polynomial in n; hence, the total running time of `Find3NRC` is $O^*(1.89^n)$. □

4.2.2 No-Rainbow 4-colorings

The proof of Theorem 4.2.1 relies on the following result.

Lemma 4.2.3. *There exists an algorithm that, given a 4-uniform hypergraph $H = (X, E)$, determines if H has a no-rainbow 4-coloring in time $O^*(2.8^n)$.*

Proof. We claim that algorithm `FindNRC` (Algorithm 4.2) solves 4-NRC in $O^*(2.8^n)$ time. `FindNRC` relies on the observation that if H has a no-rainbow 4-coloring c, then (1) there must exist a subset $A \subseteq X$ where $|A| \leq \lfloor \frac{n}{4} \rfloor$, such that all nodes in A have the same color, which is different from the colors used for $X \setminus A$, and (ii) there must exist a subset $B \subseteq X \setminus A$, where $|B| \leq \lfloor \frac{n-|A|}{3} \rfloor$, such that all nodes in B have the same color, which is different from the colors used for the nodes in $X \setminus B$. `FindNRC` tries all possible choices of A and B and, without loss of generality, assigns $c(v) = 1$, for all $v \in A$ and $c(v) = 2$, for all $v \in B$. We are now left with the problem of determining whether we can assign colors 3 and 4 to the nodes in $X \setminus (A \cup B)$ to obtain a no-rainbow 4-coloring for H.

Let c be the current coloring of H. For each $e \in E$ and each $i \in [4]$, $m_e^c(i)$ denotes the number of nodes $v \in e$ such that $c(v) = i$. Consider the situation after `FindNRC` assigns colors 1 and 2 to the nodes in A and B. There are two cases, both of which can be handled in polynomial time.

1. *There is no $e \in E$, such that, for each $i \in [2]$, $m_e^c = 1$.* Then, if we partition the nodes of $X \setminus (A \cup B)$, arbitrarily into subsets C and D and assign $c(v) = 3$ for each $v \in C$ and $c(v) = 4$ for each $v \in D$, we obtain a no-rainbow 4-coloring of H.

2. *There exists $e \in E$, such that, for each $i \in [2]$, $m_e^c = 1$.* Let e be any such edge. Then e must exactly contain two uncolored nodes, x and y. To avoid e becoming a rainbow edge, we must set $c(x) = c(y) \notin [2]$. Without loss of generality, make $c(x) = c(y) = 3$. Next, as long as there exists any hyperedge e such that $m_e^c(i) = 1$ for each $i \in [3]$, the (unique) uncolored node x in e must be assigned $c(x) = 3$, because setting $c(x) = 4$ would make e a rainbow hyperedge. Once no such hyperedges remain, we have two possibilities:

   (a) *X does not contain uncolored nodes.* Then, there does not exist a no-rainbow 4-coloring, given the current choice of A and B.

   (b) *X contains uncolored nodes.* Then, there is no $e \in E$ such that $m_e^c(i) = 1$ for each $i \in [3]$. Thus, if we set $c(u) = 4$ for each uncolored node u, we obtain a no-rainbow 4-coloring for H.

The total number of pairs (A, B) considered throughout the execution of `FindNRC` is at most $\sum_{i=1}^{\lfloor \frac{n}{4} \rfloor} \binom{n}{i} \sum_{j=1}^{\lfloor \frac{n-i}{3} \rfloor} \binom{n-i}{j}$. We have estimated this sum numerically to be $O(2.8^n)$. The time spent per pair (A, B) is polynomial in n; hence, the total running time of `FindNRC` is $O^*(2.8^n)$.

□

Proof of Theorem 4.2.1. Given $\mathcal{S}$, we construct the hypergraph $H(Q_{\mathcal{S}})$, which takes time polynomial in n, and then run `FindNRC(`$H(Q_{\mathcal{S}})$`)`, which, by Lemma 4.2.3, takes $O^*(2.8^n)$ time. If the algorithm returns a no-rainbow 4-coloring c of $H(Q_{\mathcal{S}})$, then, by Proposition 2.3.1, $\mathcal{S}$ is not decisive; if `FindNRC(`$H(Q_{\mathcal{S}})$`)` returns `fail`, then $\mathcal{S}$ is decisive. □

4.3 Reduction Rules and Fixed Parameter Tractability

4.3.1 Reduction Rules

A *reduction rule* for the decisiveness problem is a rule that replaces an instance $\mathcal{S}$ of the problem by a smaller instance $\widetilde{\mathcal{S}}$ such that $\mathcal{S}$ is decisive if and only if $\widetilde{\mathcal{S}}$ is. Here we present reduction rules that can reduce an instance of the decisiveness problem into a one whose size depends only on k. This size reduction is especially significant for taxon coverage patterns where the number of loci, k, is small relative to the number of taxa. Such inputs are not uncommon in the literature — examples of such data sets are studied in Section 6.3.

We need to introduce some definitions and notation. Let $H = (X, E)$ be a hypergraph where $X = \{x_1, x_2, \ldots, x_n\}$ and $E = \{e_1, e_2, \ldots, e_k\}$. The *incidence matrix* of H is the $n \times k$ binary matrix where $M_H[i, j] = 1$ if $x_i \in e_j$ and $M_H[i, j] = 0$ otherwise. Two rows in M_H are *copies* if the rows are identical when viewed as 0-1 strings; otherwise, they are *distinct*.

Let $\widetilde{M}_H$ denote the matrix obtained from M_H by striking out duplicate rows, so that $\widetilde{M}_H$ retains only one copy of each row in M_H. Let $\widetilde{n}$ denote the number of rows of $\widetilde{M}_H$. Then, $\widetilde{n} \leq 2^k$. $\widetilde{M}$ is the incidence matrix of a hypergraph $\widetilde{H} = (\widetilde{X}, \widetilde{E})$, where $\widetilde{X} \subseteq X$, and each $v \in \widetilde{X}$ corresponds to a distinct row of M_H. For each $v \in X$, $X(v) \subseteq X$ consists of all nodes $u \in X$ that correspond to copies of the row of M_H corresponding to v.

Given two binary strings s_1 and s_2 of length k, $s_1 \,\&\, s_2$ denotes the bitwise *and* of s_1 and s_2; $\mathbf{0}$ denotes the all-zeroes string of length k.

The next result is a direct consequence of Lemma 2.3.5.

Proposition 4.3.1. *If $\widetilde{M}_H$ has two rows r_1 and r_2 such that $r_1 \,\&\, r_2 = \mathbf{0}$ or three rows r_1, r_2 and r_3 such that $r_1 \,\&\, r_2 \,\&\, r_3 = \mathbf{0}$, then $\widetilde{H}$ and H admit no-rainbow 4-colorings.*

Corollary 4.3.2. *If $\widetilde{M}_H$ has more than 2^{k-1} rows, where k is the number of columns, then H admits a no-rainbow 4-coloring.*

Proof. Suppose $\widetilde{n} \geq 2^{k-1}$. Then, there are at least two rows r_1 and r_2 in $\widetilde{M}_H$ that are complements of each other (that is, r_2 is is obtained by negating each bit in r_1) and, thus, $r_1 \,\&\, r_2 = \mathbf{0}$. The claim now follows from Proposition 4.3.1. □

4.3.2 A Fixed-Parameter Algorithm for Decisiveness

We first present an auxiliary result.

Theorem 4.3.3. *Suppose $n \geq \widetilde{n} + 2$. H admits a no-rainbow 4-coloring if and only if $\widetilde{H}$ admits a no-rainbow r-coloring for some $r \in \{2, 3, 4\}$(17).*

Proof. (*If*) Suppose $\widetilde{H} = (\widetilde{X}, \widetilde{E})$ admits a no-rainbow r-coloring $\widetilde{c}$ for some $r \in \{2, 3, 4\}$. Let c be the coloring for H obtained by setting $c(u) = \widetilde{c}(v)$, for each $v \in \widetilde{X}$ and each $u \in X(v)$. If $\widetilde{c}$ is a no-rainbow 4-coloring of $\widetilde{H}$, then c is also a no-rainbow 4-coloring of H, and we are done.

Suppose $\widetilde{c}$ is a no-rainbow 3-coloring. Since $n \geq \widetilde{n} + 2$, there must exist $v \in \widetilde{X}$ such that $|X(v)| \geq 2$. We choose one node $u \in X(v) \setminus \{v\}$, and set $c(u) = 4$, making c a no-rainbow 4-coloring for H.

Suppose $\widetilde{c}$ is a no-rainbow 2-coloring. If there exists $v \in \widetilde{X} \setminus \{v\}$ such that $|X(v)| \geq 3$, we pick any $u, w \in \widetilde{X} \setminus \{v\}$, and set $c(u) = 3$ and $c(w) = 4$. If there is no $v \in \widetilde{X}$ such that $|X(v)| \geq 3$, there must exist $v_1, v_2 \in \widetilde{X}$ such that $|X(v_i)| \geq 2$ for $i \in \{1, 2\}$. For $i \in \{1, 2\}$, choose any $u_i \in X(v_i) \setminus \{v_i\}$ and set $c(u_i) = i + 2$.

(*Only if*) Suppose $H = (X, E)$ has a no-rainbow 4-coloring c. We show that $\widetilde{H} = (\widetilde{X}, \widetilde{E})$ admits a no-rainbow r-coloring for some $r \in \{2, 3, 4\}$.

Let $\widetilde{c}$ be the coloring of $\widetilde{H}$ where, for each $v \in \widetilde{X}$, $\widetilde{c}(v) = c(u)$, for some arbitrarily chosen node in $u \in X(v)$. If $\widetilde{c}$ is a 4-coloring of $\widetilde{H}$, $\widetilde{c}$ must be a no-rainbow 4-coloring of $\widetilde{H}$, and we are done. If $\widetilde{c}$ is a no-rainbow r-coloring for some $r \in \{2, 3, 4\}$, we are also done. Suppose that, instead, for some $r \in \{1, 2, 3\}$, $\widetilde{c}$ is an r-coloring of $\widetilde{H}$ with a rainbow edge. We argue that $\widetilde{c}$ can be transformed into a no-rainbow r-coloring of $\widetilde{H}$ for some $r \in \{2, 3, 4\}$. There are three cases.

Case 1: $\widetilde{c}$ is a 3-coloring of $\widetilde{H}$ that has a rainbow edge. We can transform $\widetilde{c}$ into a no-rainbow 4-coloring $\widehat{c}$ of $\widetilde{H}$ as follows. Consider three nodes $v_1, v_2, v_3 \in \widetilde{X}$ that are all neighbors and have

distinct colors. Since $\widetilde{X} \subseteq X$, v_1, v_2, v_3 are neighbors in H as well. Since H admits a no-rainbow 4-coloring, there must exist a node $v_4 \in X \setminus \{v_1, v_2, v_3\}$ such that $c(v_4) \neq c(v_i)$, for $i \in \{1, 2, 3\}$. Since $v_4 \notin \widetilde{X}$ there must be a copy of v_4 with a different color in $\widetilde{X}$. If we insert v_4 to $\widetilde{X}$ and remove its copy from $\widetilde{X}$, then $\widetilde{X}$ has 4 colors and the new coloring is a no-rainbow 4-coloring.

Case 2: $\widetilde{c}$ is a 2-coloring of $\widetilde{H}$, that has a rainbow edge. We can transform $\widetilde{c}$ into a no-rainbow r-coloring $\widehat{c}$ of $\widetilde{H}$, for $r \in \{3, 4\}$ as follows. Consider any two neighbor nodes $v_1, v_2 \in \widetilde{X}$ such that v_1 and v_2 have distinct colors. Note that v_1 and v_2 must be neighbors in H as well. Since H admits no-rainbow 4-coloring, there must be at least one other node $v_3 \in X$ such that $v_3 \neq v_i$, for $i \in \{1, 2\}$ and $c(v_3) \neq c(v_i)$, for $i \in \{1, 2\}$. Since $v_3 \notin \widetilde{X}$ there must be a copy of v_3 with another color in $\widetilde{H}$. If we insert v_3 with color $c(v_3)$ to $\widetilde{H}$ and remove its copy from $\widetilde{H}$, then $\widetilde{H}$ has 3 colors. If the new coloring is a no-rainbow 3-coloring we are done, otherwise the new coloring is a 3-coloring with a rainbow edge and we are in Case 1.

Case 3: $\widetilde{c}$ is a 1-coloring of $\widetilde{H}$. Then, we can transform $\widetilde{c}$ into a no-rainbow r-coloring $\widehat{c}$ of $\widetilde{H}$, for $r \in \{2, 3, 4\}$ as follows. Since H admits no-rainbow 4-coloring, we can replace one of the nodes of $\widetilde{H}$ with a copy of that in H of different color. The result is a 2-coloring for $\widetilde{H}$. If the result is a no-rainbow 2-coloring we are done. Otherwise, we have a 2-coloring of $\widetilde{H}$ with a rainbow edge and we can use Case 2 to find a no-rainbow r-coloring for $r \in \{3, 4\}$. □

Theorem 4.3.4. *Decisiveness is fixed-parameter tractable in k.*

Proof. Let $\mathcal{S}$ be the input coverage pattern. First, in $O^*(2^k)$ time, we construct $\widetilde{H}(S)$. By Theorem 2.2.3, we need to test if $\widetilde{H}(\mathcal{S})$ admits a no-rainbow r-coloring for any $r \in \{2, 3, 4\}$. If the answer is "yes" for any such r, then $\mathcal{S}$ is not decisive; otherwise $\mathcal{S}$ is decisive. By Corollary 2.3.4, the test for $r = 2$ takes polynomial time. We perform the steps for $r = 3$ and $r = 4$ using the algorithm of Section **??**. The total time is $O^*(2.8^{\widetilde{n}})$, which is $O^*(2.8^{2^k})$. □

Algorithm 4.1: No-rainbow 3-coloring of H

```
1  Find3NRC(H)
       Input: A 3-uniform hypergraph H = (X, E) such that |X| ≥ 3.
       Output: A no-rainbow 3-coloring of H, if one exists; otherwise, fail.
2      for i = 1 to ⌊n/3⌋ do
3          foreach v ∈ X do c(v) = uncolored
4          foreach A ⊆ X such that |A| = i do
5              foreach v ∈ A do c(v) = 1
6              if there is no e ∈ E such that m_e^c(1) = 1 then
7                  Arbitrarily partition X \ A into nonempty sets B, C
8                  foreach v ∈ B do c(v) = 2
9                  foreach v ∈ C do c(v) = 3
10                 return c
11             end
12             else
13                 Choose any e ∈ E such that m_e^c(1) = 1
14                 foreach uncolored node x ∈ e do
15                     c(x) = 2
16                 end
17                 while there exists e ∈ E s.t. m_e^c(i) = 1 for each i ∈ [2] do
18                     Pick any e ∈ E s.t. m_e^c(i) = 1 for each i ∈ [2]
19                     Let x be the unique uncolored node in e
20                     c(x) = 2
21                 end
22                 if X contains no uncolored node then
23                     return fail
24                 end
25                 else
26                     foreach uncolored node u ∈ X do
27                         c(u) = 3
28                     end
29                     return c
30                 end
31             end
32         end
33     end
34     return fail
```

Algorithm 4.2: No-rainbow 4-coloring of H

```
1  FindNRC(H)
       Input: A 4-uniform hypergraph H = (X, E) such that |X| ≥ 4.
       Output: A no-rainbow 4-coloring of H, if one exists; otherwise, fail.
2      for i = 1 to ⌊n/4⌋ do
3          foreach v ∈ X do c(v) = uncolored
4          foreach A ⊆ X such that |A| = i do
5              foreach v ∈ A do c(v) = 1
6              for j = 1 to ⌊(n−i)/3⌋ do
7                  foreach B ⊆ X \ A such that |B| = j do
8                      foreach v ∈ B do c(v) = 2
9                      if there is no e ∈ E such that, for each i ∈ [2], m_e^c(i) = 1 then
10                         Arbitrarily partition X \ (A ∪ B) into nonempty sets C, D
11                         foreach v ∈ C do c(v) = 3
12                         foreach v ∈ D do c(v) = 4
13                         return c
14                     end
15                     else
16                         Choose any e ∈ E such that m_e^c(i) = 1 for each i ∈ [2]
17                         foreach uncolored node x ∈ e do c(x) = 3
18                         while there exists e ∈ E s.t. m_e^c(i) = 1 for each i ∈ [3] do
19                             Pick any e ∈ E s.t. m_e^c(i) = 1 for each i ∈ [3]
20                             Let x be the unique uncolored node in e
21                             c(x) = 3
22                         end
23                         if X contains no uncolored node then return fail
24                         else
25                             foreach uncolored node u ∈ X do
26                                 c(u) = 4
27                             end
28                             return c
29                         end
30                     end
31                 end
32             end
33         end
34     end
35     return fail
```

CHAPTER 5. LOCAL SEARCH

5.1 Local Search

Let us assume we are given an r-uniform hypergraph $H = (X, E)$ where $|X| = n$. Our goal is to find a no-rainbow r-coloring for H. In this chapter, we use local search in combination with some heuristic methods to find out if a no-rainbow coloring exists for H. The idea of applying local search to this problem comes from Schöning, Moser and Scheder (20) who applied randomized algorithms and local search methods to solve $K - SAT$ problem.

Local search is a heuristic method for solving algorithmic problems. The idea of local search is to start with a candidate solution and use some rules to find a better solution. The candidate solution can be picked randomly, or we can choose one we think is within a reasonable distance from an optimum solution. We are not likely to find the best solution at the first step, so we need to find a way to do a local search to find a better coloring in the neighborhood. For example, we find a rainbow edge and try to make it no-rainbow. Although our move may negatively affect the output of the other hyperedges and change their status from no-rainbow to rainbow, we hope that applying the local search makes the current situation closer to a no-rainbow coloring.

In our problem, the solution candidate is an initial surjective coloring for the given hypergraph $H = (X, E)$.

If the initial candidate is a no-rainbow r-coloring, we are done, but if it is not a no-rainbow coloring, we need to find a way to get closer to the desired solution, which is a no-rainbow r-coloring.

How can we realize how close we are to a no-rainbow coloring? We can use different measures. For example, we can find the number of rainbow hyperedges and reduce the number of rainbow

hyperedges in each step. Here we use Hamming Distance as our measure. In our problem, the Hamming distance between two colorings c and c' equals the number of nodes $x \in X$ such that $c(x) \neq c'(x)$. We denote Hamming distance between two colorings c and c' by $d(c, c')$.

If c is a no-rainbow coloring, the region in which we are looking for a no-rainbow coloring is the set of all colorings c' within some specified Hamming distance from c; i.e.

$$Ham_g(c) = \{\tilde{c} | d(\tilde{c}, c) \leq g\}$$

which is called the *Hamming Ball* around the coloring c with radius g.

We shall consider deterministic and randomized algorithms. If there exists a no-rainbow coloring in the given graph H, a deterministic algorithm will always find it and if there does not exist any no-rainbow coloring, it gives a firm *no* as output.

If a randomized algorithm finds a no-rainbow coloring we can be sure that the coloring is indeed a no-rainbow r-coloring. However, if the algorithm says *no*, there is a probability that the algorithm failed to find an existing no-rainbow coloring. Let p be the probability that a randomized algorithm finds the correct answer. Although p is a small number, we can boost the probability of success by repeating the algorithm.

Lemma 5.1.1. *Suppose we have a randomized algorithm with some small probability p of success. Then $O(\frac{1}{p} \log n)$ repetitions of the algorithm are sufficient so that we have probability greater than or equal to $1 - \frac{1}{n^{10}}$ that at least one of the runs was successful.*

Proof. We use the fact that $1 - p \geq e^{-p}$ for any real number p. Now, suppose we repeat the algorithm t times. The probability that all repetitions fail equals $(1-p)^t \leq e^{-pt}$. If we plug in $t = \frac{10}{p} \ln n$ we get n^{-10}. So, the probability that at least one of the repetitions succeeds is at least $1 - n^{-10}$ □

It is clear that we prefer a deterministic algorithm to make sure our solution has no errors. Of course, we cannot ignore the importance of speed in the algorithms and it is the reason a randomized algorithm with a good running time can be useful.

In sections 5.3, 5.5 and 5.7 we offer randomized algorithms, while in sections 5.4, 5.6 and 5.8 we offer deterministic algorithms.

5.2 Overview

We start section 5.3 by picking a random initial coloring and do a random walk to see if we can find a no-rainbow coloring from the starting point. In each step we find a rainbow hyperedge and change the color of one of its nodes. The result is a randomized algorithm that does not guarantee finding a no-rainbow coloring in the case there exists one. The complexity for this algorithm is better than naive algorithm, but worse than Algorithm 4.2.

In section 5.4 we improve our algorithm by using a better initial candidate. The suggested initial candidate helps us to derive a deterministic algorithm that does not search the whole space. The resulting algorithm is slower than the algorithm in section 5.3, but it has the advantage of being deterministic.

In section 5.5, we suggest using more but smaller Hamming balls. Thus, we start our search with t random initial candidates and do a local search with smaller Hamming balls for each of them. The result is a randomized algorithm with better running time than the ones in sections 5.3 and 5.4.

In section 5.6 we combine the results of sections 5.4 and 5.5. We design a deterministic algorithm that uses several initial candidates and smaller Hamming balls. In order to generate the initial candidates we use *covering codes*. We also present an algorithm that freezes the color of each node after changing it. The result is a deterministic algorithm with better complexity compared to the previous sections.

In section 5.7 we apply the idea of freezing the color of one node in each step on the algorithm in 5.3. The result is a randomized algorithm with a better complexity than the algorithms mentioned in the other sections.

Finally, in section 5.8 we combine the idea of using one non-random initial candidate, first introduced in section 5.4, with the idea of freezing the color of one node in each step. The result is a simple deterministic algorithm with the same complexity as the algorithm presented in 5.6.

5.3 Randomized Algorithm

The search space for the no-rainbow coloring problem with n nodes and r colors has size r^n. Thus an exhaustive search for a no-rainbow coloring takes $\Omega(r^n)$.

Since we need to use all r colors, we can ignore the colorings that use $r-1$ or fewer colors. Then we can be more precise with search space size and say it is equal to $\left\{ {n \atop r} \right\}$ where $\left\{ {n \atop r} \right\} = \frac{1}{r!}\Sigma_{i=0}^{r}\binom{r}{i}(r-i)^n$ is the Stirling number of the second kind. The search area is too large for exhaustive search to be efficient.

In this section we present a straightforward algorithm that finds a no-rainbow coloring with probability p. We can use lemma 5.1.1 to boost the success probability.

Now we show an algorithm that uses Schöning's idea for $K-SAT$ problem. In this algorithm, we start with an initial coloring, and while the coloring is rainbow, we pick a random rainbow hyperedge and randomly change the color of one of its r nodes. We repeat this process n times.

Algorithm 5.1: `Randomized No-rainbow Coloring`

```
1 Pick a random initial coloring c̃
2 while There is at least one rainbow hyperedge and have done this at most n times: do
3 |   Pick an arbitrary rainbow hyperedge
4 |   Change the color of one of the vertices to another color.
5 end
```

Consider some arbitrary no-rainbow coloring c, and let A_k be the event that the initial assignment $\tilde{c}$ disagrees with c on exactly k vertices. Note that:

$$\Pr[A_k] = \binom{n}{k} 2^k 3^{-n}$$

The probability p that we succeed is at least the probability that in Step 2 we make a beeline towards c (reducing Hamming distance by 1 in each step until we reach a no-rainbow coloring). This means for the case of $r = 3$ we have:

$$\begin{aligned} p &\geq \Sigma_{k=0}^{n} \binom{n}{k} 2^k 3^{-n} \left(\frac{1}{6}\right)^k \\ &= 3^{-n} \Sigma_{k=0}^{n} \binom{n}{k} \left(\frac{1}{3}\right)^k \\ &= 3^{-n} \left(1 + \frac{1}{3}\right)^n \\ &= \left(\frac{4}{9}\right)^n. \end{aligned}$$

Putting this together with lemma 5.1.1 and considering the fact that each repetition runs in polynomial time, the total computation time is: $poly(n) \times (\frac{9}{4})^n = O^*(2.25^n)$.

For the case of $r = 4$ we have:

$$\begin{aligned} p &\geq \Sigma_{k=0}^{n} \binom{n}{k} 3^k 4^{-n} \left(\frac{1}{12}\right)^k \\ &= 4^{-n} \Sigma_{k=0}^{n} \binom{n}{k} \left(\frac{1}{4}\right)^k \\ &= 4^{-n} \left(1 + \frac{1}{4}\right)^n \\ &= \left(\frac{5}{16}\right)^n. \end{aligned}$$

Thus, the complexity of Algorithm 5.1 for $r = 4$ is $poly(n) \times (\frac{16}{5})^n = O^*(3.2^n)$.

More general, the complexity of Algorithm 5.1 for r is $poly(n) \times (\frac{r^2}{r+1})^n = O^*((\frac{r^2}{r+1})^n)$.

We improve this result in the next sections.

5.4 Deterministic Local Search

Here we start with a non-random solution candidate and suggest a better way to restrict the search area and find a no-rainbow coloring if one exists.

Suppose our initial coloring candidate $\tilde{c}$ assigns r different colors to r different nodes and assigns color 1 to the remaining $n - r + 1$ nodes.

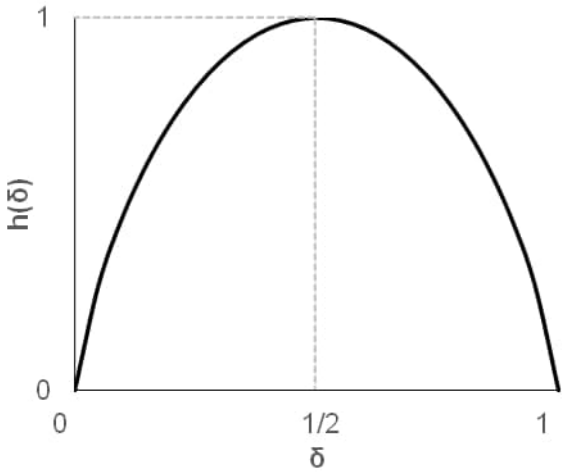

Figure 5.1 Shannon's binary enthropy function $h(\delta)$

Lemma 5.4.1. *If there exists a no-rainbow coloring c for hypergraph H, there exists at least one no-rainbow coloring c' where $d(c', \tilde{c}) \leq \frac{(r-1)n}{r} + \epsilon$.*

Proof. If c is a no-rainbow surjective coloring, it uses all r colors. By the pigeonhole principle, one of these colors is used at least $\frac{n}{r}$ times. Assume this color is b. If $b = 1$, we are done. Otherwise, we switch color 1 and b in c and we call the result c'. It is obvious that we are allowed to do this because of symmetry, and that c' is a no-rainbow coloring which satisfies $d(c', \tilde{c}) \leq (r-1)n/r + \epsilon$ where $\epsilon \leq \frac{r}{n}$. □

We start with the solution candidate mentioned in Lemma 5.4.1. We call this coloring $\tilde{c}$. Let c be a no-rainbow r-coloring. We show that the Hamming Distance between c and $\tilde{c}$ by $g = \delta n$ where δ denotes the maximum fraction of the nodes that have different color than c. In the case of $r = 3$, $\delta = 2/3$.

The number of colorings that differ from c in at most δn nodes, (i.e. $d(c, \tilde{c}) \leq \delta n$) is $\sum_{i=0}^{\delta n} \binom{n}{i}(r-1)^i$. This number behaves asymptotically like $2^{h(\delta)\cdot n}$ where $h(\delta) = -\delta \log_2(\delta) - (1-\delta)\log_2(1-\delta)$ is Shannon's binary entropy function (20). Figure 5.4 shows the graph of $h(\delta)$.

Algorithm 5.2: Deterministic Local Search Algorithm

```
1  localSearch(H, c̃, g)
2      if c̃ is a no-rainbow coloring then
3          return 1
4      end
5      Let e ∈ H be a hyperedge with rainbow color {c_1, c_2, c_3}
6      for i = 1 to 3 do
7          Change the color of c_i in c̃ to any other two colors and apply localSearch (H, c̃, g − 1)
8          if localSearch(H, c̃, g − 1) for any of two modified c̃s then
9              return 1
10         end
11     end
12 return 0
```

Algorithm 5.2 shows how local search works on the initial coloring to find a no-rainbow coloring for $r = 3$.

If $\tilde{c}$ is not a no-rainbow coloring, there exists at least one rainbow hyperedge. The idea of the `localSearch` algorithm is to find a rainbow hyperedge e and change the color of each of its 3 nodes to any of the other 2 colors. Changing the color of any of the nodes is guaranteed to make e a no-rainbow hyperedge; however, it does not mean that any change we apply moves the coloring toward the correct direction, since we may change the color of some node which has the correct color or increase the number of rainbow edges.

We know for one of the six new colorings, $d(c, \tilde{c})$ decreases by 1. If there exists a no-rainbow coloring, it is guaranteed to be found by this algorithm. The running time of algorithm 5.2 is $T(g) = 6T(g-1) = O(6^g) = O(6^{\frac{2}{3}n}) = O(3.3^n)$.

Applying the same algorithm to find a no-rainbow 4-coloring will result in a Hamming Ball with radius $g = \frac{3}{4}n$ and so $T(g) = 12T(g-1) = O(12^{\frac{3}{4}n}) = O(6.4^n)$.

Algorithm 5.2 is worse than the naive algorithm. Nevertheless, as shown in the next sections the idea of this procedure can be employed to find better algorithms.

5.5 Random Initial Coloring

The algorithm presented in section 5.4 is deterministic, but the Hamming balls are big, and so is the search region. We now show that it is possible to apply the same algorithm on several initial colorings but using smaller Hamming balls. In this section we choose the initial candidates randomly.

The idea shown in Algorithm 5.3 is to repeat the search for t times on Hamming balls with radius $g = \delta n$ to cover the whole search space.

Algorithm 5.3: `Random Local Search Algorithm`

```
1 for i = 1 to t do
2 |   Choose a random assignment c̃
3 |   if localSearch(H, c̃, g) then
4 |   |   return "There exists a no-rainbow coloring"
5 |   end
6 end
7 return "There is no no-rainbow coloring"
```

We want to find the optimal choices for t and g to cover the whole search space. We miss an existing no-rainbow coloring when there is no no-rainbow coloring in the Hamming ball of any of the t random initial colorings. We can compute this probability as follows:

$$
\begin{aligned}
\Pr(\forall i \le t : d(C_i(H), c) > \delta n) &= \prod_{i=1}^{t} \Pr(d(C_i(H), c) > \delta n) \\
&= \prod_{i=1}^{t} (1 - \Pr(d(C_i(H), c) \le \delta n)) \\
&= \prod_{i=1}^{t} \left(1 - \frac{\sum_{j=i}^{\delta n} \binom{n}{j} 2^j}{3^n}\right) \\
&\le \left(1 - \frac{\sum_{j=i}^{\delta n} \binom{n}{j} 2^j}{3^n}\right)^t \\
&\le e^{-t.\frac{\sum j=i^{\delta n}\binom{n}{j}2^j}{3^n}}, && \text{since } 1 - x \le e^{-x} \\
&= e^{-c}, && \text{when } t = \frac{c \cdot 3^n}{\sum_{j=i}^{\delta n} \binom{n}{j} 2^j}
\end{aligned}
$$

The following lemma shows the result of this computation.

Lemma 5.5.1. *If we apply the algorithm $t = \frac{c \cdot 3^n}{\sum_{j=i}^{\delta n} \binom{n}{j} 2^j}$ times, the probability that we miss an existing no-rainbow coloring is $\le e^{-c}$ when c is a constant.*

The following proposition helps us to find δ (20).

Proposition 5.5.2. *For $0 \le \delta \le 1/2$ and $n \in \mathbb{N}$ it holds that*

$$
\binom{n}{\delta n} \ge \frac{1}{\sqrt{8n\delta(1-\delta)}} \left(\frac{1}{1-\delta}\right)^{(1-\delta)n}
$$

We know $t = \frac{c \cdot 3^n}{\sum_{j=i}^{\delta n} \binom{n}{j} 2^j} \le \frac{c \cdot 3^n}{\binom{n}{\delta n} 2^{\delta n}}$ Using Proposition 5.5.2 and setting $\delta = 1/3$ for $r = 3$ we have:

$$
t \le \frac{c \cdot 3^n}{\binom{n}{\delta n} 2^{\delta n}} \le \frac{c \cdot 3^n 2^n}{3^n 2^{n/3}} \le c \cdot 2^{n/3}
$$

Thus, the complexity of the algorithm with random initial colorings for $r = 3$ is

$$
O(t \cdot 6^{\delta n}) = O(c \cdot 2^{n/3} \cdot 6^{n/3}) = O^*(12^{n/3}) = O^*(2.29^n)
$$

Considering Lemmas 5.5.2 and 5.5.1, when $r = 4$, we choose $\delta = 1/4$, which results in:

$O(t \cdot 12^{\delta n}) = O(t \cdot 12^{\delta n}) = O((3^{1/2} \cdot 12^{1/4})^n) = O(3.22^n)$.

In general, we can take $\delta = \frac{1}{r}$, and then we have

$$\binom{n}{n/r} \geq \frac{1}{\sqrt{8n}} r^{n/r} \left(\frac{r}{r-1}\right)^{(r-1)n/r} = \frac{r^n}{\sqrt{8n}(r-1)^{(r-1)n/r}}$$

Theorem 5.5.3. *The probabilistic algorithm 5.3 for no-rainbow 3-coloring has (worst-case) running time of $O^*(2.29^n)$. The probabilistic algorithm 5.3 for no-rainbow 4-coloring has (worst-case) running time of $O^*(3.22^n)$.*

Generalizing this for no-rainbow r-coloring ($r \geq 3$) this means: for every $r \geq 3$ there is a probabilistic algorithm for no-rainbow r-coloring with running time $O^*((r^{1/r} \cdot (r-1)^{\frac{r-1}{r}})^n)$.

5.6 Covering Codes

As noted earlier, we prefer a deterministic algorithm over a randomized algorithm. Thus, in this section we present a way to generate the initial coloring candidates instead of working with t random initial candidates.

Recall that Hamming radius is the maximum fraction of the number of nodes whose colors are different than their color in c, and $Ham_g(c) = \{\beta \in [r]^n | d(\beta, c) \leq g\}$ where $[r]^n$ is the set of all colorings. We define Covering Codes as follows:

Definition: A set of words $C = \{c_1, c_2, \cdots c_t\}$ over the set of colors $[r]$ of length n is called a *covering code* with Hamming radius g, if

$$\bigcup_{i=1}^{n} Ham_g(c_i) = [r]^n$$

In addition, if the Hamming balls are pairwise disjoint, C is called a *perfect code*.

We know $|Ham_g(c)| = \sum_{i=0}^{g} (r-1)^i \binom{n}{i}$. Unfortunately, there are no perfect codes with the parameters needed here. So we restrict ourselves to covering codes coming as close as possible to the ideal number of code words expressed by the so called Hamming bound:

$$\frac{r^n}{Ham_g(c)} = \frac{r^n}{\sum_{i=0}^{p} (r-1)^i \binom{n}{i}}$$

The Hamming bound is obtained when the Hamming balls are all disjoint and cover the entire search space $[r]^n$.

The following lemma is due to Moser and Scheder(16).

Lemma 5.6.1. *For every $\epsilon > 0$ there is a constant n_0 and a covering code $C_0 \in \{1, 2, \cdots, r\}^{n_0}$, with Hamming radius $\frac{n_0}{r}$, whose size is close to that of a perfect code, namely*

$$|C_0| \leq \frac{n_0^2 \cdot r^{n_0}}{\sum_{i=0}^{\frac{n_0}{r}} (r-1)^i \binom{n_0}{i}} \leq (\frac{r}{(r-1)^{1/r+h(1/r)}} + \epsilon)^{n_0} = ((r-1)+\epsilon)^{n_0 - \frac{2n_0}{r}}$$

.

Proof. The proof is similar to that in the previous section. Let ϵ and δ be given. We randomly choose $|C_0|$ code words and then compute the probability that there is a coloring that does not fall into any of the $|C_0|$ Hamming balls:

$$\begin{aligned}
\Pr(\exists c \forall i \leq |C_0| : d(C_i(H), c) > \delta n) &\leq \sum_c \prod_{i=1}^{|C_0|} P(d(C_i(H), c) > \delta n) \\
&= \sum_c \prod_{i=1}^{|C_0|} (1 - P(d(C_i(H), c) \leq \delta n)) \\
&= r^n \cdot \left(\frac{\sum_{j=0}^{\delta n} \binom{n}{j} (r-1)^j}{r^n} \right)^{|C_0|} \\
&\leq r^n \cdot e^{-|C_0| \cdot \frac{\sum_{j=0}^{\delta n} \binom{n}{j} (r-1)^j}{r^n}} && \text{since } 1 - x \leq e^{-x} \\
&= (\frac{r}{e^n})^n && \text{if we set } |C_0| = \frac{n^2 \cdot r^n}{\sum_{j=0}^{\delta n} \binom{n}{j} (r-1)^j}
\end{aligned}$$

It then holds for some n_0:

$$\begin{aligned}
|C_0| &= \frac{n_0^2 \cdot r^{n_0}}{\sum_{j=0}^{\delta n_0} \binom{n_0}{j} (r-1)^j} && (5.1) \\
&\leq \left(\frac{r}{(r-1)^{1/r+h(1/r)}} + \epsilon \right)^{n_0} && (5.2) \\
&= (r+\epsilon)^{n_0 - \frac{2n_0}{r}} && (5.3)
\end{aligned}$$

We use a probabilistic existence proof to show a code C_0 exists. Since the probabilistic estimation above for adequate size of $|C_0|$ converges toward 0 when $n \to \infty$, it follows that such codes satisfying the covering condition must exist. There exists such a code already for a concrete length n_0 which depends on ϵ and δ. □

The existence of code C_0 is proven and it can be found in polynomial time (20). For an arbitrary n we can obtain the covering code $C \subseteq \{1,2,3\}^n$ with Hamming radius δn and $|C| \leq (2^{1/3} + \epsilon)^n$ by putting code words from C_0 together. We create a set of code words such that each code words is a concatenation of code words in C_0.

$$C = \{w_1 w_2 w_3 \cdots w_n / n_0\}$$

which results in

$$|C| \leq ((2^{1/3} + \epsilon)^{n_0})^{n/n_0}$$

The fact is that C itself is a covering code. We can partition any arbitrary word w of length n into $\frac{n}{n_0}$ subwords each of length n_0. Each of these subwords is within Hamming distance δn_0 from a suitable codeword in C_0. So each word w of length n is within Hamming distance $\frac{n}{n_0}\delta n_0 = \delta n$ from a suitable codeword in C.

We use the idea of covering code to modify our randomized algorithm in the previous section and design a deterministic algorithm for no-rainbow coloring problem. We also modify the `localSearch` algorithm that we used before to find a more efficient algorithm.

We modify the procedure to *freeze* the color of each node after changing its color. After we freeze the color of a node we are never allowed to change the color again. Like the previous algorithms, we start with an initial surjective coloring $\tilde{c}$. We pick r non-neighbor vertices with r different colors and freeze the color of these nodes. In each iteration we find a rainbow hyperedge and change the color of one of its unfrozen nodes to make it a no-rainbow hyperedge. Doing this makes some hyperedges useless since they have all r nodes frozen. Also we will have some hyperedges with a reduced number of unfrozen nodes. We claim that in each iteration if the coloring is not a no-rainbow coloring or there doesn't exist any frozen rainbow hyperedge, then there exists a rainbow hyperedge with exactly $r-1$ frozen vertices. As a result, we need to worry about the color of the only node whose color is not frozen yet. In order to avoid having a rainbow hyperedge, we only need to assign one of the existing $r-1$ colors to the unfrozen node. We can

view the progress as a tree where each node in our tree stands for a state of our colored graph, and each node has $r-1$ branches. If we find a frozen rainbow hyperedge at any step, we reject it, stop investigating that branch, and move forward with the other branches. We show how to apply `localSearch&Freeze` for in Algorithm 5.4.

Algorithm 5.4: `generalLocalSearch&Freeze`$(H, \tilde{c}, g)$

if *$\tilde{c}$ is a no-rainbow coloring* **then**
  return 1
end
if *There exists a rainbow edge* **then**
  return 0
end
if *All rainbow edges have at most $r-2$ frozen nodes* **then**
  assign color c_1 to all the unfrozen nodes
  return 1
end
Let $e \in H$ be a hyperedge with rainbow color $\{c_1, c_2, \cdots, c_r\}$ where there is exactly one vertex v whose color is not frozen
Change v's color to any other $r-1$ colors, change v's status to fixed and apply `localSearch`$(H, \tilde{c}, g-1)$ to any new colorings
if `localSearch`$(H, \tilde{c}, g-1)$ *for any of the updated $\tilde{c}$s* **then**
  return 1
end
return 0

Observation 5.6.2. *Suppose that $\tilde{c}$ is a surjective r-coloring of an r-uniform hypergraph $H = (X, E)$. Suppose also that the node set X of H is partitioned into two sets:*

1. *F, consisting of all x in X such that $c(x)$ is fixed (i.e., is not allowed to change), and*
2. *$X - F$, consisting of nodes whose color is allowed to change.*

Then one of the following holds true

(i) $\tilde{c}$ is a no-rainbow r-coloring of H.

(ii) There exists a rainbow edge $e \in E$ such that $e \in F$

(iii) There exists a rainbow edge $e \in E$ such that $|e \cap F| \leq r - 1$

Lemma 5.6.3. *Consider an execution of* `localSearch&Freeze` *together with all the recursive calls that are triggered by this. There exists a hyperedge with exactly $r - 1$ frozen vertices.*

Proof. Lemma 5.6.2 shows in each iteration we either find a no-rainbow r-coloring (case 1) or we find a rainbow edge. If we find a no-rainbow coloring or we find a rainbow edge with all the nodes frozen (case 2), we are done. Thus, we only investigate case 3.

Suppose we are in case 3. Then, we have two possibilities:

a For every rainbow edge e, $|e \cap F| \leq r - 2$.

b There exists a rainbow edge e such that $|e \cap F| = r - 1$.

In case a, $\tilde{c}$ can be transformed into a no-rainbow r-coloring c as follows: Use color 1 to color all the unfrozen vertices. The result is a no-rainbow r-coloring because:

1. c is surjective,
2. no edge e in E such that $|e \cap F| = r$ in the original coloring $\tilde{c}$ is a rainbow edge, and
3. for each e in E such that $|e \cap F| \leq r - 2$ in the original coloring $\tilde{c}$, at least two nodes have the same color in the c.

In case b, choose any edge e such that $|e \cap F| = r - 1$ and let x be the single node in $e - F$. Then, there are only $r - 1$ possible colors for x. We try all the possibilities recursively (note that we may be able to rule out one or both of these possibilities immediately, but this is not a problem). □

Algorithm 5.5 applies localSearch&Freeze on the given covering code C_0.

Theorem 5.6.4. *The complexity of Algorithm 5.5 for $r = 3$ is $O(n^2 \cdot 2^{n/3} \cdot 2^{n/3}) = O^*((2^{2/3})^n) = O(1.59^n)$. The complexity of Algorithm 5.5 for no-rainbow 4-coloring is $O(n^2 \cdot 3^{n/2} \cdot 3^{n/4}) = O^*((3^{3/4})^n) = O^*(2.28^n)$.*

Generalizing Algorithm 5.5 for n and $r \geq 3$ the running time is $O(n^2 \cdot (r-1)^{n-\frac{2n}{r}}(r-1)^{\frac{n}{r}})$.

Algorithm 5.5: `coveringCodeLocalSearch&Freeze`$(H, n \geq n_0)$

for *all* $w_1 w_2 \cdots w_{n/n_0} \in C_0 \times \cdots \times C_0$ **do**
  interpret $w_1 w_2 \cdots w_{n/n_0}$ as a coloring $\tilde{c}$
  if `generalLocalSearch&Freeze`$(H, \tilde{c}, \delta n)$ **then**
    return "There exists a no-rainbow coloring"
  end
end
return "There exists no no-rainbow coloring"

5.7 Local Search and Freeze Randomized Algorithm

In section 5.3 we presented a randomized algorithm for no-rainbow coloring problem. Here we use the idea of freezing the color of one node in each step to avoid changing the color of a node over and over. We start by fixing the color of r non-neighbor nodes so that they have r different colors. As proved in lemma 5.6.3 in each step we either have at least one hyperedge with exactly $r-1$ frozen nodes or, we are done with the procedure (already found a no-rainbow r-coloring or failed to find any no-rainbow coloring). Following is the Algorithm for $r = 3$.

We can view the procedure as a tree where each node in tree represents a coloring, and each node has $r-1$ children. Thus, the probability of getting closer to the correct coloring, if there exists any, at each step is $\frac{1}{r-1}$.

Let us assume there exists a no-rainbow r-coloring c for the given hypergraph. Let A_k be the event that the initial assignment $\tilde{c}$ disagrees with c on exactly k vertices. Note that

$$\Pr[A_k] = \binom{n}{k} 2^k 3^{-n}$$

The probability p that we succeed is at least the probability that in Step 2 we make a beeline towards c, reducing disagreement by 1 on each step until we reach a no-rainbow coloring. Thus,

Algorithm 5.6: `Randomized&Freeze`

```
Pick a random initial coloring c̃
while There is at least one rainbow hyperedge which has exactly r − 1 frozen node: do
    Pick an arbitrary rainbow hyperedge with exactly r − 1 frozen nodes
    Change the color of the unfixed vertex to another color.
end
if There exists no rainbow edge then
    return 1
end
if There exists a frozen rainbow edge then
    return 0
end
if All rainbow edges have at most r − 2 frozen nodes then
    assign color c_1 to all the unfrozen nodes
    return 1
end
```

for the case of $r = 3$ we have:

$$\begin{aligned} p &\geq \Sigma_{k=0}^{n} \binom{n}{k} 2^k 3^{-n} (\frac{1}{2})^k \\ &= 3^{-n} \Sigma_{k=0}^{n} \binom{n}{k} \\ &= 3^{-n} 2^n = (\frac{2}{3})^n \end{aligned}$$

Putting this together with our claim and the fact that each iteration runs in polynomial time, we get a total work of $poly(n) \times (\frac{3}{2})^n = O^*(1.5^n)$.

For the case of $r = 4$ we have:

$$\begin{aligned} p &\geq \Sigma_{k=0}^{n} \binom{n}{k} 3^k 4^{-n} (\frac{1}{3})^k \\ &= 4^{-n} \Sigma_{k=0}^{n} \binom{n}{k} \\ &= 4^{-n} 2^n = (\frac{1}{2})^n \end{aligned}$$

Thus, the complexity of the mentioned algorithm for $r = 4$ is $poly(n) \times (2)^n = O^*(2^n)$.

Theorem 5.7.1. *Algorithm 5.6 for no-rainbow 3-coloring has the (worst-case) running time of* $O^*(1.5^n)$.

The algorithm 5.6 for no-rainbow 4-coloring has the (worst-case) running time of $O^*(2^n)$.

Generalizing this for no-rainbow r*-coloring* ($r \geq 3$) *this means: Algorithm 5.6 for no-rainbow* r*-coloring has the (worst-case) running time of* $O^*((\frac{r}{2})^n)$.

5.8 Local Search and Freeze Deterministic Algorithm

In this section, we apply the idea of freezing color of our nodes to the deterministic algorithm we presented in section 5.4.

We apply `generalLocalSearch&Freeze` on $\tilde{c}$ which maps r colors to r different nodes and assigns color 1 to every other node. We know if there exists a no-rainbow coloring, it must be in the Hamming ball of $\tilde{c}$ with maximum radius $\delta = \frac{(r-1)n}{r}$. By Lemma 5.6.2 we know we can choose a rainbow hyperedge with exactly $r-1$ frozen colors in each step. As a result we have $r-1$ choices in each step. Since the maximum distance between the candidate and the ideal solution is $(r-1)n/r$, we continue this procedure $g = (r-1)n/r$ times.

The complexity of applying Algorithm 5.4 with $\tilde{c}$ as input is $O(2^{2n/3}) = O(1.59^n)$ which is as efficient as Algorithm 5.5.

For $r = 4$ we get $O(3^{3n/4}) = O(2.28^n)$.

Theorem 5.8.1. *The* `generalLocalSearch&Freeze` *algorithm given in 5.6 for no-rainbow 3-coloring has (worst-case) running time of* $O(1.59^n)$. *The* `generalLocalSearch&Freeze` *algorithm given in 5.6 for no-rainbow 4-coloring has (worst-case) running time of* $O(2.28^n)$.

Generalizing this for no-rainbow r*-coloring* ($r \geq 3$) *we will have: the* `generalLocalSearch&Freeze` *algorithm for no-rainbow* r*-coloring has the (worst-case) running time of* $O^*((r-1)^{(r-1)n/r})$.

CHAPTER 6. PRACTICAL RESULTS[1]

6.1 Satisfiability Formulation

Let $\mathcal{S} = \{Y_1, Y_2, \ldots, Y_k\}$ be a taxon coverage pattern for X. Here we construct a Boolean formula that is satisfiable if and only if $\mathcal{S}$ is non-decisive. We use the equivalence between non-decisiveness of $\mathcal{S}$ and the existence of a no-rainbow 4-coloring of hypergraph $H(\mathcal{S})$ (Proposition 2.3.1).

Suppose $X = \{a_1, a_2, \ldots, a_n\}$. For each $i \in [n]$ and each color $q \in [4]$, define a Boolean *color variable* x_{iq}, where $x_{iq} = \texttt{true}$ if taxon a_i is assigned color q. The requirement that each a_i is assigned only one color for each $i \in [n]$, is expressed by the formula

$$\bigwedge_{i \in [n]} x_{i1} \oplus x_{i2} \oplus x_{i3} \oplus x_{i4}, \tag{6.1}$$

where $\oplus$ denotes the exclusive or.

The following formula ensures that each color $q \in [4]$ appears at least once.

$$\bigwedge_{q \in [4]} \bigvee_{i \in [n]} x_{iq} \tag{6.2}$$

To ensure that, for each $j \in [k]$, Y_j is not rainbow colored, we require that there exist at least one color that is not used in Y_j; i.e, that $\bigwedge_{i:\, a_i \in Y_j} \neg x_{iq} = \texttt{true}$, for some $q \in [4]$. The requirement that Y_j not be rainbow-colored is expressed as

$$\bigwedge_{j \in [k]} \bigvee_{q \in [4]} \bigwedge_{i:\, a_i \in Y_j} \neg x_{iq} \tag{6.3}$$

Proposition 6.1.1. *Let Φ be the Boolean formula obtained by taking the conjunction of expressions (6.1), (6.2), and (6.3). Then, $\mathcal{S}$ is non-decisive if and only if Φ is satisfiable.*

[1] **This chapter is joint work with Katherine Braught.**

Most SAT solvers require the input formula to be in conjunctive normal form (CNF). We convert each part of formula Φ of Proposition 6.1.1 into CNF as follows. Converting (6.1) to CNF is straightforward, using logical equivalence. Expression (6.2) is already in CNF. We can use DeMorgan's Theorem to convert expression (6.3) to CNF, but this yields too many clauses. Instead, we use the Tseytin transformations (see, e.g., (20)), to ensure that the number of clauses and variables grows linearly.

6.2 An Integer Linear Programming Formulation

We now formulate a 0-1 integer linear program (ILP) that is feasible if and only if $\mathcal{S}$ is non-decisive.[2] We use the same notation as in Section 6.1, and again rely on the equivalence between non-decisiveness of $\mathcal{S}$ and the existence of a no-rainbow 4-coloring of hypergraph $H(\mathcal{S})$.

Our ILP expresses the logical constraints (6.1), (6.2), and (6.3) of Section 6.1 using linear inequalities. For each $i \in [n]$ and each color $q \in [4]$, define a binary *color variable* x_{iq}, where $x_{iq} = 1$ if taxon a_i is assigned color q. To ensure that, for each $i \in [n]$, a_i is assigned only one color, we add constraints

$$\sum_{q \in [4]} x_{iq} = 1, \quad \text{for each } i \in [n]. \tag{6.4}$$

The following constraints ensure that each color $q \in [4]$ appears at least once.

$$\sum_{i \in [n]} x_{iq} \geq 1, \quad \text{for each } q \in [4]. \tag{6.5}$$

To ensure that, for each $j \in [k]$, Y_j is not rainbow colored, we require that there exist at least one color that is not used in Y_j; i.e, that $\sum_{i:\, a_i \in Y_j} x_{iq} = 0$, for some $q \in [4]$. To express this condition, we define a binary variable z_{jq} for each $j \in [k]$ and each $q \in [4]$ such that $z_{jq} = 1$ if and only if $\sum_{i:\, a_i \in Y_j} x_{iq} = 0$. We express the z_{jq}s in terms of the x_{iq}s with the following linear constraints.

$$(1 - z_{jq}) \leq \sum_{i:\, a_i \in Y_j} x_{iq} \leq n \cdot (1 - z_{jq}), \text{ for each } j \in [k] \text{ and each } q \in [4] \tag{6.6}$$

[2] For an introduction to the applications of integer linear programming, see (12).

Using the z_{jq}s, we express the requirement that Y_j not be rainbow-colored as

$$\sum_{q\in[4]} z_{jq} \geq 1, \quad \text{for each } j \in [k]. \tag{6.7}$$

Proposition 6.2.1. *$\mathcal{S}$ is non-decisive if and only if the 0-1 ILP with variables x_{iq} and z_{jq} and constraints (6.4), (6.5), (6.6), and (6.7) is feasible.*

6.3 Computational Results Using SAT and ILP

We wrote Python scripts that, as explained in Sections 6.1 and 6.2, take as input a taxon coverage pattern $\mathcal{S}$ and generate either (i) a CNF Boolean formula that is unsatisfiable if and only if $\mathcal{S}$ is decisive or (ii) an ILP model that is infeasible if and only if $\mathcal{S}$ is decisive. We used the PySAT python library (14) to solve CNF formulas. PySAT offers a variety of SAT solvers, including Glucose 3, Glucose 4, Lingeling, and CaDiCaL. We only report results for Glucose 4, as the other SAT solvers performed similarly or were slower. We used Gurobi (11) to solve ILP models. All tests were run on a four core Dell PC running Ubuntu-based Elementary OS.

We ran our scripts on the data sets analyzed in (6). Table 6.1 compares the running times of the SAT and the ILP approaches to generate a decisive submatrix. Neither approach is uniformly faster than the other. However, the SAT formulation may be preferable in practice as many good open-source SAT solvers are available.

As expected, the majority of the data sets we analyzed are non-decisive. Thus, we extended our scripts to generate decisive sub-matrices using two different simple methods:

- *Remove method:* Repeatedly remove a taxon that has the fewest number of non-zero entries until a decisive sub-matrix is found.
- *Add method:* Remove taxa as described in the remove method and add back sets of taxa in the powerset of initially removed taxa until the largest decisive sub-matrix is found.

We also used the reduction described in Section 4.3 to reduce the matrix sizes to the sizes described in Table 6.2. Figure 6.3 compares the running times of the ILP and SAT approach for

Table 6.1 Submatrix Generation Time in Seconds for ILP and SAT formulations Using Remove Method

Data Set	# Loci	# Taxa	Gurobi running time (sec)	Glucose4 running time (sec)
Allium	6	57	0.480	0.156
Asplenium	6	133	0.882	0.373
Caryophyllaceae	7	224	2.297	1.449
Eucalyptus	6	136	1.188	0.450
Euphorbia	7	131	0.556	0.343
Ficus	5	112	0.663	0.356
Iris	6	137	0.935	0.498
Meredith.mammals	6	169	0.270	0.206
Miadlikowska.fungi	9	1317	70.686	x
Primula	6	185	1.830	12.486
Rabosky.scincids	6	213	0.433	0.502
Ranunculus	7	170	1.472	1.372
Rhododendron	7	117	0.542	0.517
Rosaceae	7	529	12.886	8.966
Shi.bats	9	815	35.660	39.272
Solanum	7	187	1.915	1.081
Soltis.saxifragales	1	946	49.954	37.932
Szygium	5	106	0.365	0.205
Tolley.chameleons	6	202	0.5340	0.399

the remove method and shows a large improvement over the running times for the non-reduced matrices in Table 6.1. Table 6.2 shows the size of the final decisive sub-matrix for both the reduced and non reduced matrices. Table 6.3 compares the running time and final sizes of the add and remove method, where an "x" indicates the program timed out. On smaller matrices, the add method can find a larger decisive submatrix in similar time. On matrices with many taxa removed initially, the add method times out due the exponential number of taxa sets to be re-added.

Table 6.2 Reduced Matrices Sizes

Data Set	Rows in reduced matrix	Rows in reduced decisive submatrix	Final Expanded Decisive Matrix Rows
Allium	13	4	6
Asplenium	7	5	32
Caryophyllaceae	18	7	69
Eucalyptus	11	3	3
Euphorbia	15	11	82
Ficus	17	9	26
Iris	13	6	25
Meredith.mammals	63	62	168
Miadlikowska.fungi	30	9	320
Primula	17	4	29
Rabosky.scincids	15	10	207
Ranunculus	19	9	15
Rhododendron	14	7	63
Rosaceae	25	7	64
Shi.bats	279	35	35
Solanum	22	8	27
Soltis.saxifragales	193	16	16
Szygium	6	5	85
Tolley.chameleons	21	13	188

Table 6.3 Time to generate submatrices on reduced matrices

	Remove Method			Add Method		
Data Set	Final Size	Gurobi time (sec)	Glucose 4 time (sec)	Final Size	Gurobi time (sec)	Glucose 4 time (sec)
Allium	6	0.316	0.146	11	0.269	0.192
Asplenium	32	0.162	0.136	132	0.147	0.164
Caryophyllaceae	69	0.209	0.199	124	x	0.296
Eucalyptus	3	0.17	0.14	134	0.155	0.165
Euphorbia	82	0.167	0.142	127	0.163	0.169
Ficus	26	0.183	0.143	95	0.238	0.201
Iris	25	0.172	0.147	129	0.16	0.17
Meredith.mammals	168	0.217	0.17	168	0.318	0.291
Miadlikowska.fungi	320	0.231	0.17	1192	x	5.349
Primula	29	0.17	0.138	171	0.235	0.21
Rabosky.scincids	207	0.174	0.135	208	0.158	0.182
Ranunculus	15	0.173	0.148	164	0.194	0.2
Rhododendron	63	0.167	0.133	87	0.184	0.181
Rosaceae	64	0.2	0.15	415	x	1.769
Shi.bats	35	4.769	6.137	x	x	x
Solanum	27	0.212	0.167	151	x	0.361
Soltis.saxifragales	16	3.319	2.766	x	x	x
Szygium	85	0.18	0.169	94	0.136	0.164
Tolley.chameleons	188	0.199	0.186	194	x	0.27

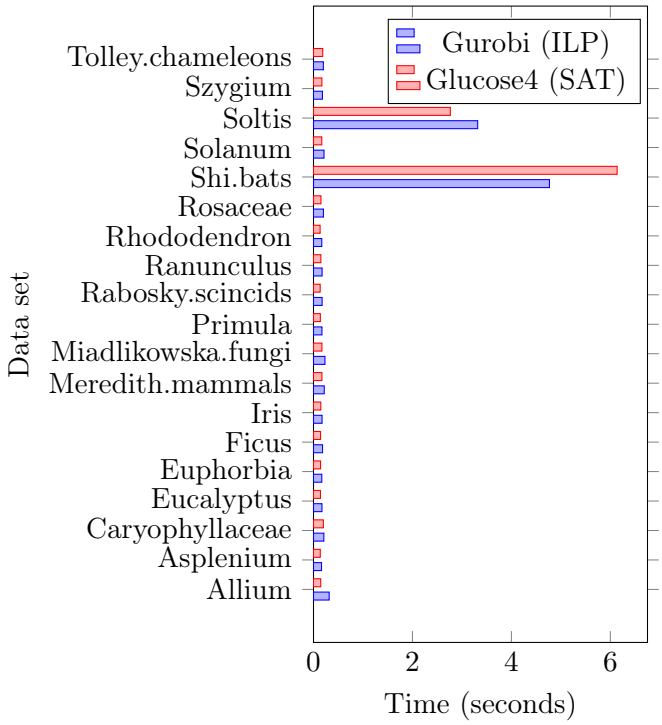

Figure 6.1 Time to Find A Decisive Submatrix on a Reduced Matrix using the Remove method

CHAPTER 7. FUTURE WORK SUMMARY AND DISCUSSION

Despite its apparent complexity, the decisiveness problem appears to be quite tractable in practice. We have presented algorithms to test decisiveness that are far superior to exhaustive enumeration. Additionally, our computational results on real data using SAT and ILP show that these approaches, combined with our reduction reduction techniques are effective in practice.

Since real data sets are likely to be non-decisive, testing for decisiveness can only be considered a first step. Indeed, if we determine that a data set is not decisive, it is useful to find a subset of the data that is decisive. In Section 6.2, we have taken some further steps in that direction, using simple heuristics. These heuristics or their implementations could potentially be improved upon, perhaps relying on the data reduction ideas of Section 4.3. One open problem is whether the doubly-exponential algorithm of Theorem 4.3.4 can be improved.

In chapter 5 we designed a deterministic algorithm using the idea of covering code. Covering codes generate a number of initial candidates with small Hamming balls to cover the whole search space. In another section of chapter 5 we showed that due to coloring symmetry we only need to cover $\frac{r-1}{r}$ of the whole space to design a deterministic algorithm. One open problem is to employ fewer covering codes to only cover the required space.

BIBLIOGRAPHY

[1] Ané, C., Eulenstein, O., Piaggio-Talice, R., and Sanderson, M. J. Groves of phylogenetic trees. *Annals of Combinatorics 13*, 2 (August 2009), 139–167.

[2] Berge, C. *Graphs and hypergraphs*. North-Holland, 1973.

[3] Berge, C. *Hypergraphs: combinatorics of finite sets*, vol. 45 of *North-Holland Mathematical Library*. Elsevier, 1984.

[4] Bininda-Emonds, O. R. P., Ed. *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, vol. 4 of *Series on Computational Biology*. Springer, Berlin, 2004.

[5] Bodirsky, M., Kára, J., and Martin, B. The complexity of surjective homomorphism problems—a survey. *Discrete Applied Mathematics 160*, 12 (2012), 1680–1690.

[6] Dobrin, B. H., Zwickl, D. J., and Sanderson, M. J. The prevalence of terraced treescapes in analyses of phylogenetic data sets. *BMC Evolutionary Biology 18* (2018), 46.

[7] Fischer, M. Mathematical aspects of phylogenetic groves. *Annals of Combinatorics 17*, 2 (2013), 295–310.

[8] Fomin, F. V., and Kratsch, D. *Exact Exponential Algorithms*. Springer, Berlin, Heidelberg, 2010.

[9] Graham, R. L., Knuth, D. E., and Patashnik, O. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, Reading, MA, 1989.

[10] Guindon, S., and Gascuel, O. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic Biology 52*, 5 (2003), 696–704.

[11] Gurobi Optimization, L. Gurobi optimizer reference manual, 2019.

[12] Gusfield, D. *Integer Linear Programming in Computational and Systems Biology: An entry-level text and course*. Cambridge University Press, 2019.

[13] Hinchliff, C. E., Smith, S. A., Allman, J. F., Burleigh, J. G., Chaudhary, R., Coghill, L. M., Crandall, K. A., Deng, J., Drew, B. T., Gazis, R., Gude, K., Hibbett, D. S., Katz, L. A., Laughinghouse IV, H. D., McTavish, E. J., Midford, P. E., Owen, C. L., Reed, R. H., Reesk, J. A., Soltis, D. E., Williams, T., and Cranston, K. A. Synthesis of phylogeny and taxonomy into a comprehensive tree of life. *Proceedings of the National Academy of Sciences 112*, 41 (2015), 12764–12769.

[14] Ignatiev, A., Morgado, A., and Marques-Silva, J. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT* (2018), pp. 428–437.

[15] Jarvis, E. D., Mirarab, S., Aberer, A. J., Li, B., Houde, P., Li, C., Ho, S. Y., Faircloth, B. C., Nabholz, B., Howard, J. T., et al. Whole-genome analyses resolve early branches in the tree of life of modern birds. *Science 346*, 6215 (2014), 1320–1331.

[16] Moser, R. A., and Scheder, D. A full derandomization of schöning's k-sat algorithm. In *Proceedings of the forty-third annual ACM symposium on Theory of computing* (2011), pp. 245–252.

[17] Parvini, G., Braught, K., and Fernández-Baca, D. Checking phylogenetic decisiveness in theory and in practice. In *Bioinformatics Research and Applications - 16th International Symposium, ISBRA 2020, Moscow, Russia, December 1-4, 2020, Proceedings* (2020), Z. Cai, I. I. Mandoiu, G. Narasimhan, P. Skums, and X. Guo, Eds., vol. 12304 of *Lecture Notes in Computer Science*, Springer, pp. 189–202.

[18] Sanderson, M. J., McMahon, M. M., and Steel, M. Phylogenomics with incomplete taxon coverage: the limits to inference. *BMC Evolutionary Biology 10* (2010), 155.

[19] Sanderson, M. J., McMahon, M. M., and Steel, M. Terraces in phylogenetic tree space. *Science 333*, 6041 (2011), 448–450.

[20] Schöning, U., and Torán, J. *The Satisfiability Problem: Algorithms and Analyses*, vol. 3 of *Mathematik für Anwendungen*. Lehmanns Media, Berlin, 2013.

[21] Scornavacca, C. *Supertree methods for phylogenomics*. PhD thesis, Univ. of Montpellier II, Montpellier, France, December 2009.

[22] Semple, C., and Steel, M. *Phylogenetics*. Oxford Lecture Series in Mathematics. Oxford University Press, Oxford, 2003.

[23] Stamatakis, A. RAxML version 8: A tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics 30*, 9 (2014), 1312–1313.

[24] Stamatakis, A., and Alachiotis, N. Time and memory efficient likelihood-based tree searches on phylogenomic alignments with missing data. *Bioinformatics 26*, 12 (2010), i132–i139.

[25] Steel, M. *Phylogeny: Discrete and random processes in evolution*, vol. 89 of *CBMS-NSF Conference Series in Applied Mathematics*. SIAM, Philadelphia, PA, USA, 2016.

[26] Steel, M., and Sanderson, M. J. Characterizing phylogenetically decisive taxon coverage. *Applied Mathematics Letters 23*, 1 (2010), 82–86.

[27] Warnow, T. Supertree construction: Opportunities and challenges. Tech. Rep. arXiv:1805.03530, ArXiV, May 2018.

[28] Wickett, N. J., Mirarab, S., Nguyen, N., Warnow, T., Carpenter, E., Matasci, N., Ayyampalayam, S., Barker, M. S., Burleigh, J. G., Gitzendanner, M. A., et al. Phylotranscriptomic analysis of the origin and early diversification of land plants. *Proceedings of the National Academy of Sciences 111*, 45 (2014), E4859–E4868.

[29] Wilkinson, M. Coping with abundant missing entries in phylogenetic inference using parsimony. *Systematic Biology 44*, 4 (1995), 501–514.

[30] Zhbannikov, I. Y., Brown, J. W., and Foster, J. A. decisivatoR: an R infrastructure package that addresses the problem of phylogenetic decisiveness. In *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics* (2013), pp. 716–717.

[31] Zhuk, D. No-rainbow problem is NP-hard, March 2020. arXiv preprint https://arxiv.org/abs/2003.11764.