UNIVERSITY OF CALIFORNIA
RIVERSIDE

Some Algorithmic Problems Concerning the Inference and Analysis of Tag SNPs, Haplotypes and Pedigrees

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Lan Liu

September 2007

Dissertation Committee:
Dr. Tao Jiang, Chairperson

Dr. Marek Chrobak
Dr. Neal Young

The Dissertation of Lan Liu is approved:

Committee Chairperson

University of California, Riverside

Acknowledgements

I would not have been able to complete this dissertation without the kind help from many people. It is a great pleasure for me to take this opportunity to express my gratitude to all of them.

My most sincere appreciation goes first and foremost to my advisor Dr. Tao Jiang, for his invaluable guidance during my graduate study at the University of California, Riverside. He is incredibly enthusiastic, energetic and always available to give me insightful advice. Moreover, I have learned many precious lessons through his exceptional research work, diligent attitude, sense of responsibility, and wonderful teaching and presentation skills. Without his constant stimulation and criticism, I would not have been able to develop the necessary skills to solve problems independently, to apply my creativity, and to become a serious researcher. I also benefit a lot from his way of bearing appropriately, and getting along with other people kindly.

I would like to also thank Dr. Marek Chrobak and Dr. Neal Young for being on my committee and for their valuable comments on the early drafts of this work. CS218 taught by Dr. Chrobak was the place where I got in touch and fell in love with algorithms. I learnt a lot from his insightful analysis to solve new problems. Dr. Young's patient guidance and ingenious suggestions led me to the generation of my first research ideas and the completion of my first paper.

I wish to express my gratitude to all the co-authors of my publications for their hard work and contributions, especially to Jing Xiao. Jing and I worked so hard to make the progress on the pedigree-related problems. It is truly a joy and an unforgettable experience to work together with Dr. Jiang on the countless Friday afternoons, on the beach in Sanya city, on airplanes and on the bus to a botanic garden. I also want to thank Yonghui Wu for many collaborations and useful discussion. Xin Chen, the former postdoc in our algorithm lab, gave me a lot of encouragement and suggestions in the early stage of my research.

I made many friends at UC Riverside, from whom I have learned a lot. I would extend my gratitude to them for their friendship and constant supports, which have made my life at UC Riverside a wonderful and colorful experience. In particular, I am indebted to my dear friends, Shawn Nematbakhsh, Min Wan, Guobiao Mei, Song Lin, Li Wei, Jian Zhou, Xiaopeng Xi, Jing Xu and Jinan Yao, as well as my labmates, Qi Fu, Zheng Fu, Zheng Liu, Serdar Bozdag, Vladimir Vacic and Christos Koufogiannakis. Their friendship would remain as my most precious asset in the rest of my life.

Finally, I wish to thank my family with my utmost gratitude. They have been extremely supportive and considerate in this endeavor. My father Donglin, mother Caifeng, dear sister Lei, brother-in-law Xuping, and cute nephew Lele are always there to provide love and support. This dissertation is dedicated to them.

ABSTRACT OF THE DISSERTATION

Some Algorithmic Problems Concerning the Inference and Analysis of Tag SNPs, Haplotypes and Pedigrees

by

Lan Liu

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2007
Dr. Tao Jiang, Chairperson

This thesis is concerned with several fundamental computational problems arising in genetics, in particular, the selection of tagSNPs, haplotype inference, and pedigree analysis. It presents several original algorithmic and complexity results concerning these problems summarized below.

We first study the complexity and approximation of the problem of reconstructing haplotypes from genotypes on a pedigree under the Mendelian Law of Inheritance and the minimum recombinant principle, called MRHC. We give some approximation algorithms and computational/approximation complexity results on many variants of MRHC. When we assume that the hereditary process was free of recombinants, we give a fast algorithm to produce a solution with running time $O(mn^2 + n^2 \log^2 n \log\log n)$, where m is the number of loci (or markers) in a genotype and n the number of individuals in the pedigree. This greatly improves on the best previous result in the literature. Furthermore, if a pedigree has no mating loops, we present an (optimal) linear-time (*i.e.* $O(mn)$ time) algorithm to generate a solution to the haplotype inference problem.

We also study the tagSNP selection problem on multiple populations using the pairwise r^2 linkage disequilibrium criterion. We propose a novel combinatorial optimization model for the tagSNP selection problem, called the minimum common tagSNP selection (MCTS) problem, and present efficient algorithms for MCTS. We assess the performance of our algorithms on real HapMap data for genome-wide tagging. The numbers of tagSNPs selected by our algorithms are shown to be almost optimal since they are very close to the corresponding lower bounds (*i.e.* the minimum number of tagSNPs needed) obtained by our method.

We then introduce a new combinatorial optimization problem, called the Minimum Common Integer Partition (MCIP) problem for input integer multisets. We prove that MCIP is APX-hard, and propose some constant-ratio approximation algorithms for MCIP when the number of input multisets is a constant.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motiviation

The secret mechanism behind phenotypic variation and inheritance has intrigued the study of genetic markers. With the discovery of genetic markers such as microsatellite DNA sequences and *Single Nucleotide Polymorphisms* (SNPs), it is now possible to provide a unique genetic map to track the variation and inheritance of genetic markers. The international HapMap project launched in October 2002, aims to discover the haplotype structure of human beings and examine the common haplotypes among populations [45].

Homologous recombination, the combination of genetic material between chromosome pairs during meiosis, is essential in diploid organisms such as humans [4]. Unfortunately, the diploid structure of humans makes it very expensive to collect haplotype data directly to display the recombination events. In a large-scale sequencing project, genotype data instead of haplotype data are collected. However, haplotype data are required in many genetic marker applications, such as linkage disequilibrium analysis and disease association mapping to name a few [37, 44]. Therefore, combinatorial algorithms and statistical methods to reconstruct haplotypes from genotypes (*i.e.* the haplotype phasing or inference problem) are urgently needed.

The input data for this problem can be SNP fragments from an individual, genotype data in a population or genotype data in a family [5, 6, 18, 7, 35]. There are many combinatorial [28, 14, 29, 36] and statistical ways [7, 8] of tackling the phasing problem. They are usually quite computationally demanding.

Some of the commonly used combinatorial methods [28, 14, 29, 36] take advantage of the availability of pedigree data. In other words, given a pedigree and the genotype information, they reconstruct a haplotype configuration for each individual in the pedigree by trying to solve the *Minimum Recombinant Haplotype Configuration* (MRHC) problem [28]. During the process of reconstruction, the minimum recombinant criterion is used as the objective function. Because this objective attempts to reduce the number of candidate haplotype configurations, it naturally preserves common haplotype structures.

In this dissertation, we mainly study the following three problems related with haplotype inference on given pedigrees.

The Approximation and Complexity of the MRHC Problem All the existing methods for the MRHC problem are time and space consuming for realistic applications. For example, when a Pentium IV computer with 256MB RAM is used to solve MRHC on an input pedigree with 29 members and 51 SNP markers, an effective combinatorial algorithm ILP takes about 5 hours to find an exact solution, whereas a well-known statistical approach, SimWalk2, takes even more than 6 days to find a haplotype configuration with the maximum likelihood [30]. While over 5 millions of SNPs have been identified in the public database dbSNP [45], there is a great need for efficient algorithms that could scale up to the whole genome level. This difficulty motivates us to analyze the hardness and approximability of MRHC problems from a theoretical point of view in Chapter 2.

The Zero-Recombinant Haplotype Configuration (ZRHC) Problem A closely related problem, called the *zero-recombinant haplotype configuration (ZRHC)* problem where we would like to enumerate all haplotype solutions that require no recombinant (if such solutions exist), was studied in [28] under a more stringent biological assumption that the pedigree is also recombination-free. ZRHC is interesting because recent genetic research has shown that human genomic DNAs can be partitioned into long blocks (called *haplotype blocks*) such that recombination within each block is rare or even nonexistent [15, 19, 25], especially when restricted to a single pedigree [29, 30]. An efficient algorithm for ZRHC could also be useful for solving the general MRHC problem as a subroutine, when the number of recombinants is expected to be small. We note in passing that recent work on haplotype inference for population data based on perfect phylogenies also assumes the data is recombination-free

[20, 21]. Observe that, when the solution for ZRHC is not unique, it would really be useful to be able to enumerate all the solutions instead of finding only one feasible solution, so that the solutions can be examined in subsequent analysis (*e.g.* likelihood distribution of haplotypes [29, 30], linkage between different haplotype blocks, etc.) by geneticists. We will give a novel efficient algorithm which can provide both a particular solution [1] and a general solution [2] to the ZRHC problem in Chapter 3.

The ZRHC Problem on Mating-Loop-Free Pedigrees When the input pedigree is a tree pedigree with no mating loops (which is often true for human pedigrees), the zero-recombinant haplotype configuration problem is called the *Loop-Free ZRHC* problem. We will provide an optimal linear-time algorithm to the Loop-Free ZRHC problem in Chapter 4.

Besides the above three problems, we also study an important problem for selecting informative SNPs described below.

The TagSNP Selection Problem The rapid development of high-throughput genotyping technologies has recently enabled genome-wide association studies to detect connections between genetic variants and human diseases. SNP is the most frequent form of polymorphism in the human genome. Common SNPs with *minor-allele frequency* (MAF) of 5% have been estimated to occur once every $\sim$600 bps [64], and there are more than 10 million verified

---

[1] A particular solution of any linear system is an assignment of numerical values to the variables in the system which satisfies the equations in the system.

[2] A general solution of any linear system is denoted by the span of a basis in the solution space to its associated homogeneous system, offset from the origin by a vector, namely by any particular solution.

SNPs in dbSNP [58]. Given these numbers, it is currently infeasible to take into account all the available SNPs to carry out association studies. This motivates the selection of a *subset* of informative SNPs, called *tagSNPs*. We will explore the tagSNP selection problem in Chapter 5 and present efficient algorithms to choose tagSNPs for multiple-population data.

We also study a new combinatorial optimization problem motivated by some biological applications that are related to comparative genomics instead of genetics as follows.

The Minimum Common Integer Partition Problem Computational molecular biology has emerged as one of the most exciting interdisciplinary fields in the past two decades, in part because various biological applications have spawned a large number of interesting combinatorial problems such as multiple sequence alignment [93], sorting by reversals [101], and recently the minimum common partition problem [91]. These problems have attracted considerable attention from computer scientists who took the challenge to design efficient and effective algorithms for solving them [86, 95, 94]. In this thesis, we introduce a new combinatorial optimization problem, called the *Minimum Common Integer Partition* problem (MCIP), which was inspired by our recent work on ortholog assignment and DNA fingerprint assembly. Some complexity results and efficient approximation algorithms for MCIP are given in Chapter 6.

1.2 Formal Definition of Biological Background and Related Problems

In this section, we give a formal definition of the MRHC, ZRHC, Loop-Free ZRHC problems as well as the issue of pedigree representation and biological background. We follow the conventions in [28].

A pedigree can be defined formally as follows.

Definition 1 *A pedigree graph is a weakly connected directed acyclic graph (DAG)* $G = (V, E)$*, where* $V = M \cup F$*,* M *stands for the male nodes,* F *stands for the female nodes. The in-degree of each node is* 0 *(founders) or* 2 *(non-founders). If the in-degree of a node is* 2*, one edge must start from a male node (called father) and the other edge from a female node (called mother) and the node itself is a child of its parents (father and mother).*

In a pedigree, the individual nodes outgoing from a mating node are called the *children*. The individual nodes with zero in-degree are called the *founders*. The induced subgraph by the father, the mother and one child adjacent to the same mating node is called a *family trio*. If there are two node-disjoint paths between two mating nodes in the pedigree graph, this pedigree has a *mating loop*. A pedigree without mating loops is called a *tree pedigree*. A pedigree where each member has at most one mate and at most one child looks like a binary tree, so this kind of pedigree is called a *binary-tree pedigree*. Figure 1.1(B) illustrates an example pedigree with a mating loop.

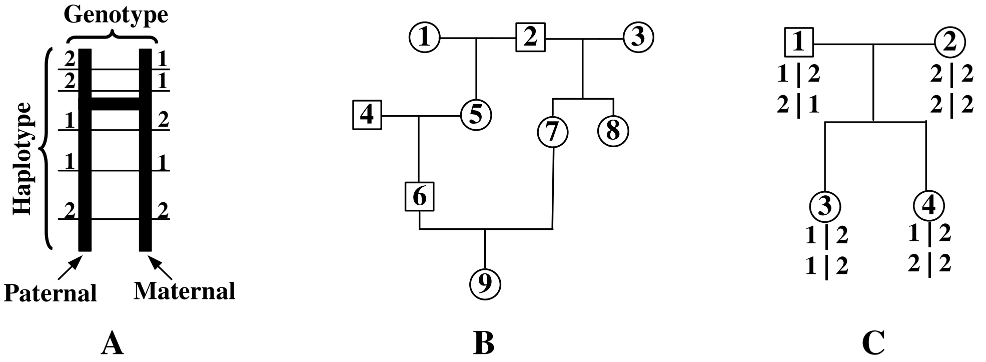

Figure 1.1: An example of a pair of chromosomes, a pedigree, and recombination event. (A). The structure of a pair of chromosomes from a mathematical point of view. In the figure, each numeric value (1 or 2) represents a marker state or an *allele*. The haplotype inherited from the father (or the mother) is called *paternal haplotype* (or *maternal haplotype*, respectively). The paternal and maternal haplotypes are thus strings 22112 and 11212, and they form the genotype $\{1, 2\}\{1, 2\}\{1, 2\}\{1, 1\}\{2, 2\}$, which is a string of unordered pairs of alleles at each locus. (B). An illustration of a pedigree with 9 members with a *mating loop*, where circles represent females and boxes represent males. Children are shown under their parents with line connections. For example, individuals 7 and 8 are children of individuals 2 and 3. Individuals without parents, such as individuals 1 and 2 are called *founders*. A pedigree with no mating loops are called *tree pedigree* conventionally. (C). An example of recombination event where the haplotypes of individual 1 recombine to produce the paternal haplotype of individual 3. The numbers inside the circles/boxes are individual IDs. Here, a "|" is used to indicate the phase of the two alleles at a marker locus, with the left allele being paternal and the right maternal. Both loci of individual 2 and the 2nd locus of individual 4 are *homozygous* while all the other loci in the pedigree are *heterozygous*.

The genome of an organism consists of *chromosomes* that are double strand DNAs. Locations on a chromosome can be labelled using *markers*, which are small segments of DNA with some specific features. A physical position of a marker on a chromosome is called a *marker locus* and a marker state is called an *allele*. In *diploid* organisms, chromosomes come in pairs. The status of two alleles at a particular marker locus of a pair of chromosomes is called a *marker genotype*. The genotype information at a locus will be denoted using a set, *e.g.* $\{a, b\}$. If the two alleles a and b are the same, the genotype is *homozygous*. Otherwise, it is *heterozygous*. If an allele is missing at some locus, it is denoted as a "*". A *haplotype* consists of all alleles, one from each locus, that are on the same chromosome. Figure 1.1(A) illustrates the above concepts, where alleles are represented by their numerical IDs.

The *Mendelian law of inheritance* states that the alleles of a child must come from the alleles of its parents at each marker locus (*i.e.* assuming no mutations within a pedigree). In other words, the two alleles at each locus of the child have different origins: one is from its father (which is called the *paternal* allele) and the other from its mother (which is called the *maternal* allele). Usually, a child inherits a complete haplotype from each parent. However, *recombination* may occur, where the two haplotypes of a parent get shuffled due to a crossover of chromosomes and one of the shuffled copies is passed on to the child. Such an event is called a recombination event and its result is called a *recombinant*. Since markers are usually short DNA sequences, we assume that recombination only occurs between markers. Figure 1.1(C) illustrates an example where the paternal haplotype of member 3 is the result of a recombinant. Paternal allele and maternal allele at each locus is separated by a "|" in this figure.

We use the term *haplotype configuration* to describes not only the paternal and maternal haplotypes of an individual, but also the grandpaternal or grandmaternal origin of each allele on the haplotypes. Observe that the number of recombinants required in a pedigree can be easily computed, once the haplotype configuration of each member of the pedigree is given.

By utilizing some reasonable biological assumptions, such as the *Mendelian law of inheritance*, and the *minimum-recombination principle* which says that genetic recombination is rare for closely linked markers and thus haplotypes with fewer recombinants should be preferred in haplotype inference [35, 36], several combinatorial approaches for inferring haplotypes from genotypes on a pedigree have been proposed recently and shown to be powerful and practical [35, 28, 29, 30, 36, 41, 42]. The MRHC, ZRHC, Loop-Free ZRHC problems are all formulated as combinatorial optimization problems using the minimum-recombination principle.

The MRHC problem is defined as follows.

Definition 2 *(MRHC) Given a pedigree and genotype information for each member of the pedigree, find a haplotype configuration for the pedigree that requires the minimum number of recombinants.*

Namely, the ZRHC problem is a special case of MRHC with the following definition.

Definition 3 *(ZRHC) Given a pedigree and genotype information for each member of the pedigree, find a haplotype configuration for the pedigree that requires no recombinant (if such solution exists).*

Furthermore, the Loop-Free ZRHC problem is a special case of ZRHC with the following definition.

Definition 4 *(Loop-Free ZRHC) Given a pedigree without mating loops and genotype information for each member of the pedigree, find a haplotype configuration for the pedigree that requires no recombinant (if such solution exists).*

1.3 Organization of the Dissertation

The dissertation is organized as follows. We have briefly given the definitions of the MRHC, ZRHC, Loop-Free ZRHC problems and other closely related problems, and introduced the related biological background in section 1.2. Next, we study the approximation and complexity of the MRHC problem in chapter 2. Then we give an efficient algorithm for solving the ZRHC problem in chapter 3. Furthermore, if the input pedigree for the ZRHC problem does not have mating loops, the problem can be solved in linear time as shown in chapter 4. The following chapter 5 is devoted to select genome-wide tagSNPs across multiple populations via the linkage disequilibrium criterion. Finally, chapter 6 gives the approximation algorithms for the MCIP problem. The conclusion is presented in chapter 7.

Chapter 2

Complexity and Approximation of the Minimum Recombinant Haplotype Configuration Problem

2.1 Introduction

In this chapter, we study the complexity and approximation of the MRHC problem and its variants from a theoretical point of view.

---

[0] An extended abstract of this paper, entitled "Complexity and approximation of the minimum recombination haplotype configuration problem", has appeared in *Proc. 16th International Symposium on Algorithms and Computation(ISAAC)*, Dec. 2005, Sanya, P.R. China. [33]

2.1.1 Mathematical Notations and Some Related Definitions

We use the following mathematical notations to represent the critical concepts mentioned in chapter 1. Let n denote the number of loci, and $a_i, b_i, c_i, d_i, e_i, f_i \in \{1, 2, *\}, 1 \leq i \leq n$. we define a vector as $(a_1, a_2, \ldots, a_n)^T$. We also define $\overrightarrow{B}, \overrightarrow{C}, \overrightarrow{D}, \overrightarrow{E}$ and $\overrightarrow{F}$similarly. We represent a genotype, haplotype, haplotype configuration and haplotype grouping by the following forms $G(\overrightarrow{A}, \overrightarrow{B}), H(\overrightarrow{A}), HC(\overrightarrow{A}, \overrightarrow{B})$ and $HG(\overrightarrow{A}, \overrightarrow{B})$ respectively.

$$G(\overrightarrow{A}, \overrightarrow{B}) = \left\{ \begin{array}{cc} a_1 & b_1 \\ a_2 & b_2 \\ \cdots & \cdots \\ a_n & b_n \end{array} \right\} \qquad H(\overrightarrow{A}) = \left\{ \begin{array}{c} a_1 \\ a_2 \\ \cdots \\ a_n \end{array} \right\}$$

$$HC(\overrightarrow{A}, \overrightarrow{B}) = \left[\begin{array}{ccc} a_1 & | & b_1 \\ a_2 & | & b_2 \\ \cdots & | & \cdots \\ a_n & | & b_n \end{array} \right] \quad HG(\overrightarrow{A}, \overrightarrow{B}) = \left[\begin{array}{ccc} a_1 & / & b_1 \\ a_2 & / & b_2 \\ \cdots & / & \cdots \\ a_n & / & b_n \end{array} \right] \tag{2.1}$$

For an individual, the *genotype* is denoted by a vector of allele pairs at every locus with the form $G(\overrightarrow{A}, \overrightarrow{B})$; the *haplotype* is denoted by a vector of alleles at every loci with the form $H(\overrightarrow{A})$; the *haplotype configuration* is denoted by an ordered haplotype pair with the form $HC(\overrightarrow{A}, \overrightarrow{B})$, where the former haplotype comes from the mother and the latter one comes from the father; the *haplotype grouping* is denoted by an unordered haplotype pair with the form $HG(\overrightarrow{A}, \overrightarrow{B})$. Given an optimization problem Q, we denote a solution to Q by $OPT(Q)$. Given an instance ϕ of problem Q, we denote a solution to ϕ by $OPT_Q(\phi)$ for convenience. For the Min 2CNF Deletion [2, 3], Min UnCut [3], $\neq$3SAT [9] and other satisfiablility problems, we usually need to check the following two properties to make sure that a truth assignment is a feasible solution. The following term "satisfiability" means different things in different problems. For instance, in the Min 2CNF Deletion and Min UnCut problems, a constraint is satisfied if it is true; but in the $\neq$3SAT problem, a clause is satisfied if its three literatures do not have the same value.

Definition 5 *Given a variable x_i , any occurrence of x_i and $\bar{x}_i$ have different values; all occurrences of x_i (or $\bar{x}_i$) must have the same values (i.e. consistency property). Given a constraint, clause or monomial, it is satisfied (i.e. satisfiablility property).*

2.1.2 Variants of MRHC and Some Related Problems

We give the definitions of the variants of MRHC and list the related problems that are going to be discussed later in the chapter.

Definition 6 *MRHC(k, j) is defined the same as MRHC except that each member in the pedigree has at most k mates and at most j children with each mate. Binary-tree-MRHC is defined as MRHC on a binary-tree pedigree. 2-locus-MRHC is MRHC on a two-locus* *pedigree without missing data. Binary-tree-MRHC* (or 2-locus-MRHC*) is defined the same as binary-tree-MRHC (or 2-locus-MRHC, respectively) except it is allowed to have missing alleles. Tree-MRHC is MRHC on a pedigree without mating loops or missing data.*

In order to discuss the hardness and approximation of the variants, we are going to make use of some related problems or properties, such as the Min UnCut [3] (*i.e.* 2-Linear-Equations Mod 2 [2]), Min UnCut(k) (the same definition as Min UnCut except that each variable occurs at most k times), Min 2CNF Deletion [2, 3] problems, consistency and satisfiablility property (see the appendix). The Min UnCut and Min 2CNF Deletion problems are known to be NP-hard [3]. We will show that the Min UnCut(k) problem is also NP-hard in this chapter. For any NP-hard minimization (or ma$x_i$mization) problem, if there is some polynomial time algorithm to give a solution with the objective value no more (or less, respectively) than $f(n)OPT$ (or $OPT/f(n)$, respectively), where f(n) can be any function of the input size n, the problem can be approximated within ratio $f(n)$; otherwise, the problem cannot be approximated.

2.1.3 Previous Complexity Results on MRHC

Qian and Bechmann proposed a ruled-based algorithm to reconstruct haplotype configurations based on six rules [36]. Their algorithm is a heuristic without theoretical analysis. Li and Jiang first proved that MRHC on two-locus pedigree is NP-hard [28]. Doi, Li and Jiang further proved that MRHC on tree pedigrees is also NP-hard in the general case [14], even though MRHC can be solved by dynamic programming algorithms when the number of members or loci in the input pedigree is bounded by a constant. However, the NP-hardness proof requires pedigrees containing individuals with an unbounded number of mates or children. It was left as an open question if the proof can be improved to work for tree pedigrees where every individual has a bounded number of mates and children.

Consistency checking of the Mendelian law of inheritance (*i.e.* the Mendelian law checking problem) is closely related to the MRHC problem. The purpose of Mendelian law checking is to determine whether the given genotype data obey the classic Mendelian law of inheritance. Mendelian law checking usually needs to be done ahead of phasing haplotype configurations. Aceto *et al.* showed that the Mendelian law checking problem is NP-hard in general, although checking the consistency on pedigrees with bi-allelic data or with no mating loops [1] can be done in polynomial time.

In this chapter, we consider a simple variant of MRHC, which involves pedigrees with members that has at most one mate and one child (*i.e.* binary-tree-MRHC). It is an open question if binary-tree MRHC is NP-hard. A polynomial-time algorithm for it, if exists, could be useful for solving the general-case MRHC problem. Another important question is whether a good approximation algorithm exists for MRHC. Here, in terms of computing the minimum-recombinant haplotype, the accuracy is sacrificed to improve the efficiency. Previously, there is no known polynomial-time approximation algorithm for MRHC with guaranteed ratio.

Table 2.1: The known hardness results of the Mendelian law checking and MRHC problems.

Pedigree	Loop?	Multi-allelic?	Unbounded number of loci?	Unbounded number of members?	Hardness
Mendelian law checking	Yes	Yes			NP-hard [1]
		No			P [1]
	No				P [1]
MRHC	Yes	No	No	Yes	NP-hard [28]
	No	No	No	Yes	P [14]
	No	No	Yes	No	P [14]
	No	No	Yes	Yes	NP-hard [14]

2.1.4 Our Results

We will consider pedigrees with bi-allelic genotype data throughout this chapter. First, we reduce $\neq$3SAT to the binary-tree-MRHC problem and show that this problem is NP-hard, which answers an open question in [14]. Second, we study the approximability of MRHC on pedigree data with the following restrictions: (i) 2-locus genotype data with missing alleles, (ii) binary tree pedigrees with missing alleles, (iii) 2-locus genotype data without missing alleles, and (iv) tree pedigrees without missing alleles. These four restricted cases of MRHC are NP-hard problems shown either in the literature [28, 14] or in this chapter. We demonstrate that for MRHC in the former two cases (i) and (ii) cannot be approximated unless P = NP. We also prove that it is NP-hard to approximate problems (iii) and (iv) within any constant ratio under the Unique Games Conjecture [2]. Moreover, we show that problem (iii) can be approximated with ratio $O(\sqrt{\log n})$ in polynomial time by reducing it to the Min 2CNF Deletion problem, Finally, we discuss the approximation of MRHC on pedigrees where each member has a bounded number of children and mates, mirroring pedigrees in real applications.

2.1.5 Organization of the Chapter

The chapter is organized as follows. We briefly give definitions of the MRHC problem and other closely related problems, and introduce the related biological background in section 2.1. We prove that binary-tree-MRHC is NP-hard and state the approximatability of MRHC on pedigrees with missing data in section 2.2. We show the approximation lower bound of MRHC on pedigrees without missing data and the approximation upper bound of 2-locus-MRHC in section 2.3. In section 2.4, we tentatively explore the approximation hardness of MRHC on the pedigrees where each member has a bounded number of mates and children. We organize our hardness results and conclude this chapter with a few remarks in section 2.5.

2.2 Approximation of MRHC on Pedigrees with Missing Data

In this section, we prove the hardness of approximating MRHC on pedigree data with missing alleles. Two variants are considered.

Lemma 2.2.1 *If it is NP-hard to decide whether* $OPT(R) = 0$ *for a minimization problem R, R cannot be approximated unless P=NP.*

Proof: This lemma is a folklore. We prove it for the completeness of the chapter. Suppose that there is an approximation algorithm A with ratio $f(n)$, where $f(n)$ is a function of input size n. On any instance, algorithm A can generate a solution S with cost smaller than $f(n) \cdot OPT(R)$ in polynomial time. If $OPT(R) = 0$, solution S has cost 0. Otherwise, solution S has a positive cost. This gives rise to a polynomial time algorithm to decide whether $OPT(R) = 0$, which contradicts the assumption that it is NP-hard to decide whether $OPT(R) = 0$. This Lemma holds. ■

2.2.1 Hardness and Approximation of Binary-tree-MHRC(*)

Theorem 2.2.2 *Binary-tree-MRHC is NP-hard.* [1]

Proof: We reduce ≠3SAT to binary-tree-MRHC. Given an instance of ≠3SAT, we assume that the Boolean formula ϕ has m clauses and each clause has three literals. In order to construct an instance of binary-tree-MRHC to enforce the consistency and satisfiability properties of an assignment to the literals in ϕ, we consider all pairs of literals in ϕ for each variable x_k (for consistency checking) and all pairs of literals (x_p , x_q), (x_p , x_s) and (x_q , x_s) for each clause $x_p \vee x_q \vee x_s$ of ϕ (for satisfiability checking). Suppose that t pairs are required for consistency checking. Clearly, t is smaller than m^2. We check exactly $3m$ literals pairs for the satisfiability property. These pairs of literals are indexed sequentially starting from 1.

We construct a pedigree with genotype data consisting of $6m + 1$ loci, indexed from 1

---

[1] This result was mostly due to Xi Chen and Jing Xiao, but we include it here for the sake of completeness.

through $6m+1$. First, we introduce an individual node M_0. Each even-numbered locus of M_0 has a heterozygous genotype $\{1\ 2\}$ and corresponds to a literal in the formula ϕ. We observe that an assignment of PS values to the even-numbered loci in M_0 naturally divides the loci (and thus the literals) into two groups, which also bipartitions the corresponding literals in ϕ. The literals in one group can be considered to have value TRUE and the others FALSE. Therefore, the PS values of the even-numbered loci in M_0 encode an assignment to the literals in the formula ϕ. The constructed pedigree is illustrated in Figure 2.1. This construction can be done in $O(m^3)$ time. Satisfiability checking or consistency checking involving complementary literals is called type I checking; it is type II checking, otherwise.

Suppose that the i^{th} pair of literals indexed in the above is (x_j, x_k). The two loci representing the occurrences of these two literals are denoted as the joth and koth loci. The pedigree has four types of individuals A_i, B_i, C_i and M_i. For convenience, we define four genotypes $G_{\neq M}, G_{=M}, G_{\neq F}, G_{=F}$ and two configurations $HC_{\neq}, HC_{=}$ as follows.

$$
\begin{gathered}
G_{\neq M} = \left\{ \begin{matrix} 1 & 1 \\ 2 & 2 \end{matrix} \right\} \quad G_{=M} = \left\{ \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix} \right\} \quad G_{\neq F} = \left\{ \begin{matrix} 2 & 2 \\ 1 & 1 \end{matrix} \right\} \\
G_{=F} = \left\{ \begin{matrix} 2 & 2 \\ 2 & 2 \end{matrix} \right\} \quad HC_{\neq} = \begin{bmatrix} 1 & | & 2 \\ 2 & | & 1 \end{bmatrix} \quad HC_{=} = \begin{bmatrix} 1 & | & 2 \\ 1 & | & 2 \end{bmatrix}
\end{gathered}
\tag{2.2}
$$

The genotype of each individual is constructed as follows:

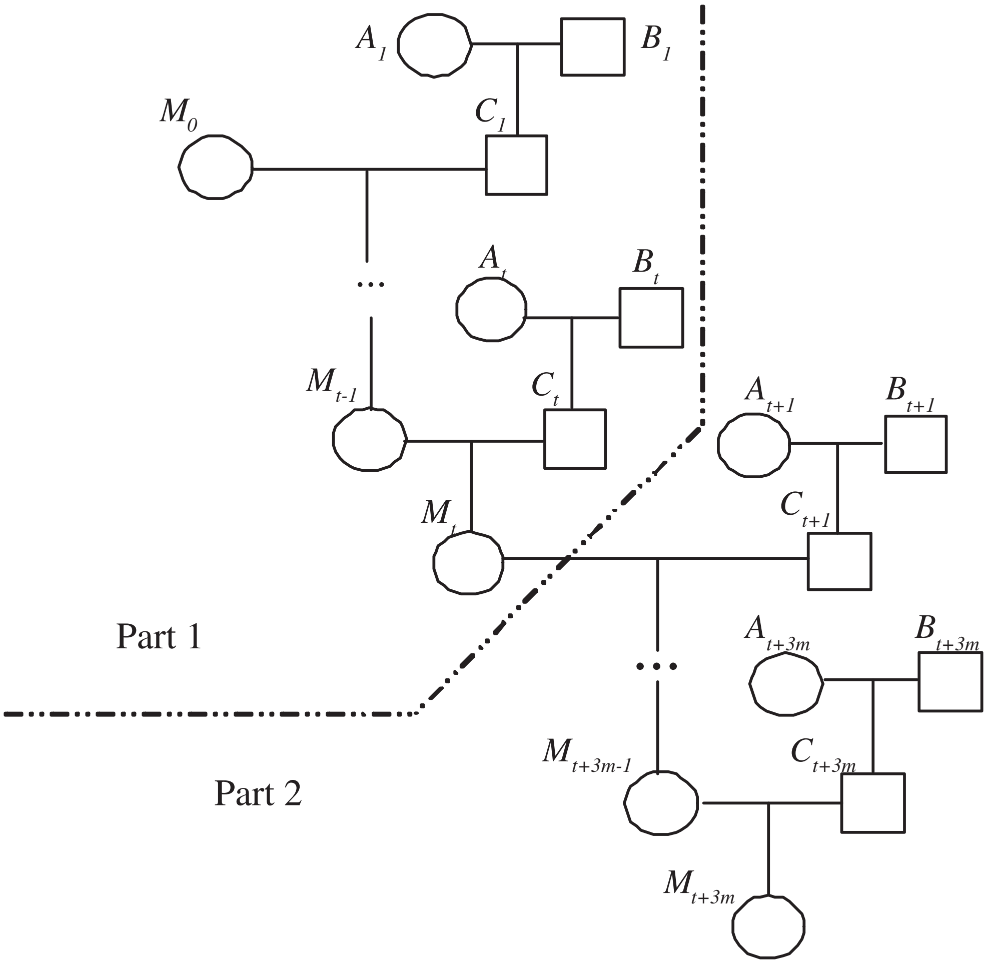

Figure 2.1: The pedigree used in the reduction from $\neq$3SAT to binary-tree-MRHC.

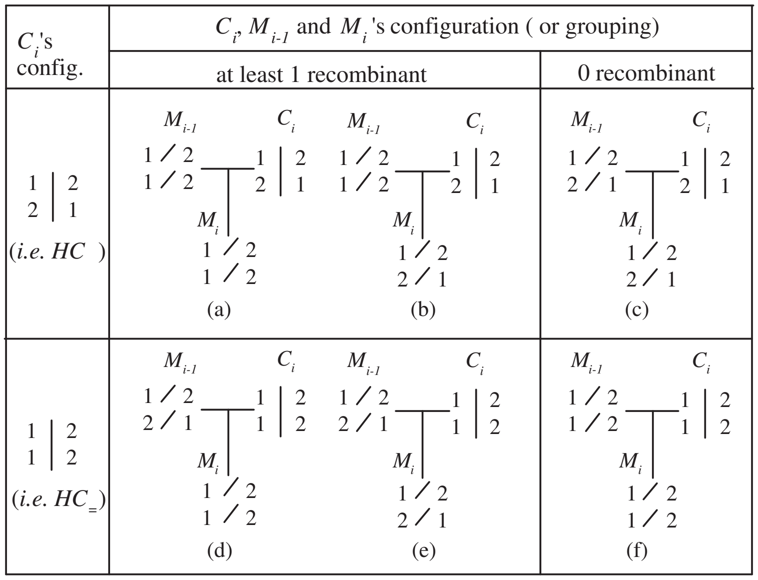

Figure 2.2: How C_i's configuration works in the reduction.

- A_i: All odd-numbered loci have genotype $\{2\ 2\}$ and all even-numbered loci have genotype $\{1\ 2\}$ except the two loci j_0 and k_0. The genotype of these two loci is $G_{\neq M}$ if the i^{th} literal pair is for type I checking; the genotype is $G_{=M}$ otherwise. ”

- B_i: All odd-numbered loci have genotype $\{2\ 2\}$ and all even-numbered loci have genotype $\{1\ 2\}$ except the two loci j_0 and k_0. The genotype of these two loci is $G_{\neq F}$ if the i^{th} pair is for type I checking; the genotype is $G_{=F}$ otherwise. ”

- C_i: All odd-numbered loci have genotype $\{2\ 2\}$ and all even-numbered loci have genotype $\{1\ 2\}$. Due to the genotype value of A_i and B_i, the two loci j_0 and k_0 of C_i will be forced to have the haplotype configuration $HC_{\neq}$ if the i^{th} pair is for type I checking; they will be forced to have the configuration $HC_=$ otherwise. ”

- M_i: All loci have genotype $\{1\ 2\}$. Because M_i's father, C_i has genotype $\{2\ 2\}$ at odd-numbered loci, M_i has haplotype configuration $\{1|2\}$ at odd-numbered loci. In other words, the maternal allele of M_i at odd-numbered loci must be 1.

There are two main parts in the pedigree. Part 1 is used to enforce the consistency property of a truth assignment, while part 2 is used to verify the satisfiability property.

- Part 1: For the i^{th} ($1 \leq i \leq t$) pair of literals indexed above, we introduce a gadget consisting of the parents-child trio A_i, B_i and C_i to do consistency checking. The two loci k_0 and j_0 standing for the occurrences of the i^{th} pair are shown in Figure 2.2. If and only if these two loci in M_{i-1} have the consistent PS value, there can be zero recombinants in this parents-child trio. Therefore, if and only if all the variables are consistent, there can be zero recombinants in part 1.
- Part 2: For the j^{th} ($1 \leq j \leq m$) clause $x_p \vee x_q \vee x_s$, we need three adjacent trios A_{t+3j-r}, B_{t+3j-r}, C_{t+3j-r} ($0 \leq r \leq 2$) to verify its satisfiability. We check whether at least one of the three pairs (x_p , x_q), (x_p , x_s) and (x_q , x_s) has different PS values. Each trio is constructed in the same way as the above. If all three literals in a clause have the same truth value, the satisfiability checking in all three corresponding trios will fail and thus at least two recombinants will be required in the pedigree. Otherwise, exactly one checking fails and thus only one recombinant would be required (which takes place when a C_i passes its haplotypes to the M_i, $1 \leq i \leq t + 3m$). Therefore, given any

haplotype configuration, there are at least m recombinants in part 2, no matter how many recombinants are in part 1.

We claim that ϕ has a satisfiable assignment if and only if there exists a haplotype configuration generating m recombinants in the whole pedigree. The proof of our claim is given as follows.

"$\Rightarrow$": Suppose ϕ has a satisfying assignment ψ to the literals (*i.e.*ψ has both consistency and satisfiability properties). Now we will show how to construct a haplotype configuration for each member based onψ. Let us take M_0 as an example. For each M_0's even-numbered locus, we can set its PS value being the boolean value of the corresponding literal in ψ. All the other M_i's have the same haplotype configurations as M_0. We also set the PS values of loci in A_i, B_i, C_i and M_i appropriately so that there is zero recombinant in part 1 of the pedigree and m recombinants in part 2. It is easy to verify that there are exactly m recombinants in the whole pedigree for the constructed haplotype configuration.

"$\Leftarrow$": Suppose that the pedigree has a haplotype configuration solution with m recombinants. Because part 2 requires at least m recombinants, there must be exactly zero recombinant in Part 1 and m recombinants in Part 2.

First, we claim that all M_i in the solution have the same haplotype configurations, which holds trivially for part 1. Now we focus on part 2. For convenience, a small pedigree for checking the satisfiability of a clause, say clause j ($1 \leq j \leq m$), is called a gadget, which includes four adjacent M_{t+3j-r}'s ($0 \leq r \leq 3$) and three adjacent trios A_{t+3j-r}, B_{t+3j-r}, C_{t+3j-r} ($0 \leq r \leq 2$). There are m gadgets in total in part 2. Observe that each gadget requires at least one recombinant, regardless how many recombinants occur in other gadgets or part 1. Therefore, each gadget must have exactly one recombinant. For each member M_i, notice that the maternal allele of every odd-numbered locus is 1, which means that every *odd-numbered* locus passes the *maternal allele* onto her child M_i. Further notice that, *even-numbered* loci are sandwiched by odd-numbered loci. Therefore, if any recombinant events occur between some M_i-1 and M_i (*i.e.* some even-numbered locus, say locus k of M_{i-1}, passes the paternal alleles to M_i), there must be at least two recombinants, one of which occurs between loci $k-1$ and k, and the other occurs between loci k and $k+1$. However, the gadget containing M_{i-1} and M_i has only one recombinant. Therefore, no recombinant can occur between any M_{i-1} and M_i (*i.e.*, all members M_i have the same haplotype configurations).

Now we can transform the haplotype configuration solution into a truth assignment. We pick an arbitrary M_i, say M_0, assign the PS values at even-numbered loci to the corresponding literals and obtain a truth assignment ψ. Because there are zero recombinants in part 1, this assignment is consistent. Because there is exactly one recombinant in the every gadget in part 2, this assignment satisfies every clause. In other words, the assignment ψ satisfies the boolean formula ϕ. Thus, binary-tree-MRHC is NP-hard. ■

Theorem 2.2.3 *It is NP-hard to decide whether OPT(binary-tree-MRHC*)=0.*

Proof: Again, we reduce $\neq$3SAT to binary-tree-MRHC. Given an instance of $\neq$3SAT, assume that the Boolean formula ϕ has m clauses and each clause has three literals. We use a similar pedigree as the one in Theorem 2.2.2. The main structure of the two parts of the pedigree remains the same. However in Part 2, instead of using three parents-child trios to check the satisfiablility of the j^{th} $(1 \leq j \leq m)$ clause, we use just one trio containing members D_j, E_j and F_j. We can force F_j's $(6j-4)^{th}$, $(6j-2)^{th}$ and $(6j)^{th}$ loci to have configuration HC_{SAT} defined as follows, by using the same technique as the one Theorem 2.2.2 (*i.e.* setting D_j and E_j's genotype appropriately).

$$HC_{SAT} = \begin{bmatrix} * & | & 1 \\ 1 & | & 2 \\ 2 & | & * \end{bmatrix} \tag{2.3}$$

This construction can still be done in O(m^3) time.

This modified part 2 of the new pedigree is illustrated in Figure 2.3. For the j^{th} $(1 \leq j \leq m)$ clause , Part 2 contains a member F_j that can be used to check if all x_p, x_q and x_s have equal truth values. If the literals in this clause do not have the same value, we can always assign appropriate alleles to the missing data positions without causing recombinants in the parents-child trio containing M_{j-1}, F_j and M_j; otherwise, there must be at least one recombinant, for the arguments similar to those demonstrated in Figure 2.2. We claim that if and only if ϕ has a satisfying assignment, there is a zero-recombinant solution for the entire

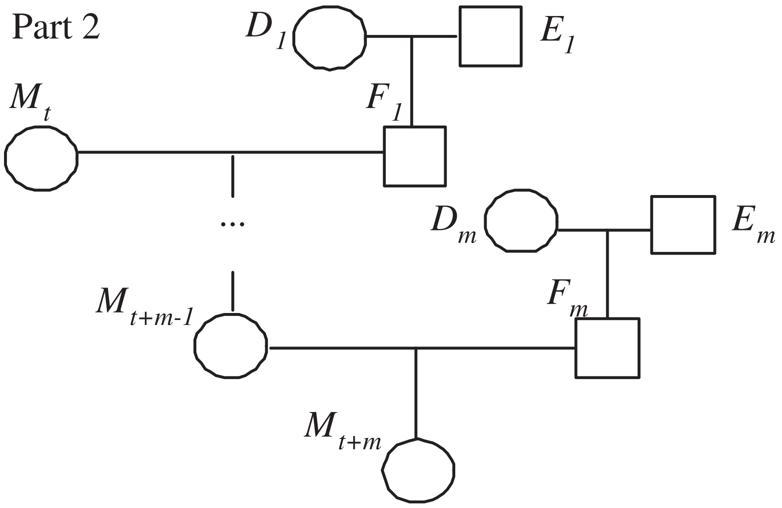

Figure 2.3: Part 2 of the constructed pedigree in the reduction.

pedigree. The details are omitted here. This completes the proof. ■

Corollary 2.2.4 *Binary-tree- MRHC* cannot be approximated unless P=NP.*

Proof: It follows obviously from Lemma 2.2.1 and Theorem 2.2.3. ■

2.2.2 Approximation of 2-loop-MHRC*

Theorem 2.2.5 *It is NP-hard to decide whether OPT(2-locus-MRHC*)=0*

Proof: We reduce 3SAT to 2-locus-MRHC*. Given an instance of 3SAT, we assume that the Boolean formula ϕ has n variables and m clauses, where each clause has three literals. We construct gadgets for each variable and clause and define a genotype G_0, two grouping HG_{TRUE} and HG_{FALSE}, a haplotype H_{TRUE} illustrated in Figure 2.4 The logical value TRUE and FALSE are represented by two alternative haplotype groupings for the genotype G_0. Without loss of generality, we will represent TRUE by HG_{TRUE} and FALSE by HG_{FALSE}. G_0, H_{TRUE}, HG_{TRUE}, and HG_{FALSE} are defined as follows.

$$
\begin{gathered}
G_0 = \left\{ \begin{matrix} 1 & 2 \\ \\ 2 & 1 \end{matrix} \right\} \qquad\qquad H_{TRUE} = \begin{bmatrix} 1 \\ \\ 1 \end{bmatrix} \\
HG_{TRUE} = \begin{bmatrix} 1 & / & 2 \\ \\ 2 & / & 1 \end{bmatrix} \quad HG_{FALSE} = \begin{bmatrix} 1 & / & 2 \\ \\ 1 & / & 2 \end{bmatrix}
\end{gathered}
\tag{2.4}
$$

We create a founder for all negative occurrences of each variable and a founder for all positive literals, and add appropriate offsprings according to the variable and clause gadgets in Figure 2.4 In each variable gadget, we let the positive(or negative) literal be male (or female, respectively). In each clause gadget, we carefully set each variable member's mate be the complementary gender. By this means, we can construct a pedigree in $O(m+n)$ time. Figure 2.4 also shows an example of the construction.

If and only if ϕ has a consistent assignment, there is a solution where no recombinant is needed in the variable gadgets. Moreover, if and only if some assignment has at least one literal with TRUE value in each clause, at least one founder has the grouping HG_{TRUE} in the corresponding clause gadget; so that the haplotype H_{TRUE} is passed through intermediate members to the last generation without incurring any recombinant. Then we claim that ϕ has

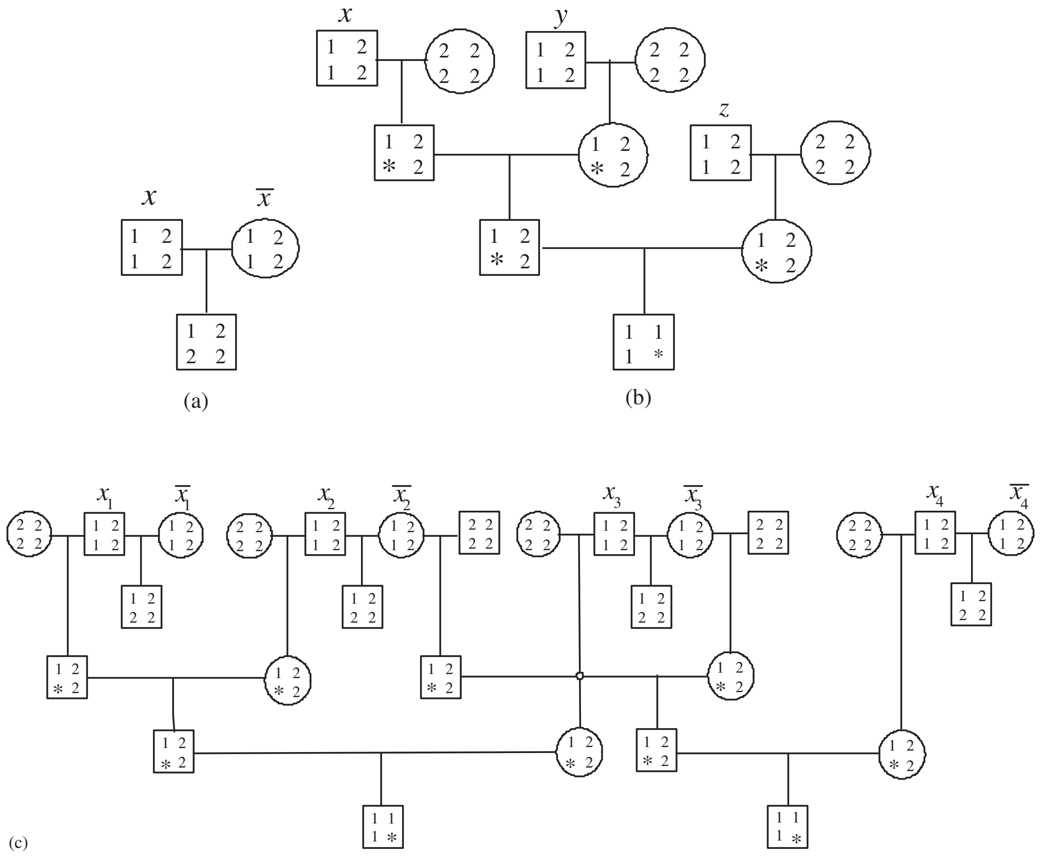

Figure 2.4: The gadgets for variables and clauses to reduce 3SAT into 2-locus-MRHC*. (a) Gadget for variable x. (b) Gadget for clause $x \vee y \vee z$. (d) An example of the reduction from 3SAT to 2-locus-MRHC* for $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee x_4)$.

a satisfiable truth assignment if and only if there is a zero-recombinant haplotype solution for the entire pedigree. This completes the proof. ■

Corollary 2.2.6 *2-locus-MRHC* cannot be approximated unless P=NP.*

Proof: This follows immediately from Lemma 2.2.1 and Theorem 2.2.5. ■

2.3 Approximation of MRHC on Pedigrees without Missing Data

In this section, we consider the approximability of the same variants of MRHC without missing data. In order to show the negative result, we need to use some gap-introducing reduction (or gap-preserving reduction) for MRHC. We will use the concept of L-reduction proposed by Papadimitriou and Yannakakis [99].

2.3.1 Approximation of Tree-MRHC

Lemma 2.3.1 *There is an L-reduction from Min UnCut to tree-MRHC that transforms a set of Boolean constraints ϕ to a tree pedigree ξ such that:*

(i) $OPT_{MinUnCut}(\phi) = OPT_{tree-MRHC}(\xi)$, and

(ii) Given a haplotype solution for ξ with k recombinants, we can construct a solution for ϕ with at most k unsatisfied clauses.

Proof: To simplify the proof, let ϕ represent both an instance of Min UnCut (or 2CNF Deletion) on a set of constraints (or clauses, respectively) and the input constraint (clause) set, ξ represent both an instance 2-locus-MRHC and also the input pedigree.

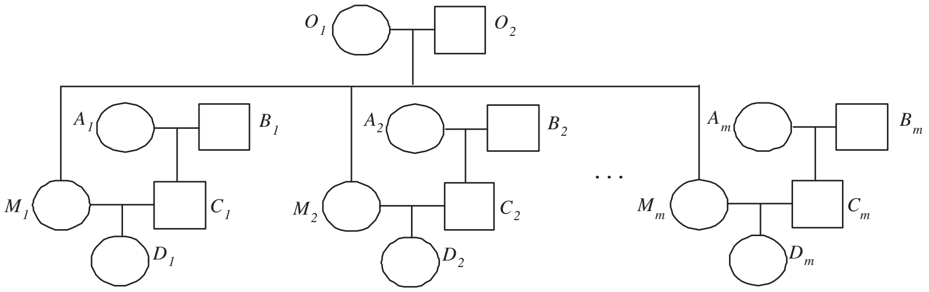

Figure 2.5: The reduction from Min UnCut to tree-MRHC.

Given an instance of Min UnCut, we assume that the Boolean constraint set ϕ has n variables and m constraints. We construct a pedigree ξ with n loci, each of which represents a variable, as illustrated in Figure 2.5 [2]. All loci in O_1, O_2, M_i, C_i and $D_i (1 \leq i \leq m)$ are heterozygous. Two haplotype configurations $HC_=$ and $HC_{\neq}$ are defined in the same way as in Theorem 2.2.2. For the i^{th} constraint of ϕ, if the constraint has the form $x_{j_1} \oplus x_{j_2} = 0$, C_i is be forced to have the configuration $HC_=$ at the j_1^{th} and the j_2^{th} loci. Otherwise, if the constraint has the form $x_{j_1} \oplus x_{j_2} = 1$, C_i is forced to have the configuration $HC_{\neq}$ at these two loci. All other loci of A_i and B_i are heterozygous.

Using the same arguments as the proofs in [14], it is easy to show this reduction is an L-reduction. The details are omitted here. ■

---

[2] The idea here is borrowed from an NP-hardness proof in Doi, Li and Jiang [14].

Theorem 2.3.2 *It is NP-hard to approximate tree-MRHC within any constant ratio under the Unique Games Conjecture [2].*

Proof: It is known NP-hard to approximate the Min UnCut problem within any constant ratio under the Unique Games Conjecture [2]. The property of L-reduction in Lemma 2.3.1 guarantees the NP-hardness of approximating tree-MRHC. ■

2.3.2 Approximation of 2-locus-MRHC

We will present a lower bound and an upper bound on the approximation ratio for the 2-locus-MRHC problem.

Negative Result for Approximating 2-locus-MRHC

Lemma 2.3.3 *There is a polynomial-time L-reduction from Min UnCut to 2-locus-MRHC that transforms a Boolean constraints set ϕ to a pedigree ξ such that*

(i) $OPT_{MinUnCut}(\phi) = OPT_{2-locus-MRHC}(\xi)$, and

(ii) Given any haplotype solution for ξ with k recombinants, we can find in polynomial time a truth assignment for ϕ with at most k unsatisfied constraints.

Proof: Given an instance ϕ of Min UnCut, we assume that the Boolean constraint set has n variables and m constraints. We create a male member (M_x) and a female member (F_x) for each variable x as the founders, and add appropriate offsprings according to the variable and constraint gadgets shown in Figure 2.6 We can construct the pedigree ξ for 2-locus-MRHC in $O(mn)$ time. We denote TRUE by HG_{TRUE} and FALSE by HG_{FALSE} in the same way as in Theorem 2.2.5.

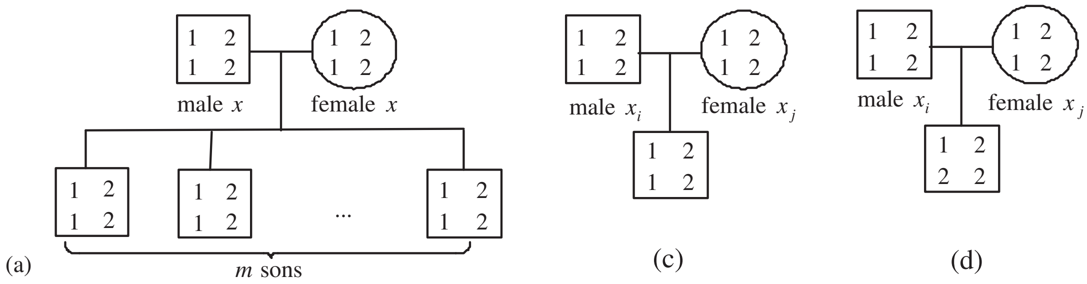

Figure 2.6: Gadgets for the polynomial-time reduction from Min UnCut to 2-locus-MRHC. (a) Gadget for variable x (b) Gadget for constraint $x_i \oplus x_j = 0$ (c) Gadget for constraint $x_i \oplus x_j = 1$.

Observe that we can let M_x and F_x have the same haplotype grouping for each variable x and obtain a haplotyope solution S with at most m recombinants in the constraint gadgets. Moreover, we claim that it is always advantageous for each pair of members M_x and F_x to have the same haplotype grouping. This is because if M_x and F_x have different haplotype groupings in some haplotype solution, each of their sons in the corresponding variable gadget would incur a recombinant and thus the variable gadget would contain a total of m recombinants. Clearly, we will a better solution by changing the haplotype grouping of M_x or F_x.

It is easy to see that we can translate in polynomial time any haplotype solution for ξ with k recombinants to a solution for ϕ with at most k unsatisfied constraints. This correspondence trivially constitutes an L-reduction. . ■

Theorem 2.3.4 *It is NP-hard to approximate 2-locus-MRHC within any constant ratio under the Unique Games Conjecture [2].*

Proof: The result follows from the above lemma and the fact that Min UnCut has no constant ratio approximation unless P = NP under the Unique Games Conjecture [2]. ■

Positive Result for Approximating 2-locus-MRHC

We first would like to reduce an instance of 2-locus-MRHC so that each member of the pedigree can be described by one Boolean variable. Since only two loci are involved, there are three types of members in a pedigree: (I) both loci are homozygous, (II) one locus is homozygous, and (III) both loci are heterozygous. A type I (or II) member has a fixed haplotype grouping. A type III member has a variable haplotype grouping.

Agarwal *et al.* recently presented a randomized polynomial-time $O(\sqrt{\log n})$ approximation algorithm for the Min 2CNF Deletion problem [3], where n is the number of variables in the input 2CNF constraints.

Lemma 2.3.5 *There is a randomized polynomial-time* $O(\sqrt{\log n})$ *approximation algorithm for 2-locus-MRHC, where* n *is the number of members in the input pedigree.*

Proof: The main idea of the proof is to transform an instance ξ of the 2-locus-MRHC problem to an instance ϕ of Min 2CNF Deletion whose solution corresponds to a haplotype solution of ξ in an equivalent way (in terms of cost). Moreover, the number of variable in ϕ

is no more than the number of members in ξ. This would give rise to an L-reduction from 2-locus-MRHC to Min 2CNF Deletion, which has an $O(\sqrt{\log n})$ approximation algorithm. In the reduction, we represent the haplotype grouping of each type III member of pedigree ξ as a Boolean variable. Clauses of size two are then used to capture all grouping combinations in a parents-child trio that would cause recombinants.

Without loss of generality, we represent grouping HG_{TRUE} by TRUE and another grouping HG_{FALSE} by FALSE in the same way as in Theorem 2.2.5. We use the following table 2.7 to show how to map each parents-child trio to a clause so that the number of recombinants is translated to the number of unsatisified clauses. In the table, alleles X and Y can be either 1 or 2, and X are Y are different.

Genotype of the Mother (***A***)	Genotype of the Father (***B***)	Genotype of the Child (***C***)	2CNF Constraint
1 2 1 2	1 2 1 2	1 1 2 2 1 1 2 2	$2(A\bigvee B)$ $(A\bigvee \overline{B})$ $(\overline{A}\bigvee B)$
		2 2 1 1 1 1 2 2	$2(\overline{A}\bigvee \overline{B})$ $(A\bigvee \overline{B})$ $(\overline{A}\bigvee B)$
		2 2 1 1 1 2 1 2 1 2 1 2 2 2 1 1	$(\overline{A}\bigvee \overline{B})$ $(A\bigvee B)$
		1 2 1 2	$(\overline{A}\bigvee C)$ $(A\bigvee \overline{C})$ $(\overline{B}\bigvee C)$ $(B\bigvee \overline{C})$
1 2 1 2	X X Y X Y X X X	1 1 2 2 1 1 2 2	A
		2 2 1 1 1 1 2 2	$\overline{A}$
		1 2 1 2	$(\overline{A}\bigvee C)$ $(A\bigvee \overline{C})$
	X X Y X	Y X X X	A
		Y X Y Y	$\overline{A}$
	Y X X X	X X X Y	A
		Y Y X Y	$\overline{A}$

Figure 2.7: Mapping a parents-child trio to clauses.

Note that, if both parents of a trio are members of types I or II, the trio cannot incur any recombinants, although we need check the Mendelian consistency of the genotypes in the trio. Hence, such trios are omitted in Figure 2.7. Let ϕ denote the 2CNF constraints consisting of all the clauses created above. Then it is easy to see that there is a one-to-one correspondence between recombinants in the pedigree and unsatisfied clauses, and the reduction is in fact an L-reduction. Hence, we have a randomized polynomial-time $O(\sqrt{\log n})$ approximation algorithm for 2-locus-MRHC. ■

Observe that the results in this section show that, in terms of approximability, the 2-locus-MRHC problem is easier than the Min 2CNF Deletion problem and harder than the Min UnCut problem. Also, Lemma 9 presents an alternative proof that 2-locus-MRHC is NP-hard, which is much easier to understand than the original proof in [28].

2.4 Approximation of MRHC(k, j)

The proof of Lemma 2.3.1 uses a pedigree that contains members with a variable number of children, although every member in the pedigree has only one mate. Can we get the same hardness result for tree-MRHC if we bound the number of mates instead of the number of children? In addition, the pedigrees in the proofs of Theorem 2.2.5 and Lemma 9 contain members with a variable number of children or mates. Another question is whether MRHC on two-locus pedigrees with a bounded number of children and mates leads to the same hardness result. In this section, we discuss the approximation of MRHC on pedigrees with bounded number of children and mates. For the convenience of comparison, we state strengthened versions of the previous theorems in the order they appear in this chapter. We use u to present an integer variable.

First, we refine Theorem 2.2.5. The hardness result in this theorem holds for 2-locus-MRHC$(u, 1)$, because some member might appear in every clause gadget and every member has at most one child in the proof of Theorem 2.2.5.

Theorem 2.4.1 *2-locus-MRHC*(4,1) cannot be approximated unless P=NP.*

Proof: According to the L-reduction from Max 3SAT to Max 3SAT(29) and the L-reduction from Max 3SAT(29) to MAX 3SAT(3) [82], deciding whether there is a satisfiable solution for 3SAT(3) is NP-hard. In the proof of Theorem 2.2.5, rather than reducing 3SAT to 2-locus-MRHC*, we can reduce 3SAT(3) to 2-locus-MRHC*. Then, in the pedigree constructed, each member has at most four mates (*i.e.* one mate in the variable gadgets and at most three mates in the clause gadgets). The construction of the pedigree introduces at most one child for each member with each mate. Hence, Theorem 2.2.5 can be tightened to 2-locus-MRHC*(4,1)). ■

Next, let us look at Lemma 2.3.1. This lemma actually works for tree-MRHC(1,u). It is natural to consider tree-MRHC on pedigrees where members have a bounded number of children with each mate. In order to decrease the number of children and mates in the pedigree, we need a bounded version of Min UnCut like the one for Max 3SAT.

In fact, there is an L-reduction from Min UnCut to Min UnCut(15) that transforms a Boolean constraints set ϕ to another Boolean constraints set ψ such that

(i) $OPT_{UnCut}(\phi) = OPT_{UnCut}(15)(\psi)$, and

(ii) Given any truth assignment for ψ with k unsatisfied constraints, we can find in polynomial time a truth assignment for ϕ with at most k unsatisified constraints.

This L-reduction from Min UnCut to Min UnCut(15) can be constructed using the same idea as the L-reduction that transforms Max 3SAT to Max 3SAT(29) in [82] with just a few minor modifications. The details of this L-reduction are omitted here. Based on the property of this L-reduction, we know that it is NP-hard to approximate Min UnCut(15) within any constant ratio under the Unique Games Conjecture [2].

Theorem 2.4.2 *It is NP-hard to approximate tree-MRHC(u,1) within any constant ratio under the Unique Games Conjecture [2].*

Proof: Given an instance ϕ of Min UnCut(15) with n variable and m constraints, we construct a pedigree ξ as illustrated in Figure 2.8 We use the same notation and idea to construct every member as those in Theorem 2.2.2. All M_i's children are designed to have the same genotype as M_i. Members C_1, C_2,..., C_t are used to check the consistency of each variable. C_{t+j} $(1 \leq i \leq m)$ is designed to check the satisfiability of the j^{th} constraint. Using

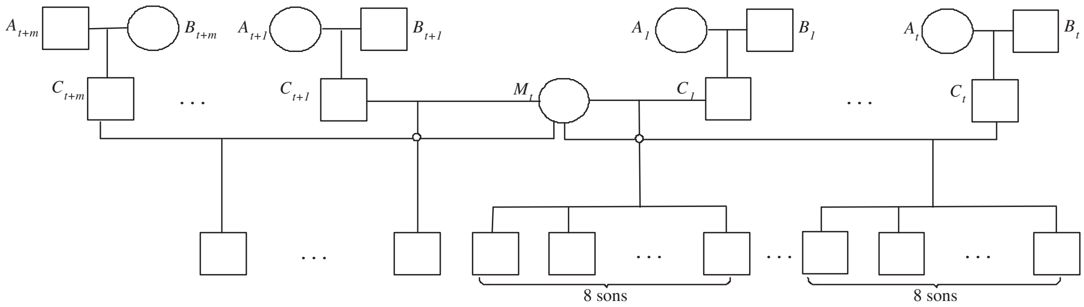

Figure 2.8: The reduction from UnCut(15) to tree-MRHC(u,8). It's easy to see that the approximability result still works for tree-MRHC(u,1).

the arguments similar to those in Lemma 9, we can prove that an optimal truth assignment for the Boolean constraints ϕ can be mapped to an optimal haplotype solution for the pedigree ξ with the same cost and vice versa. Given a haplotype solution for ξ with k recombinants, we can construct a solution for ϕ with at most k unsatisfied clauses easily. In other words, the above is an L-reduction. The details are omitted here. Hence, the approximability result holds tree-MRHC(u,8). In the above reduction, tree-MRHC(u,8) can be further tightened to tree-MRHC(u,1). We can create 8 new mates with only one child with each mate to replace C_i ($1 \leq i \leq m$) and its 8 children. These 8 new mates are duplicates of the old C_i. ■

Finally, we consider Lemma 9. The hardness result actually holds for 2-locus-MRHC(u, u), because neither the number of mates nor the number of children for a member is bounded by any constant.

Theorem 2.4.3 *It is NP-hard to approximate 2-locus-MRHC(16,15) within any constant ratio under the Unique Games Conjecture [2].*

Proof: Similar to the proof of Lemma 9. But we reduce from Min UnCut(15) instead of Min UnCut. Each member in the constructed pedigree has at most 16 mates (*i.e.* one mate in the variable gadgets and at most 15 mates in the clause gadgets) and 15 children with each mate. The details are omitted here. ■

2.5 Discussion

The results presented in this chapter are organized in Table 2.9.

First, we showed that binary-tree-MRHC is NP-hard. Binary-tree-MRHC is a simplest variant of MRHC because one mate and one child are the minimum requirement to express the inheritance of human beings.

Second, we showed some approximability results concerning the MRHC problem. With the presence of missing data, it is NP-hard to tell if an instance of 2-locus-MRHC* and binary-tree-MRHC* requires any recombinant. This gives an interesting contrast to the results in [28] where the problem of finding a zero-recombinant haplotype solution for MRHC was shown to be solvable in polynomial time. This result also implies that 2-locus-MRHC* and binary-tree-MRHC* is not approximable in polynomial time. Without the presence of missing data, 2-locus-MRHC can be approximated with the ratio $O(\sqrt{\log n})$. In addition, it is NP-hard to approximate 2-locus-MRHC and tree-MRHC within any constant ratio under the Unique Games Conjecture [2].

Our final results concern the inapproximability of MRHC on pedigrees where each member has a bounded number of mates and/or a bounded number of children with each mate.

	Loop ?	Missing data?	Unbounded Number of loci?	Unbounded number of members?	Hardness	Lower bound of approx. ratio	Assumption	The lower bound holds for	Upper bound of approx. ratio
Binary-tree-MRHC	No	No	Yes	Yes	NP				
2-locus-MRHC*	Yes	Yes	No	Yes		Any f(n)	P≠NP	2-locus-MRHC* (4,1)	
Binary-tree-MRHC*	No	Yes	Yes	Yes		Any f(n)	P≠NP	Binary-tree-MRHC*	
2-locus-MRHC	Yes	No	No	Yes		Any constant	P≠NP, the Unique Games Conjecture	2-locus-MRHC (16,15)	$O(\sqrt{\log(n)})$
Tree-MRHC	No	No	Yes	Yes		Any constant	P≠NP, the Unique Games Conjecture	Tree-MRHC(1,u) Tree-MRHC(u,1)	

Figure 2.9: Our hardness and approximation results for MRHC with bi-alleles.

Chapter 3

Fast Elimination of Redundant Linear Equations and Reconstruction of Recombination-Free Mendelian Inheritance on a Pedigree

This chapter is concerned with the question of how to efficiently infer haplotypes from genotypes of individuals related by a pedigree, assuming that the hereditary process was free of mutations (*i.e.* the Mendelian law of inheritance) and recombinants.

---

[0] An extended abstract of this paper, entitled "Fast elimination of redundant linear equations and reconstruction of recombination-free Mendelian inheritance on a pedigree", has appeared in *Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Jan. 2007, New Orleans, LA [46].

3.1 Introduction

First we briefly review the definition of the ZRHC problem. The ZRHC problem can also be stated abstractly as a simple inheritance reconstruction problem as follows. We have a pedigree connecting n individuals where each individual j has two haplotypes (*i.e.* strings) defined on m marker loci $a_{j,1} \cdots a_{j,m}$ and $b_{j,1} \cdots b_{j,m}$ inherited from j's father and mother, respectively. That is, if the parents of j are individuals j_1 and j_2, then $a_{j,1} \cdots a_{j,m} \in \{a_{j_1,1} \cdots a_{j_1,m}, b_{j_1,1} \cdots b_{j_1,m}\}$ and $b_{j,1} \cdots b_{j,m} \in \{a_{j_2,1} \cdots a_{j_2,m}, b_{j_2,1} \cdots b_{j_2,m}\}$. The haplotypes are unknown, but the genotype of each individual j is given to us in the form of string $\{a_{j,1}, b_{j,1}\} \cdots \{a_{j,m}, b_{j,m}\}$. We would like to reconstruct all the haplotype solutions that could have resulted in the genotypes. The formal definition for the ZRHC problem is given in section 1.2.

Li and Jiang presented an $O(m^3n^3)$ time algorithm for ZRHC by formulating it as a system of $O(mn)$ linear equations with mn variables over the finite field of $F(2)$ and applying Gaussian elimination [28]. Although this cubic time algorithm is reasonably fast, it is inadequate for large scale pedigree analysis where both m and n can be in the order of tens or even hundreds, and we may have to examine many pedigrees and haplotype blocks. There are, for example, over five million SNP markers in the public database dbSNP [45]. This challenge motivates us to find more efficient algorithms for ZRHC. Several attempts have been made recently in [12, 32], but the authors failed to prove the correctness of their algorithms in all cases, especially when the input pedigree has mating loops. Chan *et al.* proposed a linear-time algorithm in [11], but the algorithm only works for pedigrees without mating loops (*i.e.* the tree pedigrees).

3.1.1 Our Result

In this chapter, we present a much faster algorithm for ZRHC with running time $O(mn^2 + n^3 \log^2 n \log\log n)$. Our construction begins with a new system of linear equations over $F(2)$ for ZRHC. Although the system still has $O(mn)$ variables and $O(mn)$ equations, we prove that it can be reduced to an effectively equivalent system with $O(mn)$ equations and at most $2n$ variables, by exploring the underlying pedigree graph structure. By using standard Gaussian elimination, this already gives an improved algorithm with running time $O(mn^3)$. We then show how to reduce the number of equations further to $O(n \log^2 n \log\log n)$ (assuming that $m \geq \log^2 n \log\log n$, which usually holds in practice), by giving an $O(mn)$ time method for eliminating redundant equations in the system. Although such a fast elimination method is not known for general systems of linear equations, we again take advantage of the underlying pedigree graph structure and recent progress on low-stretch spanning trees in [17]. In particular, the low-stretch spanning tree result helps upper bound the number of equations that need be kept in the elimination process. We also show that our algorithm actually runs in time $O(mn^2 + n^3)$ time when the input pedigree is a tree pedigree with no mating loops (which is often true for human pedigrees) or when there is a locus that is heterozygous across the entire pedigree. Moreover, our algorithm produces a general solution to the original system of linear equations at the end that represents all feasible solutions to the ZRHC problem.

3.1.2 Related Work on Solving Systems of (Sparse) Linear Equations.

The search for efficient algorithms for solving systems of linear equations is a classical problem in linear algebra. Besides Gaussian elimination, methods based on fast matrix multiplication algorithms have been proposed and could achieve an asymptotic speed of $O(n^{2.376})$ on n equations with n unknowns [16, 40]. However, these methods are only of theoretical interest since they are hard to implement and do not outperform Gaussian elimination unless n is very large. Moreover, they assume that the coefficient matrix is of full rank, which is an unreasonable assumption in ZRHC (considering the linear systems derived for ZRHC so far).

Observe that the linear system given in [28] for ZRHC is actually very sparse since each of its equations has at most four variables. Thus, a plausible way to speed-up is to utilize fast algorithms for solving sparse linear systems. The Lanczos and conjugate gradient algorithms [24], and the Wiedemann algorithm [43] are some of the best known algorithms for solving sparse linear system over finite fields. The Wiedemann algorithm runs in (expected) quadratic time (which is in fact slower than our algorithm when applied to linear systems for ZRHC) while the Lanczos and conjugate gradient algorithms are only heuristics [34]. However, they use randomization and do not find all solutions. Furthermore, the algorithms cannot check if the system has no solution [23]. A randomized algorithm with quadratic expected time for certifying inconsistency of linear systems is given in [23].

3.1.3 Organization of the Chapter

The rest of our chapter is organized as follows. We will describe a new system of linear equations for ZRHC and some useful graphs derived from a pedigree in section 3.2. The $O(mn^3)$ time algorithm is presented in section 3.3, and the $O(mn^2 + n^3 \log^2 \log \log n)$ time algorithm is given in section 3.4. Section 3.5 contains an example execution of the main algorithm. Some concluding remarks are given in section 3.6. [1]

3.2 A System of Linear Equations for ZRHC and the Pedigree Graph

In this section, we first present a new formulation of ZRHC in terms of linear equations, and define some graph structures which will be used in our algorithm.

3.2.1 The Linear System

Throughout this chapter, n denotes the number of the individuals (or members) in the input pedigree and m the number of marker loci.

---

[1] Note that the results in sections 3.2 and 3.3 were mostly due to Jing Xiao's contribution and have appeared in his M.Sc. thesis. We include them here for the sake of completeness.

Without loss of generality, suppose that each allele in the given genotypes is numbered numerically as 1 or 2 (*i.e.* the markers are assumed to be *bi-allelic*, which make the hardest case for MRHC/ZRHC[28]), and the pedigree is free of genotype errors (*i.e.* the two alleles at each locus of a child can always be obtained from her respective parents). Hence, we can represent the genotype of member j as a ternary vector $\mathbf{g}_j$ as follows: $g_j[i] = 0$ if locus i of member j is homozygous with both alleles being 1's, $g_j[i] = 1$ if the locus is homozygous with both alleles being 2's, and $g_j[i] = 2$ otherwise (*i.e.* the locus is heterozygous).

For any heterozygous locus i of member j, we use a binary variable $p_j[i]$ to denote the *phase* at the locus as follows: $p_j[i] = 0$ if allele 2 is paternal, and $p_j[i] = 1$ otherwise. When the locus is homozygous, the variable is set to $g_j[i]$ for some technical reasons (so that the equations below involving $p_j[i]$ will hold). Hence, the vector $\mathbf{p}_j$ describes the paternal and maternal haplotypes of member j. Observe that the vectors $\mathbf{p}_1, \ldots, \mathbf{p}_n$ represent a complete haplotype configuration of the pedigree. In fact, the sparse linear system in [28] was based on these vectors.

Also for technical reasons, define a vector $\mathbf{w}_j$ for member j such that $w_j[i] = 0$ if its i-th locus is homozygous and $w_j[i] = 1$ otherwise.

Suppose that member j_r is a parent of member j. We introduce an auxiliary binary variable $h_{j_r,j}$ to indicate which haplotype of j_r is passed to j. If j_r gives its paternal haplotype to j, then $h_{j_r,j} = 0$; otherwise $h_{j_r,j} = 1$. Suppose that j is a non-founder member with her father and mother being j_1 and j_2, respectively. We can define two linear (constraint) equations over $F(2)$ to describe the inheritance of paternal and maternal haplotypes at j respectively, following the Mendelian law of inheritance and zero-recombinant assumption:

$$\mathbf{p}_{j_1} + h_{j_1,j} \cdot \mathbf{w}_{j_1} = \mathbf{p}_j \qquad \text{and} \qquad \mathbf{p}_{j_2} + h_{j_2,j} \cdot \mathbf{w}_{j_2} = \mathbf{p}_j + \mathbf{w}_j \tag{3.1}$$

If we let $\mathbf{d}_{j_1,j}$ be the vector $\mathbf{0}$ and $\mathbf{d}_{j_2,j} = \mathbf{w}_j$ otherwise, then the above equations can be unified into a single equation as:

$$\mathbf{p}_{j_r} + h_{j_r,j} \cdot \mathbf{w}_{j_r} = \mathbf{p}_j + \mathbf{d}_{j_r,j} \qquad (r = 1,\ 2) \tag{3.2}$$

Formally, we can express the ZRHC problem as a system of linear equations:

$$\begin{cases} p_k[i] + h_{k,j} \cdot w_k[i] = p_j[i] + d_{k,j}[i] & 1 \le i \le m,\ 1 \le j, k \le n, k \text{ is a parent of } j \\ p_j[i] = g_j[i] & 1 \le i \le m,\ 1 \le j \le n,\ g_j[i] \ne 2 \\ w_j[i] = 1 & 1 \le i \le m,\ 1 \le j \le n,\ g_j[i] = 2 \\ w_j[i] = 0 & 1 \le i \le m,\ 1 \le j \le n,\ g_j[i] \ne 2 \\ d_{k,j}[i] = w_j[i] & 1 \le i \le m,\ 1 \le j, k \le n,\ k \text{ is the mother of } j \\ d_{k,j}[i] = 0 & 1 \le i \le m,\ 1 \le j, k \le n,\ k \text{ is the father of } j \end{cases} \tag{3.3}$$

where $g_j[i], w_j[i], d_{k,j}[i]$ are all constants depending on the input genotypes, and $p_j[i], h_{k,j}$ are the unknowns. Note that, the number of p-variables is exactly mn and the number of h-variables is at most $2n$ since every child has two parents and there are at most n children in the pedigree.

Remark: Observe that for any member j, if the member itself or one of its parents are homozygous at locus i, $p_j[i]$ is fixed based on Equation (3.3). In the rest of the chapter, we will assume that all such variables $p_j[i]$ are *pre-determined* (without any conflict), and use them as "anchor points" to define some new constraints about the h-variables.

3.2.2 The Pedigree Graph and Locus Graphs

To conform with standard graph theory notations, we transform the input pedigree into a graph, called the *pedigree graph*, by connecting each parent directly to her children, as shown in Figure 3.1 (B). Although the edges in the pedigree represent the inheritance relationship between a parent and a child and are directed, we will think of the pedigree graph, and more importantly, the subsequent locus graphs, as undirected in future definitions and constructions, since each edge (j, k) of the pedigree graph (and locus graphs) will be used to represent the constraint between the vectors $\mathbf{p}_j$ and $\mathbf{p}_k$ (*i.e.* the phases at j and k) via the variable $h_{j,k}$, which is symmetric as can be seen from Lemma 3.2.1 and Corollaries 3.2.2 and 3.2.3 below. [2]

Clearly, such a pedigree graph $G = (V, E)$ may be cyclic due to mating loops or multiple children shared by a pair of parents. Let $\mathcal{T}(G)$ be any spanning tree of G. $\mathcal{T}(G)$ partitions the edge set E into two subsets: *tree edges* and *non-tree edges*. For simplicity, the non-tree edges will be called *cross edges*. Let $E^{\mathbb{X}}$ denote the set of cross edges. Since $|E| \leq 2n$ and the number of edges in $\mathcal{T}(G)$ is $n-1$, we have $|E^{\mathbb{X}}| \leq n+1$. Figure 3.1 (B) gives an example of tree edges and cross edges.

---

[2] The reader can also verify that the direction of an edge will not affect the graph traversal and the ensuing treatment of constraint equations to be discussed in the next two sections.

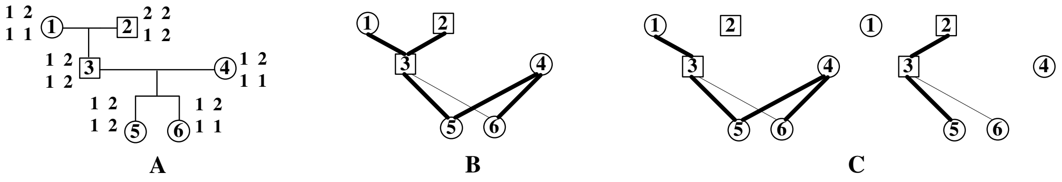

Figure 3.1: An example pedigree with genotype data and the corresponding locus graphs. (A). An example pedigree with genotype data. Here, the alleles at a locus are ordered according to their id numbers instead of phase (which is unknown). (B). The pedigree graph with a spanning tree. The tree edges are highlighted. Observe that there lies a *cycle* of length 4 in the given tree pedigree graph. (C). The locus graphs. The left graph is for the first locus, which has a cycle, while the right graph is for the second locus. The locus forests are highlighted.

For any fixed locus i, the value $w_k[i]$ can be viewed as the weight of each edge $(k, j) \in E$, where k is a parent of j. We construct the *i-th locus graph* G_i as the subgraph of G induced by the edges with weight 1. Formally, $G_i = (V, E_i)$, where $E_i = \{(k, j) |\ k \text{ is a parent of } j, w_k[i] = 1\}$. The i-th locus graph G_i induces a subgraph of the spanning tree $\mathcal{T}(G)$. Since the subgraph is a forest, it will be referred to as the *i-th locus forest* and denoted by $\mathcal{T}(G_i)$. Figure 3.1 (C) shows the locus graphs and the loucs forests of the given pedigree.

The locus graphs can be used to identify some implicit constraints on the h-variables as follows. First, we need "symmetrizing" the h-variables and d-constants: for any edge $(k, j) \in E$, define $h_{k,j} = h_{j,k}$ and $\mathbf{d}_{k,j} = \mathbf{d}_{j,k}$.

Lemma 3.2.1 *For any path $\mathcal{P} = j_0, \ldots, j_k$ in locus graph G_i connecting vertices j_0 and j_k, we have*

$$p_{j_0}[i] + p_{j_k}[i] + \sum_{r=0}^{k-1} (h_{j_r, j_{r+1}} + d_{j_r, j_{r+1}}[i]) = 0$$

Proof: The equation follows easily from Equation (3.3) and an induction on the length k of the path. ■

Note that the above constraint remains the same no matter in which direction path $\mathcal{P}$ is read, since the addition is over field $F(2)$ and the h-variables and d-constants are symmetric. From the lemma, we can see that for a cycle in G_i, the summation of all the h-variables corresponding to the edges on the cycle is a constant. The constant is said to be *associated* with the cycle.

Corollary 3.2.2 *For any cycle $\mathcal{C} = j_0, \ldots, j_k, j_0$ in G_i, there exists a binary constant b defined as $b = \sum_{r=0}^{k} d_{j_r, j_{r+1 \bmod k+1}}[i]$ such that $\sum_{r=0}^{k} h_{j_r, j_{r+1 \bmod k+1}} = b$.*

Proof: This follows from Lemma 3.2.1 and the fact that $p_{j_0}[i] + p_{j_0}[i] = 0$. ■

From Lemma 3.2.1, we can easily see that if the p-variables at the endpoints of a path are pre-determined, then the summation of all the h-variables corresponding to the edges on the path is a constant. The constant is said to be *associated* with the path. We construct constraints on h-variables as follows. Again, notice that the following constant b does not depend on the direction that path $\mathcal{P}$ is read.

Corollary 3.2.3 *Suppose that $\mathcal{P} = j_0, \ldots, j_k$ is a path in G_i connecting vertices j_0 and j_k, and the variables $p_{j_0}[i]$ and $p_{j_k}[i]$ are pre-determined. Then there exists a binary constant b defined as $b = p_{j_0}[i] + p_{j_k}[i] + \sum_{r=0}^{k-1} d_{j_r, j_{r+1}}[i]$ such that $\sum_{r=0}^{k-1} h_{j_r, j_{r+1}} = b$.*

3.3 An $O(mn^3)$ Time Algorithm for ZRHC

Since the number of h-variables is at most $2n$, our key idea is to first derive a system of $O(mn)$ linear equations on the h-variables, using paths and cycles in G_i and the pre-determined p-variables as mentioned in the last section, and then find a *general* solution to the system by using Gaussian elimination so that all inherent freedom in Equation (3.3) is kept. This new system of equations about the h-variables is clearly necessary for Equation (3.3). The crux of the construction is to show that it is also sufficient, and thus the p-variables can be determined from the values of the h-variables by a simple traversal of the locus graphs.

3.3.1 Linear Constraints on the h-Variables

We will introduce constraint equations to "cover" all the edges in each locus graph. As mentioned above, these equations connect the p-variables and will suffice to help determine their values. Note that, since the edges broken in each locus graph involve pre-determined p-variables, we do not have to introduce constraints to cover them. The constraints can be classified into two categories with respect to the spanning tree $\mathcal{T}(G)$: constraints for cross edges and constraints for tree edges.

Cross Edge Constraints. Adding a cross edge e to the spanning tree $\mathcal{T}(G)$ yields a cycle $\mathcal{C}$ in the pedigree graph G. Let $length(\mathcal{C})$ denote the length of cycle $\mathcal{C}$. Suppose that the edge e exists in the i-th locus graph G_i, and consider two cases of the cycle $\mathcal{C}$ with respect to G_i.

```
Procedure CROSS_EDGE_CONSTRAINTS
input: locus graphs $G_{[1..m]}$ and the spanning tree $\mathcal{T}(G)$
output: cross edge constraint sets $C^{\mathbb{C}}, C^{\mathbb{P}}$
begin
  $C^{\mathbb{C}} = C^{\mathbb{P}} = \emptyset$;
  for each cross edge $e$
    Suppose that $\mathcal{C}$ is the cycle in $\mathcal{T}(G) \cup \{e\}$;
    $C^{\mathbb{P}}(e) = C^{\mathbb{C}}(e) = \emptyset$;
    for each locus $i$
      if $\mathcal{C}$ is connected in $G_i$
        Let $b = \sum_{( k,j)\in \mathcal{C}} d_{k,j}[i]$;
        $C^{\mathbb{C}}(e) = C^{\mathbb{C}}(e) \uplus \{(e, b)\}$;
      else
        Suppose $\mathcal{P}$ is the path containing $e$ in $\mathcal{T}(G_i) \cup \{e\}$;
        Let vertices $j_1$ and $j_2$ be the endpoints of $\mathcal{P}$;
        Define $b = p_{j_1}[i] + p_{j_2}[i] + \sum_{( k,j)\in\mathcal{P}} d_{k,j}[i]$;
        $C^{\mathbb{P}}(e) = C^{\mathbb{P}}(e) \uplus \{(j_1, j_2, b, e)\}$;
    $C^{\mathbb{C}} = C^{\mathbb{C}} \uplus C^{\mathbb{C}}(e)$;
    $C^{\mathbb{P}} = C^{\mathbb{P}} \uplus C^{\mathbb{P}}(e)$;
end.

Procedure TREE_EDGE_CONSTRAINTS
input: locus forests $\mathcal{T}(G_{[1..m]})$ and a fixed tree edge $e_0$
output: tree edge constraint set $C^{\mathbb{T}}$
begin
  $C^{\mathbb{T}} = \emptyset$;
  for each locus $i$
    for each connected component $\mathcal{T}$ in $\mathcal{T}(G_i)$
      if $\mathcal{T}$ has no pre-determined vertices
        Arbitrarily pick a vertex $j_0$ in $\mathcal{T}$ as the seed of $\mathcal{T}$;
      else
        Arbitrarily pick a pre-determined vertex $j_0$ in $\mathcal{T}$ as
          the seed of $\mathcal{T}$;
      for each pre-determined vertex $j_1 \neq j_0$ in $\mathcal{T}$
        Let $\mathcal{P}$ be the path between $j_0$ and $j_1$;
        Define $b = p_{j_0}[i] + p_{j_1}[i] + \sum_{(k,j)\in\mathcal{P}} d_{k,j}[i]$;
        $C^{\mathbb{T}} = C^{\mathbb{T}} \uplus \{(j_0, j_1, b, e_0)\}$;
end.
```

Figure 3.2: The procedure for generating constraints.

In order to save running time, we use disjoint union (*i.e.* $\uplus$) in algorithms ZRHC_PHASE and IMPROVED_ZRHC_PHASE.

```
Algorithm ZRHC_PHASE
          [IMPROVED ZRHC_PHASE]
input: pedigree $G = (V, E)$ and genotype $\{\mathbf{g}_j\}$
output: a general solution of $\{\mathbf{p}_j\}$
begin
  Step 1. Preprocessing
    Construct a [low-stretch] spanning tree $\mathcal{T}(G)$ on $G$;
    Let $e_0$ be an arbitrary tree edge;
    for each locus $i$
      Generate the locus graph $G_i$;
      Generate the locus forest $\mathcal{T}(G_i)$;
      Identify the pre-determined nodes;

  Step 2. Constraint generation
    CROSS_EDGE_CONSTRAINTS($G_{[1..m]}, \mathcal{T}(G), C^{\mathbb{C}}, C^{\mathbb{P}}$);
    TREE_EDGE_CONSTRAINTS($\mathcal{T}(G_{[1..m]}), C^{\mathbb{T}}, e_0$ );

 [ Step 2′. Redundant Constraint Elimination
   COMPACT_CONSTRAINTS( $C^{\mathbb{C}}, C^{\mathbb{P}}, C^{\mathbb{T}}, e_0$ ); ]

  Step 3. Solve the $h$-variables
    Apply Gaussian elimination on $C^{\mathbb{C}} \biguplus C^{\mathbb{P}} \biguplus C^{\mathbb{T}}$
      to get a general solution of the $h$-variables;

  Step 4. Solve the $p$-variables by propagation
    for each locus $i$
      for each connected component $\mathcal{T}$ in $\mathcal{T}(G_i)$
        if $\mathcal{T}$ has no pre-determined vertices
          Set the $p$-variable of the seed as a free variable and
            treat it as a determined value;
        Traverse $\mathcal{T}$ by BFS starting from the seed;
          for each edge $(j, k)$ in $\mathcal{T}$
            if $p_j[i]$ is determined but $p_k[i]$ is undetermined
              $p_k[i] = p_j[i] + h_{j,k} + d_{j,k}[i]$;
  return $\{\mathbf{p}_j\}$;
end.
```

Figure 3.3: The $O(mn^3)$ time algorithm ZRHC_PHASE, and the $O(mn^2 + n^3 \log^2 n \log\log n)$ time algorithm IMPROVED_ZRHC_PHASE.

The additional instructions in IMPROVED_ZRHC_PHASE are highlighted by bold font in square brackets.

Case 1: The cycle exists in G_i. We introduce a constraint along the cycle as in Corollary 3.2.2. This constraint is called a *cycle constraint*. The set of such cycle constraints for edge e in all locus graphs is denoted by $C^{\mathbb{C}}(e)$, *i.e.*

$$C^{\mathbb{C}}(e) = \{(b, e) \mid b \text{ is associated with the cycle in } \mathcal{T}(G_i) \cup \{e\}, 1 \leq i \leq m\}$$

The set of cycle constraints for all cross edges is denoted by $C^{\mathbb{C}} = \biguplus_{e \in E^{\mathbb{X}}} C^{\mathbb{C}}(e)$.

Case 2: Some of the edges of the cycle do not exist in G_i. This means that the cycle $\mathcal{C}$ is broken into several disjoint paths in G_i by the pre-determined vertices. Since e exists in G_i, exactly one of these paths, denoted as $\mathcal{P}$, contains e. Observe that both endpoints of $\mathcal{P}$ are pre-determined and thus Corollary 3.2.3 could give us a constraint concerning the h-variables along the path. Such a constraint will be called a *path constraint*. The set of such path constraints for e in all locus graphs G_i is denoted by $C^{\mathbb{P}}(e)$, *i.e.*

$$C^{\mathbb{P}}(e) = \left\{ (k, j, b, e) \;\middle|\; \begin{array}{l} \text{in } \mathcal{T}(G_i) \cup \{e\}, b \text{ is associated with the path containing } e \\ \text{connecting two pre-determined vertices } k \text{ and } j, 1 \leq i \leq m \end{array} \right\}$$

The set of path constraints for all cross edges is denoted by $C^{\mathbb{P}} = \biguplus_{e \in E^{\mathbb{X}}} C^{\mathbb{P}}(e)$.

Tree Edge Constraints. By Corollary 3.2.3, there is an implicit constraint concerning the h-variables along each path between two pre-determined vertices in the same connected component of $\mathcal{T}(G_i)$. Therefore, for each connected component $\mathcal{T}$ of $\mathcal{T}(G_i)$, we arbitrarily pick a pre-determined vertex in the component as the *seed* vertex, and generate a constraint for the unique path in $\mathcal{T}(G_i)$ between the seed and each of the other pre-determined vertices in the component, as in Corollary 3.2.3. Such a constraint will be called a *tree constraint*. Notice that if there exists any component having no pre-determined vertices, then locus i must be heterozygous across the entire pedigree and $\mathcal{T}(G_i)$ is actually a spanning tree. Such a locus will be referred to as an *all-heterozygous* locus. For such a locus i, we arbitrarily pick a vertex in $\mathcal{T}(G_i)$ as the seed, but will not generate at tree constraints.

To conform with the notation of path constraints and for the convenience of presentation, we arbitrarily pick a tree edge denoted as e_0, and write the set of tree constraints at all loci as

$$C^{\mathbb{T}} = \left\{ (k, j, b, e_0) \;\middle|\; \begin{array}{l} \text{in a connected component of } \mathcal{T}(G_i) \text{ with seed } k, b \text{ is associated with} \\ \text{the path connecting vertices } k \text{ and a predetermined vertex } j, 1 \leq i \leq m \end{array} \right\}$$

Note that e_0 is the same for all the tree constraints, and will be used as an *indictor* to distinguish tree constraints from path constraints defined by cross edges.

Again, we need symmetrize path constraints and tree constraints: given any constraint (k, j, b, e) generated for a path connecting two pre-determined vertices k and j in a locus graph, define $(k, j, b, e) = (j, k, b, e)$. The above constructions of $C^{\mathbb{C}}$, $C^{\mathbb{P}}$ and $C^{\mathbb{T}}$ are more formally described as pseudo-code in Figures 3.2. We can easily see that

Lemma 3.3.1 $|C^{\mathbb{C}}| + |C^{\mathbb{P}}| + |C^{\mathbb{T}}| = O(mn)$.

3.3.2 Solving the Linear System for ZRHC Using the New Constraints

We now describe how to solve the system in Equation (3.3) in $O(mn^3)$ time. The pseudocode for solving the system is formally given as algorithm **ZRHC_phase** in Figure 3.3.

Here, we first construct the cycle, path and tree constraints on the h-variables, and pick a vertex as the seed for every connected component in the locus forests $\mathcal{T}(G_i)$, as described in the last subsection.

Then we solve these constraints by using Gaussian elimination to obtain a general solution of the h-variables, which may contain some *free* h-variables.

Next, for each connected component with no pre-determined vertices, we set the p-variable of the seed as a *free* variable and treat it as a determined value.

Finally, we perform a (*e.g.* breadth-first) traversal on the spanning forest $\mathcal{T}(G_i)$ of each locus graph G_i. For each connected component of $\mathcal{T}(G_i)$, we start from the seed and propagate its p-variable value to the undetermined vertices in the component by using the solution for the h-variables, which will result in functions of the *free* h-variables and at most one *free* p-variable.

Note that in the last step of the algorithm, $p_k[i]$ is expressed as a linear combination of the free variables in $p_j[i]$ and the free h-variables with an appropriate constant term.

To show the correctness of the algorithm, we need only show that the solution found by the algorithm is a feasible solution for Equation (3.3) and vice the versa. Since we determine the p-variables based on the linear system for the h-variables derived from Equation (3.3), any feasible solution to Equation (3.3) will be included in the (general) solution found by our algorithm. In other words, we do not lose any degrees of freedom in the solution process. Hence, it suffices to show that our solution satisfies Equation (3.3).

Lemma 3.3.2 *The p-variables and h-variables determined by algorithm* ***ZRHC_Phase*** *satisfy the linear system in Equation (3.3).*

Proof: Denote the solution found by our algorithm as $\mathbf{p}$ and $\mathbf{h}$. We need only care about the first equations in Equation (3.3). If $w_k[i]$ in the equation is 0, the equation only involves two pre-determined p-values, which holds trivially since step 1 of our algorithm explicitly takes care of pre-determined p-values. Otherwise, each such equation corresponds to an edge in some locus graph. Let $e = (j_1, j_2)$ be an edge in locus graph G_i. It represents an equation $p_{j_1}[i] + h_{j_1,j_2} = p_{j_2}[i] + d_{j_1,j_2}[i]$, *i.e.* the first equation on edge (j_1, j_2) in Equation (3.3).

Given any two vertices j_s and j_t in a same connected component of $\mathcal{T}(G_i)$, we denote by $\mathcal{P}(j_s, j_t)$ the unique path in the component connecting j_s and j_t. Suppose that vertex j_0 is the seed of the component. The key observation here is that for any vertex j_t, no matter vertex j_t is pre-determined or not, our solution satisfies

$$p_{j_t}[i] \;=\; p_{j_0}[i] \;+ \sum_{(k,\;j)\,\in\,\mathcal{P}(j_0,j_t)} (h_{k,j} + d_{k,j}[i]) \tag{3.4}$$

More precisely, if vertex j_t is pre-determined, Equation (3.4) holds because of the tree constraint for path $\mathcal{P}(j_0, j_t)$ defined in step 2 of the algorithm; otherwise, $p_{j_t}[i]$ is set to be the value in Equation (3.4) during the traversal from the seed j_0 to j_t in step 4. More generally, we can build a relationship between the p-values of two vertices j_s and j_t from Equation (3.4) as follows:

$$p_{j_t}[i] \;=\; p_{j_s}[i] \;+ \sum_{(k,\ j)\ \in\ \mathcal{P}(j_t,\, j_s)} (h_{k,j} + d_{k,j}[i]) \tag{3.5}$$

Notice that, as long as j_s and j_t are in the same connected component of $\mathcal{T}(G_i)$, the above equation is satisfied by our solution. With the help of Equation (3.5), we now show that our solution satisfies all equations represented by tree edges and cross edges.

Case 1: e is a tree edge. Clearly, vertices j_1 and j_2 belong to the same connected component of $\mathcal{T}(G_i)$. We replace j_s and j_t by j_1 and j_2 respectively in Equation (3.5) and obtain $p_{j_1}[i] \;=\; p_{j_2}[i] \;+ \sum_{(k,\ j)\ \in\ \mathcal{P}(j_1,\, j_2)} (h_{k,j} + d_{k,j}[i]) \;=\; p_{j_2}[i] \;+ h_{j_1,j_2} + d_{j_1,j_2}[i]$, which means that our solution satisfies the equation represented by the tree edge e.

Case 2: e is a cross edge. Denote by $\mathcal{C}$ the cycle in $\mathcal{T}(G_i) \cup \{e\}$. There are two subcases.

Case 2.1: $\mathcal{C}$ exists in G_i. In this case, we have a cycle constraint for $\mathcal{C}$ in G_i. Suppose that the cycle constraint is $0 = \sum_{(k,j)\in\mathcal{C}} (h_{k,j} + d_{k,j}[i])$. Observe that vertices j_1 and j_2 are in the same connected component of $\mathcal{T}(G_i)$, thus we have $p_{j_1}[i] \;=\; p_{j_2}[i] \;+ \sum_{(k,\ j)\ \in\ \mathcal{P}(j_1,\, j_2)} (h_{k,j} + d_{k,j}[i])$ as shown in Equation (3.5). Observe that cycle $\mathcal{C}$ can be decomposed into path $\mathcal{P}(j_1, j_2)$ and cross edge $e = (j_1, j_2)$. Therefore, summing up the above two equations, we can get $p_{j_1}[i] \;=\; p_{j_2}[i] + h_{j_1,j_2} + d_{j_1,j_2}[i]$, which means that our solution satisfies the equation represented by the cross edge e.

Case 2.2.: $\mathcal{C}$ is broken into serval disjoint paths in G_i by pre-determined vertices. Suppose that the unique path containing e is $\mathcal{P}$. Without loss of generality, suppose that vertices k_1 and k_2 are the endpoints of $\mathcal{P}$, and $\mathcal{P}$ has the form $\mathcal{P}(k_1, j_1); e; \mathcal{P}(j_2, k_2)$. Since vertices k_1 and k_2 are pre-determined, our algorithm defines a path constraint for $\mathcal{P}$ in G_i, *i.e.*, $p_{k_1}[i] = p_{k_2}[i] + \sum_{(k,\ j) \in \mathcal{P}}(h_{k,j} + d_{k,j}[i])$. Observe that vertices k_1 and j_1 are in the same connected component of $\mathcal{T}(G_i)$. Therefore, $p_{k_1}[i] = p_{j_1}[i] + \sum_{(k,\ j) \in \mathcal{P}(k_1, j_1)}(h_{k,j} + d_{k,j}[i])$ based on Equation (3.5). Similarly, vertices j_2 and k_2 are in the same connected component of $\mathcal{T}(G_i)$ and $p_{j_2}[i] = p_{k_2}[i] + \sum_{(k,\ j) \in \mathcal{P}(j_2, k_2)}(h_{k,j} + d_{k,j}[i])$. Since path $\mathcal{P} = \mathcal{P}(k_1, j_1); e; \mathcal{P}(j_2, k_2)$, summing up the above three equations yields equality $p_{j_2}[i] = p_{j_1}[i] + h_{j_1,j_2} + d_{j_1,j_2}[i]$, which means that our solution satisfies the equation represented by the cross edge e.

In conclusion, the solution produced by algorithm **ZRHC_Phase** satisfies the equations on all edges, and thus the lemma holds. ■

Theorem 3.3.3 *The running time of algorithm* ***ZRHC_Phase*** *is* $O(mn^3)$.

Proof: Step 1 needs $O(n)$ time for each locus, which takes $O(mn)$ time in total. In step 2, we need at most $O(n)$ time for generating a constraint. Since there are at most $O(mn)$ constraints, it takes at most $O(mn^2)$ time. In step 3, the system has $O(n)$ h-variables and $O(mn)$ constraints. So, Gaussian elimination requires $O(mn^3)$ time. In step 4, every edge is just visited once in the traversal. Since each h-variable is expressed as a linear combination of at most n free h-variables at the end of step 3, every p-variable is expressed as at most

n free h-variables and at most one free p-variable on the same locus. Thus, this steps takes $O(mn^2)$ time altogether. Therefore, the entire process takes $O(mn^3)$ time. ■

3.4 Speeding up the Algorithm by Fast Elimination of Redundant Equations

The bottleneck in the above algorithm is step 3 where we have to spend $O(mn^3)$ time to solve a system of $O(mn)$ equations over $O(n)$ variables. Clearly, most of the equations are redundant and can be expressed as linear combinations of other equations. The question is how to detect and eliminate these redundant equations (without using Gaussian elimination, of course). To our best knowledge, there are no methods that would eliminate redundant equations for any system of linear equations over any field faster than Gaussian elimination asymptotically in the worst case. Here, we give such a method taking advantage of the underlying pedigree structure. We first give a general method to compact path and tree constraints that correspond to paths on a cycle in the pedigree graph.

Let $\mathcal{C}$ be a cycle in the pedigree graph G induced by cross edge e_1. For convenience, we say that a path/tree constraint is on the cycle $\mathcal{C}$ if it corresponds to a path/edge on $\mathcal{C}$. The following lemma shows that the path/tree constraints on a cycle can be greatly compacted, and is the key to our algorithm to eliminate redundant constraints.

```
Procedure COMPACT_PT_CONST
input: a set $C$ of path/tree constraints
        on cycle $\mathcal{C}$ induced by cross edge $e_1$,
        and a fixed tree edge $e_0$
output: a compact constraint set $\widehat{C} \equiv C$
begin
  Construct the constraint graph $G^*$ for $C$;
  $\widehat{C} = \emptyset$;
  for each connected component $S$ of $G^*$
    Pick an arbitrary vertex $k_0 \in S$ as the root of $S$;
    Traverse $S$ by BFS starting from $k_0$;
    while there exists unvisited vertices in $G^*$
      Visit an unvisited vertex, say $k$, in the BFS order;
      for each constraint $c = (k_0, j, b, e)$ in $C$
        $\widehat{C} = \widehat{C} \uplus \{c\}$;
      for each constraint $c = (k, j, b, e)$ in $C$
                s.t. vertex $k \neq k_0$ is visited before $j$
        for each constraint $c' = (k_0, k, b', e')$ in $\widehat{C}$
          $b'' = b + b'$;
          if $\{e\} \cup \{e'\} = \{e_0, e_1\}$
            $e'' \ = \ e_1$;
          else
            $e'' \ = \ e_0$;
          Construct a new constraint $c'' = (k_0, j, b'', e'')$;
          if there exists a constraint $(k_0, j, b'' + 1, e'') \in \widehat{C}$
            exit "The input genotypes are inconsistent.";
          if $c'' \notin \widehat{C}$
            $\widehat{C} = \widehat{C} \uplus \{c''\}$;
  return $\widehat{C}$;
end.
```

Figure 3.4: The procedure for compacting path and tree constraints on a cycle.

```
Procedure COMPACT_CONSTRAINTS
input: $C^{\mathbb{C}}$, $C^{\mathbb{P}}$, $C^{\mathbb{T}}$, and a fixed tree edge $e_0$
output: compact $C^{\mathbb{C}}, C^{\mathbb{P}}$ and $C^{\mathbb{T}}$
begin
 Step 1. Removing redundant cycle constraints
  for each cross edge $e$
    Pick an arbitrary constraint, say $c = (e, b)$, from $C^{\mathbb{C}}(e)$;
    if there exists a constraint $(e, b+1) \in C^{\mathbb{C}}(e)$
      exit "The input genotypes are inconsistent.";
    $C^{\mathbb{C}} = C^{\mathbb{C}} - C^{\mathbb{C}}(e) \uplus \{c\}$;

 Step 2. Processing path constraints
  if $G$ is a tree pedigree
    for each cross edge $e$
      $\widehat{C}^{\mathbb{P}}(e) = \emptyset$;
      for each constraint $c = (k, j, b, e) \in C^{\mathbb{P}}(e)$
        if there exists a constraint $(k, j, b+1, e) \in C^{\mathbb{P}}(e)$
          exit "The input genotypes are inconsistent.";
        $\widehat{C}^{\mathbb{P}}(e) = \widehat{C}^{\mathbb{P}}(e) \uplus \{c\}$;
      $C^{\mathbb{P}} = C^{\mathbb{P}} - C^{\mathbb{P}}(e) \uplus \widehat{C}^{\mathbb{P}}(e)$;

  else if $G$ has an all-heterozygous locus
    for each cross edge $e$
      Let $(b', e)$ be the cycle constraint for $e$ in $C^{\mathbb{C}}$;
      for each constraint $c = (k, j, b, e)$ in $C^{\mathbb{P}}(e)$
        Construct a new constraint $c' = (k, j, b - b', e_0)$;
        $C^{\mathbb{T}} = C^{\mathbb{T}} \uplus \{c'\}$ ;

  else (i.e. $G$ is a general pedigree)
    for each cross edge $e$
      $\widehat{C}^{\mathbb{P}}(e)$ = COMPACT_PT_CONST($C^{\mathbb{P}}(e)$, $e_0$);
      $C^{\mathbb{P}} = C^{\mathbb{P}} - C^{\mathbb{P}}(e) \uplus \widehat{C}^{\mathbb{P}}(e)$;

 Step 3. Removing redundant tree constraints
  $C^{\mathbb{T}}$ = COMPACT_PT_CONST($C^{\mathbb{T}}$, $e_0$);
end.
```

Figure 3.5: The procedure for removing redundant constraints.

Lemma 3.4.1 *Given a set C of path/tree constraints on cycle $\mathcal{C}$, we can reduce C to an equivalent constraint set of size at most $2 \cdot length(\mathcal{C})$ in time $O(|C|)$.*

Proof: Recall that e_0 represents the fixed tree edge introduced in subsection 3.3.1 for defining the tree constraints. We use $\widehat{C}$ to denote the equivalent constraint set (to be constructed). Initially, we set $\widehat{C} = \emptyset$.

For convenience, we say a path/tree constraint *connects* vertices j and k, if the constraint has the form (k, j, b, e). To depict a more clear picture of the relationship between constraints in C, we define a *constraint graph* G^* [3] as follows. For each vertex k of the cycle $\mathcal{C}$, we create a vertex in G^*. For each path/tree constraint in C connecting k and j, we build an edge connecting k and j in G^*. Observe that the connected components in G^* naturally partition the constraints in C into disjoint subsets. We will compact the constraints in each of these disjoint subsets separately, and put the resultant equivalent constraint sets into $\widehat{C}$. More precisely, for each connected component of G^*, we pick an arbitrary vertex as the *root* of the component and construct new constraints connecting the root and the other vertices in the component. The details of the construction will be given in the next paragraph. Here, the term root is meant to be synonymous to the term seed defined in subsection 3.3.1, although each seed is defined for a single locus whereas a root may be used to deal with constraints concerning multiple loci.

Now, we give the details of how to construct $\widehat{C}$. Consider each connected component S

---

[3] Note that a constraint graph might actually be a multigraph, but this will not affect the correctness of Lemma 3.4.1.

of G^*. Suppose that its root is k_0. We process the constraints of C induced by S in the order of increasing distance between the root and the vertices connected by the constraints. In other words, we traverse S by BFS starting from the root. Suppose that we are now visiting vertex j. For each edge (k_0, j), we directly put into $\widehat{C}$ the constraints which have resulted in (k_0, j) in G^*. For each edge (k, j) where $k(k \neq k_0)$ is visited before j, our construction guarantees that $\widehat{C}$ has constraints connecting the root k_0 and k. Suppose one of such constraints is $c' = (k_0, k, b', e')$. Let $c = (k, j, b, e) \in C$ denote the constraint that has resulted in the edge (k, j) in G^*. We generate a new constraint $c'' = (k_0, j, b'', e'')$, where $b'' = b + b'$ and e'' is defined as follows:

$$e'' = \begin{cases} e_1 & \text{if } \{e\} \cup \{e'\} = \{e_0, e_1\} \\ \\ e_0 & \text{otherwise} \end{cases}$$

Because c can be represented as the summation of c'' and $c' \in \widehat{C}$, c is equivalent to c'' given $\widehat{C}$. Then we add c'' to $\widehat{C}$. Here, again the edges e_0 and e_1 are used to indicate tree and path constraints, respectively. Observe that, the above combination of constraints c' and c will result in a constraint c'' involving only tree edges, if and only if both or none of them correspond to paths containing e. Otherwise, the resultant constraint c'' will correspond to some path across e_1.

The BFS creates an equivalent constraint in $\widehat{C}$ for every constraint in C, and thus $\widehat{C} \equiv C$. Recall that each constraint in $\widehat{C}$ has the form (k_0, j, b, e), where j is a vertex in $\mathcal{C}$, e is either e_0 or e_1, and k_0 is the root of the connected component of G^* containing j. It is easy to see that two constraints (k_0, j, b', e) and (k_0, j, b'', e) that differ only in the associated b-constants are consistent with each other if and only if $b' = b''$. Hence, $\widehat{C}$ has at most $2 \cdot length(\mathcal{C})$ different constraints, or otherwise input linear system has no feasible solutions. The construction can be done in $O(|C|)$ time, as more formally described by procedure **Compact_PT_Const** in Figure 3.4. Hence, the lemma holds. ■

An immediate application of Lemma 3.4.1 is to remove redundancy from each path constraint set $C^{\mathbb{P}}(e)$, since the path constraints in $C^{\mathbb{P}}(e)$ are all on the cycle induced by e.

Corollary 3.4.2 *Given the path constraint set $C^{\mathbb{P}}(e)$, we can reduce it to an equivalent constraint set of size at most $2 \cdot length(\mathcal{C})$ in time $O(|C^{\mathbb{P}}(e)|)$, where $\mathcal{C}$ is the cycle induced by cross edge e.*

We can also use Lemma 3.4.1 to remove redundant tree constraints. Note that the construction in the proof of Lemma 3.4.1 still works if the constraints in C are all tree constraints involving no cross edge e_1. Moreover, the resultant set $\widehat{C}$ contains only constraints of the form (k_0, j, b, e_0). This implies that $|\widehat{C}| \leq n$. Therefore, the following corollary holds.

Corollary 3.4.3 *Given the tree constraint set $C^{\mathbb{T}}$, we can reduce it to an equivalent constraint set of size at most n, in $O(|C^{\mathbb{T}}|)$ time.*

3.4.1 Elimination of Redundant Cycle Constraints

Each cross edge e induces a unique cycle $\mathcal{C}$. Since every constraint in $C^{\mathbb{C}}(e)$ concerns the same set of h-variables corresponding to the edges on $\mathcal{C}$, each $C^{\mathbb{C}}(e)$ contains only one independent constraint. Moreover, these constraints are consistent with each other if and only if their associated constants are identical, which can be checked in $O(m)$ time. Because the total number of cross edges are at most $n+1$, we have

Lemma 3.4.4 *Given the cycle constraint set $C^{\mathbb{C}}$, we can reduce it to an equivalent constraint set of size at most $n+1$ in $O(mn)$ time.*

3.4.2 Elimination of Redundant Path Constraints

We will show how to reduce the path constraints $C^{\mathbb{P}}$ on a general pedigree to an equivalent set of path constraints with size $O(n \log^2 n \log\log n)$ in $O(mn)$ time (assuming $\log^2 n \log\log n < m$). Furthermore, for tree pedigrees (*i.e.* pedigrees with no mating loops) we can make the equivalent constraint set as small as $O(n)$. For pedigrees with an all-heterozygous locus across the entire pedigree, we can first transform $C^{\mathbb{P}}$ into an equivalent tree constraint set with size $O(mn)$, and then will remove its redundancy via Corollary 3.4.3. We first start with the special cases.

Elimination of Redundant Path Constraint on Tree Pedigrees. Observing that the length of each (simple) cycle in the pedigree graph of a tree pedigree is a constant (*i.e.* 4, of which an example is given in Figure 3.1(B)), we can upper bound the total number of path constraints as follows.

Lemma 3.4.5 *Given the path constraint set $C^{\mathbb{P}}$ on a tree pedigree, we can reduce it to an equivalent path constraint set of size $O(n)$ in $O(mn)$ time.*

Proof: By Corollary 3.4.2, we can reduce $C^{\mathbb{P}}(e)$ for each cross edge e to an equivalent set of at most 8 path constraints in $O(m)$ time. Since there are at most $n+1$ cross edges, the set $C^{\mathbb{P}}$ can be reduced to an equivalent set of size $O(n)$ in $O(mn)$ time and thus the lemma holds. ■

Transformation of Path Constraints on Pedigrees with an All-Heterozygous Locus. Observe that for a pedigree with an all-heterozygous locus i, each cross edge induces a cycle that exists in the locus graph G_i and has a cycle constraint in the (reduced) set $C^{\mathbb{C}}$. This allows us to transform all the path constraints into tree constraints given the cycle constraints as follows.

Corollary 3.4.6 *Given the path constraint set $C^{\mathbb{P}}$ on a pedigree with an all-heterozygous locus, we can construct a equivalent tree constraint set of size $O(mn)$ in $O(mn)$ time.*

Proof: Let us first focus on a cross edge e. Suppose that the cycle $\mathcal{C}$ is induced by e in $\mathcal{T}(G) \cup \{e\}$, the compact $C^{\mathbb{C}}(e)$ is $\{(b', e)\}$, and e_0 is the fixed tree edge defined in subsection 3.3.1. Notice that the cycle $\mathcal{C}$ has two disjoint paths connecting each pair of two vertices on the cycle. One of them consists of only tree edges and may be used to define a tree constraint, while the other contains the cross edge and may be used to represent a path constraint. Therefore, with the help of the cycle constraint (b', e), we can transform the path constraint set $C^{\mathbb{P}}(e)$ into a tree constraint set $C(e)$ of the same size as follows:

$$C(e) = \{(k, j,\ b - b',\ e_0) \mid (k, j,\ b,\ e) \in C^{\mathbb{P}}(e)\}$$

It is not hard to see that $C(e) \cup C^{\mathbb{C}}(e) \equiv C^{\mathbb{P}}(e) \cup C^{\mathbb{C}}(e)$, and $C(e)$ can be constructed in $|C^{\mathbb{P}}(e)|$ time.

Let $C = \cup_{e \in E^{\mathbb{X}}} C(e)$. Obviously, $C \equiv C^{\mathbb{P}}$ and $|C| = O(mn)$ since $|C^{\mathbb{P}}| = O(mn)$. Hence, the lemma holds. ■

Elimination of Redundant Path Constraints on a General Pedigree. Now we consider how to compact path constraints in the general case. As shown in Corollary 3.4.2, given a cross edge e inducing cycle $\mathcal{C}$, we can compact the constraints in $C^{\mathbb{P}}(e)$ so that at most $2 \cdot length(\mathcal{C})$ constraints are kept. Clearly, the compact $C^{\mathbb{P}}$ has size at most $O(n^2)$ since the number of cross edges is at most $n + 1$ and the length of a cycle containing a cross edge is at most n. This bound can be improved by observing that the total length of all cycles in G is related to the average *stretch* [17] of G with respect to the spanning tree $\mathcal{T}(G)$. Hence, we can obtain a sharper upper bound on $|C^{\mathbb{P}}|$ by using a low-stretch spanning tree $\mathcal{T}(G)$ as constructed in [17].

We first give a formal definition of the *stretch* of an unweighted connected graph with respect to a spanning tree. Given a spanning tree $\mathcal{T}$ on an unweighted connected graph $G = (V, E)$ (*e.g.* the pedigree graph), we define the stretch of an edge $(k, j) \in E$, denoted as $streth_{\mathcal{T}}(k, j)$, to be the length of the unique path (*i.e.* the number of edges on the path) in $\mathcal{T}$ between k and j. The average stretch of G with respect to $\mathcal{T}$ is then defined as $avg\text{-}stretch_{\mathcal{T}}(E) = \frac{1}{|E|} \sum_{(k,j)\in E} stretch_{\mathcal{T}}(k, j)$.

Lemma 3.4.7 *Given a pedigree* G*, we can build a low-stretch spanning tree* $\mathcal{T}(G)$ *in* $O(n \log n)$ *time such that* $|C^{\mathbb{P}}| = O(n \log^2 n \cdot \log\log n)$ *after compacting.* [4]

Proof: In [17], Elkin *et al.* showed that every unweighted connected graph $G = (V, E)$ contains a spanning tree, into which each edge of the graph can be embedded with an average stretch of $O(\log^2 n \log\log n)$. Moreover, this tree can constructed in $O(|E| \log |V|)$ time. In our situation, such a low-stretch spanning tree $\mathcal{T}(G)$ can be built in $O(n \log n)$ time because $|E| \leq 2n$ and $|V| = n$. The following inequality establishes the relationship between $|C^{\mathbb{P}}|$ and the average stretch of E with respect to $\mathcal{T}(G)$. From Corollary 3.4.2, we have

---

[4] This result was mostly due to Jing Xiao and has appeared in his M.Sc. thesis, but we include it here for the sake of completeness.

$$
\begin{aligned}
|C^{\mathbb{P}}| &= \sum_{e \in E^{\mathbb{X}}} |C^{\mathbb{P}}(e)| \\
&\leq \sum_{\mathcal{C} \text{ induced by } e \in E^{\mathbb{X}}} 2 \cdot \mathit{length}(\mathcal{C}) \\
&= \sum_{e\,=\,(k,j) \in E^{\mathbb{X}}} 2 \cdot \big(\mathit{stretch}_{\mathcal{T}}(k,j) + 1\big) \\
&\leq 2 \sum_{(k,j) \in E} \mathit{stretch}_{\mathcal{T}}(k,j) + 2n \\
&\leq 2 \cdot |E| \cdot \mathit{avg\text{-}stretch}_{\mathcal{T}}(E) + 2n \\
&\leq 2 \cdot (2n) \cdot O(\log^2 n \log\log n) + 2n \\
&= O(n \cdot \log^2 n \log\log n)
\end{aligned}
$$

■

3.4.3 Elimination of Redundant Tree Constraints and Final Algorithm

After we process (*i.e.* compact or transform) the path constraints, we eliminate redundant tree constraints and obtain a compact tree constraint set containing at most n constraints as shown in Corollary 3.4.3.

The complete algorithm for eliminating redundant cycle, path and tree constraints is given in Figure 3.5 as procedure **Compact_Constraints**. The next theorem summarizes the above discussion.

Theorem 3.4.8 *Given the constraint sets* $C^{\mathbb{C}}$, $C^{\mathbb{P}}$ *and* $C^{\mathbb{T}}$ *on a pedigree, we can reduce them to an equivalent constraint set of size* $O(n \cdot \log^2 n \log\log n)$ *in* $O(mn)$ *time. In particular, for tree pedigrees and pedigrees with an all-heterozygous locus, the equivalent constraint set has size* $O(n)$.

We can incorporate the above redundant constraint elimination procedure **Compact _Constraints** into the $O(mn^3)$ time algorithm for ZRHC to obtain an improved algorithm **Improved_ZRHC _Phase** as shown in Figure 3.3. (An example of how **Improved_ZRHC_Phase** works is given in Section 3.5). The following theorem is obvious given Theorem 3.4.8.

Theorem 3.4.9 *The algorithm* ***Improved_ZRHC_Phase*** *solves the ZRHC problem correctly on any pedigree in* $O(mn^2 + n^3 \log^2 n \log\log n)$ *time. Moreover, it solves ZRHC on tree pedigrees or pedigrees with an all-heterozygous locus in* $O(mn^2 + n^3)$ *time.*

3.5 An Example for Algorithm IMPROVED_ZRHC_PHASE

The example in the following Figure 3.6 aims to demonstrate how algorithm **Improved_ZRHC _phase** works (see Figure 3.5 for the pseudo-code of the algorithm). The input pedigree with genotype data is shown in Figure 3.6(A) and the corresponding pedigree graph is in Figure 3.6(B).

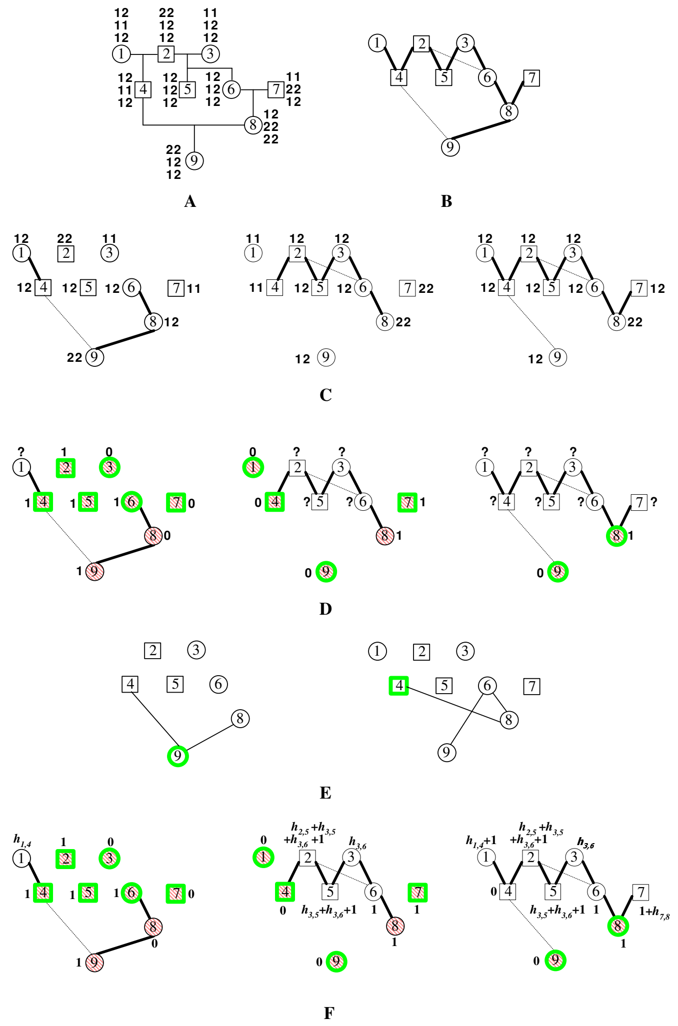

Figure 3.6: An example execution for illustrating the algorithm IMPROVED_ZRHC_PHASE. (A). An example input pedigree with genotype data. Here, the alleles at a locus are ordered according to their id numbers instead of phase (which is unknown). (B). The pedigree graph with a spanning tree. The tree edges are highlighted. (C). Locus graphs for the three loci, respectively. The locus forests are highlighted. (D). The pre-determined nodes. On the locus graphs, the pre-determined nodes have the corresponding p-values (*i.e.* the number 0 or 1 near the nodes) with the red shadow highlight, while the undetermined nodes have question marks (*i.e.*"?"). The seeds in the graphs are highlighted by thick green border. (E). The constraint graphs. The left graph is for removing redundant path constraints generated by the cross edge $e_{4,9}$, while the right graph is for tree constraints. The roots (see Lemma 3.4.1 for the definition) in the graphs are highlighted by thick green border. (F). The locus graphs with propagated p-values. The notations are the same as those in Figure 3.6(D), except that the question marks on the undermined nodes are replaced by their resolved p-values. The resolved p-values are expressed as a linear combination of the free variables in $p_j[i]$ and the free h-variables with an appropriate constant term.

Step 1, we generate the locus graphs (or forests) illustrated in Figure 3.6(C) and identify the pre-determined vertices in Figure 3.6(D). Moreover, we arbitrarily pick a tree edge, say $e_{1,4}$, as the *indicator* to distinguish tree constraints from path constraints in subsection 3.3.1.

Step 2, we generate cycle, path and tree constraints as follows. For example, given the cycle $\mathcal{C} = v_2v_5v_3v_6v_2$ in the second locus graph of Figure 3.6(D), we denote the cycle constraint $h_{2,5} + h_{3,5} + h_{3,6} + h_{2,6} = 0$ by the form $(0, e_{2,6})$. Afterwards, we have $C^{\mathbb{C}} = \{(0, e_{2,6}), (0, e_{2,6})\}$, $C^{\mathbb{P}} = \{(v_4, v_9, 0, e_{4,9}), (v_9, v_8, 1, e_{4,9})\}$ and $C^{\mathbb{T}} = \{(v_6, v_8, 0, e_{1,4}), (v_6, v_9, 1, e_{1,4}), (v_4, v_8, 1, e_{1,4})\}$.

Step 2′, we first remove redundant cycle constraint and get $C^{\mathbb{C}} = \{(0, e_{2,6})\}$. Now we need to take care path constraints. For an instance, given the path constraints $(v_4, v_9, 0, e_{4,9})$ and $(v_9, v_8, 1, e_{4,9})$ induced by the cross edge $e_{4,9}$, we draw the constraint graph illustrated in Figure 3.6(E) to help removing redundant path constraints, and get $C^{\mathbb{P}} = \{(v_4, v_9, 0, e_{4,9}), (v_9, v_8, 1, e_{4,9})\}$ based on Lemma 3.4.1. Then, we construct the constraint graph shown in Figure 3.6(E) for tree constraints and obtain $C^{\mathbb{T}} = \{(v_4, v_8, 1, e_{1,4}), (v_4, v_6, 1, e_{1,4}), (v_4, v_9, 0, e_{1,4})\}$, according to Lemma 3.4.1.

Step 3, we apply Gaussian elimination to the following linear system, which is equivalent to $C^{\mathbb{C}} \biguplus C^{\mathbb{P}} \biguplus C^{\mathbb{T}}$.

$$\begin{cases} h_{2,5}+h_{3,5}+h_{3,6}+h_{2,6}=0 & \textit{i.e.}\text{ the cycle constraint } (0,e_{2,6}) \\ h_{4,9}=0 & \textit{i.e.}\text{ the path constraint } (v_4,v_9,0,e_{4,9}) \\ h_{4,9}+h_{2,4}+h_{2,5}+h_{3,5}+h_{3,6}+h_{6,8}=1 & \textit{i.e.}\text{ the path constraint } (v_9,v_8,1,e_{4,9}) \\ h_{2,4}+h_{2,5}+h_{5,3}+h_{3,6}+h_{6,8}=1 & \textit{i.e.}\text{ the tree constraint } (v_4,v_8,1,e_{1,4}) \\ h_{2,4}+h_{2,5}+h_{5,3}+h_{3,6}=1 & \textit{i.e.}\text{ the tree constraint } (v_4,v_6,1,e_{1,4}) \\ h_{2,4}+h_{2,5}+h_{5,3}+h_{3,6}+h_{6,8}+h_{8,9}=0 & \textit{i.e.}\text{ the tree constraint } (v_4,v_9,0,e_{1,4}) \end{cases}$$

Then we obtain the following general solution, which is the solution to the linear system $C^{\mathbb{C}} \uplus C^{\mathbb{P}} \uplus C^{\mathbb{T}}$.

$$\begin{cases} h_{4,9}=0 \\ h_{6,8}=0 \\ h_{8,9}=1 \\ h_{2,6}=h_{2,5}+h_{3,5}+h_{3,6} \\ h_{2,4}=h_{2,5}+h_{3,5}+h_{3,6}+1 \end{cases}$$

where $h_{2,5}, h_{3,5}, h_{3,6}, h_{1,4}$ and $h_{7,8}$ are free variables.

Step 4, we solve the unknown p-values by propagation from the seeds and obtain all p-values as shown in Figure 3.6(F).

3.6 Concluding Remarks

It remains interesting if the time complexity for ZRHC on general pedigrees can be improved to $O(mn^2 + n^3)$ or something even lower. Another open question is how to use the algorithm to solve MRHC on pedigrees that require only a small (constant) number of recombinants.

Chapter 4

Linear-Time Reconstruction of Zero-Recombinant Mendelian Inheritance on Pedigrees without Mating Loops

4.1 Introduction

This chapter is concerned with haplotype inference when the input pedigree is a tree pedigree with no mating loops (*i.e.* the Loop-free ZRHC problem formally defined in section 1.2).

---

[0] An extended abstract of this paper, entitled "Linear-Time Reconstruction of Zero-Recombinant Mendelian Inheritance on Pedigrees without Mating Loops", has submitted to *the 7th Annual Workshop on Algorithms in*

We first give a brief review of previous results and a high-level summary of our result for the Loop-free ZRHC problem.

4.1.1 Previous Results and Our Result

Li and Jiang presented an $O(m^3n^3)$ time algorithm for ZRHC by formulating it as a system of $O(mn)$ linear equations with mn variables over the finite field $F(2)$ and applying Gaussian elimination [28]. As shown in the previous chapter, Xiao, Liu, Xia and Jiang [46] presented a much faster algorithm for ZRHC with running time $O(mn^2 + n^3 \log^2 n \log\log n)$ by introducing a new system of $O(mn)$ linear equations over $F(2)$ with mn variables and reducing it to an effectively equivalent system with only $O(n \log^2 n \log\log n)$ equations and at most $2n$ variables.

For Loop-Free ZRHC, the method proposed in the previous chapter [46] runs in time $O(mn^2 + n^3)$ to produce a general solution and in time $O(mn + n^3)$ to produce a particular solution.

Chan *et al.* [11] proposed a linear-time (*i.e.* $O(mn)$ time) algorithm to find a particular solution. It is unclear how the algorithm can be extended to produce general solutions efficiently. Moreover, the algorithm has four different stages and needs to maintain various consistency conditions (called SNP-consistency, Mendelian consistency, and endgame-consistency) and avoid some inconsistency conditions (such as the family problem) at these

---

Bioinformatics (WABI), Sept. 2007, Philadelphia, PA.

stages (see [11] for the details of the (in)consistency conditions). As a result, the analysis of the algorithm (concerning both correctness and complexity) is quite involved.

In this chapter, we present a linear-time algorithm to generate a particular solution for Loop-Free ZRHC. Moreover, the algorithm can also provide a general solution in $O(mn^2)$ time, which is optimal because the size of a general solution could be as large as $\Theta(mn^2)$. [1] Our construction is based on the system of linear equations/constraints introduced in the previous chapter[46], a careful examination of the relationship between the constraints using the so called *constraint graphs* (to be defined later on), and a novel linear-time method for solving linear systems without invoking Gaussian elimination. In the algorithm, we first build a *one-to-one* mapping from the constraints to the edges in the pedigree graph, then solve the constraints by a single BFS traversal on the pedigree graph. We also give a rigorous proof of the correctness and tight analysis of the time complexity of our algorithm.

4.1.2 Organization of the Chapter

The rest of the chapter is organized as follows. In subsection 4.1.3, we describe how to add indices to the vertices of the pedigree graph, whereas the system of linear equations for ZRHC and some useful graphs derived from the input pedigree are introduced in the previous chapter [46]. A linear-time algorithm to check if the linear equations derived from an input tree pedigree are consistent (*i.e.* if solutions for the linear system and thus ZRHC exist) and

---

[1] An instance of the Loop-Free ZRHC problem that has a general solution of size $\Theta(mn^2)$ is given in section 4.3.

solve the linear system to attain a general solution is presented in section 4.2. The method of obtaining a particular solution by the algorithm is also described in this section. Section 4.3 contains an example execution of the main algorithm and an instance of the Loop-free ZRHC problem that has a general solution of size $\Theta(mn^2)$. Section 4.4 gives concluding remarks.

4.1.3 The Pedigree Graph with Indexed Vertices

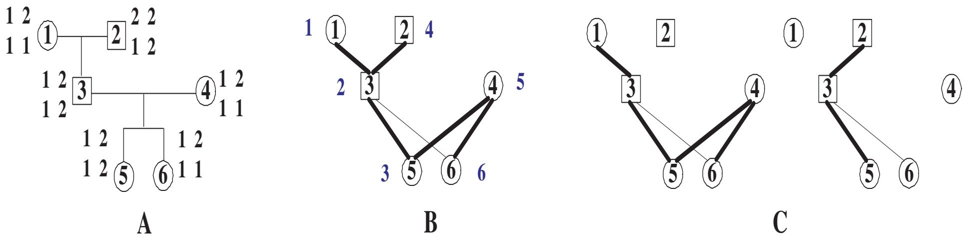

Figure 4.1: An example pedigree with indexed vertices and the corresponding locus graphs. (A). An example pedigree with genotype data. Here, the alleles at a locus are ordered according to their id numbers instead of phase (which is unknown). (B). The pedigree graph with a spanning tree. The tree edges are highlighted. Observe that there lies a *cycle* of length 4 in the given tree pedigree graph. The root of the tree is v_1. The number (in blue) next to a vertex is the index generated by a BFS traversal of the spanning tree. For instance, v_3 is v_2's predecessor in the BFS traversal where $index(v_3) = 2$ and $index(v_2) = 4$, while v_2 is v_3's biological father. (C). The locus graphs. The left graph is for the first locus, which has a cycle, while the right graph is for the second locus. The locus forests are highlighted.

Given a pedigree graph, we start a BFS traversal on $\mathcal{T}(G)$ from the root, and index the vertices in E based on the order of their first-time occurrences in the traverse. For a vertex j, we denote by $index(j)$ the index of j. For instance, $index(r) = 1$ if vertex r is the root of $\mathcal{T}(G)$. If the last edge on the path from the root to vertex j is (k, j), k is called the *predecessor* of j, and denoted by $pred(j)$. [2] Obviously, $index(pred(k)) < index(j)$ for every non-root vertex j. The example of indices are shown in Figure 4.1 (B).

4.2 A Linear-Time Algorithm for Solving Loop-Free ZRHC

As pointed out in the previous chapter [46], ZRHC has a solution if and only if the above linear system for the h-variables has a solution. Now we focus on how to solve the linear system to obtain solutions to the h-variables. We will check the consistency of the path/cycle constraints and tree constraints first, perform some transformation on the constraints to obtain an equivalent but smaller system of constraints, and then solve the h-variables for tree edges and cross edges with respect to the spanning tree $\mathcal{T}(G)$.

4.2.1 Consistency Checking

First, we transform the path/cycle constraints to eliminate redundancy. The tree constraints will be dealt with in a similar way.

Path/Cycle Consistency Checking and Redundancy Removal. Because every cycle in a tree pedigree has length four and involves only the members in a single *nuclear family* (consisting of a pair of parents and their children), the number of independent path constraints and the cycle constraints should be small. We would like to remove redundant constraints such that, after the reduction, $|C^{\mathbb{C}}(e) \uplus C^{\mathbb{P}}(e)| \leq 1$ for every cross edge e.

---

[2] Conventionally, k is called the *father* of j in graph theory. In order to distinguish the concept of father in graph theory and the concept of biological father, we use the term *predecessor* in this chapter.

```
Procedure CONSTRAINTS_CONSISTENCY
input: $C^{\mathbb{C}}$, $C^{\mathbb{P}}$, $C^{\mathbb{T}}$, and a fixed tree edge $e_0$
output: compact $C^{\mathbb{C}}, C^{\mathbb{P}}$ and $C^{\mathbb{T}}$

begin
 Step 1. Consistency checking for path/cycle constraints
  for each cross edge $e$
   Pick an arbitrary constraint, say $(j,k,b,e)$, from $C^{\mathbb{C}}(e)$;
   if there exists a constraint $(j,k,b+1,e) \in C^{\mathbb{C}}(e)$
    exit "The input genotypes are inconsistent.";
   $C^{\mathbb{P}} = \{(j,k,b,e)\}$;

  for each cross edge $e$
   Pick an arbitrary constraint, say $(b,e)$, from $C^{\mathbb{C}}(e)$;
   if there exists a constraint $(b+1,e) \in C^{\mathbb{C}}(e)$
    exit "The input genotypes are inconsistent.";
   $C^{\mathbb{C}} = \{(b,e)\}$;
   if $C^{\mathbb{P}} \neq \emptyset$ (i.e. $C^{\mathbb{P}}$ contains exactly one constraint)
    Suppose the constraint $\in C^{\mathbb{P}}$ is $(j,k,b',e)$;
    $C^{\mathbb{T}} = \{(j,k,b'+b,e_0)\}$;
    $C^{\mathbb{P}} = \emptyset$;

 Step 2. Consistency checking for tree constraints
 Construct the constraint graph $G^*$ for $C^{\mathbb{T}}$;
 $\widehat{C} = \emptyset$;
 for each connected component $S$ of $G^*$
  Generate the spanning tree $\mathcal{T}(S)$ of $S$;
  Suppose vertex $k_0$ is the vertex with the smallest index;
  Let vertex $k_0$ be the root of $\mathcal{T}(S)$;

  Traverse $\mathcal{T}(S)$ by BFS starting from $k_0$;
  for each vertex $k \in S$
   $visited(k) = false$;
  $W(k_0) = 0$;    $visited(k_0) = true$;
  for each tree edge $(j,k)$ with respect to $\mathcal{T}(S)$
   Suppose the edge represents constraint $(j,k,b,e_0)$
   Suppose $visited(j) = true$ and $visited(k) = false$;
   $W(k) = W(j) + b$;    $visited(k) = true$;
   $\widehat{C} = \widehat{C} \uplus \{(k_0,k,W(k),e_0)\}$;
  for each non-tree edge $(j,k)$ with respect to $\mathcal{T}(S)$
   Suppose the edge represents constraint $(j,k,b,e_0)$
   if $W[j] + W[k] + b = 1$ (i.e. $\exists$ a 1-weight basic cycle)
    exit "The input genotypes are inconsistent.";

  $C^{\mathbb{T}}$=$\widehat{C}$;
  return $C^{\mathbb{C}}, C^{\mathbb{P}}$ and $C^{\mathbb{T}}$
end.
```

Figure 4.2: The procedure for checking the consistency among the constraints.

```
Algorithm LOOP_FREE_ZRHC_PHASE
input: a tree pedigree $G = (V, E)$ and genotypes $\{\mathbf{g}_j\}$
output: a general solution of $\{\mathbf{p}_j\}$
begin
  Step 1. Preprocessing
    Choose an arbitrary vertex $r$ as the root of a spanning
      tree $\mathcal{T}(G)$ for the pedigree graph $G$;
    Index the vertices by a BFS traversal on $\mathcal{T}(G)$;
  Step 2. Constraint generation
    Let $e_0$ be an arbitrary tree edge with respect to $\mathcal{T}(G)$ ;
    Generate $C^{\mathbb{C}}$, $C^{\mathbb{P}}$ and $C^{\mathbb{T}}$;
Step 2′. Redundant Constraint Elimination
    CONSTRAINTS_CONSISTENCY( $C^{\mathbb{C}}$, $C^{\mathbb{P}}$, $C^{\mathbb{T}}$, $e_0$ );
  Step 3. Solve the $h$-variables
  Construct the mapping $f : C \mapsto E'$ in Equality (4.9);
  Traverse $G$ in BFS order;
  $W[r] = 0$;
  for each tree edge $(pred(k), k) \in E$
     if $(pred(k), k) \notin E'$
      Define $h_{pred(k),k}$ as a free variable;
      $W(k) = h_{pred(k),k} + W(pred(k))$;
    if $(pred(k), k) \in E'$
      Suppose $f^{-1}((pred(k),k))=(j,k,b,e_0)$ $(index(j)<index(k))$;
      $h_{pred(k),k} = b + W(j) + W(pred(k))$;
      $W(k) = b + W(j)$;
  for each cross edge $(i, j) \in E$
    if $(i, j) \notin E'$
      Define $h_{i,j}$ as a free variable;
    if $(i, j) \in E'$
      if $f^{-1}((i, j))$ is a path constraint, say, $(k_1, k_2, b, e)$
        $h_{i,j} = b + W(k_1) + W(k_2) + W(i) + W(j)$;
      else (i.e. $f^{-1}((i, j))$ is a cycle constraint, say, $(b, e)$)
        $h_{i,j} = b + W(i) + W(j)$;
  Step 4. Solve the $p$-variables $\{\mathbf{p}_j\}$ by propagation
    for each locus $i$
      for each connected component $\mathcal{T}$ in $\mathcal{T}(G_i)$
        if $\mathcal{T}$ has no pre-determined vertices
          Set the $p$-variable of the seed as a free variable and
            treat it as a determined value;
        Traverse $\mathcal{T}$ by BFS starting from the seed;
          for each edge $(j, k)$ in $\mathcal{T}$
            if $p_j[i]$ is determined but $p_k[i]$ is undetermined
              $p_k[i] = p_j[i] + h_{j,k} + d_{j,k}[i]$;
  return $\{\mathbf{p}_j\}$;
end.
```

Figure 4.3: The algorithm for solving the linear system for Loop-Free ZRHC.

We need introduce some notations for a cross edge, say, e. The cross edge e induces a unique cycle $\mathcal{C}$ on the pedigree graph G. Suppose that $\mathcal{C} = k_1 j_1 k_2 j_2 k_1$ resides in a nuclear family, where k_1 and k_2 are the parents and j_1 and j_2 are two children. The cycle $\mathcal{C}$ can be decomposed into two edge-disjoint paths $\mathcal{P}_1 = j_1 k_1 j_2$ and $\mathcal{P}_2 = j_1 k_2 j_2$. Notice that exactly one of these paths contains the cross edge e. Without loss of generality, suppose that $\mathcal{P}_1$ contains e. Figure 4.4(A) depicts the cycle $\mathcal{C}$ and paths $\mathcal{P}_1$ and $\mathcal{P}_2$ in the pedigree graph.

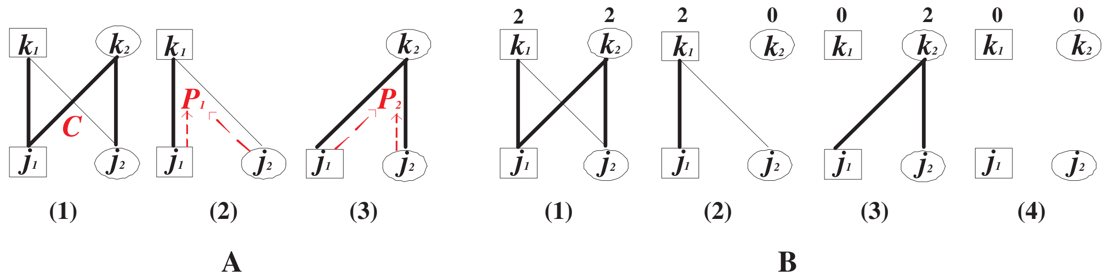

Figure 4.4: Pedigree subgraphs and locus subgraphs for two siblings in a nuclear family. (A). The cross edge e induces a unique cycle $\mathcal{C}$ on the pedigree graph G consisting of parents k_1 and k_2 and children j_1 and j_2 as shown in subfigure A(1). The tree edges on $\mathcal{T}(G)$ are highlighted. The cycle $\mathcal{C}$ can be decomposed into two edge-disjoint paths $\mathcal{P}_1 = j_1 k_1 j_2$ shown in subfigure A(2) and $\mathcal{P}_2 = j_1 k_2 j_2$ shown in subfigure A(3). Without loss of generality, suppose that $\mathcal{P}_1$ contains e. (B). A locus subgraph for cross edge e. The number on the top of a parent represents the genotype of the parent. Subfigure B(1) shows the locus subgraph if both parents are heterozygous (*i.e.* both of their genotypes are 2). Here, the label of every edge on $\mathcal{C}$ is 1 and we have a cycle constraint concerning $\mathcal{C}$. Subfigure B(2) demonstrates the locus subgraph if the parent k_1 is heterozygous while the other parent k_2 is homozygous (*e.g.* with genotype 0). Here, the labels of e_{j_1,k_2} and e_{j_2,k_2} are 0, and both j_1 and j_2 are pre-determined. Thus, we have a path constraint concerning $\mathcal{P}_1$. Subfigure B(3) displays the locus subgraph if the parent k_2 is heterozygous and the other parent k_1 is homozygous (*e.g.* with genotype 0). Here, the labels of e_{j_1,k_1} and e_{j_2,k_1} are 0, both j_1 and j_2 are pre-determined, and we obtain a tree constraint. Subfigure B(4) shows the locus subgraph if both parents are homozygous. Now, the label of every edge on the cycle $\mathcal{C}$ is 0, and there is no constraint for $\mathcal{C}$.

We now consider the path/cycle constraints generated for the cross edge e at a single locus. In the locus graph for this locus, if both parents are heterozygous, we have a cycle constraint concerning $\mathcal{C}$ as shown in Figure 4.4B(1). If exactly one of the parents is heterozygous, we have a path constraint concerning $\mathcal{P}_1$ or a tree constraint as shown in Figure 4.4B(2) or in Figure 4.4B(3), respectively. Otherwise, both parents are homozygous and there is no constraint for the cycle $\mathcal{C}$ as illustrated in Figure 4.4B(4).

Next, we consider the path/cycle constraints derived for the cross edge e at all loci. Based on the above discussions, $C^{\mathbb{C}}(e)$ (or $C^{\mathbb{P}}(e)$) contains only one independent constraint. Moreover, the constraints are consistent with each other if and only if their associated constants are identical. Therefore, we can check the consistency for $C^{\mathbb{C}}(e)$ (or $C^{\mathbb{P}}(e)$) in time $O(|C^{\mathbb{C}}(e)|)$ (or $O(|C^{\mathbb{P}}(e)|)$, respectively) and obtain an equivalent constraint set with size at most one. We further consider the following two cases:

- *Case 1*: There exists a locus where both parents discussed above are heterozygous. As shown in Figure 4.4B(1), we have a cycle constraint for $\mathcal{C}$ at the locus, which can be used to transform all the path constraints in $C^{\mathbb{C}}(e)$ into tree constraints. More specifically, suppose that the cycle constraint is (b, e). Then each path constraint (j, k, b', e) can be replaced by a tree constraint $(j, k, b + b', e_0)$. After the transformation, $C^{\mathbb{P}}(e) = \emptyset$.

- *Case 2*: At every locus, at least one parent is homozygous. Obviously, $C^{\mathbb{C}}(e) = \emptyset$.

As a summary of the above two cases, we can see that $C^{\mathbb{P}}(e) = \emptyset$ or $C^{\mathbb{C}}(e) = \emptyset$. A pseudo-code for performing this transformation is given in Figure 4.2. Hence, we have the following lemma concerning the transformation of the path and cycle constraints.

Lemma 4.2.1 *Given the path constraint set* $C^{\mathbb{P}}(e)$ *and the cycle constraint cycle* $C^{\mathbb{C}}(e)$ *for a cross edge* e *in a tree pedigree, we can reduce them to an equivalent constraint set of size at most one, in* $O(|C^{\mathbb{C}}(e) \uplus C^{\mathbb{P}}(e)|)$ *time.*

As a natural extension of Lemma 4.2.1, we have the following corollary:

Corollary 4.2.2 *Given the path constraint set* $C^{\mathbb{P}}$ *and the cycle constraint cycle* $C^{\mathbb{C}}$ *for a tree pedigree, we can reduce them to an equivalent constraint set of size at most* $|E^{\mathbb{X}}|$*, in* $O(|C^{\mathbb{C}} \uplus C^{\mathbb{P}}|)$ *time.*

Tree Constraint Consistency Checking and Redundancy Removal. To clearly demonstrate the relationship among the constraints in $C^{\mathbb{T}}$, we define a *constraint graph* G^* as follows. [3] For each vertex in the pedigree graph, we create a vertex in G^*. For each tree constraint $(k, j, b, e_0) \in C^{\mathbb{T}}$, we introduce an edge connecting vertices k and j in G^* with weight b. We denote by $E(G^*)$ the edge set in G^*. The weight of a path $\mathcal{P}$ (or a cycle $\mathcal{C}$) in the constraint graph is defined as the summation (in $F(2)$) over the weights of the edges along path $\mathcal{P}$ (or cycle $\mathcal{C}$), which is denoted as $W(\mathcal{C})$ (or $W(\mathcal{P})$, respectively). We choose a spanning forest $\mathcal{T}(G^*)$ on the constraint graph G^*. In each connected component $\mathcal{T}$ of $\mathcal{T}(G^*)$, the vertex with the smallest index is designated as the root of $\mathcal{T}$. (We set the roots in

---

[3] Note that a constraint graph can be a weighted multigraph.

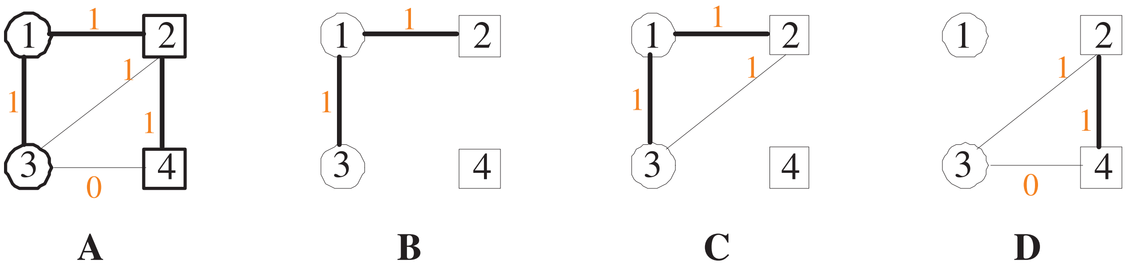

Figure 4.5: An example for constructing the constraint graph based on given constraints. (A). Given a set of tree constraints $C^{\mathbb{T}} = \{(v_1, v_2, 1, e_0), (v_1, v_3, 1, e_0), (v_2, v_3, 1, e_0), (v_2, v_4, 1, e_0), (v_3, v_4, 0, e_0)\}$, we can construct the constraint graph G^* shown in (A), where the number on an edge in G^* is the weight of the edge, *i.e.* the b-constant in the constraint represented by the edge. For example, $(v_1, v_3, 1, e_0)$ is represented by edge (v_1, v_3) with weight 1 in the constraint graph G^*. We choose a spanning tree $\mathcal{T}(G^*)$, where the tree edges of $\mathcal{T}(G^*)$ are highlighted. (B). A path connecting v_2 and v_3 on the spanning tree $\mathcal{T}(G^*)$. The weight of the path (*i.e.* the summation over weights of the edges along the path) is 0. (C). A cycle consisting of v_1, v_2 and v_3 with weight 1. The cycle contains only one cross edge in $E(G^*)$, and is thus a basic cycle. (D). A cycle consisting of v_2, v_3 and v_4 with weight 0. The cycle contains two cross edges in $E(G^*)$, and is thus not a basic cycle.

this way to facilitate solving the h-variables in subsection 4.2.2). $\mathcal{T}(G^*)$ partitions the edge set $E(G^*)$ into two subsets: *tree edges* and *cross edges*. Conventionally, a cycle containing exactly one cross edge is called a *basic cycle*. Figure 4.5 illustrates an example constraint graph and its basic cycles.

In the rest of this subsection, we will consider some basic properties of the constraint graph, which will be used to check the consistency on tree constraints. The proofs of the following lemmas are given in the appendix.

Lemma 4.2.3 *For any three vertices i, j and k in a connected component of the spanning forest $\mathcal{T}(G^*)$, the weight of the path connecting vertices j and k is equal to the total weight of the path connecting vertices i and j and the path connecting vertices i and k.*

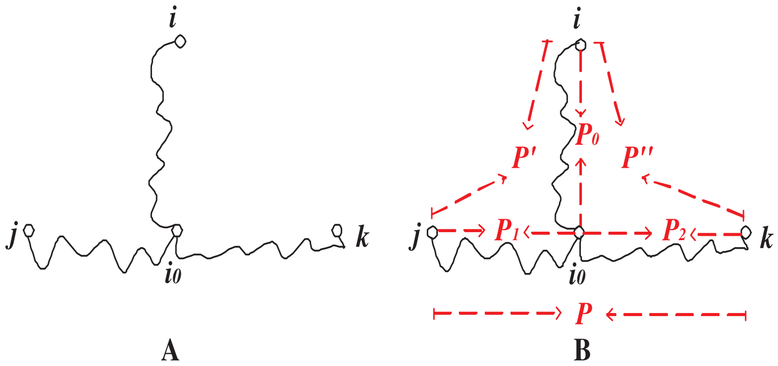

Figure 4.6: The paths generated by three vertices in a connected component in tree $\mathcal{T}(G^*)$. Suppose three vertices are i, j and k. (A) i_0 is the vertex closest to vertex i on the path between vertices j and k. (B) The paths $\mathcal{P}$, $\mathcal{P}'$, $\mathcal{P}''$, $\mathcal{P}_0$, $\mathcal{P}_1$, and $\mathcal{P}_2$.

Proof: Suppose that vertex i_0 is the vertex closest to vertex i on the path between vertices j and k. We denote by $\mathcal{P}$ ($\mathcal{P}'$, $\mathcal{P}''$,$\mathcal{P}_0$, $\mathcal{P}_1$, or $\mathcal{P}_2$) the paths between vertices j and k (vertices i and j, vertices i and k, vertices i and i_0, vertices i_0 and j, or vertices i_0 and k, respectively) as shown in Figure 4.6. Obviously, we have the following equalities:

$$\begin{aligned} W(\mathcal{P}) &= W(\mathcal{P}_1) + W(\mathcal{P}_2) \\ W(\mathcal{P}') &= W(\mathcal{P}_0) + W(\mathcal{P}_1) \\ W(\mathcal{P}'') &= W(\mathcal{P}_0) + W(\mathcal{P}_2) \end{aligned} \tag{4.1}$$

We sum up the above the equalities and obtain the following equality:

$$W(\mathcal{P}) + W(\mathcal{P}') + W(\mathcal{P}'') = 2 \cdot W(\mathcal{P}_0) + 2 \cdot W(\mathcal{P}_1) + 2 \cdot W(\mathcal{P}_2) = 0 \tag{4.2}$$

Therefore, $W(\mathcal{P}) = W(\mathcal{P}') + W(\mathcal{P}'')$ and the lemma holds. ■

Lemma 4.2.4 *In a constraint graph, the weight of every cycle can be represented as a summation over the weights of some basic cycles.*

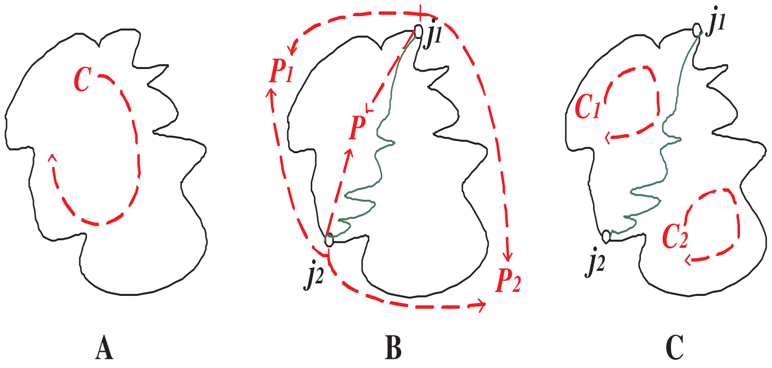

Figure 4.7: The relationship between the weights of cycles and the weights of basic cycles. (A) A cycle $\mathcal{C}$ of order $k \geq 2$, *i.e.* $\mathcal{C}$ contains k cross edges. (B) The two vertices j_1 and j_2 partition the cycle into two paths, $\mathcal{P}_1$ and $\mathcal{P}_2$. The vertices j_1 and j_2 are connected by a path $\mathcal{P}$ in the spanning forest $\mathcal{T}(G^*)$. (C) $\mathcal{P}_1$ and $\mathcal{P}$ form a cycle $\mathcal{C}_1$, and $\mathcal{P}_2$ and $\mathcal{P}$ form a cycle $\mathcal{C}_2$.

Proof: For a cycle $\mathcal{C}$ in the constraint graph G^*, we define the *order* of $\mathcal{C}$ as the number of cross edges it contains. For example, a basic cycle has order one. We prove the lemma inductively on the order of the cycle, denoted by k.

Base step. When $k = 1$, the lemma holds trivially.

Inductive step. Assume that the lemma is true for any cycle of order less than $k \geq 2$. Now we consider a cycle of order k. There exists two vertices j_1 and j_2 on the cycle that partition the cycle into two paths, denoted as $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, such that each of $\mathcal{P}_1$ and $\mathcal{P}_2$ contains at least one cross edge in $E(G^*)$. There exists a path $\mathcal{P}$ between j_1 and j_2 on the spanning forest $\mathcal{T}(G^*)$. Obviously, $\mathcal{P}_1$ and $\mathcal{P}$ form a cycle, say, $\mathcal{C}_1$. $\mathcal{P}_2$ and $\mathcal{P}$ also form a cycle, say, $\mathcal{C}_2$. An illustration of the above cycles and paths is given in Figure 4.7. Obviously, we have the following equalities:

$$\begin{aligned} W(\mathcal{C}) &= W(\mathcal{P}_1) + W(\mathcal{P}_2) \\ W(\mathcal{C}_1) &= W(\mathcal{P}) + W(\mathcal{P}_1) \\ W(\mathcal{C}_2) &= W(\mathcal{P}) + W(\mathcal{P}_2) \end{aligned} \tag{4.3}$$

Summing up these equalities, we obtain the following equality:

$$W(\mathcal{C}) + W(\mathcal{C}_1) + W(\mathcal{C}_1) = 2 \cdot W(\mathcal{P}) + 2 \cdot W(\mathcal{P}_2) + 2 \cdot W(\mathcal{P}_2) = 0 \tag{4.4}$$

Therefore, $W(\mathcal{C}) = W(\mathcal{C}_1) + W(\mathcal{C}_1)$. Observe that both $\mathcal{C}1$ and $\mathcal{C}2$ contain less than k cross edges (*i.e.* $\mathcal{C}1$ and $\mathcal{C}2$ are cycles of order lower than k), which implies that $W(\mathcal{C}1)$ and $W(\mathcal{C}2)$ can be expressed as summations of the weights of some basic cycles by the inductive hypothesis. Consequently, $W(\mathcal{C})$ can be represented as a summation over the weights of some basic cycles. ■

The next corollary follows from Lemma 4.2.4 easily.

Corollary 4.2.5 *In a constraint graph, every cycle has weight 0 if and only every basic cycle has weight 0.*

Now we present our key lemma, which suggests how to use the constraint graph constructed from $C^{\mathbb{T}}$ to check the consistency among the constraints in $C^{\mathbb{T}}$.

Lemma 4.2.6 *$C^{\mathbb{T}}$ has a solution if and only if every cycle in the corresponding constraint graph has weight 0.*

Proof:

"$\Rightarrow$" Consider a cycle $\mathcal{C} = j_0, j_1, \ldots, j_k, j_0$ in the constraint graph, denoting a set of constraints $S = \{(j_0, j_1, b_0, e_0), (j_1, j_2, b_1, e_0), \ldots, (j_k, j_0, b_k, e_0)\}$. If $C^{\mathbb{T}}$ has a solution, there exists an assignment to the h-variables satisfying all the constraints in S. In other words, for every tree constraint $(j, k, b, e_0) \in S$, we have $\sum_{(j_1,j_2)\in\mathcal{P}(j,k)} h_{j_1,j_2} = b$ where $\mathcal{P}(j,k)$ denotes the path connecting vertices j and k in the spanning tree $\mathcal{T}(G)$ of the pedigree graph G, according to the definition of tree constraints. Summing up all the tree constraints, we have

$$\sum_{(j,k,b,e_0)\in S} \sum_{(j_1,j_2)\in\mathcal{P}(j,k)} h_{j_1,j_2} = \sum_{(j,k,b,e_0)\in S} b \tag{4.5}$$

Note that, by ignoring the b-constants, we can view S as a walk in the spanning forest $\mathcal{T}(G)$ starting from vertex j_0, passing through vertices $j_1, \ldots, j_k$, and then returning to vertex j_0. We denote by $\mathcal{W}$ the walk. Rearranging the terms on the left side of Equation (4.5), we have the following equality:

$$\sum_{(j,k,b,e_0)\in S} \sum_{(j_1,j_2)\in\mathcal{P}(j,k)} h_{j_1,j_2} = \sum_{(j_1,j_2)\in E^{\mathbb{T}}} occ((j_1,j_2),\mathcal{W}) \cdot h_{j_1,j_2}, \tag{4.6}$$

where $occ((j_1,j_2),\mathcal{W})$ denotes the number of times that edge (j_1, j_2) occurs in the walk. Because $\mathcal{T}(G)$ is a tree, each edge must occur an even numbered times in the walk, which implies that each h-variable must occur an even numbered times in S. Therefore, the following equality holds

$$\sum_{(j_1,j_2)\in E^{\mathbb{T}}} occ((j_1,j_2),\mathcal{W})\cdot h_{j_1,j_2}=0 \tag{4.7}$$

It follows from Equations (4.5), (4.6) and (4.7) that $\sum_{(j,k,b,e_0)\in S} b=0$, *i.e.*, the cycle $\mathcal{C}$ has weight 0.

"$\Leftarrow$" We will show in the next subsection how to construct a solution for $C^{\mathbb{T}}$ when all the cycles in the constraint graph have weight 0. ■

Corollary 4.2.5 and Lemma 4.2.6 imply the following theorem.

Theorem 4.2.7 *$C^{\mathbb{T}}$ has a solution if and only if every basic cycle in the corresponding constraint graph has weight 0.*

Therefore, we can check the consistency of $C^{\mathbb{T}}$ by detecting the existence of 1-weight basic cycles. Notice that every basic cycle contains exactly one cross edge in $E(G^*)$. We can calculate the weights of basic cycles by first calculating the weight of the path connecting every pair of vertices in each connected component of the spanning forest $\mathcal{T}(G^*)$, and then checking if the basic cycle induced by each cross edge has weight 0. By Lemma 4.2.3, in every connected component $\mathcal{T}$ of the forest $\mathcal{T}(G^*)$, the weight of the path connecting each pair of vertices can be obtained from the weights of the paths connecting the root of $\mathcal{T}$ and each of involved vertices. In the following, we denote by $W(k)$ the weight of the path between the root and a vertex k in a connected component of $\mathcal{T}(G^*)$.

The detailed pseudo-code for checking the consistency of $C^{\mathbb{T}}$ is given in Figure 4.2. Besides consistency checking, the procedure also generates an equivalent constraint set $\widehat{C}$ for $C^{\mathbb{T}}$ based on $\mathcal{T}(G^*)$, which is defined as follows,

$$\widehat{C} = \left\{ (k_0, k, W(k), e_0) \;\middle|\; \begin{array}{l} k \in V,\, k_0 \text{ is the root of the connected component} \\ \text{containing } k \text{ in } \mathcal{T}(G^*)\text{, and } k \neq k_0 \end{array} \right\} \tag{4.8}$$

Clearly, $|\widehat{C}| \leq |V| - 1 = n - 1$. This discussion is summarized formally in the following lemma. [4]

Lemma 4.2.8 *Given the tree constraint set $C^{\mathbb{T}}$, we can reduce it to an equivalent constraint set of size at most $(n-1)$ in $O(|C^{\mathbb{T}})|$ time.*

Proof: We may assume that every basic cycle in G^* has weight 0 (otherwise, we would report the inconsistency of $C^{\mathbb{T}}$ in $O(|C^{\mathbb{T}}|)$ time as shown in Figure 4.2). We need show that $\widehat{C} \equiv C^{\mathbb{T}}$.

On one hand, observe that $\widehat{C}$ defines a new spanning forest $\mathcal{T}$ for G^*. Adding any constraint c ($c \in C^{\mathbb{T}}$) to $\mathcal{T}$ introduces a cycle, say, $\mathcal{C}$. It is easy to verify that the weight of $\mathcal{C}$ is 0, if every basic cycle in G^* has weight 0. As we can see from Equation (4.5), the summation of the weights of the constraints along cycle $\mathcal{C}$ is equal to the weight of the cycle (*i.e.* 0).

---

[4] A similar lemma is given in the previous chapter too.

In other words, the constraint c can be denoted as the summation of some constraints in $\widehat{C}$. Therefore, $\widehat{C} \Rightarrow C^{\mathbb{T}}$.

On the other hand, notice that adding any constraint $\hat{c}$ ($\hat{c} \in \widehat{C}$) to G^* introduces a cycle, denoted as $\mathcal{C}$. It is easy to verify that the weight of $\mathcal{C}$ is 0, if every basic cycle in G^* has weight 0, which implies that the constraint $\hat{c}$ can be denoted as the summation of some constraints in $C^{\mathbb{T}}$. Consequently, $C^{\mathbb{T}} \Rightarrow \widehat{C}$. ■

The following corollary holds from Corollary 4.2.2, Lemma 4.2.8, and the fact $|C^{\mathbb{C}}| + |C^{\mathbb{P}}| + |C^{\mathbb{T}}| \leq 2mn$.

Corollary 4.2.9 *Given the linear constraint set $C = C^{\mathbb{C}} \uplus C^{\mathbb{P}} \uplus C^{\mathbb{T}}$ for a tree pedigree, we can reduce it to an equivalent constraint set of size at most $|E|$ in $O(mn)$ time, where E is the set of the edges in the pedigree graph and $|E| \leq 2n$.*

4.2.2 Solving the h-Variables Efficiently

Now we return to the pedigree graph G and show how to solve the h-variables for tree edges and for cross edges separately. Our idea is as follows. We first construct a one-to-one mapping from the constraints to the edges in the pedigree graph, and then assign values to the h-variables consistent with the constraints by a BFS traversal on the pedigree graph.

Constructing a Mapping from Constraints to Edges. We want to construct a mapping f from the (reduced) constraints $C = C^{\mathbb{C}} \uplus C^{\mathbb{P}} \uplus C^{\mathbb{T}}$ to a subset $E' \subseteq E$ of the edges satisfying the following two conditions:

- *Condition 1:* For every tree/path constraint (or every cycle constraint), there exists exactly one edge of E' on the path (or the cycle, respectively) corresponding to the constraint.
- *Condition 2:* f is one-to-one.

We define the mapping $f : C \mapsto S'$ as follows, where $C = C^{\mathbb{T}} \uplus C^{\mathbb{P}} \uplus C^{\mathbb{C}}$ and the subset S' is implied by the definition.

$$f(c) = \begin{cases} (pred(k), k) & \text{if } c = (j, k, b, e_0) \in C^{\mathbb{T}} \text{ and } index(j) < index(k) \text{ (\textit{i.e.} } c \text{ is a tree constraint)} \\ (i, j) & \text{if } c = (k_1, k_2, b, e) \in C^{\mathbb{P}}\text{, where } e = (i, j) \text{ (\textit{i.e.} } c \text{ is a path constraint)} \\ (i, j) & \text{otherwise } c = (b, e) \in C^{\mathbb{C}}\text{, where } e = (i, j) \text{ (\textit{i.e.} } c \text{ is a cycle constraint)} \end{cases} \tag{4.9}$$

Clearly, the mapping for every path (or cycle) constraint satisfies condition 1. Consider a tree constraint $(j, k, b, e_0) \in C^{\mathbb{T}}$ and suppose that $index(j) < index(k)$. Because we construct the indexing from top to bottom in $\mathcal{T}(G)$, vertex j must be at the same or a higher level than vertex k. Hence, $(pred(k), k)$ must be on the path connecting vertices j and k in $\mathcal{T}(G)$. As a result, condition 1 holds for the mapping f.

Now we want to show that the mapping f is one-to-one. Obviously, each cross edge e is mapped from at most one constraint, since $|C^{\mathbb{C}}(e) \uplus C^{\mathbb{P}}(e)| \leq 1$ by Lemma 4.2.1. Let us next consider tree edges. Assume that there exists a tree edge $(pred(k), k)$ such that two constraints (j_1, k_1, b_1, e_0) and (j_2, k_2, b_2, e_0) are mapped to the edge. Without loss of generality, suppose that $index(j_1) < index(k_1)$ and $index(j_2) < index(k_2)$. According to the definition of the mapping f in Equality (4.9), $k_1 = k_2 = k$. It follows from the construction of $C^{\mathbb{T}}$ given in Equation (4.8), j_1 (or j_2) is the root of the connected component containing k_1 (or k_2, respectively) in $\mathcal{T}(G^*)$. Since $k_1 = k_2 = k$, $j_1 = j_2$. Based on Equation (4.8), $b_1 = b_2 = W(k)$. Therefore, (j_1, k_1, b_1, e_0) and (j_2, k_2, b_2, e_0) are the same constraint. This implies that the mapping f is a one-to-one mapping, *i.e.*, f satisfies condition 2.

Once the mapping f is defined, we can solve the constraints as follows. We set each h-variable corresponding to an edge in $E - E'$ as a free variable, and represent the h-variable corresponding to an edge in E' as a linear combination of the free variables based on the constraint mapped to the edge so that the constraint is satisfied.

Assigning Values to the h-Variables We need assign values to the h-variables so that all the path/cycle/tree constraints are satisfied. Given a vertex j, we denote by $W[j]$ the sum of the h-variables along the path between the root of the spanning tree $\mathcal{T}(G)$ and vertex j. We will treat h-variables for tree edges and cross edges separately, utilizing the mapping f.

- **Handling h-variables for tree edges.** We solve the h-variables by a BFS traversal on $\mathcal{T}(G)$ starting from the root r. First, we set $W[r] = 0$. If we encounter

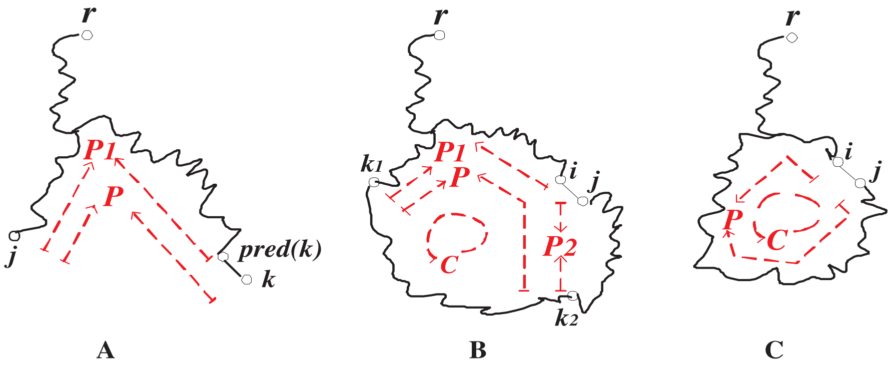

Figure 4.8: An illustration of how to solve h-variables given cycle, path and tree constraints. The spanning tree $\mathcal{T}(G)$ with root r. The tree edges are highlighted. (A) The path $\mathcal{P}$ connecting vertices j and k on $\mathcal{T}(G)$ for a given tree constraint (j, k, b, e_0), and the path $\mathcal{P}_1$ connecting vertices j and $pred(k)$. (B) The cycle $\mathcal{C}$ induced by adding $e = (i, j)$ to the spanning tree $\mathcal{T}(G)$ for given a path constraint (k_1, k_2, b, e), and the paths $\mathcal{P}$, $\mathcal{P}_1$, and $\mathcal{P}_2$ connecting vertices k_1 and k_2, vertices k_1 and i, and vertices j and k_2, respectively. Notice that $\mathcal{P}_1$ and $\mathcal{P}_2$ do not contain e while $\mathcal{P}$ does. (C) The cycle $\mathcal{C}$ induced by adding $e = (i, j)$ to the spanning tree $\mathcal{T}(G)$ for a given cycle constraint (b, e) and the path $\mathcal{P}$ connecting the vertices i and j. Note that $\mathcal{P}$ does not contain e.

a tree edge $(pred(k), k)$ in $(E - E')$, we set $h_{pred(k),k}$ as a free variable, and set $W(k) = h_{pred(k),k} + W(pred(k))$. Otherwise, we encounter a tree edge $(pred(k), k)$ in E' (assuming $f^{-1}((pred(k), k)) = (j, k, b, e_0)$ and $index(j) < index(k)$), and we set $h_{pred(k),k}$ as:

$$h_{pred(k),k} = b + W(j) + W(pred(k)) \tag{4.10}$$

Note that since $index(j) < index(k)$ and $index(pred(k)) < index(k)$, $W(j)$ and $W(pred(k))$ are known at this time. We also set $W(k) = b + W(j)$. An illustration of the vertices j, $pred(k)$ and k is given in Figure 4.8(A). After the BFS traversal, we determine $W[j]$ for every vertex j. We will use $W[j]$ to solve the h-variables for cross edges.

- **Handling h-variables for cross edges:** If we encounter a cross edge $(i, j) \in (E - E')$, we set $h_{i,j}$ as a free variable. Otherwise, we encounter a cross edge $(i, j) \in E'$, and have $|C^{\mathbb{C}}(e) \uplus C^{\mathbb{P}}(e)| = 1$ based on Lemma 4.2.4 and Equation (4.9). We assign $h_{i,j}$ as follows.
  - *Case 1:* $C^{\mathbb{C}}(e) \uplus C^{\mathbb{P}}(e)$ contains a path constraint (k_1, k_2, b, e). An illustration of the vertices k_1, k_2, i and j is demonstrated in Figure 4.8(B). We set

    $$h_{i,j} = b + W(k_1) + W(k_2) + W(i) + W(j) \tag{4.11}$$

  - *Case 2:* $C^{\mathbb{C}}(e) \uplus C^{\mathbb{P}}(e)$ contains a cycle constraint (b, e). An illustration of the vertices i and j is given in Figure 4.8(C). We set

    $$h_{i,j} = b + W(i) + W(j) \tag{4.12}$$

Figure 4.3 displays a pseudo-code of the above procedure. The algorithm is called **Loop_Free_ZRHC_Phase**, which generates a general solution. Note that if we assign a particular binary value to each free variable in the algorithm, the algorithm obtains a particular solution. The correctness of the algorithm is given in the following lemma.

Lemma 4.2.10 *The algorithm* ***Loop_Free_ZRHC_Phase*** *correctly solves the ZRHC problem on a tree pedigree.*

Proof: Clearly, given two vertices j and k, the sum of the h-variables along the path connecting vertices j and k on $\mathcal{T}(G)$ is equal to $W(i) + W(j)$, similar to Lemma 4.2.3.

Given a tree constraint (j, k, b, e_0), we denote by $\mathcal{P}$ the path connecting vertices j and k on $\mathcal{T}(G)$. Furthermore, we denote by $\mathcal{P}_1$ the path connecting vertices j and $pred(k)$ on $\mathcal{T}(G)$ as illustrated in Figure 4.8(A). The weight of $\mathcal{P}$ (*i.e.* the sum of the h-variables along the path) is

$$\begin{aligned} W(\mathcal{P}) &= W(\mathcal{P}_1) + h_{pred(k),k} && \text{(by the definition of weights)} \\ &= (W(j) + W(pred(k))) + (b + W(pred(k)) + W(j)) && \text{(by Lemma 4.2.3 and Equation(4.10))} \\ &= b \end{aligned} \tag{4.13}$$

In other words, the assignment of the h-variables by algorithm **Loop_Free_ZRHC_Phase** satisfies the tree constraint.

Given a path constraint (k_1, k_2, b, e), let us focus on a cycle $\mathcal{C}$ induced by adding edge $e = (i, j)$ to the spanning tree $\mathcal{T}(G)$. On the cycle $\mathcal{C}$, we denote by $\mathcal{P}$ ($\mathcal{P}_1$, or $\mathcal{P}_2$) the path connecting vertices k_1 and k_2 (vertices k_1 and i, or vertices j and k_2, respectively) as shown in in Figure 4.8(B). Notice that $\mathcal{P}_1$ and $\mathcal{P}_2$ do not contain e while $\mathcal{P}$ does. The weight of $\mathcal{P}$ is

$$\begin{aligned} W(\mathcal{P}) &= W(\mathcal{P}_1) + W(\mathcal{P}_2) + h_{i,j} && \text{(by the definition of weights)} \\ &= (W(k_1) + W(i)) + (W(k_2) + W(j)) && \text{(by Lemma 4.2.3)} \\ &\quad +(b + W(k_1) + W(k_2) + W(i) + W(j)) && \text{(by Equation (4.11))} \\ &= b \end{aligned} \tag{4.14}$$

That is, the assignment satisfies the path constraint.

Given a cycle constraint (b, e), a cycle $\mathcal{C}$ is induced by adding edge $e = (i, j)$ to the spanning tree $\mathcal{T}(G)$. On the cycle $\mathcal{C}$, we denote by $\mathcal{P}$ the path connecting vertices i and j as shown in Figure 4.8(C). Note that, $\mathcal{P}$ does not contain e. The weight of $\mathcal{C}$ (*i.e.* the sum of the h-variables along the cycle) is

$$\begin{aligned} W(\mathcal{C}) &= W(\mathcal{P}) + h_{i,j} && \text{(by the definition of weights)} \\ &= (W(i) + W(j)) + (b + W(i) + W(j)) && \text{(by Lemma 4.2.3 and Equation (4.12))} \\ &= b \end{aligned} \tag{4.15}$$

In conclusion, the solution found by algorithm **Loop_free_ZRHC_Phase** satisfies $C^{\mathbb{T}} \uplus C^{\mathbb{P}} \uplus C^{\mathbb{C}}$. Notice that, each (solved) h-variable is represented as a linear combination of free variables and an appropriate constant term. In other words, the algorithm actually finds a general solution for the h-variables. Such a solution for the h-variables can be used to further find a general solution for the p-variables easily as shown in Figure 4.3. Notice that if we assign arbitrary binary values to the free variables, the algorithm would then generate a particular solution. ■

Theorem 4.2.11 *The algorithm* **Loop_Free_ZRHC_Phase** *produces a particular solution to the ZRHC problem on a tree pedigree in $O(mn)$ time, and a general solution in $O(mn^2)$ time.*

Proof: Step 1 takes $O(mn)$ time to generate the locus graphs and step 2 takes $O(mn)$ time to construct the constraints $C^{\mathbb{C}}$, $C^{\mathbb{P}}$ and $C^{\mathbb{T}}$. As we can see from the pseudo-code in Figure 4.2 and Corollary 4.2.9, step $2'$ takes $O(mn)$ time to check the consistency of the constraints and transform the constraints into a set of equivalent constraints with size at most $2n$. According to the pseudo-code in Figure 4.3, step 3 needs $O(n)$ time to construct the mapping f and produce a particular solution for the h-variables, and $O(n^2)$ time to obtain a general solution for the h-variables. As we can see from Figure 4.3, step 4 takes $O(mn)$ time to produce a particular solution for the p-variables and $O(mn^2)$ time to obtain a general solution for the p-variables. ■

4.3 An Example Execution of the Main Algorithm and Tight Time Complexity Analysis

4.3.1 An Example Execution of Algorithm LOOP_FREE_ZRHC_PHASE

The example in the following Figure 4.9 could help demonstrate how the algorithm **Loop_Free_ZRHC_Phase** works. An input pedigree with genotype data is shown in Figure 4.9(A) and the corresponding pedigree graph is in Figure 4.9(B).

In step 1 of the algorithm, we generate the locus graphs (and forests) as illustrated in Figure 4.9(C) and identify the pre-determined vertices in Figure 4.9(D). Moreover, we arbitrarily pick a tree edge, say $e_{1,4}$, as the *indicator* to distinguish tree constraints from path In step 2, we generate the cycle, path and tree constraints. For example, for cycle $\mathcal{C} = v_2v_5v_3v_6v_2$ in the second locus graph of Figure 4.9(D), we denote the cycle constraint $h_{2,5}+h_{3,5}+h_{3,6}+h_{2,6} = 1$ as $(1, e_{3,6})$. This gives us the constraint sets $C^{\mathbb{C}} = \{(1, e_{3,6}), (1, e_{3,7})\}$, $C^{\mathbb{P}} = \{(v_5, v_6, 1, e_{3,6}), (v_5, v_7, 1, e_{3,7})\}$ and $C^{\mathbb{T}} = \{(v_4, v_9, 0, e_{1,4}), (v_4, v_6, 0, e_{1,4}), (v_5, v_9, 0, e_{1,4}), (v_6, v_9, 0, e_{1,4}), (v_7, v_9, 0, e_{1,4})\}$.

In step $2'$, we first remove redundant cycle constraints and obtain $C^{\mathbb{C}} = \{(1, e_{3,6}), (1, e_{3,7})\}$ (*i.e.* no change). Then we take care of the path constraints. For an instance, for path constraint $(v_5, v_6, 1, e_{3,6})$, we generate an equivalent constraint $(v_5, v_6, 0, e_{1,4})$ with the help of cycle constraint $(1, e_{3,6})$. Similarly, we can transform path constraint $(v_5, v_7, 1, e_{3,7})$ into a

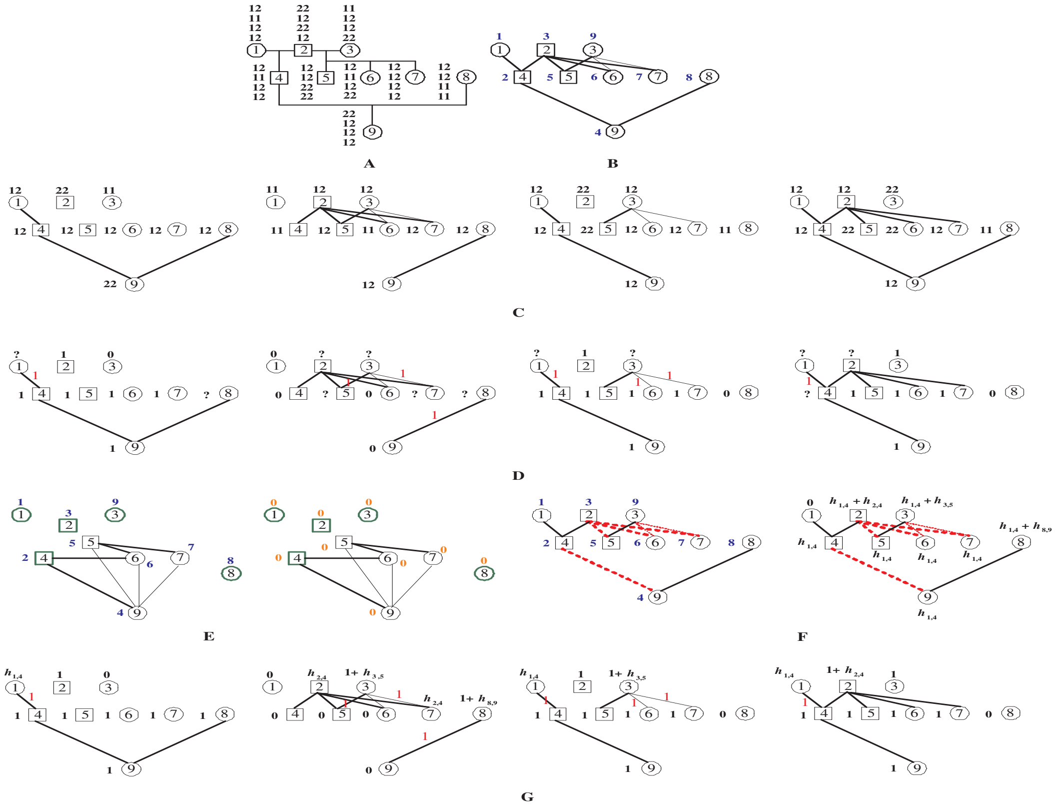

Figure 4.9: An example execution for illustrating algorithm LOOP_FREE_ZRHC_PHASE. The figure is best viewed on the computer screen. (A) An example input pedigree with genotype data. (B) The pedigree graph with a highlighted spanning tree. The blue number beside each vertex is its BFS traversal index. (C) The locus graphs for the four loci, respectively, with the locus forests highlighted. (D) The pre-determined nodes. In the locus graphs, the p-values for each pre-determined nodes are shown beside the nodes, while the undetermined nodes have question marks. The number on each edge is the d-constant for the edge. (E) The constraint graph G^*. Tree edges in the spanning tree $\mathcal{T}(G^*)$ are highlighted. The number beside each vertex in the left graph is the BFS traversal index of the vertex. The root (*i.e.* the vertex with the smallest index) of each connected component in $\mathcal{T}(G^*)$ is highlighted by thick green border. The number in orange color on every vertex in the right graph is the weight (*i.e.* the summation of b-constants) of the path connecting the vertex and the root in a connected component of $\mathcal{T}(G^*)$. (F) The mapping $f : C \mapsto E'$ is constructed, where E' is shown in red dotted line. We show the weight of the path connecting every vertex and the root of $\mathcal{T}(G)$ (*i.e.* summation of the h-variables along the path) as label on the vertex. (G) The locus graphs with propagated p-values. The notations are the same as those in (D), except that the question marks on the undermined nodes are replaced by their resolved p-values. The resolved p-values are expressed as a linear combination of the free p-variables (which do not exist in the example) and the free h-variables with an appropriate constant term.

tree constraint $(v_5, v_7, 0, e_{1,4})$ with the help of cycle constraint $(1, e_{3,7})$. After the transformation, $C^{\mathbb{P}} = \emptyset$. Now $C^{\mathbb{T}} = \{(v_4, v_9, 0, e_{1,4}), (v_4, v_6, 0, e_{1,4}), (v_5, v_9, 0, e_{1,4}), (v_6, v_9, 0, e_{1,4}), (v_7, v_9, 0, e_{1,4}), (v_5, v_6, 0, e_{1,4}), (v_5, v_7, 0, e_{3,7})\}$. Figure 4.9(E) demonstrates the constraint graph G^* for $C^{\mathbb{T}}$ and the spanning forest $\mathcal{T}(G^*)$. We calculate the weight of each path (*i.e.* the summation of the b-constants along the path) connecting a vertex and the root in a connected component of the spanning forest $\mathcal{T}(G^*)$. The weights of the paths will help to detect the existence of 1-weight basic cycle in the constraint graph, and thus check the consistency of $C^{\mathbb{T}}$. Next, we obtain an equivalent new tree constraint set $C^{\mathbb{T}} = \{(v_4, v_5, 0, e_{1,4}), (v_4, v_6, 0, e_{1,4}), (v_4, v_7, 0, e_{1,4}), (v_4, v_9, 0, e_{1,4})\}$, using Equation (4.8).

In step 3, we construct the mapping $f : C \mapsto E'$ as shown in Equation (4.9). For example, for cycle constraint $(1, e_{3,4})$, $f((1, e_{3,4})) = (v_3, v_4)$. This results in the subsets $E' = \{e_{2,5}, e_{2,6}, e_{2,7}, e_{4,9}, e_{3,6}, e_{3,7}\}$ and $E - E' = \{e_{1,4}, e_{2,4}, e_{3,5}, e_{8,9}\}$, where E' is shown in red dotted line in Figure 4.9(F). We set the h-variables for each edge in $E - E'$ as free variables, and denote the h-variables for each edge in E' as a linear combination of free variables and an appropriate constant. Figure 4.9(F) shows the weight of the path (*i.e.* the summation of the h-variables along the path) connecting every vertex and the root of $\mathcal{T}(G)$. Utilizing the mapping f, the algorithm LOOP_FREE_ZRHC_PHASE obtains the following general solution:

$$\begin{cases} h_{2,5} = h_{2,4} \\ h_{2,6} = h_{2,4} \\ h_{2,7} = h_{2,4} \\ h_{4,9} = 0 \\ h_{3,6} = 1 + h_{3,5} \\ h_{3,7} = 1 + h_{3,5} \end{cases}$$

where $h_{1,4}, h_{2,4}, h_{3,5}$ and $h_{8,9}$ (*i.e.* the variables mapped to the edges in $E - E'$) are free variables.

In step 4, we solve the unknown p-values by propagation from the seeds of each connected component and obtain all p-values as shown in Figure 4.9(G). The details of propagation algorithm are given in [46].

4.3.2 An Instance of Loop-Free ZRHC Whose General Solution Has Size $\Theta(mn^2)$

For the pedigree illustrated in Figure 4.10 with the heterozygous genotype for every member at all m loci, no linear constraints on h-variables will be generated. In other words, all h-variables are free variables. Suppose that $n = 2k + 1$ is an odd number. In a specific locus graph, we set the p-value of v_1 as a free variable, and obtain the other p-variables as a linear combination of the free variables and an appropriate constant term. As we can see

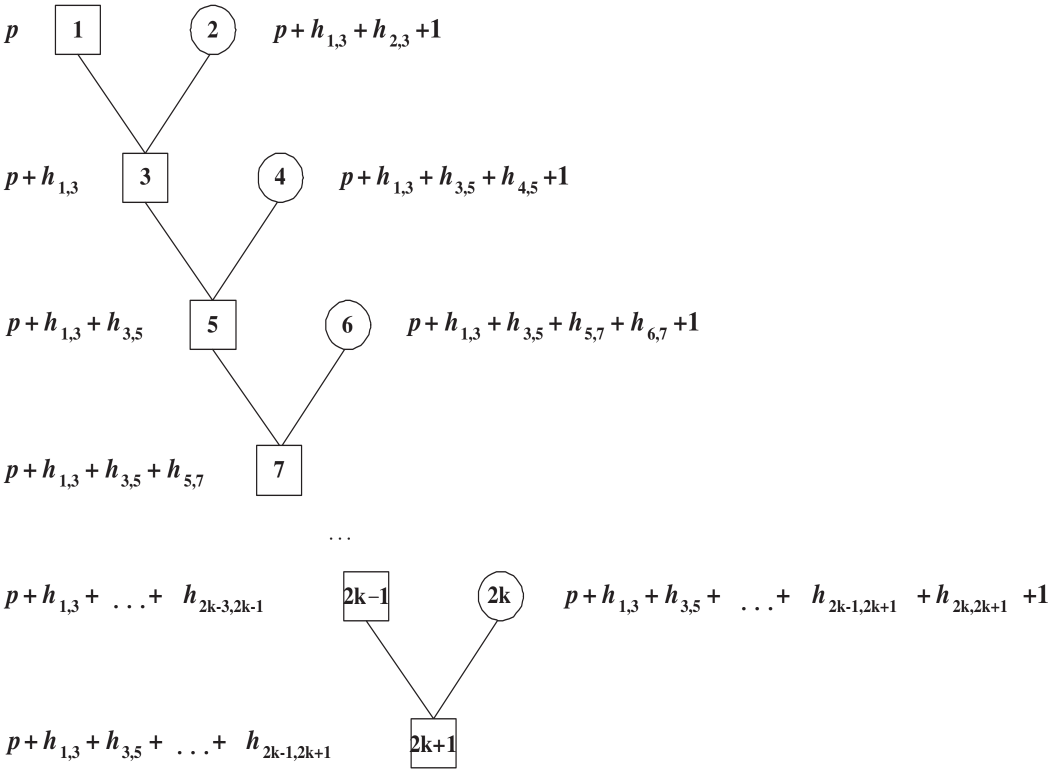

Figure 4.10: An instance of Loop-Free ZRHC whose general solution has size $\Theta(mn^2)$. A pedigree with n members, where $n = 2k + 1$ is an odd number. The genotype in the pedigree are heterozygous for every member at all m loci. The term close to each vertex is the p-value denoted as a linear combination of free h-variables and a free p-variable.

from Figure 4.10, for vertex $2i + 1 (i \leq k)$, its p-value involves $i + 1$ free variables, and for vertex $2i (i \leq k)$, its p-value involves $i + 2$ free variables. Therefore, all p-values in the locus graph require a total of $(\sum_{1 \leq i \leq k}(i+1) + \sum_{1 \leq i \leq k}(i+2)) = \Omega(k^2) = \Omega(n^2)$ free variables. Furthermore, the p-values in all locus graphs involve $\Omega(mn^2)$ free variables. In conclusion, the general solution has size $\Omega(mn^2)$.

4.4 Concluding Remarks

It still remains open if one could extend the algorithm **Loop_Free_ZRHC_Phase** to solve the ZRHC problem on a general pedigree in linear time.

Chapter 5

Efficient Algorithms for Genome-Wide TagSNP Selection across Populations via the Linkage Disequilibrium Criterion

5.1 Introduction

The selection of tagSNPs *in silico* is a well-studied research topic. Existing computational methods for tagSNP selection can be classified into two categories: *haplotype-based* methods [49, 19, 63, 65, 70, 74, 77, 78, 80] and *haplotype-independent* methods [53, 61, 62, 66, 67, 68, 71, 73, 72]. The haplotype-based methods require phased multi-locus haplotypes,

---

[0] An extended abstract of this paper, entitled "Efficient Algorithms for Genome-Wide TagSNP Selection across Populations via the Linkage Disequilibrium Criterion", has accepted by *Proc. 6th Annual International Conference on Computational Systems Bioinformatics (CSB)*, Aug. 2007, San Diego , CA [47].

whereas the haplotype-independent methods do not require haplotype information. The main shortcoming of haplotype-based methods is that the preprocessing step (*i.e.* the inference of haplotypes from genotypes) is computationally demanding. In addition, since there is not an authoritative inference method, the haplotypes generated by the existing haplotype inference methods are often quite different [55, 78, 81]. Consequently, the tagSNPs selected by the haplotype-based methods would be quite different. Recently, Carlson *et al.* [53] proposed a haplotype-independent method that employs the r^2 *linkage disequilibrium* (LD) statistical criterion to measure the association between SNPs. The tagSNPs selected by this method are shown to be effective in disease association mapping studies, because the measure r^2 is directly related to the statistical power of association mapping. Because this method has comparable performance at a lower computational cost than many other methods [79, 73], tagging approaches based on r^2 LD statistics have gained popularity among researchers in the SNP community [50, 53, 56, 68, 72, 79].

Most approaches using the r^2 criterion require that tagSNPs be defined within a single population, because LD patterns (see the caption of Figure 5.1(A) for a definition) are quite susceptible to population stratification [53]. In two populations with different evolutionary histories, a pair of SNPs having remarkably different allele frequencies and very weak LD may show strong LD in the admixed population (see such an example in Table 5.1). Recent study [54] shows that the LD patterns and allele frequencies across populations are very different [54, 75] in fact. For example, among the populations collected in the HapMap project (*i.e.* YRI, CEU, CHB and JPT), 81% of the SNPs in YRI population have a near perfect proxy (*i.e.* SNPs that have $r^2 \geq 0.8$ with other SNPs), while in the other three populations, 91% of the SNPs have a near perfect proxy [45]. Therefore, tagSNPs picked from the combined populations or one of the populations might not be sufficient to capture the variations in all populations. In order to maintain the power of association mapping, we need generate a common (or universal) tagSNP set to type all the populations with sufficient accuracy.

Table 5.1: r^2 statistics for a pair of SNP markers in a single and admixed populations. One SNP has alleles denoted as A and a while the other SNP has alleles denoted as B and b. Population 3 is an even mixture of populations 1 and 2.

	Population 1		
	B	b	
A	0.9025	0.0475	0.95
a	0.0475	0.0025	0.05
	0.95	0.05	$r^2 = 0$

	Population 2		
	B	b	
A	0.0025	0.0475	0.05
a	0.0475	0.9025	0.95
	0.05	0.95	$r^2 = 0$

	Population 3		
	B	b	
A	0.4525	0.0475	0.5
a	0.0475	0.4525	0.5
	0.5	0.5	$r^2 = 0.65$

A simple approach to select a universal tagSNP set is to tag one population first and then select a supplementary set for each of the other populations one by one [50, 69, 68]. For instance, we can select a tagSNP set for non-African populations and a supplement for populations with significant African ancestry [69]. However, this sequential approach might not give a satisfactory solution, as the tagSNP set selected for one population might be far from being adequate to type the SNPs of the remaining populations. As a result, the supplementary tagSNP sets are large and the total number of tagSNPs chosen is far from the optimum.

Moreover, the performance of the approach is sensitive to the specific order of the input populations. In order to generate the smallest set of tagSNPs on K populations, one would have to execute the tagging procedure $K!$ times considering all possible orderings, which would be extremely inefficient for genome-wide tagging. We can improve the performance of the tagging approach by evaluating multiple populations at the same time. When choosing tagSNPs, we prefer those with "good properties" with respect to the collection of populations as a whole. An example of our tagging strategy is given in Figure 5.1.

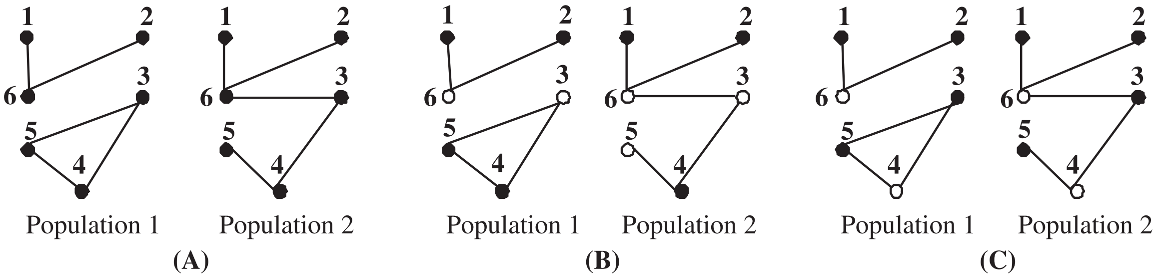

Figure 5.1: Different LD patterns in each of the two populations and mixed population. (A). LD patterns in two populations. The vertices denote the SNP markers and the edges denote pairs of markers with strong LD (*i.e.* the r^2 measure between the markers is greater than a given threshold). (B). Tagging results of the above simple sequential approach. We first choose markers 3 and 6 to tag population 1 and then choose an additional marker 5 to tag population 2. Three markers are selected in total to tag both populations. (C). Tagging results of an improved approach. We select markers 4 and 6 considering both populations simultaneously. Only two markers are selected in total to tag both populations.

Previous work on tagSNP selection based on the linkage disequilibrium criterion. There is a large body of scientific literature on the problem of selecting tagSNPs based on the r^2 LD criterion. Carlson *et al.* suggested a greedy procedure called LD-Select, which works as follows: (i) select the SNP with the maximum number of proxies, (ii) remove the SNP and its proxies from consideration, and (iii) repeat the above two steps until all SNPs have been tagged [53]. This algorithm is very simple, however it may miss solutions with the smallest number of tagSNPs in general, as pointed out in [72].

More recently, Qin *et al.* implemented a comprehensive search algorithm called FESTA, which first breaks down a large set of markers into disjoint pieces (called *precincts*), and then performs an exhaustive search on each piece if the estimated computational cost is below a certain threshold [72]. FESTA usually gives a better solution than LD-Select, but due to the fact that it employs exhaustive search, it is too slow to be practical for genome-wide tagSNP selection. The above two methods are only applicable to single population tagSNP selection. Recently, Howie *et al.* presented an algorithm for multiple populations by extending LD-Select, called MultiPop-TagSelect. MultiPop-TagSelect combines the tagSNPs selected for each population by LD-Select to produce a universal tagSNP set for a collection of populations [59]. The algorithm works reliably, and it could in principle be used with any tagSNP selection method for single populations. However, its accuracy highly depends on the performance of the single-population tagSNP selection method. Magi *et al.* [68] also designed a software tool called REAPER which is rather similar to LD-Select if applied to a single population. To select a universal tagSNP set for several populations, it first selects a tagSNP set for one population, and then it selects a supplement for the remaining populations one by one.

As mentioned above, the performance of the method crucially depends on the choice of the initial tagSNP set and the ordering of the populations. It is not clear, moreover, how one should select tagSNPs for the first population so as to minimize the size of the final solution.

Our contribution on tagSNP section based on the linkage disequilibrium criterion. In this chapter, we take a different approach to the multi-population tagSNP selection problem. Contrary to the previous methods, we do not generate a tagSNP set for each individual population separately, but rather we evaluate all the populations at the same time. The method that we propose could be used to generate a universal or cosmopolitan tagSNP set for multi-ethnic, ethic-unknown or admixed populations [59].

The main idea of our approach is to transform a multi-population tagSNP selection problem, called the *minimum common tagSNP selection* (MCTS) problem (to be defined more precisely later in the chapter), into a minimum common dominating vertex set problem on multiple graphs. Each graph corresponds to one of the populations under consideration. The vertices in a graph correspond to the SNP markers of the population, and there is an edge between two markers when they are in strong LD (according to some given threshold). To find an optimal solution MCTS, we first decompose it into disjoint subproblems, each of which is essentially a connected component of the union graph[1] and represents a precinct as defined in [72]. Then, for each precinct, we apply three data reduction rules repeatedly to further reduce the size of the subproblem, until none of the rules can be applied anymore. Finally, the reduced subproblems are solved by either a simple greedy approach (similar to *cosmopolitan*

---

[1] Given graphs $G_i = (V_i, E_i)(1 \leq i \leq k)$, the union graph is defined as $G = (V, E)$, where $V = \bigcup_i V_i$ and $E = \bigcup_i E_i$.

tagging [50]) or a more sophisticated Lagrangian relaxation heuristic. Both algorithms will be explained in detail later in the chapter. Along with the solution produced by our algorithm, we also obtain lower bounds on the minimum number of tagSNPs required, which allows us to quantitatively assess how close our solution is from the true optimum.

We evaluate the performance of our method on real HapMap data for genome-wide tagging. The experimental results demonstrate that our algorithms run 3 to 4 orders of magnitude faster than the existing single-population tagging programs like FESTA, LD-Select and the multiple-population tagging method MultiPop-TagSelect. Our method also greatly reduces the required tagSNPs compared to LD-Select on a single population and MultiPop-TagSelect on multiple populations. Moreover, the numbers of tagSNPs selected by our algorithms are almost optimal since they are very close to the corresponding lower bounds provided by our method. For example, the gap between our solution and the lower bound is 1061 SNPs with r^2 threshold being 0.5 and 142 SNPs with the r^2 threshold being 0.8, given the entire human genome with 2,862,454 SNPs (MAF being 5%).

The rest of the chapter is organized as follows. In Section 5.2, we first propose a combinatorial optimization model for the MCTS problem and then present a computational complexity result. In Section 5.3, we introduce three rules to reduce the size of the problem, and devise a greedy tagging algorithm, called GreedyTag, and a Lagrangian relaxation heuristic, called LRTag. After showing the experimental results in Section 5.4, we conclude the chapter with some remarks about the performance of our tagging method in Section 5.5.

5.2 Formulation of the MCTS Problem

Consider K distinct populations and a set V of biallelic SNP markers v_1, v_2, …, v_n. Since the r^2 coefficient is unreliable for rare SNPs when the sample size is small [53], we will consider only SNPs with MAF $\geq 5\%$. The set of SNPs might be different from population to population. We use $V_i \subseteq V$ to denote the SNP set in population i. Clearly, we have $V = V_1 \cup V_2 \cup \cdots \cup V_k$.

For a pair of SNP markers v_{j_1} and v_{j_2} in a population i (for any $1 \leq i \leq K$), the r^2 coefficient between them is denoted by $r_i^2(v_{j_1}, v_{j_2})$. Markers v_{j_1} and v_{j_2} are said to be in *high LD* in population i, if $r_i^2(v_{j_1}, v_{j_2}) \geq \gamma_0$, where γ_0 is a pre-defined threshold (γ_0 will be set to 0.5 or higher in our study). Moreover, v_{j_1} (or v_{j_2}) is considered being the *tagSNP* or *proxy* for v_{j_2} (or v_{j_1}, respectively) in population i. For convenience, we define E_i to be the set containing all the high-LD marker pairs in population i, *i.e.* $E_i = \{(v_{j_1}, v_{j_2}) | r_i^2(v_{j_1}, v_{j_2}) \geq \gamma_0,\ v_{j_1}, v_{j_2} \in E_i\}$. Now we can formally define the MCTS problem.

MINIMUM COMMON TAGSNP SELECTION(MCTS)

Instance: A collection of K populations and a set V of biallelic SNP markers. Each population i ($1 \leq i \leq K$) has its marker set $V_i \subseteq V$ and LD patterns $E_i = \{(v_{j_1}, v_{j_2}) | r_i^2(v_{j_1}, v_{j_2}) \geq \gamma_0,\ v_{j_1}, v_{j_2} \in V_i\}$, where γ_0 is a pre-defined threshold.

Feasible solution: A subset $T \subseteq V$ such that for any marker $v \in V_i, v \notin T$ from some population i, there exists a marker v' in $T \cap V_i$ with $(v, v') \in E_i$ (that is, $r_i^2(v, v') \geq \gamma_0$).

Objective: Minimize |T|.

It is easy to observe that any feasible solution to the MCTS problem is a common dominating vertex set in the graphs $\{G_i | 1 \leq i \leq K\}$, where $G_i = (V_i, E_i)$. In particular, the smallest set of tagSNPs for a single population is a minimum dominating vertex set of the corresponding graph. Obviously, the MCTS problem is NP-hard, since it is a generalization of the minimum dominating vertex set problem, which is known to be NP-hard [52].

Theorem 5.2.1 *The MCTS problem is NP-hard.*

We introduce some additional notations to be used later. To differentiate the occurrences of a marker in different populations, we use v_j^i to represent the j^{th} marker appearing in the i^{th} population. Given a marker $v_j \in V$, we define the following two sets:

$$\begin{aligned} N^i(v_j) &= \{v_{j'}^i | (v_j, v_{j'}) \in E_i, v_j, v_{j'} \in V_i\} \cup \{v_j^i | v_j \in V_i\} \\ N^*(v_j) &= \bigcup_{1 \leq i \leq K} N^i(v_j) \end{aligned} \tag{5.1}$$

The set $N^i(v_j)$ represents the subset of markers in strong-LD with v_j in population i, and the set $N^*(v_j)$ represents the union of such subsets for all the populations. Note that, $N^i(v_j)$ is empty if $v_j \notin V_i$. Given a marker $v_j \in V_i$ in population i, we define the following set:

$$C(v_j^i) = \{v_{j'} | (v_j, v_{j'}) \in E_i, v_j, v_{j'} \in V_i\} \cup \{v_j\} \tag{5.2}$$

The set $C(v_j^i)$ is the subset of markers each of which can tag the occurrence v_j^i, whereas $N^*(v_j)$ is the subset of occurrences that the marker v_j can tag.

Based on the above definitions, the MCTS problem can also be viewed as the following set cover problem. Given the universe $\mathcal{U} = \bigcup_{1 \le i \le K} \{v_j^i | v_j \in V_i\}$ and the collection $\mathcal{C} = \{N^*(v_j) | v_j \in V\}$, find a subcollection of sets from $\mathcal{C}$ to cover $\mathcal{U}$. Clearly, the number of sets in a minimum set cover is equal to the number of markers in a minimum tagSNP set.

As a consequence, approximation algorithms that solve set cover can be applied to the MCTS problem. In practice, greedy algorithms are commonly used for set cover due to their simplicity and effectiveness. The simplest greedy algorithm for set cover, which picks the set that covers the most number of uncovered elements each time, achieves an approximation ratio of $\log(m)$, where m is the number of elements to be covered [76]. This implies a $\log(Kn)$ approximation algorithm for MCTS, $|\mathcal{U}| \le Kn$.

However, the approximation ratio of $\log(Kn)$ could be too large in practice, due to the fact that V may contain millions of markers and $n = |V|$. Therefore, the solution produced by the above greedy approach may not be sufficiently small. We aim to design efficient heuristics to provide better solutions.

5.3 Optimization Techniques to Solve the MCTS Problem

In principle, a minimum common tagSNP set can be found by exhaustive search. In reality, there are millions of SNP markers, and it is infeasible to conduct the exhaustive search. Since human chromosomes consist of high-LD regions (*i.e.* haplotype blocks) interspersed with *recombination hotspots*, we partition the markers into precincts such that markers in strong LD will belong in the same precinct. In this way, we could narrow down the search space and thus improve the efficiency of our algorithm.

In order to deal with multiple populations, we extend the concept of precinct defined originally in [72]. We say that two markers are in the same *precinct* if and only if they are in strong LD in some population. Based on the simple observation that no marker in a precinct can tag a marker in another tag a marker in another precinct, we can obtain a minimum tagSNP set for the combining the minimum tagSNP sets for each precinct. The precincts can be easily identified by running a breath first search (BFS) in the union graph G. By partitioning the markers into precincts, we decompose the original problem into a set of disjoint subproblems of much smaller sizes. We then select tagSNPs for each precinct independently, which could save a lot of running time.

5.3.1 Data Reduction Rules

In order to further reduce the subproblem sizes and improve efficiency, we introduce three simple data reduction rules.

- Rule 1: *Pick all irreplaceable markers*. If a marker v_j has no proxy from population i (that is, v_j is a singleton in $G_i = (V_i, E_i)$), then marker v_j must be in the minimum tagSNP set.
- Rule 2: *Remove less informative markers*. Given two markers $v_{j'}$ and v_j, if $N^*(v_{j'}) \subseteq N^*(v_j)$, we say that v_j is more *informative* than $v_{j'}$. Similarly, given a set of markers v_{j_1}, v_{j_1},..., v_{j_k}, if $N^*(v_{j_1}) \subseteq N^*(v_{j_2}) \subseteq \ldots \subseteq N^*(v_{j_k})$, v_{j_k} is called the *maximally informative* SNP marker in the set. It is clear that we can discard less informative SNPs and only keep those maximally informative ones without degrading the quality of the solution.
- Rule 3: *Remove less stringent occurrences*. Given two occurrences $v_{j'}^{i'}$ and v_j^i, if $C_j^i \subseteq C_{j'}^{i'}$, we say that $v_{j'}^{i'}$ is *less stringent* than v_j^i. Similarly, given a set of occurrences $v_{j1}^{i_1}$, $v_{j2}^{i_2}$,..., $v_{j_k}^{i_k}$, if $C_{j_k}^{i_k} \subseteq \ldots \subseteq C_{j2}^{i_2} \subseteq C_{j1}^{i_1}$, the occurrence $v_{j_k}^{i_k}$ is called the *most stringent* occurrence in the set. Observe that the markers selected to tag the most stringent occurrences will also tag the less stringent occurrences. Therefore, we consider only the most stringent occurrences and discard the others.

The above rules can also be viewed as data reduction rules applied to a 0/1 matrix obtained as follows. Given the notations of the occurrence set $\mathcal{U}$, the marker set V and the neighborhood collections $\mathcal{C}$ introduced in previous section, the rows in the matrix represent $\mathcal{U}$, the columns denote V, and each cell (i, j) indicates whether the marker corresponding to column j can tag the occurrence corresponding to row i (*i.e.* the value of a cell is set to 1 if the marker can tag the occurrence, and 0 otherwise). Thus, Rule 2 is equivalent to redundant column deletion, and Rule 3 is equivalent to redundant row deletion.

The above rules can be applied repeatedly and in any combination whenever applicable. The reduced problem obtained after the application of the above data reduction rules will be subject to our greedy algorithm or Lagrangian relaxation algorithm, as explained next.

5.3.2 A Greedy Algorithm for MCTS

Greedy algorithms are often desirable due to their simplicity and efficiency. The greedy algorithm, *GreedyTag*, below is adapted from the greedy algorithm for the set cover problem as presented in [76]. By first applying the above data reduction rules, we will show later in the chapter that GreedyTag greatly outperforms the other greedy algorithms such as LD-Select and MultiPop-TagSelect. Moreover, a lower bound, called *GreedyTag_lb*, is produced by GreedyTag, which is equal to the number of tagSNPs selected by data reduction Rule 1. Even though the lower bound is is somewhat loose since we only consider Rule 1, it turned out to be pretty tight in our experiments on real data (see Section 5.4 for more details). Due to space constraint, we present the pseudo-code of GreedyTag in the following figure.

Algorithm 1 (GreedyTag: Greedy Algorithm for TagSNP Selection in Multiple Populations)

```
Input: A set $V$ of biallelic SNP markers and their pairwise $r^2$ LD statistics in $K$ distinct populations. A pre-defined threshold $\gamma_0$ for $r^2$ LD statistics.
Output: A feasible tagSNP set $T \subseteq V$, and a lower bound $LB$.
Begin
  Partition markers into precincts. Let the set of precincts be $\mathcal{P}$.
  For each precinct $p \in \mathcal{P}$ {the following will be executed in parallel on a multi-processor machine}
    Let $U$ be the set of SNPs and $W$ the set of marker occurrences in $p$.

    Step 1: Apply the three data reduction rules.
    $T_p \Leftarrow \emptyset$; $LB_p \Leftarrow 0$; UPDATED $\Leftarrow$ true;
    While UPDATED { execute the optimal rules iteratively}
      UPDATED $\Leftarrow$ false;
      If there exists an irreplaceable marker $v_j \in U$ {Rule 1}
        $U \Leftarrow U - \{v_j\}$;   $W \Leftarrow W - N^*(v_j)$; {$N^*(v_j)$ is defined in Equation (5.1) }
        $T_p \Leftarrow T_p \cup \{v_j\}$;   $LB_p \Leftarrow LB_p + 1$;   UPDATED $\Leftarrow$ true;
      If there exists a less informative marker $v_j \in U$ {Rule 2}
        $U \Leftarrow U - \{v_j\}$;   UPDATED $\Leftarrow$ true;
      If there exists a less stringent occurrence $v_j^i \in W$ {Rule 3}
        $W \Leftarrow W - \{v_j^i\}$; UPDATED $\Leftarrow$ true;
    For each $v_j \in U$
      $D(v_j) \Leftarrow N^*(v_j) \cap W$;

    Step 2: Select tagSNPs greedily.
    While $W$ is non-empty {there are markers to be tagged}
      Let $v_{j_0} \Leftarrow argmax_{v_j \in U} |D(v_j)|$;
      $T_p \Leftarrow T_p \cup \{v_{j_0}\}$; $U \Leftarrow U - \{v_{j_0}\}$; $W \Leftarrow W - N^*(v_{j_0})$;
      For each $v_j \in U$
        $D(v_j) \Leftarrow D(v_j) \cap W$;

  $T \Leftarrow \bigcup_{p \in \mathcal{P}} T_p$;   $LB \Leftarrow \sum_{p \in \mathcal{P}} LB_p$
  Output $T$ , $LB$ {output the solution and lower bound}
End
```

Algorithm 2 (LRTag: Lagrangian relaxation Algorithm for TagSNP Selection in Multiple Populations)

```
Input: A set $V$ of $n$ biallelic SNP markers and their pairwise $r^2$ LD statistics in $K$ distinct populations. A pre-defined threshold $\gamma_0$ for $r^2$ LD statistics. A pre-defined initial scalar $\alpha_0$ and threshold $\alpha_{min}$ for subgradient optimization. A pre-defined maximum number $Iter_{max}$ of iterations and a pre-defined threshold $K_{max}$ of maximum trials.
Output: A feasible tagSNP set $T \subseteq V$, and a lower bound $LB$.
Begin
  Partition markers into precincts. Let the set of precincts be $\mathcal{P}$
  For each precinct $p \in \mathcal{P}$ $\{$the following will be executed in parallel on a multi-processor machine$\}$
    Let $U$ be the set of SNPs and $W$ be the set of marker occurrences in $p$.
    Step 1: Apply the data reduction rules and obtain a temporary tagSNP set $T_p$ and a lower bound $LB_p$.
    $\{$The same as the rules in GreedyTag algorithm (see Algorithm 1 for details)$\}$.
    Step 2: Select tagSNPs under a Lagrangian relaxation framework.
    Generate Lagrangian relaxation formula as in Equation (5.4);
    $k \Leftarrow 0;\ \alpha \Leftarrow \alpha_0;\ Iter \Leftarrow 0;$ Initialize $\boldsymbol{\lambda}$ being an arbitrary non-negative vector; $LB_{p1} \Leftarrow 0;\ T_{p1} \Leftarrow U;$
    While $(\alpha > \alpha_{min})$ and $(Iter < Iter_{max})$
      $Iter \Leftarrow Iter + 1;\ new\_LB \Leftarrow \sum_{1 \le i \le K, 1 \le j \le n} \lambda_{i,j};\ new\_T \Leftarrow \emptyset;$
      $\{$Calculate a new lower bound $new\_LB\}$
      For each $v_j \in U$
        $s_j \Leftarrow 1 - \sum_{v^i_{j'} \in\ N^*(v_j)} \lambda_{i,j'};$ $\{N^*(v_j)$ is defined in Equation (5.1) $\}$
        If $s_j \le 0$   $t_j \Leftarrow 1;$   Else   $t_j \Leftarrow 0;$
        $new\_LB \Leftarrow new\_LB + s_j \cdot t_j;$
      $\{$Obtain a feasible tagSNP set $new\_T$ by the RCH method [51]$\}$
      For each $v_j \in U$      $RCH\_s_j \Leftarrow s_j;$
      For each $v^i_j \in W$      $RCH\_\lambda_{i,j} \Leftarrow \lambda_{i,j};$
      For each $v^i_j \in W$
        If $\sum_{v_{j'} \in\ C(v^i_j)} t_{j'} < 1$   $\{C(v^i_j)$ is defined in Equation (5.2) $\}$
          $RCH\_s_m \Leftarrow min\{RCH\_s_{j'} : v_{j'} \in\ C(v^i_j)\};$
          $RCH\_\lambda_{i,j} \Leftarrow RCH\_\lambda_{i,j} + RCH\_s_m;$
          For each $v_{j'} \in C(v^i_j)$
            $RCH\_s_{j'} \Leftarrow RCH\_s_{j'} - RCH\_s_m;$
            If $RCH\_s_{j'} \le 0$    $t_{j'} \Leftarrow 1;$
      For each $v_j \in U$
        If $t_j = 1$    $new\_T \Leftarrow new\_T \cup \{v_j\};$
      $\{$Update the lower bound $LB_{p1}$ and the tagSNP set $T_{p1}$ $\}$
      If $new\_LB \le LB_{p1}$    $k \Leftarrow k + 1;$
        If $(k \ge K_{max})$    $\alpha \Leftarrow \alpha/2;\ k \Leftarrow 0;$
      Else   $LB_{p1} \Leftarrow new\_LB;\ k \Leftarrow 0;$
      If $|new\_T| < |T_{p1}|$    $T_{p1} \Leftarrow new\_T;$
      $\{$Update the Lagrangian multipliers $\boldsymbol{\lambda}$ by the subgradient optimization method $\}$
      For each $v^i_j \in W$    $\nabla\lambda_{i,j} \Leftarrow 1 - \sum_{v_{j'} \in C(v^i_j)} t_{j'};$
      $\boldsymbol{\lambda} \Leftarrow max\{\mathbf{0}, \boldsymbol{\lambda} + \alpha \frac{|T_{p1}| - LB_{p1}}{||\nabla\boldsymbol{\lambda}||^2} \nabla\boldsymbol{\lambda}\};$
    $\{$Combine the solutions from step 1 and step 2$\}$
    $T_p \Leftarrow T_p \cup T_{p1};\ LB_p \Leftarrow LB_p + LB_{p1};$
  $T \Leftarrow \bigcup_{p \in \mathcal{P}} T_p;\ LB \Leftarrow \sum_{p \in \mathcal{P}} LB_p$
  Output $T,\ LB$ $\{$output the solution and the lower bound$\}$
End
```

5.3.3 A Lagrangian Relaxation Algorithm for MCTS

A subset T of SNPs can be denoted by its characteristic vector $\mathbf{t} = t_1 t_2 \ldots t_n$, where $t_i = 1$ if $v_i \in T$, and $t_i = 0$ otherwise. It is thus easy to formulate the following integer linear program for MCTS.

$$\begin{aligned} &\text{Minimize} && |T| = \textstyle\sum_{1 \le j \le n} t_j \\ &\text{subject to} && \textstyle\sum_{v_{j'} \in\, C(v_j^i)} t_{j'} \ge 1 \qquad 1 \le i \le K \text{ and } 1 \le j \le n \\ & && t_j \in \{0,1\}, 1 \le j \le n \end{aligned} \tag{5.3}$$

Our second algorithm for MCTS is based on the Lagrangian relaxation framework. We assign a non-negative vector $\boldsymbol{\lambda} = \lambda_{11}\lambda_{12}\ldots\lambda_{K,n}$ of Lagrangian multipliers to the inequalities, and obtain the following relaxed integer program.

$$\begin{aligned} &\text{Minimize} && L(\mathbf{t}, \boldsymbol{\lambda}) = \textstyle\sum_{1 \le j \le n} t_j + \sum_{1 \le i \le K, 1 \le j \le n} \lambda_{i,j}(1 - \sum_{v_{j'} \in\, C(v_j^i)} t_{j'}) \\ &\text{subject to} && t_j \in \{0,1\}, \lambda_{i,j} \ge 0, 1 \le i \le K, 1 \le j \le n \end{aligned} \tag{5.4}$$

For a given $\boldsymbol{\lambda}$, define $L(\boldsymbol{\lambda}) = min\ L(\mathbf{t}, \boldsymbol{\lambda})$. Observe that the size of any feasible tagSNP set T would be an upper bound for $L(\boldsymbol{\lambda})$ in (5.4), and any $L(\boldsymbol{\lambda})$ would be a lower bound for $|T|$. Hence, we look for $max\ L(\boldsymbol{\lambda})$, which gives the best lower bound for $min\ |T|$.

For any given $\boldsymbol{\lambda}$, we can easily obtain $L(\boldsymbol{\lambda})$ in (5.4) as follows. For convenience, we define $s(t_j)(1 \le j \le n)$ as:

$$s(t_j) = 1 - \sum_{1 \le i \le K,\ v_{j'} \in\ C(v_j^i)} \lambda_{i,j'} = 1 - \sum_{v_{j'}^i \in\ N^*(v_j)} \lambda_{i,j'} \tag{5.5}$$

which are the *Lagrangian costs* associated with t_j in (5.4). Rearranging the terms in (5.4), we have the objective function $L(\mathbf{t}, \boldsymbol{\lambda}) = \sum_{1 \le i \le K, 1 \le j \le n} \lambda_{i,j} + \sum_{1 \le j \le n} s(t_j) \cdot t_j$. In order to minimize the objective function, we have to set $t_j = 0$ if $s(t_j) > 0$, $t_j = 1$ if $s(t_j) < 0$, and t_j an arbitrary value if $s(t_j) = 0$.

The vector $\mathbf{t}$ obtained above may not be a feasible solution to (5.3). In other words, some occurrence might not be tagged by any marker in $T = \{v_j | t_j = 1, 1 \le j \le n\}$ induced by the characteristic vector $\mathbf{t}$. We will adopt a strategy *reduced cost heuristic* (RCH) introduced by Balas and Ho [51] to deal with this issue (the details are given in the pseudo-code in the figure for Algorithm 2).

Next we need find a good multiplier vector $\boldsymbol{\lambda}$, *i.e.* one that gives a near optimal lower bound. We utilize a standard optimization technique called *subgradient optimization* [51], which iteratively updates the solution toward the subgradient direction to reach the optimum. We can define

$$S(\lambda_{i,j}) = 1 - \sum_{v_{j'} \in\ C(v_j^i)} t_{j'} \tag{5.6}$$

which simplifies $L(\mathbf{t}, \boldsymbol{\lambda}) = \sum_{1 \le j \le n} t_j + \sum_{1 \le i \le K, 1 \le j \le n} S(\lambda_{i,j}) \cdot \lambda_{i,j}$. Obviously, $\nabla \boldsymbol{\lambda} = (\nabla \lambda_{11}, \nabla \lambda_{12}, ..., \nabla \lambda_{K,n})$ where $\nabla \lambda_{i,j} = S(\lambda_{i,j})$. Starting from a initial setting $\boldsymbol{\lambda}^0$, we sequentially generate $\boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2, \boldsymbol{\lambda}^3, \ldots$, based on the following formula

$$\boldsymbol{\lambda}^{k+1} = max\{\mathbf{0}, \boldsymbol{\lambda}^k + \alpha_k \frac{|T^*| - L^*}{||\nabla \boldsymbol{\lambda}^k||^2} \nabla \boldsymbol{\lambda}^k\} \tag{5.7}$$

where T^* is the smallest feasbile tagSNP set found so far (*i.e.* the best upper bound for $max\ L(\mathbf{t})$), L^* is the largest of $max\ L(\mathbf{t})$ found so far (*i.e.* the best lower bound for $max\ L(\mathbf{t})$), and $\{\alpha_0, \alpha_1, \ldots\}$ is a decreasing sequence of pre-defined scalars.

The pseudo-code for the Lagrangian relaxation algorithm, *LRTag*, is given in the figure for Algorithm 2. In the algorithm, we start from a initial setting of $\boldsymbol{\lambda}^0$, generate a solution to $\mathbf{t}^0$ and extend it to a valid tagSNP set as mentioned above. Then we update $\boldsymbol{\lambda}^0$ into $\boldsymbol{\lambda}^1$ according to the formula (5.7). We repeat the process until we cannot improve $\boldsymbol{\lambda}$ or a pre-defined number of maximum iterations is reached. Over the entire iterative process, the smallest feasible set of tagSNPs found by LRTag would be output as a solution to the MCTS problem, and the largest $L(\mathbf{t})$ would be a lower bound for tagSNP selection, called *LRTag_lb*.

5.4 Experimental Results

In our experiments, we test the algorithms GreedyTag and LRTag on the Hapmap populations, and compare their performance and efficiency with single-population tagging programs LD-Select and FESTA, and a multiple-population tagging program MultiPop-TagSelect. For convenience, we will also denote by GreedyTag the cardinality of a feasible tagSNP set obtained by the GreedyTag algorithm. We use similar notations for LRTag, LD-Select, FESTA and MultiPop-TagSelect.

Both of our algorithms calculate lower bounds on the minimum number of required tagSNPs, one of which is found by GreedyTag (*i.e.* GreedyTag_lb) and the other by LRTag (*i.e.* LRTag_lb). We define *gap* as the difference between the highest lower bound and the cardinality of the smallest tagSNP set found by our algorithms, which will be used to measure the quality of the solutions.

We apply all the methods on the entire human genome data involving chromosomes 1 through 22 and on all ENCODE regions (ENm010, ENm013, ENm014, ENr112, ENr113, ENr123, ENr131, ENr213, ENr232 and ENr321) genotyped by the HapMap project (release #19, NCBI build 34, October 2005). For the ENCODE data, we estimate the r^2 statistics by using a two-marker EM algorithm to compute the maximum-likelihood values of the four gamete frequencies, which is also commonly adopted by LD-Select and HaploView [50]. For the entire human genome data, we directly download the r^2 statistics from the HapMap website [57], generated by HaploView to save computational cost. Note that, HaploView only calculates LD for markers up to 250 kbps apart, which is reasonable because the LD for markers that are farther than 250 kbps would normally be very weak anyway, and high LD in such a case can happen only purely by chance. In order to save running time for dealing with the entire human genome data, we prune the LD pattern data downloaded from the HapMap website by keeping only entries with r^2 greater than or equal to 0.5.

We ran all the programs on a 32-processor SGI Altix 4700 supercomputer system with 1.6HZ CPU and 64 GB shared memory in the Computer Science Department, University of California - Riverside. Our GreedyTag and LRTag algorithms used up to 15 threads in parallel, while each of the other programs is single-threaded.[2]

5.4.1 Tagging the ENCODE Regions

A dense set of SNPs across ten large genomic regions have been produced by the HapMap ENCODE project. These regions serve as the foundation to evaluate the development of methodologies and technologies for detecting functional elements in human DNA. Each region is about 500Kb in length and has an SNP density about 1 SNP per 600 bps.

Table 5.2: Summary of tagSNPs identified by FESTA, LD-Select, GreedyTag and LRTag for a single population, CEU, on all ENCODE regions.

Region	ENm010	ENm013	ENm014	ENr112	ENr113	ENr123	ENr131	ENr213	ENr232	ENr321
# SNP	525	692	904	947	1080	864	990	612	457	544
$r^2 \geq 0.5$										
# precinct	39	27	47	52	40	30	83	42	64	52
(upper bound)										
LD-Select	62	38	65	84	77	69	112	62	72	68
FESTA	57	35	63	76	73	65	107	61	70	65
GreedyTag	56	35	63	76	73	62	107	61	70	64
LRTag	56	35	63	76	73	62	107	61	70	64
(lower bound)										
LRTag_lb	55	35	63	76	73	62	107	60	70	64
GreedyTag_lb	50	33	63	72	69	54	101	55	70	62
Gap	1	0	0	0	0	0	0	1	0	0
$r^2 \geq 0.8$										
# precinct	116	69	121	139	131	129	175	105	106	107
(upper bound)										
LD-Select	123	82	129	152	146	139	189	110	115	109
FESTA	122	79	129	152	143	139	186	110	114	109
GreedyTag	122	79	129	152	143	139	186	110	114	109
LRTag	122	79	129	152	143	139	186	110	114	109
(lower bound)										
GreedyTag_lb	122	79	129	152	143	139	186	110	114	109
LRTag_lb	122	79	129	150	143	139	186	107	110	109
Gap	0	0	0	0	0	0	0	0	0	0

---

[2] Note that if a program runs in time of t with 15 threads, then its running time with one thread would be $15t$. This transformation can be used to compare the running times of our programs and those of the other programs on a single thread mode.

Table 5.3: The time efficiency of FESTA, GreedyTag and LRTag for selecting tagSNPs from a single population, CEU, on all ENCODE regions. The running times are obtained on a 32-processor SGI Altix 4700 supercomputer system.

Region	ENm010	ENm013	ENm014	ENr112	ENr113	ENr123	ENr131	ENr213	ENr232	ENr321
$r^2 \geq 0.5$										
FESTA	3h14m	3h16m	4h51m	3h18m	14h24m	64h4m	5h13m	2h24m	1h38m	45m19s
GreedyTag	1s	1s	1s	1s	1s	1s	1s	1s	1s	1s
LRTag	1s	1s	1s	2s	1s	2s	1s	1s	1s	1s
$r^2 \geq 0.8$										
FESTA	1s	3s	28m20s	44m6s	12m8s	50m16s	3s	1s	2s	1s
GreedyTag	<1s	<1s	<1s	<1s	<1s	<1s	<1s	<1s	<1s	<1s
LRTag	<1s	<1s	<1s	<1s	<1s	<1s	<1s	<1s	<1s	<1s

Tagging ENCODE regions for a single population. We tag each HapMap population separately by LD-Select, FESTA, and our new algorithms GreedyTag and LRTag. For illustration purposes, we only show the results for tagging the CEU population and compare the performance of the above algorithms in Table 5.2.

When the r^2 threshold is set as 0.5, the number of tagSNPs selected by our algorithm is on the average 9.3% of the total number of markers (the actual percentage number ranges from 5.1% to 15.3%). With a more stringent r^2 threshold of 0.8, the average number of tagSNPs rises to 17.6% of the total number of markers (ranging from 11.4% to 24.9%). The same trend was observed when applying our algorithms on the other populations (results are not shown due to space constraint).

On each ENCODE region, we observe that the gap between LRTag_lb and LRTag is at most one with the r^2 threshold being 0.5, and there is no gap when the r^2 threshold is set as 0.8. This demonstrates that our algorithm LRTag found near-optimal solutions in all test cases. In general, LRTag and GreedyTag always generated the smallest sets of tagSNPs, FESTA selected at most three more tagSNP, and LD-Select might select up to eight more tagSNPs.

Since our algorithms and FESTA are all near-optimal, we compare the time efficiency of these programs in Table 5.3. Because LD-Select takes genotype data as input and the other programs take pairwise LD data as input, we do not compare LD-Select's running times directly with those of the others here (generally speaking, it takes LD-Select from 30m to 2h on an ENCODE region). From Table 5.3, we can see that the running time of FESTA varied widely from 1s to 64h on different regions, while our algorithms GreedyTag and LRTag consistently took 1-2s on all regions. In conclusion, our algorithms were 3 to 4 orders of magnitude faster than FESTA in most of the cases, and our algorithms found slightly smaller sets of tagSNPs.

Tagging ENCODE regions for multiple populations. We tag each and the entire ENCODE regions for all four HapMap populations by MultiPop-TagSelect, GreedyTag and LRTag. The tagging results of these methods on each ENCODE region are summarized in Table 5.4. We also highlight the results for region ENm013 and for the entire ENCODE region in Figure 5.2.

With the r^2 threshold set as 0.5, the number of tagSNPs selected by our algorithms is on the average 18.3% of the total number of markers (the actual percentage number ranges from 11.0% to 34.5%). With a more stringent r^2 threshold of 0.8, the average number of tagSNPs

Table 5.4: Summary of tagSNPs identified by MultiPop-TagSelect, GreedyTag and LRTag for all HapMap populations on ENCODE regions.

Region	ENm010	ENm013	ENm014	ENr112	ENr113	ENr123	ENr131	ENr213	ENr232	ENr321
# SNP	783	1063	1261	1158	1485	1221	1186	900	777	1025
$r^2 \geq 0.5$										
# precinct	65	38	48	44	67	38	73	56	126	53
(upper bound)										
MultiPop -TagSelect	206	150	201	238	260	181	306	200	286	233
GreedyTag	179	117	164	184	228	141	257	173	268	193
LRTag	179	117	164	184	228	141	257	173	268	193
(lower bound)										
LRTag_lb	178	117	162	182	226	141	256	173	268	193
GreedyTag_lb	169	107	149	168	218	139	250	173	264	189
Gap	1	0	2	2	0	0	1	0	0	0
$r^2 \geq 0.8$										
# precinct	156	111	146	101	209	106	194	170	210	191
(upper bound)										
MultiPop -TagSelect	338	275	321	425	454	329	462	324	402	396
GreedyTag	322	255	305	391	437	300	445	318	392	377
LRTag	322	255	305	389	437	300	445	318	392	377
(lower bound)										
LRTag_lb	322	255	305	389	437	300	445	318	392	377
GreedyTag_lb	319	253	303	374	435	300	443	318	392	377
Gap	0	0	0	0	0	0	0	0	0	0

increases to 33.7% (ranging from 24.0% to 50.5%). We observe that LRTag always performs the best in these tests, followed by the GreedyTag algorithm, and MultiPop-TagSelect always performs worst. When r^2 threshold is set as 0.5, LRTag requires 16.4% fewer markers on the average than MultiPop-TagSelect. When the r^2 threshold is 0.8, LRTag usually requires 5.1% fewer markers on the average than MultiPop-TagSelect.

The gap between LRTag_lb and LRTag is at most two for each ENCODE region and totally six for all ENCODE regions with the r^2 threshold being 0.5. There is no gap with the r^2 threshold being 0.8.

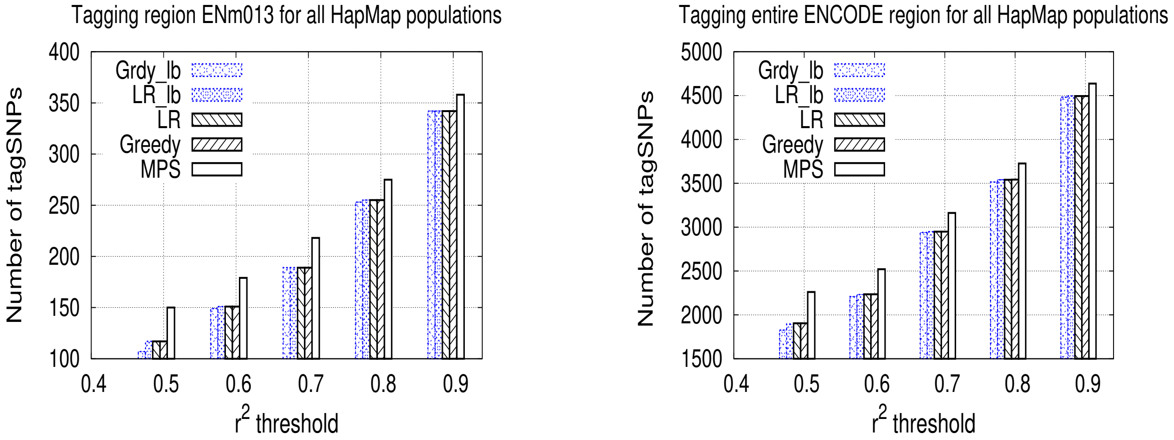

Figure 5.2: Performance comparison of various algorithms to tag entire ENCODE regions. (A). Tagging for HapMap populations on region ENm013 with 783 markers. (B). Tagging for HapMap populations on all ENCODE regions with 10,859 markers. Here, "Grdy_lb" stands for "GreedyTag_lb", "LRTag_lb" stands for "LRTag_lb", "LR" stands for "LRTag", "Greedy" stands for "GreedyTag", and "MPS" stands for "MultiPopTagSelect".

5.4.2 Genome-wide Tagging

Because both LD-Select and MultiPop-TagSelect (written in Perl) took more than 20 hours to tag a single chromosome, we re-implemented their algorithms in C++, called *LD-Select** and *MultiPop-TagSelect**, respectively, in order give a fair comparison of the programs. Since FESTA's "greedy-exhaustive hybrid search" is very computational demanding and hard to emulate, we exclude FESTA from the following comparative study.

Tagging the human genome for a single population. We apply LD-Select*, GreedyTag and LRTag on each HapMap population separately. For illustration purposes, we only discuss the results for tagging the CEU population and compare the performance of the above three algorithms. The details can be found in Table 5.5.

Table 5.5: Summary of the tag SNPs selected by LD-Select, GreedyTag and LRTag for a single population, CEU, on each human chromosome.

Chromosome	1	2	3	4	5	6	7	8	9	10
# SNP	151195	181499	143472	130823	138817	149514	113037	122646	100352	110942
$r^2 \geq 0.5$										
# precinct	15752	29426	12901	11906	11998	11831	10512	9900	9438	10153
(upper bound)										
LD-Select*	21865	36238	19063	17212	17765	17921	15418	15140	13800	14882
GreedyTag	20806	35083	17984	16295	16769	16815	14584	14203	13066	14041
LRTag	20800	35065	17977	16286	16756	16798	14577	14196	13058	14038
(lower bound)										
LRTag_lb	20793	35059	17958	16279	16736	16784	14569	14182	13049	14031
GreedyTag_lb	20123	34202	17155	15675	15965	16086	14021	13568	12530	13477
Gap	7	6	19	7	20	14	8	14	9	7
$r^2 \geq 0.8$										
# precinct	35990	51098	31916	28650	29931	30632	26181	26120	23739	25186
(upper bound)										
LD-Select*	38944	54612	35092	31590	32978	33723	28754	28822	26008	27698
GreedyTag	38534	54081	34602	31124	32502	33229	28362	28394	25666	27302
LRTag	38534	54080	34601	31123	32501	33227	28362	28393	25665	27302
(lower bound)										
LRTag_lb	38534	54080	34600	31123	32501	33225	28361	28391	25664	27301
GreedyTag_lb	38269	53687	34310	30824	32189	32962	28083	28110	25396	27077
Gap	0	0	1	0	0	2	1	2	1	1
Chromosome	12	13	14	15	16	17	18	19	20	21
# SNP	100437	84184	68485	58491	57083	47505	62666	29341	51206	27955
$r^2 \geq 0.5$										
# precinct	9960	7476	6751	6740	7184	6764	6534	5291	5874	3270
(upper bound)										
LD-Select*	14243	10996	9703	9364	9962	8656	9115	6464	7972	4470
GreedyTag	13554	10374	9215	8930	9503	8355	8652	6286	7637	4258
LRTag	13548	10370	9212	8923	9500	8354	8649	6284	7634	4257
(lower bound)										
LRTag_lb	13539	10363	9203	8920	9500	8353	8646	6283	7630	4256
GreedyTag_lb	13048	9988	8867	8589	9241	8180	8332	6190	7380	4125
gap	9	7	9	3	0	1	3	1	4	1
$r^2 \geq 0.8$										
(upper bound)										
# precinct	23809	18509	16391	15629	16869	13942	15262	10019	13177	7390
LD-Select*	25887	20221	17723	16908	18194	14778	16498	10494	14194	7912
GreedyTag	25579	19967	17546	16722	18012	14670	16299	10420	14052	7844
LRTag	25579	19967	17545	16722	18012	14669	16299	10420	14052	7844
(lower bound)										
LRTag_lb	25578	19966	17545	16722	18012	14668	16299	10420	14051	7844
GreedyTag_lb	25387	19778	17405	16608	17836	14588	16181	10382	13943	7774
Gap	1	1	0	0	0	1	0	0	1	0

With the r^2 threshold set as 0.5, the number of tagSNPs selected by our algorithms is 14.4% of the total number of markers on the average (the actual percentage ranges from 11.2% to 21.4%). With a more stringent r^2 threshold of 0.8, the average number of tagSNPs increases to 26.6% (ranging from 22.2% to 35.5%). Similar trends were observed when applying our algorithms to the other populations (the results are not shown due to space limitation).

We observe that LRTag always performs the best, followed by the GreedyTag algorithm, and LD-Select* always performs the worst. With the r^2 threshold set as 0.5, LRTag usually requires 4.9% fewer tagSNPs (the actual percentage number ranges from 2.8% to 6.3%) on average than LD-Select* on each chromosome. When the r^2 threshold is increased to 0.8, LRTag usually requires 1.2% fewer tagSNPs (ranging from 0.07% to 1.5%) on average than LD-Select*.

We can see that, on each chromosome, the gap between the lower bound from LRTag_lb and the upper bound obtained by LRTag is on the average 7 (the actual number ranges from 1 to 20) with the r^2 threshold set as 0.5 and less than 1 (ranging from 0 to 2) with the r^2 threshold set being 0.8. This demonstrates that LRTag finds near-optimal solutions in all test cases even for genome-wide tagging on a single population. In fact, the performance of GreedyTag is not bad either.

Tagging the human genome for multiple populations. Finally, we tag the entire human genome for all four HapMap populations by MultiPop-TagSelect, GreedyTag and LRTag. We summarize the tagging results of these methods on each chromosome in Table 5.6 and then highlight the results for chromosome 3 and all chromosomes in Figure 5.3.

With the r^2 threshold set as 0.5, the number of tagSNPs selected by our methods is on the

Table 5.6: Summary of the tagSNPs selected by MultiPop-TagSelect, GreedyTag and LRTag for all HapMap populations on each human chromosome.

Chromosome	1	2	3	4	5	6	7	8	9	10
# SNP	216357	249136	196535	182273	187924	205496	155224	170136	138047	156089
$r^2 \geq 0.5$										
# precinct	16234	26836	12835	12251	12414	11862	10332	10101	9254	10220
(upper bound)										
MultiPop -TagSelect*	64892	126408	56978	52828	54087	54454	45943	46927	41341	45226
GreedyTag	59126	122372	51266	47650	48661	48817	41169	42206	37289	40713
LRTag	55016	117537	47450	44223	45186	44987	38150	39149	34439	37554
(lower bound)										
LRTag_lb	54942	117511	47362	44141	45102	44878	38090	39076	34381	37486
GreedyTag_lb	53937	117155	46330	43239	44145	43845	37280	38161	33534	36713
Gap	74	26	88	82	84	109	60	73	58	68
$r^2 \geq 0.8$										
# precinct	42450	56135	35192	33434	33211	33228	28543	28948	25485	28277
(upper bound)										
MultiPop -TagSelect*	100062	155505	89195	82835	84998	86313	72024	74934	65442	70817
GreedyTag	94797	150664	84091	78077	80188	80981	67818	70678	61708	66676
LRTag	94797	150664	84090	78076	80186	80980	67817	70677	61706	66674
(lower bound)										
LRTag_lb	94788	150660	84079	78072	80174	80964	67808	70667	61699	66663
GreedyTag_lb	94362	150393	83585	77674	79663	80507	67461	70285	61321	66291
Gap	9	4	11	4	12	16	9	10	7	11
Chromosome	12	13	14	15	16	17	18	19	20	21
# SNP	141943	119080	94528	81687	79898	64645	89024	40549	70877	39400
$r^2 \geq 0.5$										
# precinct	10086	7810	6532	6667	7328	6952	6875	5127	6139	3290
(upper bound)										
MultiPop -TagSelect*	42362	33477	28465	27847	28987	23601	27109	15768	23243	13100
GreedyTag	38563	30183	25706	25408	26432	21931	24789	14785	21319	12010
LRTag	35493	27927	23932	23721	24791	20647	23174	14007	19994	11253
(lower bound)										
LRTag_lb	35449	27881	23903	23686	24761	20636	23141	14001	19971	11238
GreedyTag_lb	34833	27306	23440	23229	24294	20366	22697	13814	19601	11035
Gap	44	46	29	35	30	11	33	6	23	15
$r^2 \geq 0.8$										
# precinct	27027	21084	17723	17526	18943	16278	17866	11289	15438	8366
(upper bound)										
MultiPop -TagSelect*	65521	52863	44226	42380	43913	34289	41893	22274	35251	19990
GreedyTag_lb	61828	49797	41867	40250	41726	32862	39833	21465	33676	19060
LRTag_lb	61826	49796	41867	40250	41724	32862	39833	21464	33675	19060
(lower bound)										
LRTag_lb	61816	49791	41860	40247	41721	32860	39832	21464	33673	19059
GreedyTag_lb	61450	49498	41642	40029	41497	32740	39625	21377	33525	18996
Gap	10	5	7	3	3	2	1	0	2	1

average 27.3% of the total number of markers (the actual percentage ranges from 21.9% to 47.2%). With a more stringent r^2 threshold of 0.8, the average number of tagSNPs increases to 46.0% (ranging from 29.4% to 60.4%). Based on Table 5.6, we observe that LRTag always

performs slightly better than GreedyTag and significantly better than MultiPop-TagSelect*. With the r^2 threshold set as 0.5, LRTag requires 6.8% fewer tagSNPs on average (the actual number ranges from 4.0% to 8.0%) than MultiPop-TagSelect* on each chromosome. With the r^2 threshold set as 0.8, LRTag requires 3.6% fewer markers on average (ranging from 2.7% to 4.3%) than MultiPop-TagSelect*.

The gap between the lower bound from LRTag_lb and upper bound of LRTag is on the average 48 for each chromosome (the actual number ranges from 6 to 109) with the r^2 threshold set as 0.5, and 6.5 (ranging from 0 to 16) with the r^2 threshold set being 0.8, as shown in Table 5.6.

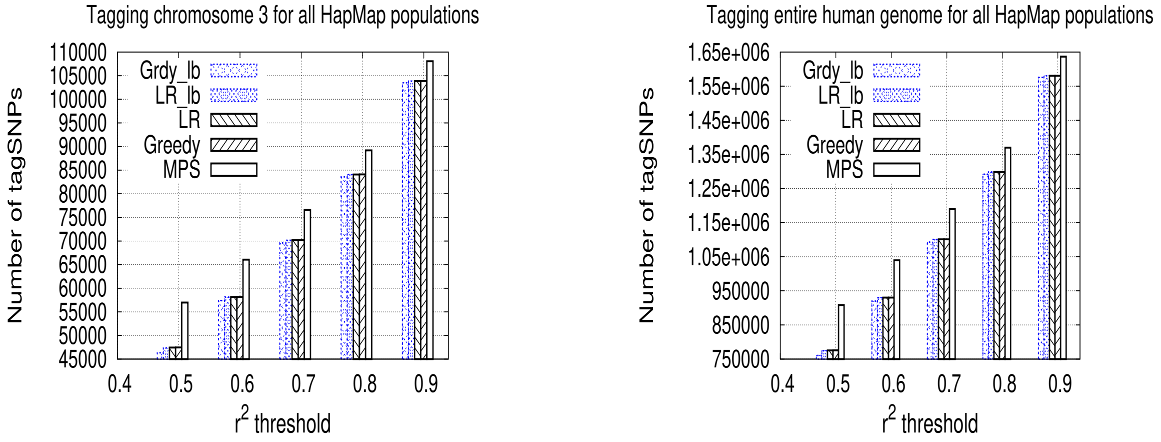

Figure 5.3: Performance comparison of various algorithms to tag entire human genome. (A). Tagging chromosome 3 for all HapMap populations with 196,535 markers. (B). Tagging the entire human genome for all HapMap populations with 2,862,454 markers. See the caption of Figure 5.2 for the definitions of the legends in the subfigures.

5.5 Concluding Remarks

Our LRTag and GreedyTag algorithms run quickly on ENCODE regions and the entire human genome for both single and multiple populations. On an ENCODE region with the r^2 threshold being 0.5, it takes our algorithms no more than 2 seconds to tag a single population (as shown in Table 5.3) and less than 7 seconds to tag multiple populations (as displayed in Table 5.7. On a human chromosome, it takes no more than 4 minutes to tag a single population (as shown in Table 5.8) and less than 12 minutes to tag on multiple populations (as displayed in Table 5.9. For r^2 thresholds greater than 0.5, our algorithms run faster. Hence, for any given r^2 threshold, it takes our algorithms less than a minute to tag the ENCODE region and less than an hour to tag the human genome.

All the lower and upper bounds produced by the discussed methods are shown in Figure 5.2 and Figure 5.3. In the figures, we tag the ENCODE regions and human genome on the HapMap populations with the r^2 thresholds being 0.5, 0.6, 0.7 and 0.8 separately. From all these test cases, we observe that LRTag always chooses the smallest set of tagSNPs, closely followed by GreedyTag, while MultiPop-TagSelect chooses the largest set of tagSNPs.

LRTag_lb always provides the best lower bound and LRTag the best upper bound among all methods considered. The simple greedy algorithm, GreedyTag, chooses slightly more tagSNPs than LRTag and the lower bound GreedyTag_lb is slightly lower than LRTag_lb, which indicates that the data reduction rules in Section 5.3 are very powerful.

Table 5.7: The speeds of GreedyTag and LRTag for tagging the entire ENCODE region for all HapMap populations with the r^2 threshold being 0.5. The running time is evaluated on a 32-processor SGI Altix 4700 supercomputer system.

Region	ENm010	ENm013	ENm014	ENr112	ENr113	ENr123	ENr131	ENr213	ENr232	ENr321
LRTag	1s	4s	3s	5s	7s	5s	1s	1s	1s	2s
GreedyTag	1s	4s	4s	5s	7s	6s	1s	1s	1s	2s

Table 5.8: The speeds of GreedyTag and LRTag for tagging the human genome for a single population, CEU, with the r^2 threshold being 0.5. The running time is evaluated on a 32-processor SGI Altix 4700 supercomputer system.

Chromosome	1	2	3	4	5	6	7	8	9	10	11
LRTag	1m18s	1m44s	1m28s	1m12s	1m27s	3m7s	1m3s	1m15s	57s	1m6s	1m8s
GreedyTag	1m17s	1m41s	1m16s	1m15s	1m24s	3m11s	58s	1m16s	57s	1m6s	1m10s
Chromosome	12	13	14	15	16	17	18	19	20	21	22
LRTag	56s	50s	34s	28s	23s	46s	31s	9s	23s	11s	10s
GreedyTag	56s	50s	37s	27s	20s	47s	31s	10s	22s	11s	9s

Table 5.9: The speeds of GreedyTag and LRTag for tagging the entire human genome for all HapMap populations with the r^2 threshold being 0.5. The running time is evaluated on a 32-processor SGI Altix 4700 supercomputer system.

Chromosome	1	2	3	4	5	6	7	8	9	10	11
LRTag	3m4s	2m2s	3m9s	2m51s	3m37s	11m4s	2m12s	3m45s	2m24s	2m49s	2m20s
GreedyTag	3m11s	1m13s	3m43s	2m46s	3m20s	10m45s	2m25s	2m52s	2m18s	2m55s	2m16s
Chromosome	12	13	14	15	16	17	18	19	20	21	22
LRTag	2m55s	2m11s	1m28s	1m	48s	1m10s	1m17s	27s	56s	25s	30s
GreedyTag	3m	2m27s	1m16s	52s	23s	1m9s	1m16s	27s	50s	25s	30s

When the r^2 threshold increases, the size of the precincts decreases. Consequently, the gap between the lower bound and the upper bound decreases. For the entire human genome with 2,862,454 markers, the gap between LRTag and LRTag_lb is 1061 when the r^2 threshold is 0.5, and 142 when the r^2 threshold increases to 0.8. The small gap shows that LRTag finds near-optimal solutions for genome-wide tagging.

Chapter 6

On the Minimum Common Integer Partition Problem

In this chapter we consider an algorithmic problem that is motivated by problems from comparative genomics and microarray-based DNA fingerprinting instead of genetics.

6.1 Introduction

First, we give a formal definition of the Minimum Common Integer Partition (MCIP) Problem. By a *partition* of a positive integer n we mean a multiset [1] $\{n_1, n_2, \cdots, n_r\}$ of positive integers that add up to exactly n, *i.e.* $\sum_{i=1}^{r} n_i = n$, where n_i is called a *part* of n [83, 85].

---

[0] An extended abstract of this paper, entitled "On the Minimum Common Integer Partition Problem", has accepted by *Proc. 6th Conference on Algorithms and Complexity (CIAC)*, May. 2006, Rome, Italy [48].

[1] Recall that a multiset is a set-like object in which order is ignored, but multiplicity is explicitly significant.

Given a multiset $S = \{x_1, x_2, \cdots, x_m\}$ of integers with a partition for each integer x_i, $1 \leq i \leq m$, we can define an *integer partition* of S as the multiset union of these partitions, that is $\biguplus_{i=1}^{m} P(x_i)$. By definition, S is an integer partition of itself. A multiset is said to be a *common integer partition* of a sequence of multisets $S_1, S_2, \ldots, S_k (k \geq 2)$ if it is an integer partition of every multiset S_i, $1 \leq i \leq k$. The minimum common integer partition problem is thus defined as follows: given a sequence of multisets $S_1, S_2, \cdots, S_k$ of integers, find a common integer partition of them with the minimum cardinality. We denote the minimum common integer partition by MCIP($S_1, S_2, \cdots, S_k$) (or simply MCIP when the input multisets are clear from the context). Note that, now MCIP denotes both the MCIP problem and also its solution on a particular instance, but this overloading is a common pratice and should not cause any confusion given the context. For simplicity, we also denote by MCIP(S_1, S_2, $\cdots$, S_k) (or simply k-MCIP) the restricted version of the MCIP problem when the number of input multisets is fixed to be k throughout the paper.

For example, the integer 3 has only three partitions, *i.e.*, $\{3\}$,$\{2, 1\}$, and $\{1, 1, 1\}$, while the integer 10 has 190569292 partitions [83]. We can see that the number of partitions increases quite rapidly with the integer n. For multiset $S = \{3, 3, 4\}$, $\{2, 2, 3, 3\}$ is an integer partition of S and $\{1, 1, 2, 2, 4\}$ is another one. For a pair of multisets $S = \{3, 3, 4\}$ and $T = \{2, 2, 6\}$, both $\{2, 2, 3, 3\}$ and $\{1, 1, 2, 2, 4\}$ are common integer partitions of S and T, while the first one gives the minimum cardinality, *i.e.*, MCIP$(S, T) = \{2, 2, 3, 3\}$. Note that the minimum common integer partition is not necessarily unique. So, the notation MCIP$(S_1, S_2, \cdots, S_k)$ is not really a function, strictly speaking. But we will use it as a function throughout the paper for simplicity.

The necessary and sufficient condition for a sequence of multisets $S_1, S_2, \ldots, S_k$ to have a common integer partition is that they have the same summation over their integer elements. Multisets with this property are called *related*. Verifying whether a sequence of multisets of integers are related can be done easily in linear time, and thus for the rest of the paper we will assume, without loss of generality, that the input multisets are all related.

Clearly, the MCIP problem is NP-hard since it generalizes the well-known Set Partition problem [88]. In this paper, we show that the MCIP problem is APX-hard and hence has no polynomial-time approximation algorithm (PTAS) unless P = NP. We also present a $\frac{5}{4}$-approximation algorithm for the 2-MCIP using a heuristic for the *Maximum Set Packing* problem, and a $\frac{3k(k-1)}{3k-2}$-approximation algorithm for the general k-MCIP problem, where $k \geq 3$.

6.1.1 Biological Background

Although the MCIP problem is quite a natural extension of the Set Partition problem, its formulation was mainly motivated by our recent work on ortholog assignment and DNA fingerprint assembly in computational molecular biology. The following gives a brief account of the background.

Since it contains discussions that involve the knowledge of some biological experiments, the reader who is not interested in the biological relevance may feel free to skip some (or all) of the paragraphs in this subsection.

Ortholog assignment. Orthologous genes are typically the evolutionary and functional counterparts in different species, and therefore the prediction (or assignment) of orthologs is a common task in computational biology. While it is usually done using sequence homology search [100], we have recently proposed an alternative and promising approach to assign orthologs via genome rearrangement [90, 91]. This new approach has inspired us to formulate several interesting combinatorial optimization problems, *e.g.*, Signed Reversal Distance with Duplicates (SRDD), Minimum Common String Partition (MCSP), and Maximum Cycle Decomposition (MCD), which have attracted increasing attention from the algorithms community [87, 94, 92, 97]. In particular, the MCSP problem, which is the most related to MCIP, is defined as follows: Given two input strings, partition them into the same collection of substrings so that the number of resultant substrings is minimized. For example, the MCSP for $\{aaabbbccc, bbbaaaccc\}$ is $\{aaa, bbb, ccc\}$. The restricted version of MCSP where the number of symbols that occur in an input string multiple times (called duplicated symbols; the other symbols are called singletons) is no more than l in each input string, is denoted by MCSP-l. It is known that the MCSP-l problem is NP-hard [89], when $l \geq 1$. In other words, even when there is only one symbol with multiple copies in each input string, we still cannot find the MCSP in polynomial time unless P=NP.

It is easy to transform an instance of MCSP-1 into an instance of 2-MCIP where each integer represents the size of a block consisting of only the duplicated symbol so that an optimal solution to the 2-MCIP problem would in most cases give an optimal solution to the MCSP-1 problem with the same cardinality [89]. Therefore, we hope that the study of MCIP will help the design of good approximation algorithms for MCSP-1 and MCSP in general.

DNA fingerprint assembly. In the ongoing *Oligonucleotide Fingerprinting Ribosomal Genes* (OFRG) project [102], we collaborate with microbiologists and statisticians to provide a high-throughput method for identifying different microbial organisms. Briefly, the microbiologists build an rDNA clone library after DNA extraction and *Polymerase Chain Reaction* (PCR) amplification. The rDNA clones are assigned fingerprints (binary strings where 0 indicate non-binding between a clone and a probe, and 1 otherwise) through a series of hybridization experiments, each using a single 10-nucleotide DNA probe. These 10-nucleotide DNA probes comprise a probe set and the size of the probe set determines the length of a fingerprint. Then, clones are identified by clustering their fingerprints with those of known sequences. By mapping sequence data to hybridization patterns, clones can be identified (or at least differentiated). Compared with direct sequencing, the method saves significant cost without sacrificing too much discriminating ability.

Although OFRG is a cost-effective approach, we are trying to scale it up in order to process a large number of samples from applications such as identifying microorganisms involved in the development of the mucosal and systemic immune system. One possible way of enhancing OFRG is inspired by new (but proven) technologies such as microbead clone libraries and multiplex flow cytometry. By producing clone libraries on microbeads, we are able to simultaneously hybridize a set of probes to thousands of clones in seconds, which is a significant improvement over the current array platform. However, we will still need multiple hybridizations, each using a different probe (sub)set, as the size of the desired probe set in OFRG exceeds the maximum discriminating size of the cytometry technology. Thus we obtain a *partial fingerprint* from each run of hybridization because only a subset of the probes are used in each hybridization.

The *DNA fingerprint assembly problem* aims at inferring a *complete fingerprint* (with respect to the overall probe set) for each clone from partial fingerprints by minimizing the total number of distinct complete fingerprints. We assume that all the probe subsets share a small number of common probes which are called the *linking probes*. That is, these linking probes will be used for each run of hybridization. A complete fingerprint can thus be obtained from partial fingerprints that share the same bits on the linking probes. More specifically, after each run of the hybridization, we assign a *weight* to each distinct partial fingerprint as the number of clones that produced this partial fingerprint in the hybridization. Then we divide all partial fingerprints into groups based on their bits on the positions of linking probes. The partial fingerprints in a group are compatible with each other and may correspond to the same complete fingerprint. For each group, the fingerprint assembly problem can be viewed as MCIP$(S_1, S_2, \cdots, S_k)$, with k being the number of the probe subsets (*i.e.* the number of hybridizations) and S_i containing the weight of each partial fingerprint in this particular group from the ith hybridization. Hence, complete fingerprints for each group can be obtained by combining their respective partial fingerprints via the minimum common integer partition of the weights. Such a solution would represent the minimum number of *distinct* complete fingerprints (or clones) that have produced the group of partial fingerprints.

6.2 Some Basic Facts

Throughout the paper, we assume that the multisets given as input to MCIP are related as mentioned before. We denote the size of the minimum common integer partition by $|MCIP(S_1, S_2, \cdots, S_k)|$ (or simply $|k\text{-}MCIP|$ if the input multisets are clear from the context). Because every integer in any input multiset will be partitioned into one or more integers in the minimum common integer partition, the following lemma gives a trivial, but useful lower bound.

Lemma 6.2.1 $|MCIP(S_1, S_2, \cdots, S_k)| \geq max(|S_1|, |S_2|, \cdots, |S_k|)$*, where* $|\cdot|$ *is the size of a multiset.*

Proof. The proof is obvious and thus omitted. ■

In the case of 2-MCIP, we use $\langle S, T\rangle$ to denote the two input multisets, where $S = \{x_1, x_2, \cdots, x_m\}$ and $T = \{y_1, y_2, \cdots, y_n\}$ such that $\sum_{i=1}^{m} x_i = \sum_{i=1}^{n} y_i$. A greedy algorithm that constructs a common integer partition of $\langle S, T\rangle$ is to iteratively add the smaller one of two integers randomly selected from the two input multisets. More precisely, the algorithm can be described in pseudo-code as in Figure 6.1, and runs in time linear in n. The following lemma gives an upper bound for 2-MCIP, which is very useful in the subsequent discussion.

Lemma 6.2.2 $|MCIP(S,T)| \leq |S| + |T| - 1$.

Proof. After each iteration of the algorithm 2-APPROX-MCIP(S,T), the total size of S and T shall decrease by one or two while the multiset CIP expands by one integer. In the last iteration, the two integers remaining in S and T must be equal, and thus the total size of S and T shall decrease by exactly two. Therefore, the common integer partition returned from the algorithm contains no more than $|S| + |T| - 1$ integers. ■

As its name suggests, the algorithm 2-APPROX-MCIP(S,T) is a 2-approximation algorithm for the problem of 2-MCIP.

Lemma 6.2.3 *The algorithm* 2-APPROX-MCIP*(S,T) achieves an approximation ratio of 2.*

Proof. The result is trivially implied by Lemma 6.2.1 and Lemma 6.2.2. ■

Given a common integer partition $CIP(S,T)$ of $\langle S,T\rangle$, we say that x_i is *mapped* to y_j if there exists an element in $CIP(S,T)$ such that it is a part of x_i as well as a part of y_j. Notice that an integer in S (or T) can be mapped to two or more integers in T (or S). Two integers a_1 and a_h in $\langle S, T \rangle$, *i.e.*, $a_1 \in S \biguplus T$ and $a_h \in S \biguplus T$, are said to be *connected* if there exist a sequence of integers $a_2,\cdots,a_{h-1}$ in $\langle S, T \rangle$ such that a_i is mapped to a_{i+1}, for each $i \in [1, h-1]$. Thus, all the integers that are connected to each other in S and T will constitute a *connected component* (or simply *component*) of $\langle S, T \rangle$. We say that these connected components are *induced* by the given common integer partition $CIP(S, T)$.

Lemma 6.2.4 *Suppose that $CIP(S, T)$ denotes a common integer partition of S and T. Then*

1. *every connected component $\langle S_1, T_1 \rangle$ induced by $CIP(S, T)$ is a pair of related multisets;*
2. *for every connected component $\langle S_1, T_1 \rangle$, all the integers in $CIP(S, T)$ that are parts of integers in S_1 or T_1 constitute a common integer partition $CIP(S_1, T_1)$ of S_1 and T_1 such that $|CIP(S_1, T_1)| \geq |S_1| + |T_1| - 1$.*

Proof.

(1) Based on the common integer partition $CIP(S, T)$, each part of an integer x_i in S_1 corresponds to a distinct part of exactly one integer y_j in T in a one-to-one fashion. In this case, x_i and y_j are mapped to each other, and by the definition of connected components, y_j will be included in T_1, implying that $\sum_{x \in S_1} x \leq \sum_{y \in T_1} y$. Similarly, we have $\sum_{x \in S_1} x \geq \sum_{y \in T_1} y$. Therefore, S_1 and T_1 are two related multisets.

(2) Since the multiset under consideration (*i.e.* $CIP(S_1, T_1)$) consists of all integers from $CIP(S, T)$ that are parts of integers in S_1 or T_1 and nothing else, it is clearly a common integer partition of S_1 and T_1.

To see that $|CIP(S_1, T_1)| \geq |S_1| + |T_1| - 1$, we construct an undirected graph based on the integers in S_1, T_1 and $CIP(S_1, T_1)$: for each integer in S_1 (or T_1), a vertex is created and, for each integer in $CIP(S_1, T_1)$ which is a part of x_i as well as a part of y_j, for some x_i and y_j, an edge is created between the vertices for x_i and y_j. We denote by $|V|$ the number of vertices in the graph and by $|E|$ the number of edges. Observe that there is a one-to-one correspondence between the vertices and the integers in $S_1 \biguplus T_1$, and hence $|V| = |S_1| + |T_1|$. Further observe that there is a one-to-one correspondence between the edges and the integers in $CIP(S_1, T_1)$, and thus $|E| = |CIP(S_1, T_1)|$. We can see that the constructed graph is connected and may have multiple edges between a pair of vertices, which means that $|E| \geq |V| - 1$. Thus $|CIP(S_1, T_1)| \geq |S_1| + |T_1| - 1$ holds. ■

6.2.1 The Maximum Related Multiset Partition

In this subsection, we define a new combinatorial optimization problem, *maximum related multiset partition (MRMP)*, to assist solving the MCIP problem.

S_1 and T_1 are said to be a pair of *related submultisets* of two related multisets S and T if S_1 is a (nonempty) submultiset of S, T_1 is a (nonempty) submultiset of T, and they are related. We write $\langle S_1, T_1 \rangle \subseteq \langle S, T \rangle$ to denote the related submultisets. Obviously,$\langle S, T \rangle \subseteq \langle S, T \rangle$.

Furthermore, S and T are said to be *basic* if they have one and only one pair of related submultisets, namely $\langle S, T\rangle$. For example, consider $S = \{3, 3, 4\}$ and $T = \{2, 2, 6\}$. They have three pairs of related submultisets: $\langle\{3, 3\}, \{6\}\rangle$, $\langle\{4\}, \{2, 2\}\rangle$, and $\langle S, T\rangle$. Therefore, S and T are not a pair of basic related multisets. An example of two basic related multisets is $\langle\{1, 4\}, \{2, 3\}\rangle$.

A *multiset partition* (or simply *partition*) of a multiset S is a sequence of disjoint submultisets $S_1, S_2, \cdots, S_l$ of S whose union is S, *i.e.* $S = \biguplus_{i=1}^{l} S_i$. By definition, S is a multiset partition of itself. It is important to remember that multiset partition and the integer partition are two different concepts in this paper. Given two multisets S and T of integers, a sequence of multiset pairs $\langle S_1, T_1\rangle, \langle S_2, T_2\rangle, \cdots, \langle S_l, T_l\rangle$ is called a *related multiset partition* if $\{S_1, S_2, \cdots, S_l\}$ is a multiset partition of S, $\{T_1, T_2, \cdots, T_l\}$ is a multiset partition of T, and, moreover, for each $i \in [1, l]$, S_i and T_i are a pair of related multisets. The maximum related multiset partition problem is then defined as to find a related multiset partition of two given multisets S and T, maximizing the number of related multiset pairs in the partition. We denote by $MRMP(S, T)$ (or 2-$MRMP$) the maximum related multiset partition of S and T, and by $|MRMP(S, T)|$ (or $|2\text{-}MRMP|$) the size of the partition, *i.e.*, the number of related multiset pairs in the partition.

Lemma 6.2.5 *Given a common integer partition $CIP(S, T)$, we can transform it into a related multiset partition of S and T, denoted as $RMP(S, T)$, such that $|RMP(S, T)| \geq |S| + |T| - |CIP(S, T)|$.*

Proof. Based on the given common integer partition $CIP(S, T)$, $\langle S, T\rangle$ can be decomposed into l connected components $\langle S_1, T_1\rangle$, $\langle S_2, T_2\rangle$, $\cdots$, $\langle S_l, T_l\rangle$. By Lemma 6.2.4, each connected component is a pair of related multisets and disjoint with any other component. Therefore, all the l connected components naturally give a related multiset partition (denoted as $RMP(S, T)$) of $\langle S, T\rangle$, such that $|RMP(S, T)| = l$. Let $CIP(S_i, T_i)$ denote the common integer partition of $\langle S_i, T_i\rangle$ induced from $CIP(S, T)$. We can see that the union of all $CIP(S_i, T_i)$ will be the common integer partition $CIP(S, T)$, *i.e.*, $CIP(S, T) = \biguplus_{i=1}^{l} CIP(S_i, T_i)$. Since each $\langle S_i, T_i\rangle$ is a connected component, by Theorem 6.2.4, $|CIP(S_i, T_i)| \geq |S_i| + |T_i| - 1$ holds for each $i \in [1, l]$. Therefore, $|CIP(S, T)| = \sum_{i=1}^{l} |CIP(S_i, T_i)| \geq \sum_{i=1}^{l}(|S_i| + |T_i| - 1)| = |S| + |T| - |RMP(S, T)|$. ■

The following lemma establishes the relationship between MCIP and MRMP, showing their (complementary) equivalence.

Lemma 6.2.6 *If S and T are two related multisets, then* $|MCIP(S, T)| + |MRMP(S, T)| = |S| + |T|$.

Proof. Assume that $\langle S_1, T_1\rangle$, $\langle S_2, T_2\rangle$, $\cdots$, $\langle S_l, T_l\rangle$ is a maximum related multiset partition of S and T with $l = |MRMP(S, T)|$. For each $i \in [1, l]$, $\langle S_i, T_i\rangle$ is a pair of basic related multisets, and by Lemma 6.2.2, the minimum common integer partition MCIP(S_i, T_i) is of size less than or equal to $|S_i| + |T_i| - 1$, *i.e.*, $|MCIP(S_i, T_i)| \leq |S_i| + |T_i| - 1$. We can also see that the union of all MCIP(S_i, T_i) forms a common integer partition $CIP(S, T)$ of S and T, *i.e.*, $CIP(S,T) = \biguplus_{i=1}^{l} MCIP(S_i,T_i)$, and its size is $|CIP(S,T)| = \sum_{i=1}^{l} |MCIP(S_i,T_i)| \leq \sum_{i=1}^{l}(|S_i|+|T_i|-1)| = |S|+|T|-|MRMP(S,T)|$. Therefore, we have $|MCIP(S,T)| \leq |CIP(S,T)| \leq |S|+|T|-|MRMP(S,T)|$.

By Lemma 6.2.5, given a minimum common integer partition MCIP(S,T), we can transform it into a related multiset partition $RMP(S,T)$ such that $|RMP(S,T)| \geq |S|+|T|-|MCIP(S,T)|$. Because $|MRMP(S,T)| \geq |RMP(S,T)|$, we have $|MCIP(S,T)| \geq |S|+|T|-|MRMP(S,T)|$. ■

Since a pair of basic related multisets S and T cannot be partitioned further into related submultisets, *i.e.*, $|MRMP(S,T)| = 1$, the following lemma is trivially implied by Lemma 6.2.6.

Lemma 6.2.7 *If S and T are a pair of basic related multisets, then* $|MCIP(S,T)| = |S|+|T|-1$.

The following lemmas will be crucial to the approximation algorithms. We define the size of a pair of related multisets S and T as the sum of the size of S and the size of T, *i.e.*, $|\langle S,T\rangle| = |S|+|T|$.

Lemma 6.2.8 *If the minimum size of any related submultiset of S and T is c, then* $|MCIP(S,T)| \geq \frac{c-1}{c}(|S|+|T|)$.

Proof. Assume that $\{\langle S_1,T_1\rangle, \langle S_2,T_2\rangle, \cdots, \langle S_l,T_l\rangle\}$ are the basic related multisets induced by the minimum common integer partition MCIP(S,T), such that $|MCIP(S,T)| =$

$\sum_{i=1}^{l}(|S_i| + |T_i| - 1)$. Since $|\langle S_i, T_i\rangle| = |S_i| + |T_i| \geq c$ for each $i \in [1, n]$, we have $|MCIP(S,T)| = \sum_{i=1}^{l}(|S_i| + |T_i| - 1) = \sum_{i=1}^{l} \frac{|S_i|+|T_i|-1}{|S_i|+|T_i|}(|S_i| + |T_i|) \geq \sum_{i=1}^{l}(1 - \frac{1}{c})(|S_i| + |T_i|) = \frac{c-1}{c}(|S| + |T|)$. ■

Lemma 6.2.9 *Given two related multisets, $S = \{x_1, x_2, \cdots, x_m\}$ and $T = \{y_1, y_2, \cdots, y_n\}$. If x_i and y_j are a pair of identical integers, then $\{x_i\} \biguplus MCIP(S \setminus \{x_i\}, T \setminus \{y_j\})$ is a minimum common integer partition of S and T,* i.e., $|MCIP(S,T)| = |MCIP(S\setminus\{x_i\}, T\setminus\{y_j\})| + 1$.

Proof. Assume that MCIP(S,T) is a minimum common integer partition of S and T. Let $MRMP(S,T)$ denote the maximum related multiset partition induced by MCIP(S,T). If x_i and y_j are in the same related submultiset $\langle S_1, T_1\rangle$ of $MRMP(S,T)$ such that $\langle S_1, T_1\rangle \neq \langle\{x_i\}, \{y_j\}\rangle$, then we can further decompose $\langle S_1, T_1\rangle$ into two related submultisets $\langle\{x_i\}, \{y_j\}\rangle$ and $\langle S_1 \setminus \{x_i\}, T_1 \setminus \{y_j\}\rangle$, which contradicts the definition of the maximum related multiset partition. If x_i and y_j are in two different related submultiset $\langle S_1, T_1\rangle$ and $\langle S_2, T_2\rangle$ of $MRMP(S,T)$, respectively, then we can obtain a new maximum related multiset partition by replacing $\langle S_1, T_1\rangle$ and $\langle S_2, T_2\rangle$ with $\langle\{x_i\}, \{y_j\}\rangle$ and $\langle S_1 \biguplus S_2 \setminus \{x_i\}, T_1 \biguplus T_2 \setminus \{y_j\}\rangle$. Moreover, by Lemma 6.2.6, the new maximum related multiset partition gives another minimum common integer partition in which x_i is mapped to y_j, implying that $|MCIP(S,T)| = |MCIP(S \setminus \{x_i\}, T \setminus \{y_j\})| + 1$. ■

Unfortunately, the result in Lemma 6.2.9 cannot be extended to the case of k multisets when $k \geq 3$. An interesting counterexample is $\{6, 5, 1, 4, 2\}$, $\{6, 5, 1, 3, 3\}$, $\{6, 4, 2, 3, 3\}$. Their minimum common integer partition is of size 6, but any common integer partition including 6 as an element is of size at least 7. In the following, we will use a procedure **remove_common_integer**$(S_1, S_2, \cdots, S_k)$ to remove all common integer elements existing in every multiset of $\{S_1, S_2, \cdots, S_k\}$ (and add them into the solution). The optimality of this operation is guaranteed only when $k = 2$, as shown in Lemma 6.2.9.

6.3 Hardness of Approximation

It is easy to see that MCIP is NP-hard because there is a straightforward reduction from the *Set Partition* problem. This section is devoted to proving that MCIP is APX-hard.

In the sequel, we prove the APX-completeness of 2-MCIP by an L-reduction from the *Maximum Bounded 3-Dimensional Matching* problem (denoted as MAX 3DM-3). The MAX 3DM-3 problem is defined as follows: Given a set $D \subseteq X \times Y \times Z$, where X, Y and Z are disjoint sets and moreover, each element in X, Y and Z occurs in at least one and at most three triples in D [98], the goal is to find a matching $M \subseteq D$ for D of the maximum cardinality, *i.e.*, a largest set $M \subseteq D$ such that no two elements in M agree in any coordinate. In this problem, without loss of generality, we can assume that $n = |X| \leq |Y| \leq |Z|$. Since each element in X occurs at least once and at most three times in D, the number of triples is at least n and at most $3n$, *i.e.*, $n \leq |D| \leq 3n$. It also implies that $|Y| \leq 3n$ and $|Z| \leq 3n$.

Further observe that each triple can intersect at most six other triples, which implies that the maximum matching contains at least $|D|/7$ triples. Let $|MAX\ 3DM\text{-}3|$ denote the size of maximum matching of $|D|$. It is easy to see that $\lceil \frac{n}{7} \rceil \leq |MAX\ 3DM\text{-}3| \leq n$.

Let $X = \{x_1, x_2, \cdots, x_{|X|}\}$, $Y = \{y_1, y_2, \cdots, y_{|Y|}\}$, $Z = \{z_1, z_2, \cdots, z_{|Z|}\}$, and $D = \{d_1, d_2, \cdots, d_{|D|}\}$ where $d_i = (x_{i^X}, y_{i^Y}, z_{i^Z})$ for each $i \in [1, |D|]$ and i^X (i^Y or i^Z, respectively) is the corresponding index of the integer x_{i^X} (y_{i^Y} or z_{i^Z}, respectively) in X (Y or Z, respectively). We can define a function f to construct an instance of 2-MCIP as follows:

1. A multiset $\tilde{X} = \{\tilde{x}_i | \tilde{x}_i = 4^i, \forall x_i \in X\}$;
2. A multiset $\tilde{Y} = \{\tilde{y}_i | \tilde{y}_i = 4^{|X|+i}, \forall y_i \in Y\}$;
3. A multiset $\tilde{Z} = \{\tilde{z}_i | \tilde{z}_i = 4^{|X|+|Y|+i}, \forall z_i \in Z\}$;
4. A multiset $\tilde{D} = \{\tilde{d}_i | \tilde{d}_i = \tilde{x}_{i^X} + \tilde{y}_{i^Y} + \tilde{z}_{i^Z}, \forall d_i \in D\}$;
5. An integer $e = \sum_{i=1}^{|D|} \tilde{d}_i - \sum_{i=1}^{|X|} \tilde{x}_i - \sum_{i=1}^{|Y|} \tilde{y}_i - \sum_{i=1}^{|Z|} \tilde{z}_i$.
6. Two multisets $S = \tilde{D}$ and $T = \tilde{X} \cup \tilde{Y} \cup \tilde{Z} \cup \{e\}$.

Since each element in X, Y and Z is assumed to occur at least once in D while some elements occur more than once, it always holds that $e > 0$. Obviously, $\sum S = \sum T$. Therefore, $\langle S, T \rangle$ is an instance of 2-MCIP that we can obtain in time linear in n.

Let $|2\text{-}MCIP|$ denote the size of the minimum common integer partition of $\langle S, T \rangle$. Then, we have the following lemma.

Lemma 6.3.1 *For any instance of MAX 3DM-3,* $|2\text{-}MCIP| \leq 70 \cdot |MAX\ 3DM\text{-}3|$.

Proof. By Lemma 6.2.2, we have $|2\text{-}MCIP| \leq |X| + |Y| + |Z| + |D| \leq 10n$. On the other hand, we have shown that $|MAX\ 3DM\text{-}3| \geq \lceil \frac{n}{7} \rceil$. The lemma thus follows. ■

Given a common integer partition $2\text{-}CIP$ of $\langle S, T \rangle$, we define a function g to construct a subset (denoted as $3DM\text{-}3$) of D by including all the triples $d_i = (x_{i^X}, y_{i^Y}, z_{i^Z})$ $(1 \leq i \leq |D|)$ whose corresponding integers $\tilde{d}_i = \tilde{x}_{i^X} + \tilde{y}_{i^Y} + \tilde{z}_{i^Z}$ are not connected to the integer e in the common integer partition $2\text{-}CIP$.

Lemma 6.3.2 *For any instance D of MAX 3DM-3, the subset $3DM\text{-}3$ constructed by the function g is a matching of D.*

Proof. Let $S_1 = \{\tilde{d}_{\varepsilon_1}, \cdots, \tilde{d}_{\varepsilon_k}\}$ and $T_1 = \{\tilde{x}_{\chi_1}, \cdots, \tilde{x}_{\chi_l}, \tilde{y}_{\gamma_1}, \cdots, \tilde{y}_{\gamma_m}, \tilde{z}_{\zeta_1}, \cdots, \tilde{z}_{\zeta_n}\}$ include all the integers that are not connected to e in the given common integer partition $2\text{-}CIP$ of S and T. It can be seen that, $3DM\text{-}3 = \{d_{\varepsilon_1}, \cdots, d_{\varepsilon_k}\}$, and $\langle S_1, T_1 \rangle$ are a pair of related multisets, *i.e.*,

$$\sum_{i=1}^{k} \tilde{d}_{\varepsilon_i} = \sum_{i=1}^{l} \tilde{x}_{\chi_i} + \sum_{i=1}^{m} \tilde{y}_{\gamma_i} + \sum_{i=1}^{n} \tilde{z}_{\zeta_i}$$

By definition, $\tilde{d}_{\varepsilon_i} = \tilde{x}_{\varepsilon_i^X} + \tilde{y}_{\varepsilon_i^Y} + \tilde{z}_{\varepsilon_i^Z}$, for each $i \in [1, k]$. Thus,

$$\sum_{i=1}^{k} \tilde{x}_{\varepsilon_i^X} + \sum_{i=1}^{k} \tilde{y}_{\varepsilon_i^Y} + \sum_{i=1}^{k} \tilde{z}_{\varepsilon_i^Z} = \sum_{i=1}^{l} \tilde{x}_{\chi_i} + \sum_{i=1}^{m} \tilde{y}_{\gamma_i} + \sum_{i=1}^{n} \tilde{z}_{\zeta_i} \tag{6.1}$$

In order to prove that $3DM\text{-}3$ is a matching of D, it is sufficient to show that the following three pairs of index sets are identical: $\{\varepsilon_1^X, \cdots, \varepsilon_k^X\} = \{\chi_1, \cdots, \chi_l\}$, $\{\varepsilon_1^Y, \cdots, \varepsilon_k^Y\} = \{\gamma_1, \cdots, \gamma_m\}$ and $\{\varepsilon_1^Z, \cdots, \varepsilon_k^Z\} = \{\zeta_1, \cdots, \zeta_n\}$, since no integer element has two copies in T_1. Also notice that, by definition, no two integer elements in T are of equal value.

Let us first assume that $\tilde{x}_{\chi_1}(= 4^{\chi_1})$ be the smallest integer in $S_1 \biguplus T_1$, and apply mod 4^{χ_1+1} to Equation (6.1); that is,

$$\begin{aligned} &\text{Equation (6.1)} \qquad \bmod 4^{\chi_1+1} \\ \Rightarrow \; &\sum_{i=1}^{k} \tilde{x}_{\varepsilon_i^X} \equiv \tilde{x}_{\chi_i} \quad \bmod 4^{\chi_1+1} \end{aligned}$$

This is because any integer in S_1 or T_1 rather than $\tilde{x}_{\chi_1}$ is divisible by 4^{χ_1+1}. On the other hand, the integer $\tilde{x}_{\chi_1}$ occurs at most three times in the multiset $\{\tilde{x}_{\varepsilon_i^X} | 1 \leq i \leq k\}$, and the base that we use to define the integers in S and T is four. Therefore, the above modulo equivalence implies that there is exactly one integer element of $\tilde{x}_{\chi_1}$ in S_1.

If the smallest integer in $S_1 \biguplus T_1$ is $\tilde{x}_{\varepsilon_1^X}$, we can use the same arguments as above to show that there is exactly one integer element of $\tilde{x}_{\varepsilon_1^X}$ in S_1 and also in T_1. Therefore, we can remove the smallest integer from S_1 and T_1, and then repeat the above procedure until the three pairs of index sets are shown to be identical. ■

Let $|2\text{-}MRMP|$ be the size of the maximum related multiset partition of S and T.

Lemma 6.3.3 $|2\text{-}MRMP| = |MAX\ 3DM\text{-}3| + 1$.

Proof. We can see that, each triple $d_i = (x_{i^X}, y_{i^Y}, z_{i^Z})$ in a maximum matching naturally leads to a pair of related submultisets, ***i.e.***, $\{\tilde{d}_i\}$ and $\{\tilde{x}_{i^X}, \tilde{y}_{i^Y}, \tilde{z}_{i^Z}\}$. In addition, there is a pair of related submultisets that contain the integer e. Therefore, $|2\text{-}MRMP| \geq |MAX\ 3DM\text{-}3| + 1$.

In a maximum related multiset partition, there is only one pair of related submultisets that contain the integer e. In any other pair of related submultisets, there exists at least one integer $\tilde{d}_i$, whose corresponding triple d_i will be included in a matching of D. Therefore, $|2\text{-}MRMP| - 1 \leq |MAX\ 3DM\text{-}3|$. ■

Let $|2\text{-}RMP|$ be the size of a related multiset partition of S and T, induced by a given common partition 2-CIP.

Lemma 6.3.4 $|MAX\ 3DM\text{-}3| - |3DM\text{-}3| \leq |2\text{-}CIP| - |2\text{-}MCIP|$.

Proof. We have shown in the proof of the previous lemma that, given a pair of related submultisets that does not include e, there exists at least one integer $\tilde{d}_i$, whose corresponding triple d_i will be included in a matching of D. Therefore, $|2\text{-}RMP| - 1 \leq |3DM\text{-}3|$. By Lemma 6.3.3, we have $|MAX\ 3DM\text{-}3| - |3DM\text{-}3| \leq |2\text{-}MRMP| - |2\text{-}RMP|$. On the other hand, $|2\text{-}MCIP| + |2\text{-}MRMP| = |X| + |Y| + |Z| + |D| + 1$ by lemma 6.2.6, and $|2\text{-}CIP| + |2\text{-}RMP| \geq |X| + |Y| + |Z| + |D| + 1$ by Lemma 6.2.5. Therefore, $|2\text{-}MRMP| - |2\text{-}RMP| \leq |2\text{-}CIP| - |2\text{-}MCIP|$, from which the lemma follows. ■

Lemma 6.3.5 *MAX 3DM-3* $\leq_L$ *2-MCIP.*

```
Algorithm 2-APPROX-MCIP($S$, $T$)
input Two related multisets $S$ and $T$
output A common integer partition $CIP$
        of $S$ and $T$
begin
  $CIP := \emptyset$;
  while $S \neq \emptyset$ do
    arbitrarily pick $x_i \in S$ and $y_j \in T$;
    $S := S \setminus \{x_i\}$;     $T := T \setminus \{y_j\}$;
    $z := \min(x_i, y_j)$;  $CIP := CIP \biguplus \{z\}$;
    if $x_i \neq z$          $S := S \biguplus \{x_i - z\}$;
    if $y_i \neq z$          $T := T \biguplus \{y_i - z\}$;
end.
```

Figure 6.1: A 2-approximation algorithm for 2-MCIP.

Proof. By Lemmas 6.3.1 and 6.3.4, the quadruple $(f, g, 70, 1)$ discussed above gives an L-reduction from MAX 3DM-3 to the 2-MCIP problem [99]. ■

Theorem 6.3.6 *The k-MCIP problem is APX-complete, for any $k \geq 2$.*

Proof. Since the MAX 3DM-3 problem is APX-complete [98] and MAX 3DM-3 $\leq_L$ 2-MCIP by Lemma 6.3.5, 2-MCIP is APX-hard. In addition, by Lemma 6.2.3, there exists a polynomial-time 2-approximation algorithm for 2-MCIP, which implies that 2-MCIP is APX-complete. In Section 6.5, we will present a k-approximation algorithm for k-MCIP, which implies that k-MCIP is APX-complete, for any $k \geq 2$. ■

```
Algorithm $\frac{5}{4}$-Approx-MCIP($S$, $T$)
input Two related multisets $S$ and $T$
output A common integer partition $CIP$ of $S$ and $T$
begin
    remove_common_integer($S$,$T$);
    approximate_set_packing($S$,$T$);
    $CIP := CIP(S_1, T_1) \biguplus CIP(S_2, T_2)$;
    $CIP := CIP \biguplus$ 2-APPROX-MCIP$(S_3, T_3)$;
    return $CIP$;
end.
```

Figure 6.2: A $\frac{5}{4}$-approximation algorithm for 2-MCIP.

6.4 Approximation of 2-MCIP via Maximum Set Packing

In this section, we will give a $\frac{5}{4}$-approximation algorithm for the 2-MCIP problem by considering basic related submultisets of sizes three and four between S and T. As mentioned earlier, we assume that there are no common integer elements between the two input multisets S and T, without loss of generality.

We can construct an instance of the *Maximum Set Packing* problem [82], in which the collection C consists of all the basic related submultisets of sizes three and four between S and T. Since the cardinality of each multiset in C is bounded from the above by a constant, it is actually an instance of the *Maximum k-Set Packing* problem where $k = 4$. Hurkens and Schrijver [96] show that the Maximum k-Set Packing problem is approximable within ratio $k/2 + \epsilon$ for any $\epsilon > 0$. For the weighted version of the Maximum k-Set Packing problem, where each set is given a non-negative weight, Arkin and Hassin [84] show that it is approximable within ratio $k - 1 + \epsilon$ for any $\epsilon > 0$.

In the following, we consider a special weighted Maximum k-Set Packing problem on C, where the weight for each basic related multiset of size three is 2 and the weight for a multiset of size four is 1, and the goal is to find a collection of disjoint multisets of maximum total weight. Call any collection of pairwise disjoint multisets a *packing*. We design a heuristic algorithm, which is implemented in the procedure **approximate_set_packing**(S,T), to find a packing as follows: first find a *maximal* set packing, and then recursively replace a multiset of size four in the packing by a multiset of size three, or replace a multiset of size three by two multisets of size three, or add some multiset into the packing so that the resultant collection is still a packing (but with one more multiset of size three after a replacement or with one more multiset after an addition), until no such replacement or addition could be made further.

The above heuristic algorithm can be made to run in $O(|U| \cdot |C|^2)$ time, where U denotes the universe of the elements in Set Packing, *i.e.*, the multiset union of all multisets in C. In our case, $|U| \leq m + n$. To see this running time, first, given a packing P, we define a mapping $f_P : U \mapsto C \cup \{\emptyset\}$ as follows: for $\forall u \in U$, $f_P(u) = c$ if there exists a multiset $c \in P$ such that $u \in c$, and $f_P(u) = \emptyset$ otherwise. Second, given a multiset $c \in C$, the multisets in the packing P that are not disjoint with c can be found in constant time by looking up the mapping function f_P, because c has only three or four elements. Furthermore, given two multisets c_1 and c_2 in C, we can find in constant time a multiset p in P such that if p is replaced by c_1 and c_2 then $P^{'} = P \biguplus \{c_1, c_2\} \setminus \{p\}$ is still a packing of C, or report no such a multiset p in P exists. Third, after each replacement or addition, updating the mapping function for $P^{'}$ can also be done with f_P in constant time. Therefore, in our heuristic algorithm, a replacement or addition at each iteration can be made in $|C|^2$ time as we may enumerate every pair of multisets in C for a possible replacement or addition. Finally, observe that at most $|U|$ replacements could be made as the number of multisets of size three in the found packing increases by one after each replacement; also observe that at most $|U|$ additions could be made as the number of multisets in a maximal set packing is at most $|U|$.

Let q_3 and q_4 denote the numbers of basic related multisets of sizes three and four in the packing found by our heuristic algorithm, and q_3^* and q_4^* the numbers of basic related multisets of sizes three and four in an optimal weighted set packing, respectively. It is obvious that $2q_3 + q_4 \leq 2q_3^* + q_4^*$. Moreover, we can obtain the following relationship. [2]

Lemma 6.4.1 $2q_3^* + q_4^* \leq 4(q_3 + q_4)$.

Proof. Let $Q_{i,j}^*$, where $i \in \{3, 4\}$ and $1 \leq j \leq i$, be a collection of multisets of size i in the optimal set packing that intersect j multisets in the packing found by our heuristic algorithm, and $q_{i,j}^*$ be the cardinality of $Q_{i,j}^*$. Because the packing found by our heuristic is maximal, we can see that $q_3^* = \sum_{j=1}^{3} q_{3,j}^*$ and $q_4^* = \sum_{j=1}^{4} q_{4,j}^*$. Observe that every multiset of size three (and four) in the packing found by the heuristic intersects at most three (and four, respectively) multisets in the optimal packing, which implies that $\sum_{j=1}^{3} j \cdot q_{3,j}^* + \sum_{j=1}^{4} j \cdot$

---

[2] The $(k/2 + \epsilon)$-approximation algorithm given by Hurkens and Schrijver [96] can also find a packing of C satisfying the inequality in Lemma 6.4.1, but only in quasi-polynomial time.

$q^*_{4,j} \leq 3q_3 + 4q_4$. Furthermore, no two multisets in $Q^*_{3,1}$ can intersect a same multiset in the packing of the heuristic and none of multisets in $Q^*_{3,1}$ intersect a multiset of size four in the packing of the heuristic either, implying that $q^*_{3,1} \leq q_3$. Therefore, it follows that $2q^*_3 + q^*_4 = 2\sum_{j=1}^{3} q^*_{3,j} + \sum_{j=1}^{4} q^*_{4,j} \leq q^*_{3,1} + (\sum_{j=1}^{3} j \cdot q^*_{3,j} + \sum_{j=1}^{4} j \cdot q^*_{4,j}) \leq 4(q_3 + q_4)$.

■

Let q'_3 and q'_4 be the numbers of basic related submultisets of sizes three and four in the related multiset partition induced by a given minimum common partition MCIP(S, T). It is obvious that $2q'_3 + q'_4 \leq 2q^*_3 + q^*_4$. The following is a tighter lower bound for 2-MCIP.

Lemma 6.4.2 $|MCIP(S, T)| \geq \frac{4}{5}(m + n) - \frac{1}{5}(2q^*_3 + q^*_4)$, *where* $m = |S|$ *and* $n = |T|$.

Proof. Based on the given minimum common integer partition MCIP(S, T), we can partition $\langle S, T\rangle$ into three pairs of disjoint related submultisets: $\langle S_1, T_1\rangle$, which consists of integer elements in the basic related submultisets of size three; $\langle S_2, T_2\rangle$, which consists of integer elements in the basic related submultisets of size four; and $\langle S_3, T_3\rangle$, which includes the remaining elements in $\langle S, T\rangle$ such that, $S = S_1 \uplus S_2 \uplus S_3$ and $T = T_1 \uplus T_2 \uplus T_3$. Therefore, we have

$$\begin{aligned} |MCIP(S, T)| &= |MCIP(S_1, T_1)| + |MCIP(S_2, T_2)| + |MCIP(S_3, T_3)| \\ &= 2q'_3 + 3q'_4 + |MCIP(S_3, T_3)| \end{aligned} \qquad (6.2)$$

Since all the basic related submultisets of $\langle S_3, T_3 \rangle$ induced by MCIP(S,T) are of size at least five, by Lemma 6.2.8, we have

$$|MCIP(S_3,T_3)| \geq \frac{4}{5}(m+n-3q_3'-4q_4')$$

and thus

$$|MCIP(S,T)| \geq \frac{4}{5}(m+n) - \frac{1}{5}(2q_3'+q_4') \geq \frac{4}{5}(m+n) - \frac{1}{5}(2q_3^*+q_4^*)$$

from which the lemma follows. ■

The following lemma gives a tighter upper bound for 2-MCIP.

Lemma 6.4.3 $|MCIP(S,T)| \leq m+n-q_3-q_4-1$.

Proof. Observe that we can partition $\langle S, T \rangle$ into three pairs of disjoint related submultisets: $\langle S_1, T_1 \rangle$, which consists of integer elements in the q_3 basic related submultisets of size three; $\langle S_2, T_2 \rangle$, which consists of integer elements in the q_4 basic related submultisets of size four; and $\langle S_3, T_3 \rangle$, which includes the remaining elements in $\langle S, T \rangle$, *i.e.*, $S = S_1 \uplus S_2 \uplus S_3$ and $T = T_1 \uplus T_2 \uplus T_3$. Therefore, we have

$$\begin{aligned} |MCIP(S,T)| &\leq |MCIP(S_1,T_1)| + |MCIP(S_2,T_2)| + |MCIP(S_3,T_3)| \\ &\leq 2q_3 + 3q_4 + |MCIP(S_3,T_3)| \end{aligned} \tag{6.3}$$

$$|MCIP(S_3,T_3)| \leq m+n-3q_3-4q_4-1$$

and thus

$$|MCIP(S,T)| \leq m+n-q_3-q_4-1$$

from which the lemma follows. ■

As mentioned earlier, we run the procedure **approximate_set_packing**(S,T) to find the three disjoint submultisets $\langle S_1,T_1\rangle$, $\langle S_2,T_2\rangle$ and $\langle S_3,T_3\rangle$. A $\frac{5}{4}$-approximation algorithm for 2-MCIP can then be obtained, as illustrated in Figure 6.2. The algorithm runs in time $O((m+n)^9)$, which is dominated by the running time of the procedure **approximate_set_packing**(S,T), as there are $m+n$ elements in the universe and the size of the collection C could reach $\Theta((m+n)^4)$ in the worst case. We believe that the running time can be further reduced by a more careful implementation and analysis of the procedure **approximate_set_packing**(S,T).

Theorem 6.4.4 *The algorithm $\frac{5}{4}$-APPROX-MCIP is a $\frac{5}{4}$-approximation algorithm for 2-MCIP.*

Proof. By Lemmas 6.4.2 and 6.4.3, the approximation ratio α given by algorithm $\frac{5}{4}$-APPROX-MCIP is

$$\alpha \leq \frac{m+n-q_3-q_4-1}{\frac{4}{5}(m+n)-\frac{1}{5}(2q_3^*+q_4^*)} = \frac{5}{4}\cdot\frac{m+n-q_3-q_4-1}{m+n-\frac{1}{4}(2q_3^*+q_4^*)}$$

It suffices to show that $m+n-q_3-q_4-1 \leq m+n-\frac{1}{4}(2q_3^*+q_4^*)$, which is equivalent to showing $2q_3^*+q_4^* \leq 4(q_3+q_4+1)$. By lemma 6.4.1, we know that $2q_3^*+q_4^* \leq 4(q_3+q_4)$.

```
Algorithm $k$-APPROX-MCIP($S_1$, $S_2$,$\cdots$, $S_k$)
input A sequence of related multisets $S_1$, $S_2$,$\cdots$, $S_k$
output A common integer partition $CIP$ of $S_1$, $S_2$,$\cdots$, $S_k$
begin
    $CIP$ := 2-APPROX-MCIP($S_1, S_2$);
    for $i = 3$ to $k$ do
        $CIP$ := 2-APPROX-MCIP($CIP, S_i$);
    return $CIP$;
end.
```

Figure 6.3: A k-approximation algorithm for k-MCIP.

Therefore, $\alpha \leq \frac{5}{4}$. ■

6.5 Approximation of k-MCIP

In this section, we will discuss how to approximate the general k-MCIP ($k \geq 3$) problem.

Using the algorithm 2-Approx-MCIP(S,T) in the previous section, we give an approximation algorithm to solve the k-MCIP ($k \geq 3$) problem, as described in Figure 6.3. First, we give an upper bound on the performance of this algorithm.

Lemma 6.5.1 $|MCIP(S_1, S_2, \cdots, S_k)| \leq \sum_{i=1}^{k} |S_i| - k + 1$.

Proof. After the multiset S_j is processed in the algorithm k-Approx-MCIP(S_1, S_2,$\cdots$, S_k), by Lemma 6.2.2, the size of the common integer partition found so far is upper bounded by $\sum_{i=1}^{j} |S_i| - j + 1$, which holds until j increases up to k. ■

```
Algorithm $\frac{3k(k-1)}{3k-2}$-APPROX-MCIP($S_1$, $S_2$,$\cdots$, $S_k$)
input A sequence of related multisets $S_1$, $S_2$,$\cdots$, $S_k$
output A common integer partition $CIP$ of $S_1$, $S_2$,$\cdots$, $S_k$
begin
    remove_common_integer($S_1$, $S_2$,$\cdots$, $S_k$);
    $CIP := k$-APPROX-MCIP$(S_1, S_2, \cdots, S_k)$;
    return $CIP$;
end.
```

Figure 6.4: A $\frac{3k(k-1)}{3k-2}$-approximation algorithm for k-MCIP.

Theorem 6.5.2 *The algorithm k-APPROX-MCIP is a k-approximation algorithm for the k-MCIP ($k \geq 2$) problem.*

Proof. By Lemma 6.2.1 and Lemma 6.5.1, the size of the common integer partition CIP returned from k-APPROX-MCIP(S_1, S_2,$\cdots$, S_k) is such that $max\{|S_1|, |S_2|, \cdots, |S_k|\} \leq |MCIP(S_1, S_2, \cdots, S_k)| \leq |CIP(S_1, S_2, \cdots, S_k)| \leq \sum_{i=1}^{k} |S_i| - k + 1$, from which the theorem follows. ■

As described in Figure 6.4, the algorithm k-APPROX-MCIP can be slightly improved by employing the procedure **remove_common_integer**$(S_1, S_2, \cdots, S_k)$. To show that this improved algorithm achieves an approximation ratio less than k, we need the following lemma.

Lemma 6.5.3 *If there is no integer element common to all the multisets in $\{S_1, S_2, \cdots, S_k\}$, then it holds that $|MCIP(S_1, S_2, \cdots, S_k)| \geq \frac{3k-2}{3k(k-1)} \sum_{i=1}^{k} |S_i|$.*

Proof. We can see that, there is always a multiset among S_1, S_2,$\cdots$, S_k such that its size is no less than $\frac{1}{k} \sum_{i=1}^{k} |S_i|$. Without loss of generality, we assume that this multiset is S_k.

In an optimal solution MCIP, with respect to S_k, we can divide the elements in a multiset $S_i (1 \leq i \leq k-1)$ into two disjoint submultisets: S_i^1, which consists of elements that are mapped to exactly one (identical) integer in S_k; and S_i^2, which is the complement submultiset of S_i^1, *i.e.*, $S_i^2 = S_i \setminus S_i^1$. Accordingly, with respect to any S_i, S_k can be divided into two disjoint multisets: $S_{k,i}^1$, which consists of elements that are mapped to an integer in S_i^1; and $S_{k,i}^2$, which is the complement submultiset of $S_{k,i}^1$, *i.e.*, $S_{k,i}^2 = S_k \setminus S_{k,i}^1$. Obviously, S_i^1 and $S_{k,i}^1$ are a pair of related multisets, as are S_i^2 and $S_{k,i}^2$.

Notice that, we can always choose a particular multiset S_j, where $1 \leq j \leq k-1$, such that $2|S_j| - |S_j^1| \geq \frac{1}{k-1}\sum_{i=1}^{k-1}(2|S_i| - |S_i^1|)$. In addition, $\sum_{i=1}^{k-1}|S_i^1| \leq (k-2)|S_k|$ holds because $S_1, S_2, \cdots, S_k$ have no common elements. We have

$$
\begin{aligned}
&|MCIP(S_1, S_2, \cdots, S_k)| \\
\geq\ & \frac{2}{3}(|S_k| + |S_j| - 2|S_j^1|) + |S_j^1| \quad \text{(By Lemma 6.2.8)} \\
=\ & \frac{1}{3}\{2|S_k| + 2|S_j| - |S_j^1|\} \\
\geq\ & \frac{1}{3}\{2|S_k| + \frac{1}{k-1}\sum_{i=1}^{k-1}(2|S_i| - |S_i^1|)\} \quad (2|S_j| - |S_j^1| \geq \tfrac{1}{k-1}\textstyle\sum_{i=1}^{k-1}(2|S_i| - |S_i^1|)) \\
=\ & \frac{1}{3}\{2|S_k| + \frac{1}{k-1}(2\sum_{i=1}^{k-1}|S_i| - \sum_{i=1}^{k-1}|S_i^1|)\} \\
\geq\ & \frac{1}{3}\{2|S_k| + \frac{1}{k-1}(2\sum_{i=1}^{k-1}|S_i| - (k-2)|S_k|)\} \quad (\textstyle\sum_{i=1}^{k-1}|S_i^1| \leq (k-2)|S_k|) \\
=\ & \frac{1}{3}\{\frac{k}{k-1}|S_k| + \frac{2}{k-1}\sum_{i=1}^{k-1}|S_i|\} \\
=\ & \frac{1}{3}\{\frac{k-2}{k-1}|S_k| + \frac{2}{k-1}\sum_{i=1}^{k}|S_i|\} \\
\geq\ & \frac{1}{3}\{\frac{k-2}{k-1} \cdot \frac{\sum_{i=1}^{k}|S_i|}{k} + \frac{2}{k-1}\sum_{i=1}^{k}|S_i|\} \quad (|S_k| \geq \tfrac{1}{k}\textstyle\sum_{1}^{k}|S_i|) \\
=\ & \frac{3k-2}{3k(k-1)}\sum_{i=1}^{k}|S_i|
\end{aligned}
$$

■

Theorem 6.5.4 *The algorithm $\frac{3k(k-1)}{3k-2}$-APPROX-MCIP is a $\frac{3k(k-1)}{3k-2}$-approximation algorithm for the k-MCIP ($k \geq 2$) problem.*

Proof. We consider lower and upper bounds of the size of MCIP$(S_1, S_2, \cdots, S_k)$. Let q

denote the number of common integers of $\{S_1, S_2, \cdots, S_k\}$ used in a given minimum common integer partition MCIP$(S_1, S_2, \cdots, S_k)$, and q^* the maximum number of common integers, which is always returned by the procedure **remove_common_integer**$(S_1, S_2, \cdots, S_k)$. Obviously, $q \leq q^*$. By Lemma 6.5.3, we have

$$|MCIP(S_1, S_2, \cdots, S_k)| \geq q + \frac{3k-2}{3k(k-1)} \sum_{i=1}^{k} (|S_i| - q)$$

On the other hand, it follows from the definition of the algorithm $\frac{3k(k-1)}{3k-2}$-APPROX-MCIP and Lemma 6.5.1 that

$$|MCIP(S_1, S_2, \cdots, S_k)| \leq q^* + \sum_{i=1}^{k} (|S_i| - q^*)$$

Therefore, the approximation ratio α achieved by the algorithm $\frac{3k(k-1)}{3k-2}$-APPROX-MCIP is

$$\alpha \leq \frac{q^* + \sum_{i=1}^{k}(|S_i| - q^*)}{q + \frac{3k-2}{3k(k-1)} \sum_{i=1}^{k}(|S_i| - q)} = \frac{3k(k-1)}{3k-2} \cdot \frac{\sum_{i=1}^{k} |S_i| - (k-1)q^*}{\sum_{i=1}^{k} |S_i| - \frac{k}{3k-2} q}$$

Now we show that $(k-1)q^* \geq \frac{k}{3k-2}q$. Since $q \leq q^*$, it is sufficient to prove that $k-1 \geq \frac{k}{3k-2}$, which is obvious to hold for any $k \geq 2$. ■

Clearly, the algorithm $\frac{3k(k-1)}{3k-2}$-APPROX-MCIP$(S_1,\ S_2, \cdots,\ S_k)$ runs in $O(\sum_{i=1}^{k} |S_i| \cdot log(\sum_{i=1}^{k} |S_i|))$ time. Let us compare Theorem 6.5.4 with Theorem 6.5.2. Clearly, $\frac{3k(k-1)}{3k-2}$ is always smaller than k, for any $k \geq 2$. For example, when $k = 2$, the above algorithm gives approximation ratio 1.5, and when $k = 3$, its approximation ratio is $\frac{18}{7}$, which is much better than the ratio 3 in Theorem 6.5.2. However, when k becomes large, $\frac{3k(k-1)}{3k-2}$ is only slightly smaller than k, since $\frac{3k(k-1)}{3k-2} = \Theta(k)$. It is an interesting open question whether k-MCIP has an approximation algorithm with a ratio that is asymptotically better than k.

6.6 Concluding Remarks

It is interesting to observe that although 2-MCIP is in some sense similar to other integer partition/summation problems such as Knapsack and Bin Packing, it is much more difficult to approximate. For example, Knapsack and Bin Packing all have an FPTAS (fully polynomial-time approximation scheme) or asymptotic PTAS, but Theorem 6.3.6 implies that it is unlikely for 2-MCIP to have a PTAS.

Chapter 7

Future Work

It remains interesting to find the extremes to which the approximation and complexity hardness results hold for the MRHC problem. We also aims to find some approximation algorithms for the general MRHC problem on multiple loci. For the ZHRC problem, we wonder if the time complexity on general pedigrees can be improved to $O(mn^2 + n^3)$ or something even lower. Another open question is how to use the algorithm algorithms for ZRHC to solve MRHC on pedigrees that require only a small (constant) number of recombinants. We also try to extend the algorithm for the Loop-Free ZRHC problem to solve the ZRHC problem on a general pedigree in linear time. For the MTCS problem, we are working on how to further narrow the gap between the lower bound and upper bound to select tagSNPs for data with many populations either empirically or theoretically. On the MCIP problem, it is interesting to further improve the approximation ratios for two or multiple input multisets.

Bibliography

[1] L. Aceto *et al.*. The complexity of checking consistency of pedigree information and related problems. *J. Comp. Sci. Tech.*, 19(1):42-59, 2004.

[2] S. Khot. On the power of 2-Prover 1-Round Games. *Proc. of the 34th ACM Symposium on Theory of Computing (STOC'02)*, 767-775, 2002.

[3] A. Agarwal *et al.* $O(\sqrt{\log n})$ approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems. *Proc. STOC'05*, 573-581, 2005.

[4] L. Jorde. Where we are hot, they are not. *Science*, 308:60-62, 2005.

[5] L. Li, J.H. Kim, and M. S.Waterman. Haplotype reconstruction from SNP alignment. *Proc. RECOMB'03*, 207-216, 2003.

[6] R. Lippert, et al. Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Briefings in Bioinformatics* 3(1): 23-31, 2002.

[7] L. Excoffier and M. Slatkin. Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population. *Mol. Biol. Evol.*, 12: 921-927, 1995.

[8] M. Stephens, N. J. Smith, and P. Donnelly. A new statistical method for haplotype reconstruction from population data. *Am. J. Hum. Genet.*, 68(4):978-989, 2001.

[9] T.J. Schaefer. The complexity of satisfiability problems. *Proc. of the 10th STOC*, 216-226, 1978.

[10] G. R. Abecasis, S. S. Cherny, W. O. Cookson and L. R. Cardon, Merlin–rapid analysis of dense genetic maps using sparse gene flow trees. *Nat Genet*, 30(1):97-101, 2002.

[11] M. Y. Chan, W. Chan, F. Chin, S. Fung, and M. Kao Linear-Time Haplotype Inference on Pedigrees without Recombinations *Proc. of the 6th Annual Workshop on Algorithms in Bioinformatics (WABI06)*, to appear, 2006.

[12] F. Chin and Q. Zhang. Haplotype inference on tightly linked markers in pedigree data. *Unpublished manuscript*, 2005.

[13] F. Chin, Q. Zhang, and H. Shen. k-Recombination Haplotype Inference in Pedigrees. *Proc. of the International Workshop on Bioinformatics Research and Applications (ICCS)*, 985-993, 2005.

[14] K. Doi, J. Li and T. Jiang. Minimum recombinant haplotype configuration on tree pedigrees. *Proc. of the 3rd Annual Workshop on Algorithms in Bioinformatics (WABI03)*, 339-353, 2003.

[15] M. J. Daly, J. D. Rioux, S. F. Schaffner, T. J. Hudson, and E. S. Lander. High-resolution haplotype structure in the human genome. *Nat Genet*, 29(2):229–32, 2001.

[16] D. Coppersmith, and S. Winograd. Matrix multiplication via arithmetic programming. *J. Symb. Comput.* 9:251-280, 1990.

[17] M. Elkin, Y. Emeky, D. A. Spielman, S. Teng. Lower-Stretch Spanning Trees *Proc. 37th ACM Symposium on Theory of computing(STOC'05)*, 494-503, 2005.

[18] E. Eskin, E. Halperin, and R. M. Karp. Large scale reconstruction of haplotypes from genotype data. *In Proc. RECOMB'03*, pages 104–113, 2003.

[19] S. B. Gabriel, S. F. Schaffner, H. Nguyen, J. M. Moore, J. Roy, B. Blumenstiel, J. Higgins, M. DeFelice, A. Lochner, M. Faggart, S. N. Liu-Cordero, C. Rotimi, A. Adeyemo, R. Cooper, R. Ward, E. S. Lander, M. J. Daly, and D. Altshuler. The structure of haplotype blocks in the human genome. *Science*, 296(5576):2225–9, 2002.

[20] D. Gusfield. Haplotyping as perfect phylogeny: conceptual framework and efficient solutions. *In Proc. RECOMB'02*, pages 166–175, 2002.

[21] D. Gusfield. An overview of combinatorial methods for haplotype inference. *Lecture Notes in Computer Science (2983): Computational Methods for SNPs and Haplotype Inference.*, 9–25, 2004.

[22] D. F. Gudbjartsson, K. Jonasson, M. L. Frigge, and A. Kong. Allegro, a new computer program for multipoint linkage analysis. *Nat Genet*, 25(1):12–13, 2000.

[23] M. Giesbrecht, A. Lobo, and B. Saunders. Certifying inconsistency of sparse linear systems. *International Symposium on Symbolic and Algebraic Computation* pp. 113-119, 1998.

[24] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopins Univeristy Press, Baltimore, Maryland, third edition, 1996.

[25] L. Helmuth. Genome research: Map of the human genome 3.0. *Science*, 293(5530):583–585, 2001.

[26] B. V. Halldórsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph and S. Istrail. A survey of computational methods for determining haplotypes. *Lecture Notes in Computer Science (2983): Computational Methods for SNPs and Haplotype Inference.*, 26–47, 2004.

[27] L. Kruglyak, M. J. Daly, M. P. Reeve-Daly and E. S. Lander. Parametric and nonparametric linkage analysis: a unified multipoint approach. *Am J Hum Genet*, 58(6):1347–63, 1996.

[28] J. Li and T. Jiang. Efficient rule-Based haplotyping algorithm for pedigree data. *Proc. 7th Annual Conference on Research in Computational Molecular Biology (RECOMB'03)*, 197-206, 2003.

[29] J. Li and T. Jiang. An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming. *Proc. RECOMB'04*, 20-29, 2004.

[30] J. Li and T. Jiang. Computing the minimum recombinant haplotype configuration from incomplete genotype data on a pedigree by Integer Linear Programming. *Journal of Computational Biology* 12(6), 719-739, 2005.

[31] S. Lin and T. P. Speed. An algorithm for haplotype analysis. *J Comput Biol*, 4(4):535–46, 1997.

[32] X. Li, Y. Chen and J. Li. An efficient algorithm for the zero-recombinant haplotype cofiguration problem. *Unpublished manuscript*, 2006.

[33] L. Liu, X. Chen, J. Xiao and T. Jiang. Complexity and approximation of the minimum recombination haplotype configuration problem. *Proc. 16th International Symposium on Algorithms and Computation (ISAAC'05)*, 370-379, 2005.

[34] B. LaMacchia and A. Odlyzko. Solving large sparse linear systems over finite fields. *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology. Lecture Notes In Computer Science* 537, pp. 109 - 133, 1990.

[35] J. R. O'Connell. Zero-recombinant haplotyping: applications to fine mapping using SNPs. *Genet Epidemiol*, 19 Suppl 1:S64–70, 2000.

[36] D. Qian and L. Beckmann. Minimum-recombinant haplotyping in pedigrees. *Am J Hum Genet*, 70(6):1434–1445, 2002.

[37] H. Seltman, K. Roeder, and B.D. Devlin. Transmission/disequilibrium test meets measured haplotype analysis: family-based association analysis guided by evolution of haplotypes. *Am J Hum Genet*, 68(5):1250–1263, 2001.

[38] E. Sobel, K. Lange, J. O'Connell, and D. Weeks. Haplotyping algorithms. *T. Speed and M. Waterman, eds., Genetic Mapping and DNA Sequencing, IMA Vol in Math and its App*, 81:89–110, 1996.

[39] E. Sobel and K. Lange. Descent graphs in pedigree analysis: applications to haplotyping, location scores, and marker-sharing statistics. *Am J Hum Genet*, 58(6):1323–37, 1996.

[40] V. Strassen. Gaussian Elimination is not optimal *Numer. Math.* 13, pp. 354-356, 1969

[41] P. Tapadar, S. Ghosh, and P. P. Majumder. Haplotyping in pedigrees via a genetic algorithm. *Hum Hered*, 50(1):43–56, 2000.

[42] E. M. Wijsman. A deductive method of haplotype analysis in pedigrees. *Am J Hum Genet*, 41(3):356–73, 1987.

[43] D. Wiedemann. Solving sparse linear euquations over finite fields. *IEEE Trans. Inf. Theory*, IT-32:52-62, 1986

[44] S. Zhang *et al.* Transmission/Disequilibrium test based on haplotype sharing for tightly linked markers. *Am J Hum Genet*, 73(3):566–579, 2003.

[45] The international HapMap Consortium. International HapMap Project. *Nature*, 426:789-796, 2003.

[46] J. Xiao, L. Liu, L. Xia, T. Jiang. Fast Elimination of Redundant Linear Equations and Reconstruction of Recombination-Free Mendelian Inheritance on a Pedigree. *Proc. of 18th Annual ACM-SIAM Symoposium on Discrete Algorithms (SODA'07)*, 655-664, 2007.

[47] Lan Liu, Yonghui Wu, Stefano Lonardi and Tao Jiang. Efficient Algorithms for Genome-Wide TagSNP Selection across Populations via the Linkage Disequilibrium Criterion. *To appear in Proc. 6th Annual International Conference on Computational Systems Bioinformatics (CSB)*, 2007.

[48] X. Chen, L. Liu, Z. Liu and T. Jiang. On the Minimum Common Integer Partition Problem. *Proc.of the 6th Conference on Algorithms and Complexity*, 236-247, 2006.

[49] H. Avi-Itzhak *et al.*. Selection of Minimum Subsets of Single Nucleotide Polymorphisms to Capture Haplotype Block Diversity. *Proc. Pac. Symp. Biocomput.*, 466-477, 2003.

[50] P. Bakker *et al.*. Transferability of tag SNPs in Genetic AssociationStudies in Multiple Populations *Nat. Genet.*, 38:1298-1303, 2006.

[51] E. Balas and M. C. Carrera. A Dynamic Subgradient-Based Branch-And-Bound Procedure for Set Covering. Operations Research, 44:875-890, 1996.

[52] R. Bar-Yehuda and S. Moran. On Approximation Problems related to the Independent Set and Vertex Cover Problems. Disc. Appl. Math., 9:1-10, 1984.

[53] C.S. Carlson *et al.*. Selecting a Maximally Informative Set of Singlenucleotide Polymorphisms for Association Analyses using Linkage Disequilibrium. *Am. J. Hum. Genet.*, 74:106-20, 2004.

[54] D.F.Conrad *et al.* A Worldwide Survey of Haplotype Variation and Linkage Disequilibrium in the Human Genome *Nature Genet.* 38:1251-1260, 2006.

[55] K. Ding *et al.* The Effect of Haplotype-block Definitions on Inference of Haplotype-block Structure and htSNPs Selection. *Mol. Biol. Evol.*, 22: 148-159, 2005.

[56] D. A. Hinds *et al.*. Whole-Genome Patterns of Common DNA Variation in Three Human Populations *Science*, 307:1072-1079, 2005.

[57] HapMap LD data. `http://www.hapmap.org/downloads/ld_data/2005-10/`

[58] The International SNPWorking Group. A Map of Human Genome Sequence Variation Containing 1.42 Million Single Nucleotide Polymorphisms. *Nature*, 409: 928-933, 2001.

[59] B.N.Howie *et al.* Efficient Selection of Tagging Single-Nucleotide Polymorphisms in Multiple Populations *Hum. Genet.*, 120:58-68, 2006.

[60] Whole-Genome Patterns of Common DNA Variation in Three Human Populations *Science*, 307:1072-1079, 2005.

[61] E. Halperin, G. Kimmel, and R. Shamir. Tag SNP Selection in Genotype Data for Maximizing SNP Prediction Accuracy. *Bioinformatics*, 21(Suppl 1):i195-i203, 2005.

[62] J. Hampe, S. Schreiber and M. Krawczak. Entropy-based SNP Selection for Genetic Association Studies. *Hum Genet.* 114(1):36-43, 2003.

[63] G.C. Johnson *et al.*. Haplotype Tagging for the Identification of Common Disease Genes. *Nat. Genet.*, 29:233-237, 2001.

[64] L. Kruglyak and D. Nickerson. Variation is the Spice of Life. *Nat. Genet.*, 27:234-236, 2001.

[65] X. Ke and L.R. Cardon. Efficient Selective Screening of Haplotype Tag SNPs. *Bioinformatics*, 19:287-288, 2003.

[66] Z. Lin and R.B. Altman. Finding Haplotype Tagging SNPs by Use of Principal Components Analysis. *Am. J. Hum. Genet.*, 75:850-861, 2004.

[67] Z. Liu, S. Lin and M. Tan. Genome-Wide Tagging SNPs with Entropy-Based Monte Carlo Method. *J. Comput. Biol.*, 13(9):1606-1614, 2006.

[68] R. Magi, L. Kaplinski and M. Remm The Whole Genome TagSNP Selection and Transferability Among HapMap Populations *Pacific Symposium on Biocomputing*, 11:535-543, 2006.

[69] A. C. Need and D.B. Goldstein Genome-wide Tagging for Everyone *Nat. Genet.*, 38:1227-1228, 2006.

[70] N. Patil *et al.*. Blocks of Limited Haplotype Diversity Revealed by High-resolution Scanning of Human Chromosome 21. *Science*, 294:1719-1723, 2001.

[71] T. M. Phuong, Z. Lin and R. B. Altman. Choosing SNPs Using Feature Selection *Proc. Computaional Systems Bioinformatics Conference(CSB)*, 301-309, 2005.

[72] Z. S. Qin, S. Gopalakrishnan and G. R. Abecasis. An Efficient Comprehensive Search Algorithm for TagSNP Selection using Linkage Disequilibrium Criteria *Bioinformatics*, 22: 220-225, 2006.

[73] D.O. Stram *et al.*. Choosing Haplotypetagging SNPs based on Unphased Genotype Data using a Preliminary Sample of Unrelated Subjects with an Example from the Multiethnic Cohort Study. *Hum. Hered.*, 55:27C36, 2003.

[74] P. Sebastiani *et al.*. Minimal Haplotype Tagging. *Proc. Natl. Acad. Sci. USA*, 100:9900-9905, 2003.

[75] S. L. Sawyer, et al. Linkage Disequilibrium Patterns Vary Substantially among Populations. *Eur J Hum Genet*, 13:677-686, 2005.

[76] V. V. Vazirani. Approximation Algorithms, 2003.

[77] N. Wang *et al.* . Distribution of Recombination Crossovers and the Origin of Haplotype Blocks: The Interplay of Population History, Recombination, and Mutation. *Am. J. Hum. Genet.* 71: 1227-1234,2002.

[78] K. Zhang *et al.*. A Dynamic Programming Algorithm for Haplotype Partitioning. *Proc. Natl. Acad. Sci. USA*, 99:7335-7339, 2002.

[79] K. Zhang and L. Jin. HaploBlockFinder: Haplotype Block Analyses. *Bioinformatics* 19:1300-1301, 2003.

[80] K. Zhang *et al.*. HapBlock: Haplo-type Block Partitioning and Tag SNP Selection Software Using a Set of Dynamic Programming Algorithms. *Bioinformatics* 21:131-134, 2005.

[81] E. Zeggini *et al.*. Characterisation of the Genomic Architecture of Human Chromosome 17q and Evaluation of Different Methods for Haplotype Block Definition. *BMC Genet.* 6: 21, 2005.

[82] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*, Springer, 1999.

[83] G.E. Andrews. *The Theory of Partitions*, Addison-Wesley, 1976.

[84] E.M. Arkin and R. Hassin. On local search for weighted packing problems. *Math. Oper. Res.* 23, pp. 640-648, 1998.

[85] G.E. Andrews and K. Eriksson. *The Integer Partitions*, Cambridge, 2004.

[86] S. Altschul and D. Lipman. Trees, stars, and multiple sequence alignment. *SIAM Journal on Applied Math.* 49(1), pp. 197-209, 1989.

[87] M. Chrobak, P. Lolman, and J. Sgall. The greedy algorithm for the minimum common string partition problem. *Proc. of 7th International Workshop on Approximation Algorithms for Combinatiorial Optimization Problems (APPROX)*, pp. 84-95, 2004.

[88] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. *Introduction to algorithms*, The MIT Press, 2nd edition, p. 1017, 2001.

[89] X. Chen. The minimum common partition problem revisited. *manuscript*, 2005.

[90] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Computing the assignment of orthologous genes via genome rearrangement. *Proc. of 3rd Asia Pacific Bioinformatics Conference (APBC'05)*, pp. 363-378, 2005.

[91] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. The assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(4), pp. 302-315, 2005.

[92] Z. Fu. Assignment of orthologous genes for multichromosomal genomes using genome rearrangement. *UCR CS Technical report*, 2004.

[93] D. Gusfield. *Algorithms on Strings, Tree, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, 1997.

[94] A. Goldstein, P. Kolman, and J. Zheng. Minimum common string partition problem: hardness and approximations. *Proc. of 15th International Symposium on Algorithms and Computation* (ISAAC), LNCS 3341, pp. 473-484, 2004.

[95] S. Hannenhalli and P.A. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). *Proc. 27th Ann. ACM Symp. Theory of Comput.* (STOC'95), pp. 178-189, 1995.

[96] C. Hurkens and A. Schrijver. On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM J. Discrete Mathematics*, 2, pp. 68-72, 1989.

[97] P. Kolman. Approximating reversal distance for strings with bounded number of duplicates in linear time. *Proc. of 30 International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pp. 580-590, 2005.

[98] V. Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. *Information Processing Letters*, 37: 27-35, 1991.

[99] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Computer and System Sciences*, 43: 425-440, 1991.

[100] M. Remm, C. Storm, and E. Sonnhammer. Automatic clustering of orthologs and inparalogs from pairwise species comparisons. *J. Mol. Biol.*, 314, pp. 1041-1052, 2001.

[101] D. Sankoff. Mechanisms of genome evolution: models and inference. *Bull. Int. Stat. Instit.* 47, pp. 461-475, 1989.

[102] L. Valinsky, A. Scupham, G.D. Vedova, Z. Liu, A. Figueroa, K. Jampachaisri, B. Yin, E. Bent, R. Mancini-Jones, J. Press, T. Jiang, and J. Borneman. Oligonucleotide Fingerprinting of Ribosomal RNA Genes (OFRG), pp. 569-585. In G. A. Kowalchuk, F. J. de Bruijn, I. M. Head, A. D. L. Akkermans, J. D.van Elsas (eds.) *Molecular Microbial Ecology Manual* (2nd ed). Kluwer Academic Publishers, Dordrecht, The Netherlands, 2004.