

การปรับปรุงเวลาของกำหนดการพลวัตสำหรับแฮพลอไทป์ขนาดยาวในปัญหาการอนุมานแฮพลอไทป์ใน
พันธุ์ประวัติที่มีการขาดหายของแอลลีล



นายอัมเรศ คชรรัตน์

สถาบันวิทยบริการ

จุฬาลงกรณ์มหาวิทยาลัย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคณนา ภาควิชาคณิตศาสตร์

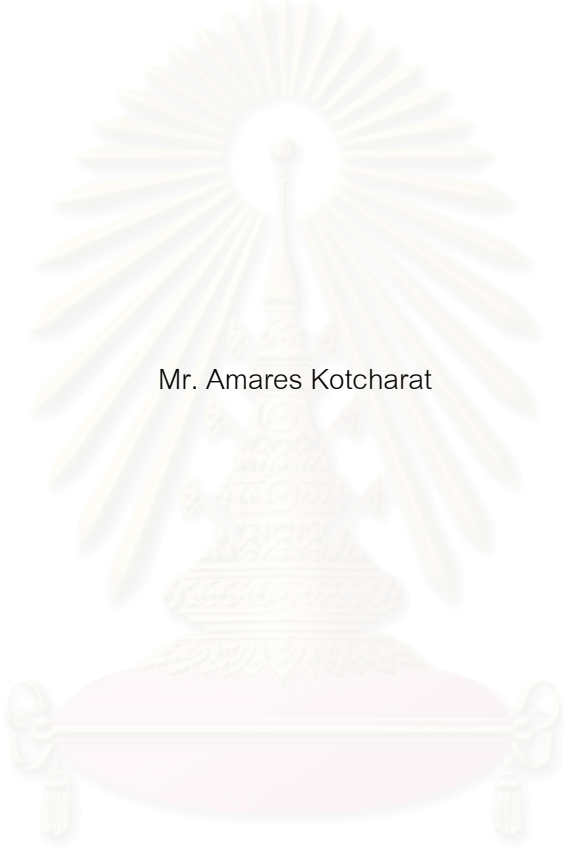
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2547

ISBN 974-17-6505-3

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

IMPROVEMENT OF DYNAMIC PROGRAMMING TIME FOR A LONG HAPLOTYPE IN THE PROBLEM
OF HAPLOTYPE INFERENCE ON A PEDIGREE CONTAINING SOME MISSING ALLELES



Mr. Amares Kotcharat

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computational Science

Department of Mathematics

Faculty of Science

Chulalongkorn University

Academic Year 2004

ISBN 974-17-6505-3

Thesis Title IMPROVEMENT OF DYNAMIC PROGRAMMING TIME FOR A
LONG HAPLOTYPE IN THE PROBLEM OF HAPLOTYPE
INFERENCE ON A PEDIGREE CONTAINING SOME MISSING
ALLELES

By Mr. Amares Kotcharat

Field of study Computational Science

Thesis Advisor Professor Dr. Chidchanok Lursinsap

Accepted by the Faculty of Science, Chulalongkorn University in Partial
Fulfillment of the Requirements for the Master's Degree

..... Dean of the Faculty of Science
(Professor Dr. Piamsak Menasveta)

THESIS COMMITTEE

..... Chairman
(Assistant Professor Dr. Krung Sinapiromsaran)

..... Thesis Advisor
(Professor Dr. Chidchanok Lursinsap)

..... Member
(Assistant Professor Dr. Paisan Nakmahachalasint)

..... Member
(Assistant Professor Tuenchai Kosakul)

..... Member
(Dr. Rath Pichyangkura)

อัมเรศ กษรรัตน์ : การปรับปรุงเวลาของกำหนดการพลวัตสำหรับแฮพลอไทป์ขนาดยาวในปัญหาการอนุมานแฮพลอไทป์ในพันธุ์ประวัติที่มีการขาดหายของแอลลีล. (IMPROVEMENT OF DYNAMIC PROGRAMMING TIME FOR A LONG HAPLOTYPE IN THE PROBLEM OF HAPLOTYPE INFERENCE ON A PEDIGREE CONTAINING SOME MISSING ALLELES) อ. ที่ปรึกษา : ศาสตราจารย์ ดร. ชิดชนก เหลือสินทรัพย์, 138 หน้า. ISBN 974-17-6505-3.

งานวิจัยนี้เกี่ยวข้องกับปัญหาการอนุมานแฮพลอไทป์จากข้อมูลจีโนไทป์ของสมาชิกแต่ละคนในพันธุ์ประวัติโดยใช้เกณฑ์การเกิดรีคอมบิเนชันน้อยที่สุดซึ่งมีประโยชน์ในการลดค่าใช้จ่ายของเทคนิคเชิงปฏิบัติการในทางตรง โดยส่วนใหญ่แล้ววิธีที่ใช้หาผลเฉลยแม่นยำตรงของปัญหาการอนุมานแฮพลอไทป์แบบนี้จะเหมาะสมกับแฮพลอไทป์ขนาดสั้นเท่านั้น จากงานวิจัยหนึ่งซึ่งใช้วิธีกำหนดการพลวัตในการแก้ปัญหานี้ เราได้นำเสนอการปรับปรุงบางอย่างเพื่อให้วิธีดังกล่าวมีความเหมาะสมกับการอนุมานแฮพลอไทป์ขนาดยาวและสามารถใช้กับข้อมูลที่มีการขาดหายของแอลลีล รวมทั้งยังมีการปรับปรุงให้ใช้ได้กับกรณีที่มีบางแอลลีลไม่สอดคล้องกับหลักการถ่ายทอดของเมนเดล ผลการทดลองแสดงให้เห็นว่าวิธีการที่นำเสนอให้ผลในแง่ของเวลาในการคำนวณดีกว่าวิธีเดิมเมื่อใช้กับการอนุมานแฮพลอไทป์ขนาดยาว (มากกว่า 20 ตำแหน่ง) ในพันธุ์ประวัติขนาดปานกลาง (15-25 คน)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

ภาควิชา.....คณิตศาสตร์..... ลายมือชื่อนิสิต.....
สาขาวิชา.....วิทยาการคอมพิวเตอร์..... ลายมือชื่ออาจารย์ที่ปรึกษา.....
ปีการศึกษา...2547

4472500823 : MAJOR Computational Science

KEY WORD: Haplotyping / Haplotype Inference / Dynamic Programming / Missing Allele

AMARES KOTCHARAT : IMPROVEMENT OF DYNAMIC PROGRAMMING TIME FOR A LONG HAPLOTYPE IN THE PROBLEM OF HAPLOTYPE INFERENCE ON A PEDIGREE CONTAINING SOME MISSING ALLELES. THESIS ADVISOR : PROF. CHIDCHANOK LURSINSAP, Ph.D., 138 pp. ISBN 974-17-6505-3.

This study concerns with the problem of inferring haplotypes from a genotype data for each member in a pedigree using a minimum-recombinant criterion, which is useful for reducing the cost of typical laboratory techniques. Most available methods for finding the exact solutions of this problem showed the feasibility for only short haplotypes. Based on the recent method that uses a dynamic programming algorithm, we propose some improvements to make it feasible with long haplotypes and able to work with the data containing some missing alleles. Our improvements also allow the occurrence of a few Mendelian inconsistent alleles which can possibly appear in the data. The experimental results show that the computing time of our method outperforms the original method for the case of inferring long haplotypes (more than 20 loci) in a moderate-size pedigree (15-25 members).

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Department.....Mathematics.....Student's signature.....

Field of study.....Computational Science.....Advisor's signature.....

Academic year...2004.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. I am deeply indebted to my advisor, Prof. Dr. Chidchanok Lursinsap, whose wide knowledge, excellent guidance, and stimulating suggestions helped me in all the time of research and writing of this thesis.

I would like to thanks Asst. Prof. Dr. Krung Sinapiromsaran, Asst. Prof. Dr. Paisan Nakmahachalasint, Asst. Prof. Tuenchai Kosakul, and Dr. Rath Pichyangkura who looked closely at the final English version of this thesis for grammar, correcting, and offering suggestions for improvement. And also, I would like to give a special thanks to Asst. Prof. Tuenchai Kosakul and Dr. Rath Pichyangkura for their valuable comments on genetic concepts behind this thesis.

I am grateful to Assoc.Prof.Suchada Siripant, Apichart Suratane and all my colleagues at the Advanced Virtual and Intelligent Computing (AVIC) Center, Department of Mathematics, Chulalongkorn University, who give me a numerous help, support, and interest.

Especially, I would like to give my special thanks to my parents for their understanding and financial support during my research.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

TABLE OF CONTENTS

	Page
ABSTRACT (THAI).....	iv
ABSTRACT (ENGLISH).....	v
ACKNOWLEDGEMENTS.....	vi
LIST OF TABLES.....	x
LIST OF FIGURES.....	xi
 CHAPTER	
I INTRODUCTION.....	1
1.1 Genetics Background.....	1
1.2 Haplotype Inference on a Pedigree.....	8
1.3 Objective of the Research.....	9
1.4 Scope of the Research.....	12
1.5 Assumptions and Limitations.....	12
1.6 Benefit of the Research.....	13
1.7 Outline of the Research.....	13
II LITERATURE REVIEWS.....	14
III BACKGROUND KNOWLEDGES.....	18
3.1 A Brief Review of Genetics.....	18
3.2 Dynamic Programming.....	22
3.2.1 Shortest Path Problem.....	22
3.2.2 Computational Efficiency of a Dynamic Programming.....	26
3.2.3 Characteristics of Dynamic Programming Applications.....	28
3.2.4 Optimization Method of Dynamic Programming.....	29
IV PROBLEM FORMULATION.....	30
4.1 Preliminary Definitions.....	30
4.2 The problem formulation.....	36

TABLE OF CONTENTS (Continued)

CHAPTER	Page
V HAPLOTYPE INFERENCE ON A PEDIGREE USING DYNAMIC PROGRAMMING.....	40
5.1 The Computational Framework.....	40
5.2 Local Haplotype Inference by a Mendelian Law.....	42
5.3 Detection of Looped Nodes.....	46
5.4 The Pedigree Tree and the Sub Rooted Nodes.....	48
5.5 Elimination of the Haplotype Configurations.....	49
5.5.1 The Algorithm for Eliminating the Haplotype Configurations for a Non-looped Pedigree.....	50
5.5.2 The Problem of the Looped Pedigree.....	51
5.5.3 The Elimination Process.....	54
5.6 Finding the Set of Pedigree Haplotype Configurations with Minimum Number of Inconsistent Alleles.....	56
5.7 Local Grandparent Inference by a Mendelian Law.....	64
5.8 Finding the Set of All Possible Pedigree Grandparent Configurations....	66
5.8.1 Finding the Set of All Possible Family Grandparent Configurations.....	66
5.8.2 Enumerating All Possible Pedigree Grandparent Configurations...	68
5.9 Finding the Grandparent Solutions with Minimum Number of Recombinants by a Dynamic Programming.....	69
5.10 Getting the Haplotype Solutions by Mapping from the Grandparent Solutions.....	75
VI EXPERIMENTAL RESULTS.....	76
6.1 Comparing the Computing Time with PedPhase.....	77
6.1.1 Comparing by the Number of Loci.....	79
6.1.2 Comparing by the Number of Members.....	82

TABLE OF CONTENTS (Continued)

CHAPTER	Page
6.2 The Effect of each Parameter to the Computing Time.....	84
6.2.1. Testing by the number of members.....	84
6.2.2 Testing by the number of loci.....	86
6.2.3 Testing by the Rate of Missing Alleles.....	87
6.3 The Effect of each Parameter to the Accuracy.....	88
6.3.1 Testing for the Rate of Local-Inferred Alleles.....	88
6.3.2 Testing by the Number of Members.....	89
6.3.3 Testing by the Number of loci.....	90
6.3.4 Testing by the Rate of Missing Alleles.....	90
6.3.5 Testing by the Number of Recombination Points.....	91
6.3.6 Testing by the Rate of Homozygous Loci.....	91
6.4 The Effect of Each Parameter to the Number of Solutions.....	92
6.4.1 Testing by the Number of Members.....	93
6.4.2 Testing by the Number of loci.....	94
6.4.3 Testing by the Rate of Missing Alleles.....	95
6.5 Evaluation.....	95
VII CONCLUSION.....	99
7.1 Conclusion.....	99
7.2 Summary of Contributions.....	100
7.3 Further Works.....	100
7.4 Recommendation.....	102
REFERENCES.....	103
APPENDIX A SIMULATING THE PEDIGREE DATA.....	108
APPENDIX B USING THE PROGRAMS.....	111
VITAE.....	125

LIST OF TABLES

Table	Page
5.1 The detail of genotype information.....	43
5.2 The local haplotype inference rules for a particular genotype of the members having some offspring.....	44
5.3 The local haplotype inference rules for a particular genotype of the members having no offspring.....	45
6.1 The computing times varying by the number of loci between our proposed method and the DP-by-Member of PedPhase (by fixing the number of members as 15, 20, and 25).....	80
6.2 The computing times varying by the number of loci between our proposed method and the DP-by-Locus of PedPhase (#Members = 7).....	82
6.3 The computing times varying by the number of members between our proposed method and two dynamic programming algorithms of PedPhase...	83
6.4 The computing time varying by the number of members.....	85
6.5 The computing time varying by the number of loci.....	86
6.6 The computing time varying by the rate of missing alleles.....	87
6.7 The accuracy varying by the number of members.....	90
6.8 The accuracy varying by the number of loci.....	90
6.9 The accuracy varying by the rate of missing alleles.....	90
6.10 The accuracy varying by the number of recombination points.....	91
6.11 The accuracy varying by the rate of homozygous loci.....	92
6.12 The number of solutions varying by the number of members.....	93
6.13 The number of solutions varying by the number of loci.....	94
6.14 The number of solutions varying by the rate of missing alleles.....	95

LIST OF FIGURES

Figure	Page
3.1 The actual road mileages between cities.....	23
3.2 The example of a large network for the shortest path problem.....	27
4.1 The conventional representation of a pedigree (left) and its corresponding representation as a weakly connected directed acyclic graph (right).....	31
4.2 The looped pedigree represented by a conventional form (left) and a graph form (right).....	31
4.3 The representation of genotypes and haplotypes for each member in a pedigree.....	33
4.4 The representation of grandparent vectors corresponding to the relation between the haplotypes of the offspring and its parents.....	34
4.5 The example of an input data.....	38
4.6 The example of a haplotype solution corresponding to the input data in Figure 4.5. This solution has one recombinant as shown in member 16.....	39
5.1 The inference process.....	42
5.2 The looped pedigree and the corresponding looped nodes.....	46
5.3 The looped pedigree and its corresponding tree structure.....	49
5.4 The example for the sub-rooted nodes of a pedigree. In this pedigree, node A, F, K, H, and I are sub-rooted nodes.....	49
5.5 The example looped pedigree with the sets of possible haplotype configurations of each member.....	52
5.6 The performing steps of the haplotype configuration elimination process for the non-looped pedigree (left) and the pedigree with one loop (right).....	53
5.7 The separation of the looped pedigree into non-looped pedigrees.....	55
5.8 The pedigrees and their corresponding sub-rooted nodes. The highlighted nodes are the sub-rooted nodes with the numbers that represent the traversal orders.....	57

LIST OF FIGURES (Continued)

Figure	Page
5.9 The pedigree used as an example for the algorithm of finding the minimum number of inconsistent alleles.....	62
5.10 The example of possible family grandparent configurations corresponding to one family haplotype configuration (here, 'gm' = grandmother and 'gf' = grandfather).....	67
5.11 The example of possible pedigree grandparent configurations corresponding to one pedigree haplotype configuration in which the possible family grandparent configurations for each sub-rooted node are defined as in the figure.....	69
5.12 The network representing the problem of finding grandparent solutions that have minimum number of recombinants.....	70
5.13 The pedigree with the assignment of grandparent configurations at two adjacent loci of each non-founder member. From these configurations, the number of recombination points is 3 as indicated by the arrows.....	71
6.1 The computing times varying by the number of loci between our proposed method and the DP-by-Member of PedPhase (by fixing the number of members as 15, 20, and 25).....	81
6.2 The computing times varying by the number of loci between our proposed method and the DP-by-Locus of PedPhase (#Members = 7).....	82
6.3 The computing times varying by the number of members between our proposed method and two dynamic programming algorithms of PedPhase...	83
6.4 The computing time varying by the number of members.....	85
6.5 The computing time varying by the number of loci.....	86
6.6 The computing time varying by the rate of missing alleles.....	87
6.7 The accuracy varying by the rate of missing alleles.....	91
6.8 The number of solutions varying by the number of members.....	93
6.9 The number of solutions varying by the number of loci.....	94

LIST OF FIGURES (Continued)

Figure	Page
6.10 The number of solutions varying by rate of missing alleles.....	95



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER I

INTRODUCTION

1.1 Genetics Background

Human Genome

The human body is made up of trillions of cells. In the cell, there is a ball-shaped structure called the nucleus which contains thread-like structures called chromatin. The chromatin is a long strand of the genetic material bound together with proteins called deoxyribonucleic acid, or DNA. The sequence of DNA is same as the blueprint of life. DNA provides the information that directs all cellular activities and specifies the development plan of multi-cellular organisms. The creation of various functions and forms of cells in an organism is controlled by the DNA codes. The regions of the DNA sequence that can be coded for proteins are called genes. The human genome is the complete DNA sequence, involving all genetic materials that make up humans.

DNA is a polymer. The monomer units of DNA are nucleotides, and the polymer is known as a “polynucleotide”. Base nucleotide consists of a 5-carbon sugar (deoxyribose), a nitrogen containing base attached to the sugar, and a phosphate group. There are four different types of nucleotides found in DNA, differing only in the nitrogenous base. The four nucleotides are given one letter abbreviation as shorthand for the four bases as follows: A is for adenine, G is for guanine, C is for cytosine, and T is for thymine.

Single Nucleotide Polymorphisms (SNPs)

In the human genome, there are over 3 billions nucleotides. Most nucleotides are similar across the population. About 0.1% of nucleotides in the human genome are varied across the population. A Single Nucleotide Polymorphism (SNP) is a change in DNA that occurs when a single nucleotide (A, T, C, or G) in the DNA

sequence is altered. For example, in the DNA sequence AAGT, a SNP occurs when the G base changes to C, and the sequence becomes AACT. SNPs are generally biallelic systems; that is, there are only two alleles that an individual may have for a particular marker. The biallelic system can be described by the infinite sites model (Halliburton, 2004). Under this model, each nucleotide position (site) within a DNA sequence is considered independently. It assumes that the mutation rate per nucleotide is low enough that most nucleotides do not mutate, and those that do, mutate only once. Thus, any given nucleotide site will show at most only two alleles (different nucleotide cause different gene) in the population.

SNPs are the simplest forms and the most common source of genetic variations in the human genome. Variations in DNA sequence can have a major impact on how humans respond to disease; environmental insults such as bacteria, viruses, toxins, and chemicals; drugs; and other therapies. This makes SNPs of great value for biomedical research and for developing pharmaceutical products or medical diagnostics. It can be concluded that, SNP is a small change in a sequence of DNA. In fact, it involves only a single chemical change. SNPs, by definition, do not cause health problems for people that have them, but they can be useful when studying diseases in the general population, as they are natural variations that we all have.

Haplotype

To study at particular regions of interest on the human genome, researchers usually use a genetic map as a tool. Genetic maps have landmarks known as genetic markers or “markers” for short. The term “marker” is used very broadly to describe any observable variations that result from an alternation, or mutation, at a single genetic locus (the specific location in the DNA sequence is called locus). The commonly used DNA markers are Single Nucleotide Polymorphisms (SNPs) and microsatellite polymorphisms (a variable number of repetitions of a very small number of nucleotides within a sequence). In general, any one of a number of alternative forms of the marker at each locus in the DNA sequence is called allele. Each allele presents the

genetic variant in the human genome. A sequence of alleles from a single chromosome is called haplotype (some literatures may specifically define the haplotype as the group of closely linked alleles that are found in a single chromosome and tend to be inherited together).

In most researches on the problem of haplotype inference, the data that have been analyzed and focused on are typically the SNP haplotypes. The SNP haplotype refers to the sequence of SNP markers on a single chromosome. SNPs are the most common form of genomic variation that is used for disease association. The study of SNPs and their diversity in a population is central to disease association research. The genetic variant influences how people differ in their risk of disease or their response to drugs. It is widely anticipated that the study of variation in the human genome will provide a means of predicting risk of a variety of complex diseases.

Each SNP marker can be analyzed independently from other markers and it succeeds in identifying individual genes responsible for monogenic disease such as Huntingtons and cystic fibrosis. However, most common diseases such as diabetes, cancer, stroke, heart disease, depression, and asthma, are affected by multiple genes and environmental factors. So when trying to map disease gene, it is more effective and informative to analyze SNP markers in a region of interest simultaneously than single SNP markers. And also, when using SNP haplotype, only the information from a relatively small set of SNPs that capture most of the common patterns of SNP haplotypes can be used to test for association with a particular disease, which can reduce the complexity of SNPs study.

Several studies of SNP haplotype structures in the human genome in various populations have been published recently, especially, the international HapMap project which is a large-scale project aiming to determine the common patterns of DNA sequence variation in the human genome, by characterizing sequence variants, their frequencies, and correlations between them, in DNA samples from various populations (The international HapMap consortium, 2003). Such knowledge may provide valuable

information on studies of human evolutionary history, genome wide association, and lead to the development of more efficient strategies to identify genetics variants that increase susceptibility to human diseases (Zhoa, Pfeiffer, and Gail, 2003).

Haplotype Block

Recently, several studies have proposed that the human genome has a block-like structure such that it can be decomposed into large blocks referred to as haplotype blocks (Daly et al., 2001; Patil et al. 2001; Gabriel et al., 2002). A haplotype block is the apparent haplotypic structure of the recombining portions of the genome, in which sets of consecutive co-inherited alleles are separated by short boundaries. In each block, there is little to no evidence for historical recombination and only a few common haplotypes are observed. At any two consecutive blocks, they are separated by a short region called hotspot at which recombination could be inferred in the history of sample. It is now becoming clear that chromosomal recombination is mostly limited to specific hotspots. At present, there is still a debate about the origins of haplotype blocks and whether the boundaries correspond to recombination hotspots.

Within each block, a very small number of common haplotypes (three to five) typically capture about 90% of all chromosomes in each population (Gabriel et al., 2002). This finding is very important to disease association studies, since once the blocks and common haplotypes are identified, one can hopefully obtain a much stronger association between a haplotype and a disease phenotype. Moreover, rather than typing every individual SNP in a particular block, one can choose few representative SNPs that suffice to determine all common haplotypes of the block (Sebastiani et al., 2003). These representative SNPs are called haplotype tagging SNPs (htSNPs) which are able to distinguish the haplotypic variations in a population. Using such tag SNPs allows a major saving in typing costs.

The finding of haplotype block has the initiative of developing of a haplotype map which is expected to be a tool that will allow the discovery of sequence variants that affect common disease, will facilitate development of diagnostic tools, and

will enhance our ability to choose targets for therapeutic intervention (The international HapMap consortium, 2003).

Haplotype mapping is often carried out as part of a genome scan. In a population isolate, the appearance of a rare Mendelian disease is almost always attributable to a single founder gene or mutation. The disease allele can be identified by searching for a common haplotype signature (the haplotype surrounding a particular disease susceptibility allele) shared among patients. As the ancestral haplotype signature is passed from generation to generation, it is disrupted by recombination. Partial conservation of the haplotype signature in a patient strongly suggests that the disease locus resides in the conserved region of the haplotype. (Peltonen, Palotie, and Lange, 2000)

Haplotyping

In diploid organisms (such as human), each cell nucleus contains two nearly identical copies of each chromosome arranged in pair, one in each pair inherited from each parent which is the nature of genetic inheritance for diploid organisms. So at each locus on the chromosome pair, there are two alleles, each one from each parent. The known allele pair at each locus is called genotype which itself does not specify which allele comes from the father or mother. The term “genotype” can refer to the alleles that a person has at a particular locus or for many loci across the genome. The DNA sequence from a chromosome pair may be represented by a genotype or separately represented by two haplotypes. A method that discovers what genotype a person has is called genotyping while a method for determining the haplotypes is called haplotyping.

Currently, a variety of technologies offers practical tools with high-throughput for genotyping, such as the 5' nuclease assay (Taq-Man), the oligonucleotide ligation assay (OLA), and Sequenom's matrix-assisted laser desorption/ionization (Jenkins and Gibson, 2002). In contrast, current experimental methods for haplotyping are technically complicated and cost prohibitive. The direct

laboratory haplotyping assays, such as long-range allele-specific PCR (MichalatosBeloin et al., 1996) or diploid-to-haploid conversion (Douglas et al., 2001), are expensive and low-throughput. Thus, a more sensible strategy is to infer haplotypes from the genotype data using the computational methods, either with or without pedigree information. This strategy is still the major choice in most haplotype-based studies, because of both the lower cost of genotyping and the availability of fast and accurate haplotyping algorithms.

Given the genotype data of each individual, the haplotype of each individual can be inferred using the computational methods (Bonizzoni et al., 2003). There are several methods for inferring haplotypes from genotype data, which can be separated into two categories, statistical methods and rule-based methods. Both methodologies can be applied to pedigree data and population data (that have no pedigree information). Statistical approaches (Lin and Speed, 1997; Fallin and Schork, 2000; Stephens, Smith, and Donnelly, 2001; Stephens and Donnelly, 2003) estimate haplotype frequencies in addition to the haplotype configuration for each individual. But the algorithms of most statistical approaches are usually very time-consuming and thus cannot handle large data sets. On the other hand, rule-based approaches are usually very fast, although they normally do not provide any numerical assessment of the reliability of their results. By utilizing some reasonable biological assumptions such as the minimum recombination principle, rule-based methods have been proven to be powerful and practical (Gusfield, 2001; Eskin, Halperin, and Karp, 2003; Quin and Beckmann, 2002).

Genotyping Error

All large genotype data sets usually have errors caused from the errors in the genotyping process. Genotyping errors can be due to operational errors or genotype scoring errors. Because of an increased use of robotic workstations, stringent quality control procedures, and optimized experimental conditions, the occurrence of operational errors has been greatly reduced for high-throughput genotyping

technologies developed in recent years. In contrast, genotype scoring errors remain a significant challenge for automated scoring programs. (Kang et al., 2004)

There are two types of genotyping errors: those inconsistent with Mendelian inheritance commonly known as “Mendelian errors” and those consistent with Mendelian inheritance. The genotype is considered to be Mendelian consistent if the alleles in the offspring (child) come from its parent’s alleles. If an allele does not inherit from the parent, it is said to be inconsistent. Given sufficient pedigree and marker data, Mendelian errors can be detected by incorporating checks for consistency with Mendelian inheritance. Many genotyping laboratories are content to detect and delete only Mendelian errors while it is much harder to detect the errors that are consistent with Mendelian inheritance. (Sobel, Papp, and Lange, 2002)

Detecting errors using only Mendelian consistency checking has the detection rate about 25% to 30% of the real genotyping errors and the errors are especially difficult to detect for biallelic markers (Gordon, Heath, and Ott, 1999). Thus, checking for Mendelian inconsistency cannot exclude all genotyping errors. The errors in the pedigree data not only cause the inconsistency of Mendelian inheritance, they also cause some artifacts of the recombination event. So it is necessary to be aware of the possible occurrences of both inconsistencies and recombination events in the pedigree data that have a chance of having errors.

Missing Genotype

Determining the genotype value in the genotyping process is performed by genotype scoring. For the automated scoring programs, genotype scoring errors remain a significant challenge. In circumstances when genotype clusters are not sufficiently separated, which can be caused by wide variations in fluorescence signals for different subjects and unbalanced amplifications of the two alternative alleles for heterozygotes, genotype scoring is typically performed manually. However, this is extremely time consuming and error prone (humans are likely to make errors due to fatigue or oversight when manual scoring becomes routine). (Kang et al., 2004)

When the genotype clusters are not sufficiently separated, the unsure determinations of genotypes may result in errors. Instead, as another choice, the ambiguous genotypes may be designated as missing genotype (missing data). For this work, we will use the term “missing alleles” to refer to the ambiguous alleles from the genotyping process. The occurrence of missing alleles in a pedigree data is considered to be significant in this work.

1.2 Haplotype Inference on a Pedigree

We are interested in the problem of haplotype inference from the genotype data on a pedigree, which is the process of determining the haplotypes from the genotype data for each individual in the pedigree. For this problem, the optimality criterion generally used for finding the haplotype solutions is finding haplotype configurations with minimum number of recombinants (Bonizzoni et al., 2003). This criterion comes from a minimum recombination principle which basically says that a genetic recombination is rare and thus haplotypes with fewer recombinants should be preferred in a haplotype inference (Li and Jiang, 2003). The minimum recombination principle is well supported by practical data, especially with the data from the region within a haplotype block (Daly et al., 2001; Patil et al. 2001; Gabriel et al., 2002).

In the case that the genotype data are obtained from the region within a block (haplotype block), there should be no recombination event in the pedigree data. Accordingly, the optimality criterion used for inferring the haplotypes should be modified from the minimum number of recombinants to be no recombinant. But despite the nature of haplotype block, there is still a chance (although very less) for the real pedigree data to have recombination events. These abnormal recombination events can probably occur by the occurrence of genotyping errors which are often found in the practical data. This means that whether the pedigree data come from either the region that probably has recombination events or the region in a block which should have no recombination, there is still a chance for the occurrence of recombination events. Thus,

it is more preferable to use the minimum number of recombinants as an optimality criterion for this inference problem. And so, in this work, we will use the minimum number of recombinants as an optimality criterion for inferring haplotype on a pedigree in order to allow the inference method to be able to support the pedigree data in general cases.

For this work, the problem of haplotype inference from the genotype data on a pedigree can be stated as follows. Given a pedigree and the genotype information of each member in the pedigree with possibly contains missing data; we want to find the haplotype configurations for each member such that there is minimum number of recombination events in the whole pedigree. We call this problem the Minimum-Recombinant Haplotype Configuration (MRHC) problem, which firstly proposed by Tapadar, Ghosh, and Majumder (2000). There are two ways to solve this optimization problem, one that gives the approximated solutions and one that gives the exact solutions. The methods that give the approximated solutions mostly base on the greedy algorithms, which use a shorter running time and can support large data sets. But the solution does not guarantee the optimal solutions. In contrast, the methods that give the exact solutions which guarantee the optimal solutions perform slowly on the large data sets. This is the trade off between these two types of methods.

1.3 Objective of the Research

We are interested in the method for finding the exact solutions of the MRHC problem. From the work of Li and Jiang (2003), it has been proved that finding exact solutions for the MRHC problem is an NP-hard, which means that the computing time increases exponentially by the size of data (the haplotype length and the pedigree size). Although there are some recent works that proposed the methods for finding the exact solutions of the MRHC problem (Doi, Li, and Jiang, 2003; Li and Jiang, 2004), these works still have some limitations.

The first limitation is the restriction on the size of data. The results from the previous works show that their methods have restrictions on the size of pedigree and the haplotype length (Doi, Li, and Jiang, 2003; Li and Jiang, 2004). This restriction comes from the fact that the computing time of this problem increases exponentially by the size of data. In the human genome, haplotype blocks tend to be approximately from a few kilobases (kb) to hundreds kilobases in size (Daly et al., 2001; Patil et al. 2001; Gabriel et al., 2002) and therefore it probably contains up to a hundred of SNPs that travel as a group. Since SNPs occur (on average) every 1,000–2,000 bases (The International SNP Map Working Group, 2001), so the number of SNP in each block can be up to a hundred. This means that the length of haplotype to be inferred can be up to a hundred of loci. But the exponential nature of the computing time limits a typical algorithm to be feasible for only the haplotype of about less than 20 loci in length. This restriction makes the current methods impractical for the case of inferring long haplotypes (more than 20 loci in length).

The major point that we are interested is to develop a method that can work with long haplotypes. By considering from the available methods that have been proposed for solving this problem, we are interested in the method of Doi, Li, and Jiang (2003) which uses the dynamic programming to solve the MRHC problem. The results of this method show that using the dynamic programming can make the computing time linear with increasing of the haplotype length, but it is restricted to work only on very small size of a pedigree (not over 10 of members). The first objective of this work is to improve this method of Doi, Li, and Jiang so that the computing time of the dynamic programming is reduced and also supports larger sizes of a pedigree.

Beside the improvement of computing time, there is another limitation that we will consider. From the method of Doi, Li, and Jiang (2003), it cannot handle the pedigree data that have missing alleles. In practice, pedigree data often contain a significant amount of missing alleles. For example, as much as 14.5% of the alleles belonging to a block could be missing in the pedigree data studied in the work of Gabriel et al., 2002. So another objective is to make the inference process handle the

missing alleles, by incorporating the algorithm for imputing the missing alleles to the method of Doi, Li, and Jiang based on the Mendelian law of inheritance.

The cases that the alleles are Mendelian inconsistent can be caused from the real mutation events or the error of the genotype data (genotyping errors). The case of real mutation events may be very rare but the case of genotyping errors is often found. Although there are many works that proposed the method for identifying the Mendelian inconsistencies in the pedigree data (O'Connell and Weeks, 1998; Aceto et al., 2003), there is no investigation for the case of pedigree data containing some missing alleles. Since this work allows the occurrence of missing alleles, as a consequence, we cannot use the existing methods to handle the problem of Mendelian inconsistent data. All of current available methods for solving MRHC problem assume that the genotype data must be consistent with Mendelian inheritance (Mendelian law). The minor objective of this work is to improve the inference method, so that it can work with the data that possibly contain some Mendelian inconsistent alleles, which has never been investigated in the previous works. The method that we will use for solving this problem is to add a pre-process for finding all possible pedigree haplotypes that has minimum number of inconsistent alleles. This method based on our assumption which assumes that the pedigree haplotypes with fewer inconsistent alleles are preferred in a haplotype inference.

In summary, the objective of this work is to improve the method of Doi, Li, and Jiang (2003) so that:

1. The dynamic programming time (computing time) is reduced and feasible for inferring long haplotypes (more than 20 loci) in a moderate size pedigree (15-25 members).

2. The inference process can handle the missing alleles, by incorporating an algorithm for imputing the missing alleles based on the Mendelian law of inheritance.

3. The occurrence of Mendelian inconsistent alleles is allowed by assuming only a few (this is a minor objective).

1.4 Scope of the Research

This research will focus on the method of finding the minimum recombination haplotype configuration. We will not emphasize on the improvement of the accuracy for the haplotype inference problem. We only emphasize on the improvement of the current dynamic programming method for MRHC problem so that it runs efficiently with respect to the time and data size. We also consider the problem of missing alleles and inconsistent genotype data. The algorithm for handling the inconsistent genotypes is not the error correction algorithm; it is just the way to ignore those inconsistent genotypes which cause the conventional method to be disrupted. So there will be no statistical assessment of the reliability of this algorithm.

Inferring the haplotypes based on the minimum-recombination criterion normally results in many haplotype solutions. This work will not concern with the method for selecting the best one. We may think of this work as the pre-filtering phase for the haplotype inference problem.

The genotype data are assumed to be any types of genetic markers. The data that we used in the research comes from the simulation method. We develop a simulation program for generating the pedigree data so that we can easily incorporate and adjust all variables that will effect on the MRHC problem. The experiment on the real data will be ignored in this research.

1.5 Assumptions and Limitations

The main assumption for this work is that the genotype data for the inference problem is assumed to be from the group of loci in which the recombination

events very rarely occur. The data may be from a region within a block (haplotype block) which should have no recombination event or from a group of any interesting loci.

The occurrence of inconsistent genotypes is assumed to be very rare so that they cannot make any significant effects on the reliability of haplotype solutions.

At present, the public pedigree data are unavailable. Then one limitation is the lack of the real testing data. In this work, we will use the synthetic data which are generated from our simulation model. We assume that we can incorporate all necessary variables that can affect on the results of the algorithm. This means that the simulated data can be used to test and evaluate the algorithm almost as same as the real data.

Since this work only focuses on the improvement of the current dynamic programming method that works with long haplotypes, the other limitation is that the proposed method will has no significant improvement on the aspect of the pedigree size.

1.6 Benefit of the Research

This work improves the current algorithm for the problem of haplotype inference on a pedigree so that the computational method can be used more practically in the real-life usages.

1.7 Outline of the Research

This research is organized as follows. The next chapter is the literature review. In Chapter 3, the biological background for the haplotype inference problem and also the concept of the dynamic programming algorithm will be introduced. The formulation of the problem will be described in Chapter 4. The details of our proposed method will be presented in Chapter 5. The experimental results will be presented in Chapter 6 and the conclusion is in Chapter 7.

CHAPTER II

LITERATURE REVIEW

A study of the haplotype diversity in a population became central to disease association research. The haplotypes was used for genome wide association studies and in the study of population histories. Unfortunately, current experimental techniques to infer the haplotypes of an individual are both expensive and time consuming. In contrast, a variety of current technologies offers practical tools with high-throughput for genotype determination. Therefore, several methods of inferring haplotypes from the genotype data have been proposed (Bonizzoni et al., 2003).

The previous works of inferring haplotypes from genotype data can be fit into two categories, statistical methods and rule-based methods. Statistical approaches (Lin and Speed, 1997; Fallin and Schork, 2000; Stephens, Smith, and Donnelly, 2001; Stephens and Donnelly, 2003) estimate haplotype frequencies in addition to the haplotype configurations for each individual. The statistical approaches are time consuming and only used the population genotype data without a pedigree structure. The inferred haplotypes from the statistical methods that perform inference without a pedigree data are not guaranteed to be consistent with the Mendelian law. On the other hand, rule-based approaches typically use the genotype data with a pedigree structure. The addition information from the pedigree structure makes the rule-base approaches faster than the statistical approaches. The haplotype inference with a pedigree can result in a high degree of accuracy because the constraints provided by other members in a pedigree would force the algorithm to settle on a unique haplotype as being the most probable.

Haplotyping analysis in a pedigree considers the whole space of all possible distinct haplotype configurations and uses some criteria to select the most probable haplotype configurations.

Tapadar, Ghosh, and Majumder (2000) firstly used the minimum recombination principle for the problem of haplotype inference on a pedigree. The minimum recombination principle states that the genetic recombination is rare and thus haplotypes with only a few of recombinants should be included in a haplotype inference (the rationale for and pitfalls of ignoring those haplotype configurations that does not have minimum number of recombinants require further investigation). The haplotype inference problem based on this principle can be formulated as an optimization problem in which the solutions are those haplotype configurations that have minimum number of recombinants. The method that Tapadar, Ghosh, and Majumder proposed uses a genetic algorithm to find the optimal solutions. But their method is hard to extend to handle the missing genotypes and is not expected to find all minimum-recombinant haplotype configurations in a limited computing time. In this paper, we refer to the haplotype inference using the minimum recombination principle as the minimum-recombinant haplotype configuration (MRHC) problem.

Qian and Beckmann (2002) proposed a six-rule minimum-recombinant haplotyping (MRH) algorithm that exhaustively searches all possible minimum-recombinant haplotype configurations in a large pedigree with many markers and allows missing genotype data to be imputed during the haplotyping process. This method performs very well for small pedigrees, but it runs extremely slow on data of moderate sizes, especially for data with biallelic markers.

Li and Jiang (2003) showed that MRHC problem is in general NP-hard, which means that the computing time increases exponentially with respect to the data size. They developed an iterative heuristic algorithm, called block-extension, for finding MRHC which is more efficient than the previous work of Qian and Beckmann. This algorithm assumed that the input genotype data are consistent with the Mendelian law. The missing alleles are imputed using random sampling method based on the allele frequency. Although this algorithm does not give the exact or real optimal solutions (the haplotype configurations with minimum number of recombinants), the experimental

results show that the solutions from the block-extension algorithm are often close to the optimal solutions when the minimum number of recombinants required is small.

Doi, Li, and Jiang (2003) proposed two dynamic programming algorithms for finding the exact solutions of MRHC problem, the locus-based and the member-based dynamic programming algorithms. Their algorithms cannot handle missing data and the algorithms are developed for small data size. The member-based algorithm is appropriated for pedigrees of small sizes (less than 10 members) and the locus-based algorithm is appropriated for non-looped pedigrees with a small number of marker loci.

Li and Jiang (2004) develop an effective integer linear programming (ILP) formulation of the MRHC problem with missing data. The ILP is solved by a branch-and-bound strategy that utilizes a partial order relationship (and some other special relationships) among variables to decide the branching order. This method can give the exact solutions and also incorporate a consistent imputation of missing alleles in the optimization process. Their experiment on the simulated data shows very efficient and highly accurate results. But we still in doubt with the results of their algorithm because when we used their program, PedPhase, to test with our simulated data, we found that the solutions are not exactly optimal.

In this work, we are interested in the method that can work with long haplotypes. By considering available methods, we are interested in the method of Doi, Li, and Jiang (2003) which uses dynamic programming algorithm to solve the MRHC problem. The results from their method showed the linear computing time using the dynamic programming, though it was restricted to work only on very small size of a pedigree (not over 10 members). This shows the feasibility of the dynamic programming method to be used for the case of inferring long haplotypes.

Also, all previous works assume that the input genotype data must be consistent with the Mendelian law. The genotype is consistent with the Mendelian law if the alleles in the offspring (child) come from its parents' alleles. If the allele does not

inherit from the parents, it is said to be Mendelian inconsistent. The case that the pedigree data contain some Mendelian inconsistencies will be also considered in this work.

For this research, we intend to develop an algorithm for inferring long haplotypes from the genotype data on a pedigree that contains some missing alleles. We will improve the dynamic programming method of Doi, Li, and Jiang (2003) so that it works efficiently with respect to the computing time and data size; and also, it works with the data containing some missing alleles and with the rare occurrence of Mendelian inconsistent alleles.



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER III

BACKGROUND KNOWLEDGES

3.1 A Brief Review of Genetics

Genetics is about the transfer of information among many different levels of an organism. The foundation level is a molecule called DNA. The information in DNA is organized into genes. Genes, in turn, make up chromosomes. Every cell in an individual contains a genome. At last, the population level involves how individuals are genetically connected to their families and larger populations through inheritance patterns. The genetic terms related to each level are described as follows.

DNA

DNA, an abbreviation for deoxyribonucleic acid, is a macro-molecule polymer found in the nucleus of cells. DNA is in the form of twisted double strand (double helix) held together by weak bonds between base pairs of nucleotides. The four nucleotides in DNA contain the bases adenine (A), guanine (G), cytosine (C) and thymine (T). In nature, base pairs form only between A and T and between G and C, thus the base sequence of each single strand can be deduced from that of its partner. DNA carries the genetic information necessary for the organization and functioning of most living cells. The nucleotide sequence on the DNA acts like a blueprint of life. It directs all cellular activities and specifies the development plan of multi-cellular organisms. The specific location in the DNA sequence is called locus.

Mutation

Mutation is defined as any heritable change in the genetic material. In the DNA sequence, mutation is both the process by which a gene or chromosome changes structurally and the end result of that process. Mutation is the ultimate source of all genetic variations; without mutation, the biological diversity that exists today could not have evolved. The mutation can be caused by an error in replication or some

external influence such as exposure to chemicals and radiation. The mutation of primary concern in this work is the point mutation. The point mutation is the change of a single nucleotide in the nucleotide sequence. When a particular nucleotide is replaced with the other one, the reading codon of the gene will change and may result in a new protein. So, mutation is an alternation in the sequence of DNA, which can potentially lead to health problems.

Gene

Gene is the basic unit of heredity in a living cell. It is a sequence of contiguous nucleotides located in a particular position on a particular chromosome that encodes for a specific RNA molecule or protein. Genes are the “blueprint” for the growth and development of the organisms. The alternative forms of a gene at any specific locus are called alleles. In general, the term “allele” is used as one of the variant forms of a DNA sequence at a particular locus on a chromosome (such as the genetic markers).

Chromosome

A chromosome is a thread-like structure found in the nucleus of a cell. It is a very long strand of DNA, which carries the genetic material in the form of genes, regulatory elements and other intervening nucleotide sequences. Each cell in the human body contains 23 pairs of chromosomes. One chromosome in each pair comes from the mother and one from the father. Egg and sperm cells have only 23 chromosomes each. Chromosomes number 1-22 are autosomes and the last pair is the sex chromosomes, XX for a female and XY for a male. Each chromosome contains about 2,000 genes. The total human genome consists of about 35,000 genes.

Recombination

During meiosis (the type of cell division that produces the gametes, egg and sperm), two chromosomes of each pair become physically close. And then, the exchange of portions between two closed chromosomes may occur. The resulting chromosomes are the combination of the portions from the two original chromosomes.

This process is called recombination and the process of portions exchanging is called crossing-over.

Genetic Markers

Just like the city maps that have buildings served as landmarks, genetic maps have landmarks known as genetic markers or “markers” for short. The term “marker” is used very broadly to describe any observable variations that result from an alternation, or mutation, at a single genetic locus (the specific location in the DNA sequence is called locus). Genetic marker is a segment of DNA at a known physical location on a chromosome. A marker can be a gene or a section of DNA with no known function. They can be used to track the inheritance pattern of genes that have not yet been identified, but whose approximate locations are known. The commonly used DNA markers are SNPs and microsatellite polymorphisms. Like the form of gene, each different forms of the genetic marker is also called allele.

SNP (single nucleotide polymorphism) is a variation in the DNA sequence that occurs when a single nucleotide (A, T, C, or G) in the genome sequence is altered. For a variation to be considered as SNP, it must occur in at least 1% of the population. SNPs, which make up about 90% of all human genetic variations, occur every 100 to 300 bases along the 3-billion-base human genome. Most of the SNPs have only two alleles (called biallelic).

Microsatellite is defined as a unit of variable number of repeating of about 2 to 5 nucleotides (e.g., CACACA). The average number of repeats in a cluster is usually about 10 to 20. At each locus there may be dozens of different alleles corresponding to different numbers of repeats. In the human genome, there are at least 50,000 microsatellite loci.

Genetic Inheritance

In diploid organisms (such as human), each cell nucleus contains two nearly identical copies of each chromosome arranged in pair, one in each pair inherited

from each parent which is the nature of genetic inheritance for diploid organisms. So at each locus on the chromosome pair, there are two alleles, each one from each parent. The known allele pair at each locus is called genotype which itself does not specify which allele comes from the father or the mother. . The term “genotype” can refer to the alleles that a person has at a particular locus or for many loci across the genome. At a particular locus, if it has two identical alleles, it is said to be homozygous otherwise it is said to be heterozygous.

The nature of the genetic inheritance of the diploid organisms is corresponding to the Mendelian law of inheritance. In the Mendelian law, it states that the alleles in an offspring (child) must inherit from the alleles of the parents. For a particular allele in an offspring, if it inherits from at least one of the parents, we will say that this allele is consistent with the Mendelian law or it is Mendelian consistent, otherwise it is inconsistent with the Mendelian law.

Pedigree

To study the inheritance of the genetic traits in the population, the biologists usually use a pedigree. Generally, the pedigree is a family tree diagram that shows how a particular genetic trait or disease has been inherited. But for this work, the pedigree is used to show the genotype data corresponding to each member and to show the relations of the genotypes between each related member. In the pedigree, all members connect to each other by some relations such as mate, offspring (child) or parent, and so the pedigree can be represented as a linked graph.

Typically, the researches on the computational method for the problem of haplotype inference on a pedigree need a large number of pedigree samples for the experimental testing. We have found that the public sources for obtaining the pedigree data that represent the genotypes of each member is hard to find, especially if we are not the researchers that directly concern in the field of genetics. Most of researches on the problem of haplotype inference on a pedigree usually use the simulated version of the pedigree data.

To simulate the pedigree data, one must understand the basic of population genetics. Population genetics is the field that considers the factors that determine the evolution of a population, such as a natural selection, a genetic drift, a mutation, a recombination, and a gene flow. We can simulate the genetic inheritance in the population using the evolution models that have been presented by many theories in population genetics. In population genetics, the population is usually separated into generations. Each generation contains the group of population that is alive in the same period of time. The genetic inheritance from one generation the other consecutive generation is controlled by many of genetic variables. Based on this generation-based model, we can build the simulation program for generating the genotype data of the population for many generations. Therefore, the genotype data of the population in the form of pedigree structure can be obtained from the simulation program.

3.2 Dynamic Programming

(This section is adapted from Winston, 2004)

Dynamic programming is one of a technique used to solve an optimization problem. In general, dynamic programming obtains solutions by breaking up a large, unwieldy problem into a series of smaller, more tractable problems. The idea is to find the solutions from the subproblems which have smaller size, and then, use the results of these subproblems to find the solutions of the larger problems until ending with the original size of the problem.

3.2.1 Shortest Path Problem

In this section, we will introduce the idea of the dynamic programming by demonstrating the shortest path problem and showing how the dynamic programming can be used to solve this problem. The example of the shortest path problem can be described as follows.

Tom lives in the city A, but he plans to drive to the city J to seek fame and fortune. Tom's budgets are limited, so he has decided to spend each night on his trip at a friend's house. Tom has friends in the cities B, C, D, E, F, G, H and I. Tom knows that after one day's drive he can reach the cities B, C, or D. After two days of a driving, he can reach the cities E, F, or G. After three days of a driving, he can reach the cities H or I. Finally, after four days of a driving, he can reach the city J. To minimize the number of miles traveled, where should Tom spend each night of the trip? The actual road mileages between cities are given in Figure 3.1.

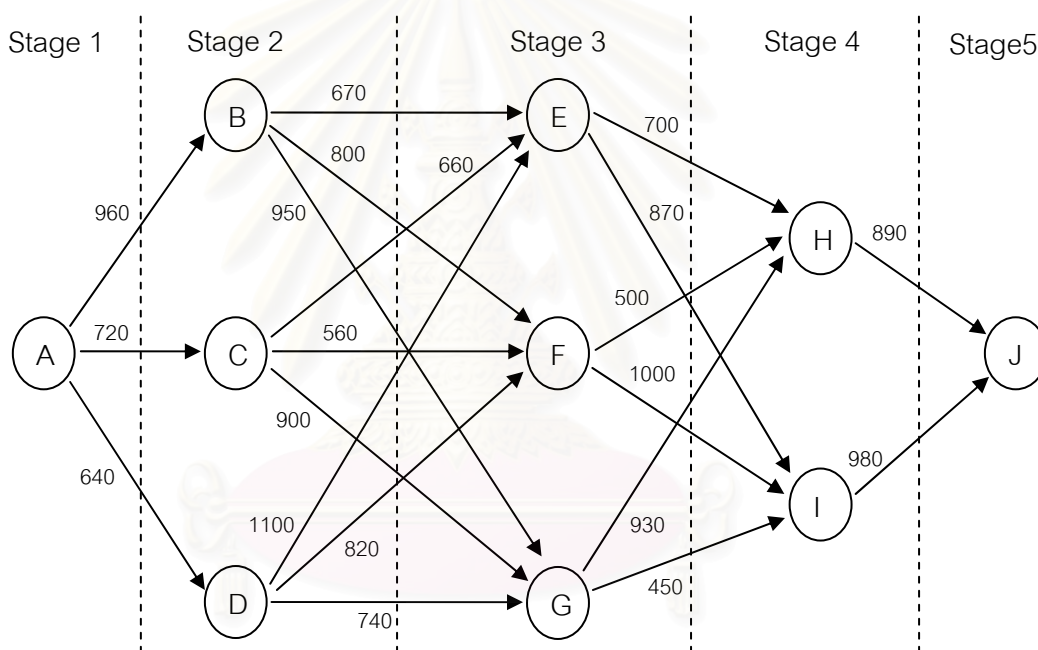


Figure 3.1 The actual road mileages between cities.

We have classified all the cities that tom can stay at the beginning of the n^{th} day of his trip as stage n cities. Tom needs to know the shortest path between the cities A and J. We will find it by beginning with smallest subproblem, two stages problem, and extend to the larger problem until ending with the five stages problem. The idea of this method is that we should begin by solving an easy problem that will eventually help us to solve a complex problem. Hence, we begin by finding the shortest path from the city A to each city in stage 2, which will make the new three subproblems.

Then, we use this information to find the shortest path from the city A to each city in stage 3. With this information in hand, we are able to find the shortest path from the city A to each city in stage 4. Finally, we can find the shortest path from the city A to the city J in the last stage.

We define $c_{i,j}$ as the road mileage between the city i and the city j . For example, $c_{C,E} = 660$ is the road mileage between the city C and the city E. Let $f_t(i)$ be the length of the shortest path from the city A to the city i in stage t .

The computation for stage 2

We first determine the shortest path from the beginning city A to each city in stage 2. We immediately see that there is only one path from the city A to each city in stage 2. Then the shortest paths can be computed as follows. $f_2(B) = 960$, $f_2(C) = 720$, and $f_2(D) = 640$.

The computation for stage 3

We now work on the next stage. Let's consider the city E. The shortest path from the city A to the city E must be one of the following.

Path 1: Go from the city A with the shortest path to the city B and, then, go from the city B to the city E.

Path 2: Go from the city A with the shortest path to the city C and, then, go from the city C to the city E.

Path 3: Go from the city A with the shortest path to the city D and, then, go from the city D to the city E.

The length of path 1 can be written as $f_2(B) + c_{B,E}$. The length of path 2 can be written as $f_2(C) + c_{C,E}$. And the length of path 3 can be written as $f_2(D) + c_{D,E}$. Hence, the shortest distance from the city A to the city E can be written as

$$f_3(E) = \min \left\{ \begin{array}{l} f_2(B) + c_{B,E} = 960 + 670 = 1,630 \\ f_2(C) + c_{C,E} = 720 + 660 = 1,380 \\ f_2(D) + c_{D,E} = 640 + 1,100 = 1,740 \end{array} \right\}$$

Thus, the shortest path from the city A to E is the path A-C-E which has the distance of 1,380 miles. We can see that to obtain this result, we have made use of our knowledge of $f_2(B)$, $f_2(C)$, and $f_2(D)$.

Similarly, we can find the shortest path from the city A to the cities F and G as follows.

$$f_3(F) = \min \left\{ \begin{array}{l} f_2(B) + c_{B,F} = 960 + 800 = 1,760 \\ f_2(C) + c_{C,F} = 720 + 560 = 1,280 \\ f_2(D) + c_{D,F} = 640 + 820 = 1,460 \end{array} \right\}$$

$$f_3(G) = \min \left\{ \begin{array}{l} f_2(B) + c_{B,G} = 960 + 950 = 1,910 \\ f_2(C) + c_{C,G} = 720 + 900 = 1,620 \\ f_2(D) + c_{D,G} = 640 + 740 = 1,380 \end{array} \right\}$$

The shortest path from the city A to F is the path A-C-F with the distance of 1,280 miles and the shortest path from the city A to G is the path A-D-G with the distance of 1,380 miles.

The computation for stage 4

Given the knowledge of $f_3(E)$, $f_3(F)$, and $f_3(G)$, we can easily find the shortest path from the city A to each city in stage 4 by the same method used for stage 3. The computation can be written as follows.

$$f_4(H) = \min \left\{ \begin{array}{l} f_3(E) + c_{E,H} = 1,380 + 700 = 2,080 \\ f_3(F) + c_{F,H} = 1,280 + 500 = 1,780 \\ f_3(G) + c_{G,H} = 1,380 + 930 = 2,310 \end{array} \right\}$$

$$f_4(I) = \min \left\{ \begin{array}{l} f_3(E) + c_{E,I} = 1,380 + 870 = 2,250 \\ f_3(F) + c_{F,I} = 1,280 + 1000 = 2,280 \\ f_3(G) + c_{G,I} = 1,380 + 450 = 1,830 \end{array} \right\}$$

Here, we can see that the shortest path from the city A to H has the distance of 1,780 miles and consists of the shortest path from the city A to F appended with the path from the city F to H, which is the path A-C-F-H. Similarly, the shortest path from the city A to I is the path A-D-G-I with the distance of 1,830 miles.

The computation for stage 5

We can now find the solution of the Tom's problem, the shortest path from the city A to the city J. For the last stage, we have the knowledge of $f_4(H)$ and $f_4(I)$. The computation can be written as follows.

$$f_5(J) = \min \left\{ \begin{array}{l} f_4(H) + c_{H,J} = 1,780 + 890 = 2,670 \\ f_4(I) + c_{I,J} = 1,830 + 980 = 2,810 \end{array} \right\}$$

From the computation above, the shortest path from the city A to the city J consists of the shortest path from the city A to H appended with the path from the city H to J. Thus, the solution is the path A-C-F-H-J with the distance of 2,670 miles. And this path is the optimal path for Tom to drive from the city A to the city J.

3.2.2 Computational Efficiency of a Dynamic Programming

For the example of the shortest path problem described above, it would have been an easy matter to determine the shortest path by enumerating all possible paths which, after all, there are only $3(3)(2) = 18$ paths. Thus, in this example problem, the use of dynamic programming did not really serve much purpose. But for the larger networks, however, dynamic programming is much more efficient for determining a shortest path than the explicit enumeration of all paths. To see this, consider the network in Figure 3.2. In this network, it is possible to travel from any node in stage k to any node in stage $k+1$. Let $c_{i,j}$ denote the distance between node i and node j . Suppose that

we want to determine the shortest path from node 1 to node 27. One way to solve this problem is explicit enumeration of all paths. There are 5^5 possible paths from node 1 to node 27. It takes five additions to determine the length of each path. Thus, explicitly enumerating the length of all paths requires $5^5(5) = 15,625$ additions.

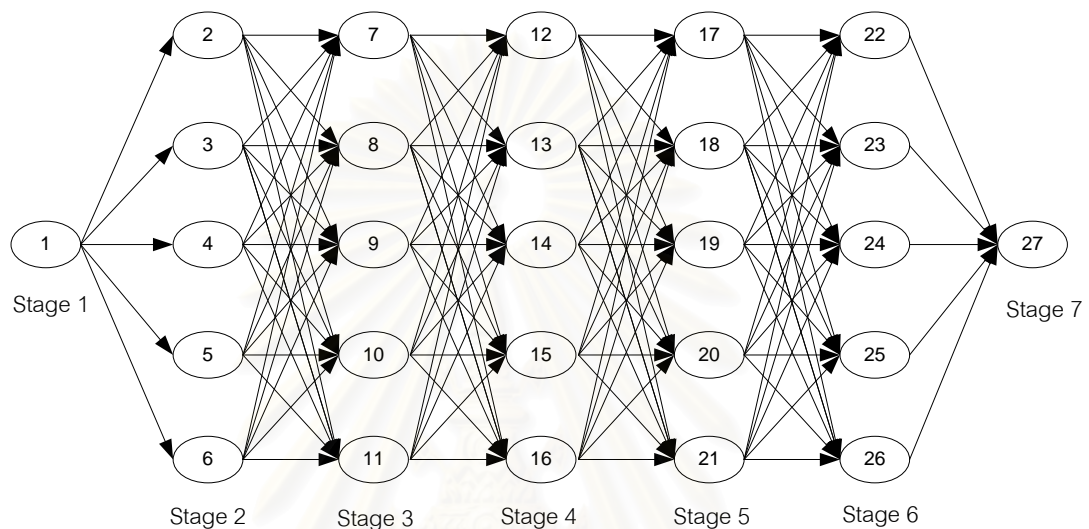


Figure 3.2 The example of a large network for the shortest path problem.

Suppose we use dynamic programming to determine the shortest path from node 1 to node 27. Let's $f_t(i)$ be the length of the shortest path from node 1 to node i in stage t . To determine the shortest path from node 1 to node 27, we begin by finding $f_2(2)$, $f_2(3)$, $f_2(4)$, $f_2(5)$, and $f_2(6)$. This does not require any additions. Then we consider at the next stage by finding $f_3(7)$, $f_3(8)$, $f_3(9)$, $f_3(10)$, and $f_3(11)$. We repeat for each stage until we can find $f_7(27)$. To find each $f_t(i)$, for example $f_3(7)$, we use the following equation:

$$f_3(7) = \min_j \{ f_2(j) + c_{j,7} \} \quad (j = 2, 3, 4, 5, 6)$$

Determining $f_3(7)$ in this manner requires five additions. Thus, the calculation of all the $f_3(\cdot)$'s requires $5(5) = 25$ additions. Similarly, the calculation of all the $f_4(\cdot)$'s requires 25 additions, all the $f_5(\cdot)$'s requires 25 additions, all the $f_6(\cdot)$'s requires 25 additions, and for the last $f_7(27)$ requires 5 additions. Thus, in total,

dynamic programming requires $4(25) + 5 = 105$ additions to find the shortest path from node 1 to node 27. Because the explicit enumeration requires 15,625 additions, we can see that dynamic programming requires only 0.007 times as many additions as explicit enumeration. For larger networks, the computational savings affected by dynamic programming are even more dramatic.

Beside additions, determination of the shortest path in a network requires comparisons between the lengths of path. If explicit enumeration is used, then $5^5 - 1 = 3,124$ comparisons must be made (that is, compare the length of the first two paths, then compare the third path with the shortest path of the first two paths, and so on). If dynamic programming is used, for $t = 3, 4, 5, 6, 7$, determination of each $f_t(i)$ requires $5-1 = 4$ comparisons. Thus to find shortest path from node 1 to node 27, dynamic programming requires a total of $20(5-1) + 4 = 84$ comparison. Again, dynamic programming comes out far superior to the explicit enumeration.

3.2.3 Characteristics of Dynamic Programming Applications

The applications or problems that are appropriated to be solved by the dynamic programming method usually have some common characteristics as follows.

Characteristic 1: The problem can be divided into stages with a decision required at each stage.

Characteristic 2: Each stage has a number of states associated with it. By a state, we mean the information that is needed at any stage to make the optimal decision.

Characteristic 3: The decision chosen at any stage describes how the state at the current stage is transformed into the state at the next stage.

Characteristic 4: Given the current state, the optimal decision for each of the remaining stages must not depend on previously reached states or previously chosen decisions. This idea is known as the principle of optimality.

Characteristic 5: If the states for the problem have been classified into one of T stages, there must be a recursion that relates the cost or reward earned between two adjacent states.

3.2.4 Optimization Method of Dynamic Programming

The first step of the dynamic programming is to formulate the problem as the recursion. For the minimization problem, the recursion can be write in general form as

$$f_t(i) = \min_j \{ c_{j,i} + f_{t-1}(j) \}$$

where $t \in \{1, \dots, T\}$, $i \in \{1, \dots, N^t\}$ and $j \in \{1, \dots, N^{t-1}\}$.

Let T be the total number of stages. N^t denotes the number of states in stage t and N^{t-1} denotes the number of states in stage $t-1$. $f_t(i)$ is a function of an optimal value for the subproblem that end at the stage t with the state i . Each $f_t(i)$ will has the solutions (can be one or more) associated with it. Here, one particular solution of the problem is the selection of specific states for each stage (one state for each stage).

$c_{j,i}$ denotes a cost or weight for the transition or link from the state j of the previous stage $t-1$ to the state i of the current stage t . Or we can say that when the state of the previous stage is j , if we choose the state i for the current stage t , it will cost $c_{j,i}$.

We can now describe how to make optimal solutions. Let's $f_1(\cdot)$ be assigned with the fixed value, in general, is 0. We begin by finding the optimal solution for each state associated with the second stage by determining $f_2(\cdot)$ using the recursion equation above. Each $f_t(i)$ has only one optimal value but can have many solutions associated with it. Then, we find $f_3(\cdot)$ using the knowledge of $f_2(\cdot)$. We continue in this fashion until we have computed $f_T(\cdot)$. Then, the final optimal solutions of the problem are chosen from $\min_i f_T(i)$ (there may be many solutions).

CHAPTER IV

PROBLEM FORMULATION

There are many works on the problem of haplotype inference. These works can be separated into two categories, inference by population data with and without pedigree. We are interested in the problem of haplotype inference on a pedigree based on the minimum recombination principle. This problem can be solved by two types of methods. One is a greedy algorithm which results in approximated solutions. The other one is a global optimization which results in exact solutions. From our observation, there are two methods that have been proposed for finding the exact solution of this problem, a dynamic programming method and an integer linear programming method. This work involves the improvement of the dynamic programming method in order to reduce the computing time and make it work with the pedigree data that contain some missing alleles, and also, the inconsistent alleles.

4.1 Preliminary Definitions

This work focuses on the problem of inferring haplotypes from a pedigree with the genotype information for each member in the pedigree. The genotype information may contain some missing alleles which are the alleles that could not be specified from the genotyping process. The genotype information may contain some inconsistent alleles which are the alleles that do not inherit from the parents. For the simplification, we will define some necessary biological terms for this work as follows.

DEFINITION 4.1.1 A conventional representation of a pedigree can be transformed to a graph representation. A pedigree can be represented as a weakly connected directed acyclic graph $P = \{V, E\}$, where V is a set of nodes (vertices) and E is a set of edges. $V = M \cup F$, where M stands for the male nodes and F stands for the female nodes. The in-degree of each node will be 0 for the founder nodes (have

no parents) and 1 or 2 for the non-founder nodes. There are only two types of edges, the edges starting from one of the parent nodes to its mate node and the edges starting from one of the parent nodes to its offspring (child) nodes.

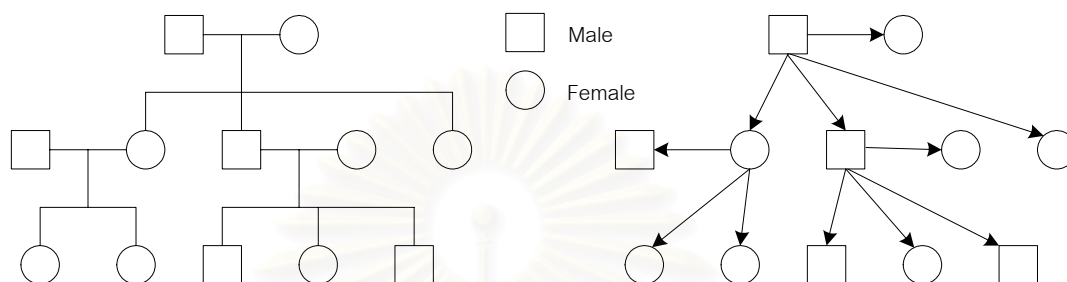


Figure 4.1 The conventional representation of a pedigree (left) and its corresponding representation as a weakly connected directed acyclic graph (right).

DEFINITION 4.1.2 For a particular pedigree, if there exists two nodes that can connect to each other by more than one path, it will be said to be a looped pedigree (the pedigree that has loops). For the graph representation, we may consider the looped pedigree as the pedigree in which there exists a node that has the in-degree equal to 2.

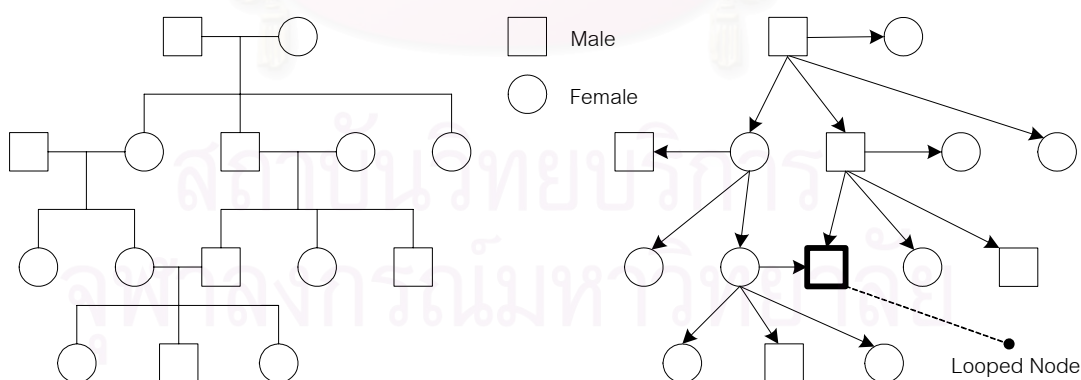


Figure 4.2 The looped pedigree represented by a conventional form (left) and a graph form (right).

DEFINITION 4.1.3 For a particular looped pedigree, a looped node is the node (the members of the pedigree) that has the in-degree equal to 2. The looped node can be assigned in different way depending on the way we build the pedigree graph.

DEFINITION 4.1.4 A haplotype of length m is a vector $H = \langle a_1, a_2, \dots, a_m \rangle$ in which a_j represents the allele at the locus j on the haplotype and $a_j \in \{0, 1, 2, \dots, k\}$ in which k is the number of possible values of allele at each locus. The value 0 of a_j is used to represent the missing alleles.

DEFINITION 4.1.5 A genotype sequence (genotype vector) of length m is a vector $G = \langle g_1, g_2, \dots, g_m \rangle$ in which g_j is a genotype at the locus j on the genotype sequence (sometime we may use the term "genotype" to refer to a genotype sequence). A genotype of locus j is an unordered pair of alleles that comes from two associated haplotypes and is represented as $g_j = (a_{j,1}, a_{j,2})$. So the genotype vector of length m may be represented by $G = \langle (a_{1,1}, a_{1,2}), (a_{2,1}, a_{2,2}), (a_{3,1}, a_{3,2}), \dots, (a_{m,1}, a_{m,2}) \rangle$ in which a genotype $(a_{j,1}, a_{j,2})$ is an unordered pair of alleles at the locus j on the genotype. For a particular genotype, each allele in the allele pair comes from one of two haplotypes that compose the genotype. For example, two haplotypes of length 4 with the value $(0, 2, 1, 2)$ and $(1, 2, 2, 1)$ can be combined into the genotype $\langle (0,1), (2,2), (2,1), (1,2) \rangle$.

DEFINITION 4.1.6 A genotype of a particular locus is said to be homozygous if its allele pair made of two identical values, otherwise it is said to be heterozygous. For example, $(1,1)$ and $(2,2)$ are homozygous while $(1,2)$ is heterozygous.

DEFINITION 4.1.7 For a particular genotype (a pair of alleles), if we can specify that which allele is the paternal allele (an allele that inherit from the father) and which one is the maternal allele (an allele that inherit from the mother), we will say that this genotype has been haplotyped.

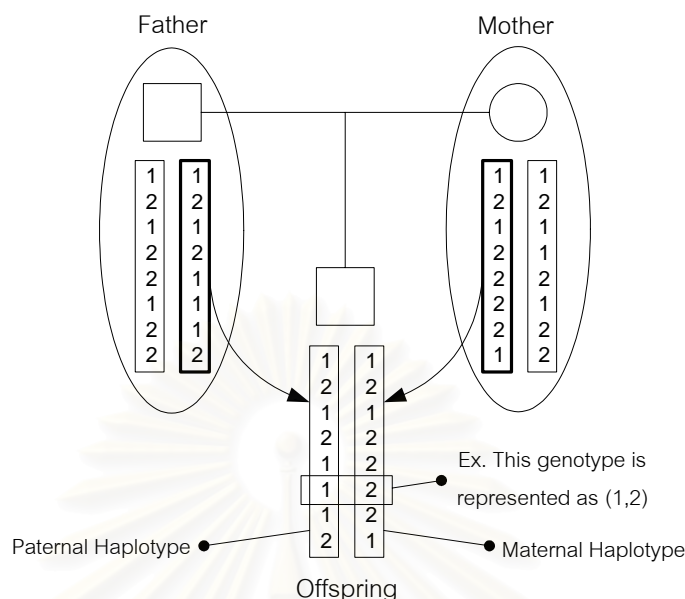


Figure 4.3 The representation of genotypes and haplotypes for each member in a pedigree.

DEFINITION 4.1.8 For a known paternal allele at a particular locus on the paternal haplotype of any offspring, an allele is said to be Mendelian consistent if it is identical to at least one of the alleles in the father's genotype at the same locus. And for a known maternal allele at a particular locus on the maternal haplotype of any offspring, an allele is said to be Mendelian consistent if it is identical to at least one of the alleles in the mother's genotype at the same locus.

DEFINITION 4.1.9 The allele that is not consistent with the Mendelian law will be referred to as an inconsistent allele (short for Mendelian inconsistent).

DEFINITION 4.1.10 For a genotype at a particular locus on the genotype sequence of any offspring, it is said to be Mendelian consistent if its paternal and maternal alleles can be assigned so that both alleles are Mendelian consistent. If the genotype is not Mendelian consistent, it will be referred to as an inconsistent genotype (short for Mendelian inconsistent).

DEFINITION 4.1.11 For a known paternal allele at a particular locus on the paternal haplotype of any offspring, a grandparent of this allele refers to the haplotype source of this allele. If this allele inherits from the paternal haplotype of the father, its grandparent will be specified as 'grandfather' denoted by the value 1 and if this allele inherits from the maternal haplotype of the father, its grandparent will be specified as 'grandmother' denoted by the value 2. This definition is also used for the maternal allele in similar way.

DEFINITION 4.1.12 A grandparent vector corresponding to a particular haplotype is represented as $S = \{v_1, v_2, \dots, v_m\}$ in which v_j is the grandparent value (haplotype source) of an allele at the locus j of the haplotype and $v_j \in \{0, 1, 2\}$ in which the value 0 denotes the unknown grandparent. For any haplotype, it must have one grandparent vector corresponding to it. When considering at any locus, we may call the grandparent of the paternal allele as the paternal grandparent and the grandparent of the maternal allele as the maternal grandparent.

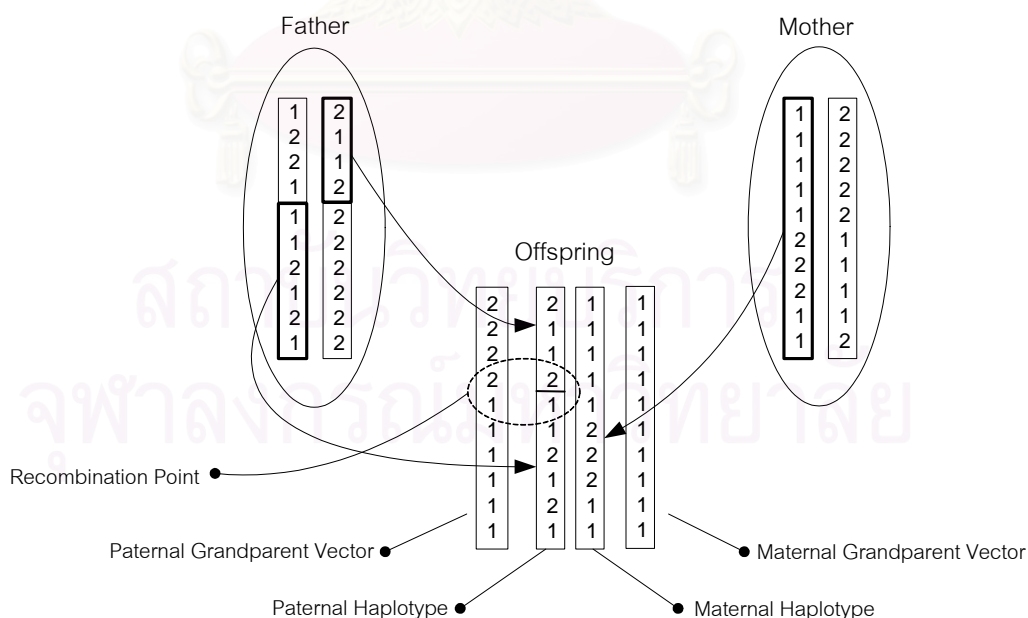


Figure 4.4 The representation of grandparent vectors corresponding to the relation between the haplotypes of the offspring and its parents.

DEFINITION 4.1.13 A haplotype is said to be a recombinant haplotype if it is a combination of parts from the paternal and maternal haplotypes of the parents. The positions in the grandparent vector in which two adjacent grandparents are different are called recombination points. The recombination point will be used for counting of recombination events.

DEFINITION 4.1.14 The term “haplotype configuration” refers to the specification of the paternal and maternal alleles for the genotype at any loci. For a particular genotype, a haplotype configuration is denoted by $h = (a^P | a^m)$ where a^P is a paternal allele and a^m is a maternal allele.

DEFINITION 4.1.15 The term “family haplotype configuration” refers to the specification of all haplotype configurations of all members in a particular family at one particular locus.

DEFINITION 4.1.16 The term “pedigree haplotype configuration” refers to the specification of all haplotype configurations of all members in the pedigree at one particular locus.

DEFINITION 4.1.17 The term “grandparent configuration” refers to the specification of the grandparents for the paternal and maternal allele at any loci. For a particular genotype with known paternal and maternal alleles, a grandparent configuration is denoted by $s = (v^P | v^m)$ where v^P is a grandparent value of the paternal allele and v^m is a grandparent value of the maternal allele.

DEFINITION 4.1.18 The term “family grandparent configuration” refers to the specification of all grandparent configurations of all offspring in a particular family at one particular locus.

DEFINITION 4.1.19 The term “pedigree grandparent configuration” refers to the specification of all grandparent configurations of all non-founder members in the pedigree at one particular locus.

4.2 The Problem Formulation

We are interested in the method of finding the solution for the MRHC problem. To infer the haplotypes from the pedigree with the genotype information for each member in the pedigree, we use the minimum recombination principle which basically says that genetic recombination is rare and thus haplotype configurations with fewer recombinants should be preferred in a haplotype inference. There are some recent works that propose the methods for solving this problem but the one that we are interested is the dynamic programming method. We will improve the dynamic programming method of Doi, Li, and Jiang (2003) so that it runs efficiently with respect to the computing time and size of data; and also, it can work with the data containing some missing alleles and with the rare occurrence of Mendelian inconsistent alleles. The formulation of our problem can be stated as follows.

INPUT: A pedigree with the genotype information (genotype vector) for each member. The input genotype vector may contain some missing alleles and also some alleles may be Mendelian inconsistent. The example of input data is shown in Figure 4.5.

OUTPUT: A set of haplotype solutions. Each haplotype solution refers to a specification of haplotype configurations for all loci of all members in the pedigree that require minimum number of recombination events. Note that, these haplotype solutions are obtained based on the set of pedigree haplotype configurations that have minimum number of inconsistent alleles. The example of one haplotype solution corresponding to the input data in Figure 4.5 is shown in Figure 4.6.

Our inference method will consist of the processes of local inferring for the haplotype and grandparent configurations, finding the set of all possible pedigree haplotype configurations that have minimum number of inconsistent alleles and also get the corresponding grandparent configurations, using dynamic programming to find the minimum-recombinant pedigree grandparent configurations, and then get the final solutions as the haplotype configurations that require minimum number of recombination events.



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

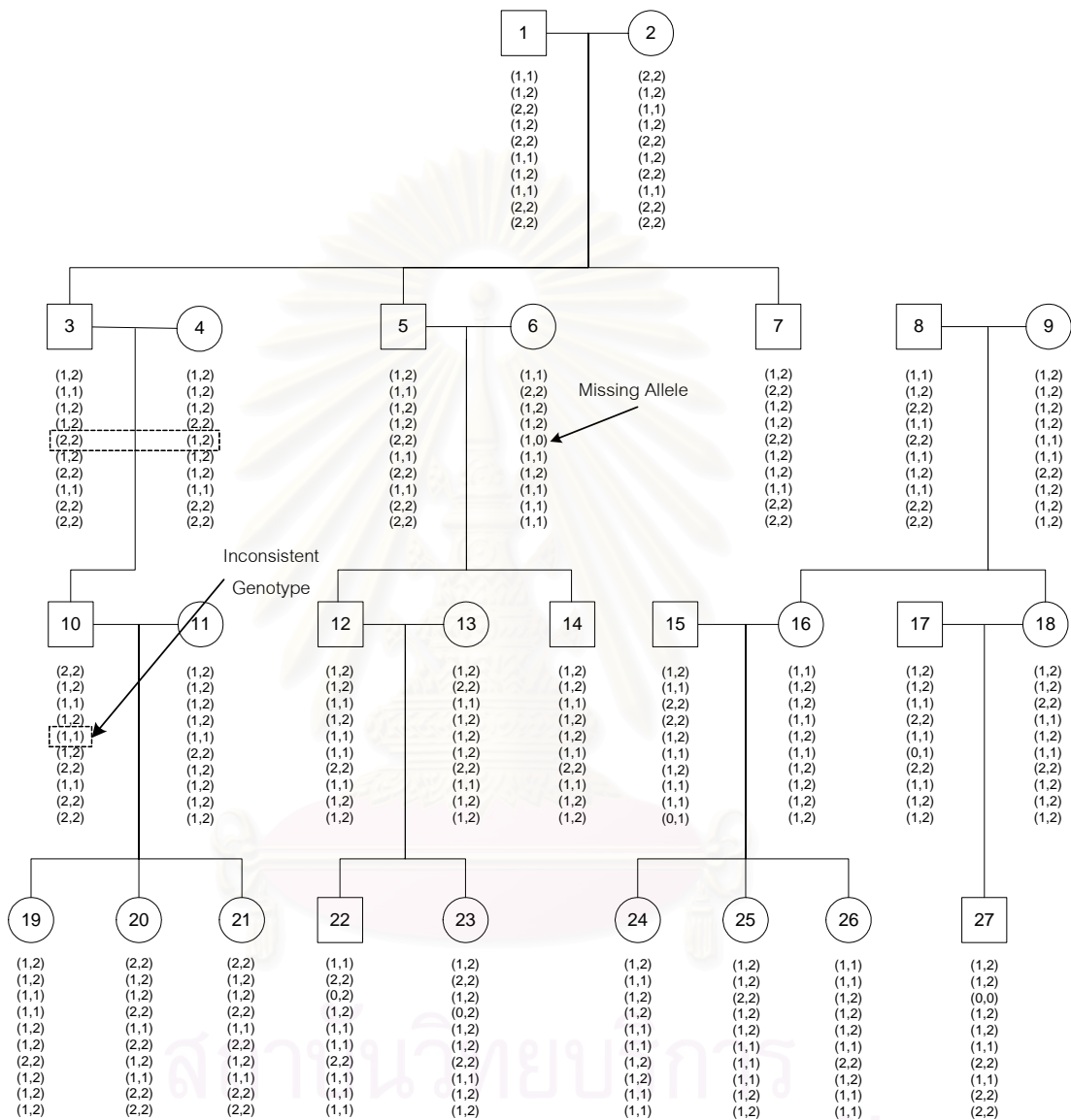


Figure 4.5 The example of an input data.

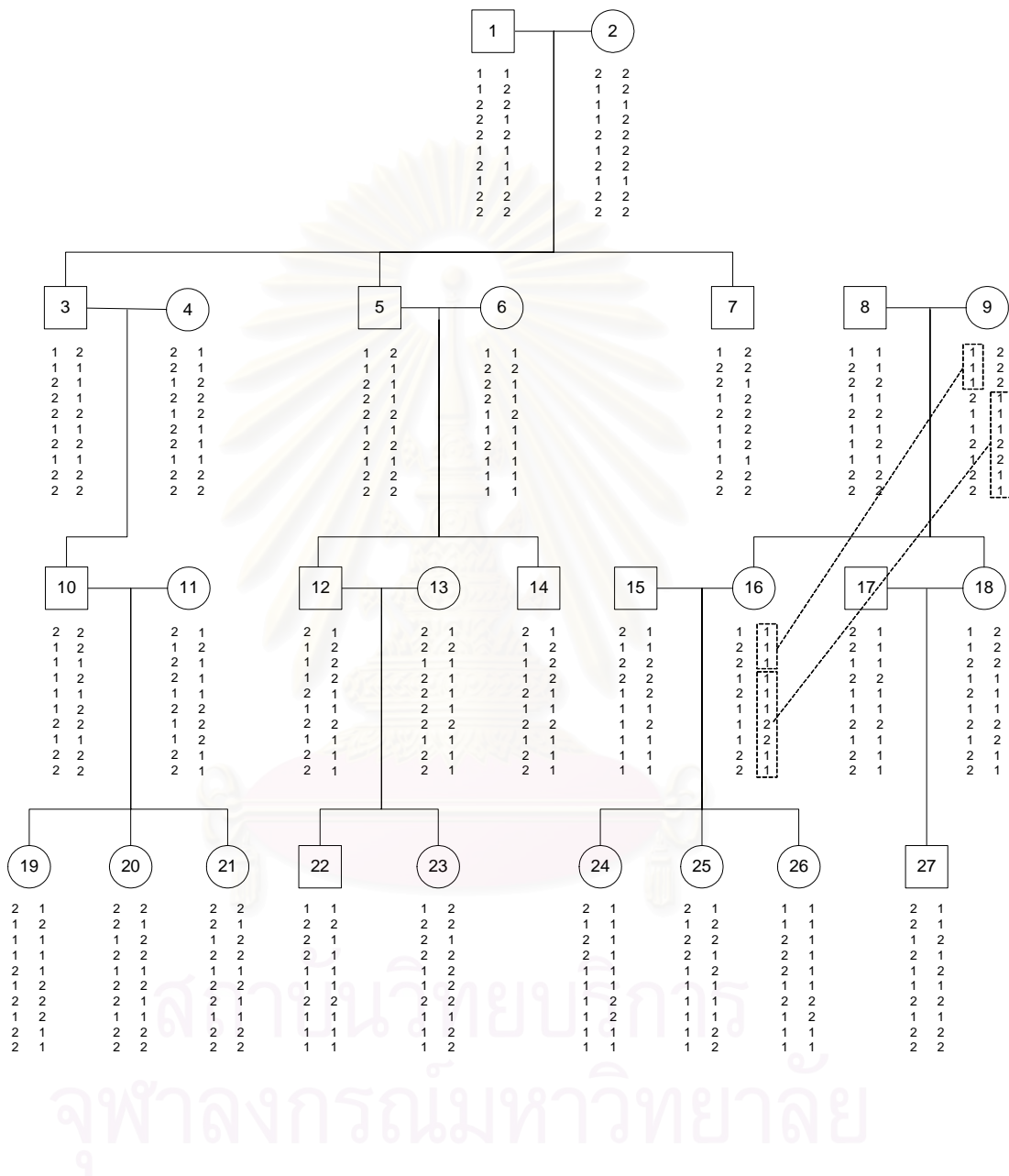


Figure 4.6 The example of a haplotype solution corresponding to the input data in Figure 4.5. This solution has one recombinant as shown in member 16.

CHAPTER V

HAPLOTYPE INFERENCE ON A PEDIGREE USING DYNAMIC PROGRAMMING

5.1 The Computational Framework

In general, finding haplotype and grandparent configurations for the pedigree that requires minimum number of recombination events consists of two main processes, the pre-inference process and the optimization process. The pre-inference process is the process of direct inferring for some haplotype or grandparent configurations based on the Mendelian law of inheritance. There are some haplotype and grandparent configurations that we can directly infer in order to make them consistent with the Mendelian law. And also, we can eliminate some possible haplotype configurations that are not consistent with the Mendelian law in the pre-inference phase. The optimization process is the process of finding the set of haplotype or grandparent configurations that requires minimum number of recombination events. Generally, the optimization methods can be separated into two categories, the methods that find the exact solutions and the methods that find the near-optimal solutions. In this work, we will use the method that finds the exact solutions, a dynamic programming.

Typically, the methods for the problem of haplotype inference on a pedigree usually suffer from the problem of missing alleles. When the genotype data contain some missing alleles, we must try to assign the values for those missing alleles based on the Mendelian law. The problem is that, when assigning each missing allele, it must be Mendelian consistent within not only its associated family but also the whole pedigree. Checking for the Mendelian consistency of the whole pedigree is quit difficult. From our observation in the previous works, we have seen only one work that can completely handle this missing allele problem. Li and Jiang (2004) proposed the method for finding the minimum-recombinant haplotype configurations with missing alleles using integer linear programming method. Their method implicitly embedded the constraints

that control the Mendelian consistency of each allele assignment. Although their method works efficiently for handling of missing alleles, it can only work in the framework of integer linear programming method.

The other problem is the problem of inconsistent alleles. From our observation in the previous works, there are no methods that handle the problem of inconsistent alleles. Those methods assume that the input genotype data must be Mendelian consistent. Our work will handle this problem. The assumption that we use to handle this problem is quite similar to the minimum recombination principle. We assume that the occurrence of inconsistent alleles is very rare, both by natural events and by genotyping errors. Then, the haplotype configuration of the pedigree should have very few or no inconsistent alleles. So we can state that the haplotype configuration that has minimum number of inconsistent alleles should be preferred in a haplotype inference. We will assume the priority of minimum-inconsistent criterion over the minimum-recombinant criterion. Our concept is to find the set of haplotype configurations that have minimum number of inconsistent alleles at first and, then, use this set of haplotype configurations to find the final solutions with the minimum-recombinant criterion.

Both solving for the haplotype configurations with minimum number of inconsistent alleles and for the haplotype configurations with minimum number of recombinants are optimization problems which will be solved using a dynamic programming method. For the case of solving for the minimum-recombinant haplotype configurations, our method will not directly solve it similar to the previous work. Instead, we will first solve for the minimum-recombinant grandparent configurations; and then, use the resulted solutions to infer the corresponding haplotype configurations later. This modification is aimed to reduce the computing time.

The computational framework of our method can be represented by the process flow as shown in Figure 5.1.

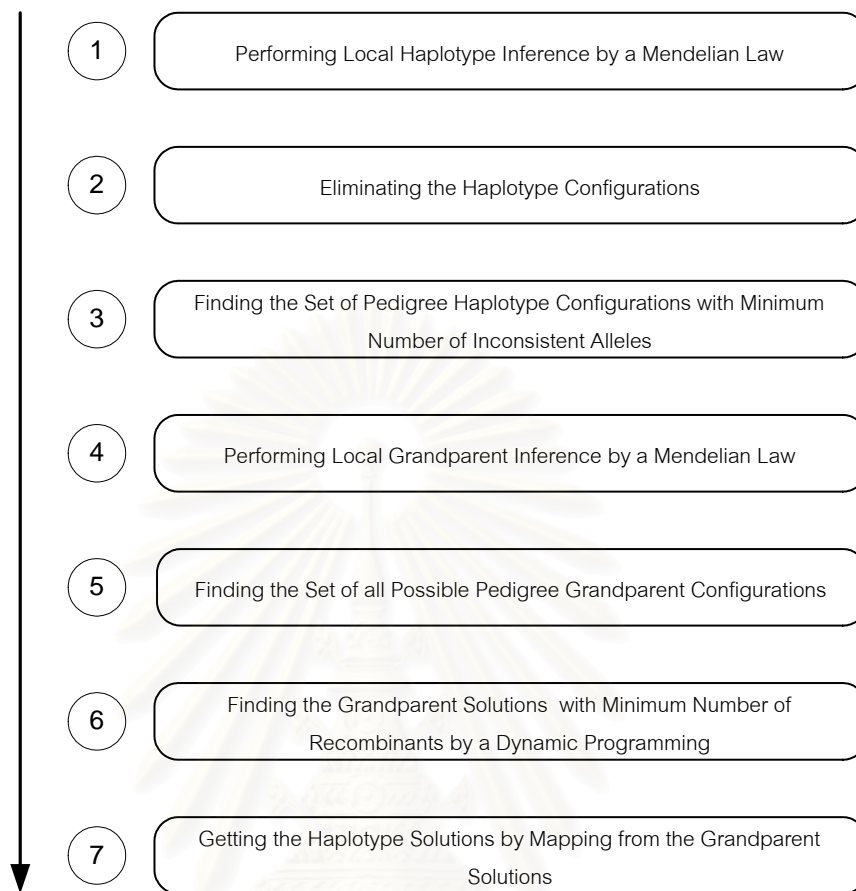


Figure 5.1 The inference process.

5.2 Local Haplotype Inference by a Mendelian Law

From the genotype data of the pedigree, there are usually some alleles that we can directly infer based on a Mendelian law. Local haplotype inference refers to the process of direct inferring the paternal and maternal alleles from a genotype at a particular locus of a particular trio (father, mother, and child) regardless of other members in a pedigree. The local haplotype inference concerns only the inferring that will not affect the optimal criterion of a whole pedigree. The process of local haplotype inference will be performed as the first phase of the framework. This includes imputation of missing alleles and the specification for the paternal and maternal alleles. The concept used for the local haplotype inference is to choose the configuration that is

Mendelian consistent or if the consistency is not possible, chooses the one that has fewer inconsistent alleles.

After we have known some paternal or maternal alleles, we could directly infer the grandparents for some of these known paternal or maternal alleles. Since in the later phases, there may be some more information for inferring the grandparents thus we will perform the inference of the grandparents once in the later phase. The detail of local grandparent inference will be described in the later section.

The local haplotype inference can be performed only for the members that have parents because the inference rules depend on the information of the parents. We use the letter F to represent a father, M to represent a mother, and O to represent an offspring. The inference will be performed by considering at one locus at a time. For each inference, the genotype information of each member in the trio will be used. Here, each member will be represented in the forms $F\{information\}$ for a father, $M\{information\}$ for a mother, and $O\{information\}$ for an offspring. The part $\{information\}$ represents the characteristic of the genotype. The detail of genotype information is listed in Table 5.1.

Table 5.1 The detail of genotype information.

Information	Description
(a_1, a_2)	The allele pair of the genotype (here, if the value of a_k is 0, it means the missing allele).
$(a_1 a_2)$	The genotype of which the paternal and maternal alleles have been specified already (a_1 represents the paternal allele and a_2 represents the maternal allele).
a^f	The allele value of the father's genotype (used for the case that the father's genotype is homozygous).
a^m	The allele value of the mother's genotype (used for the case that the mother's genotype is homozygous).

Information	Description
A and B	Two allele values where $A \neq B$
$homo$	The genotype is homozygous.
$\sim homo$	The genotype is not homozygous (it is heterozygous or has some missing alleles).
$homo = A$	The genotype is homozygous and the values of both alleles are A .
$homo \neq A$	The genotype is homozygous and the value of both alleles are not equal to A .
$A \in (.,.)$	At least one of the alleles of the genotype has the value A .

Imputing an allele on the member that has some offspring may affect the consistency in its offspring's genotypes. To consider the consistencies for whole pedigree is out of the scope of this local inference process. So we will ignore the imputation of missing alleles for the member that has some offspring, and only the specifications for paternal and maternal alleles will be performed for these members. The local haplotype inference rules are set for two cases. A first set of rules is for the members that have some offspring and the second set is for the members that have no offspring (the members in both cases must have parents). Two sets of inference rules are shown in Table 5.2 and Table 5.3.

Table 5.2 The local haplotype inference rules for a particular genotype of the members having some offspring.

Condition on genotypes	Resulted Genotype
Case $O\{(A, B)\}$ - The genotype has no missing allele and is heterozygous.	
1. $F\{homo \neq B\}$ AND $M\{B \in (.,.)\}$	$O\{(A B)\}$
2. $F\{homo \neq A\}$ AND $M\{A \in (.,.)\}$	$O\{(B A)\}$
3. $F\{A \in (.,.)\}$ AND $M\{homo \neq A\}$	$O\{(A B)\}$

Condition on genotypes	Resulted Genotype
4. $F\{B \in (\cdot, \cdot)\}$ AND $M\{homo \neq B\}$	$O\{(B A)\}$
Case $O\{(A,0)\}$ - The genotype has one missing allele.	
5. $F\{homo=A\}$	$O\{(A 0)\}$
6. $M\{homo=A\}$	$O\{(0 A)\}$
7. $F\{homo \neq A\}$ AND $M\{\sim homo \text{ and } A \in (\cdot, \cdot)\}$	$O\{(0 A)\}$
8. $F\{\sim homo \text{ and } A \in (\cdot, \cdot)\}$ AND $M\{homo \neq A\}$	$O\{(A 0)\}$

Table 5.3 The local haplotype inference rules for a particular genotype of the members having no offspring.

Condition on genotypes	Resulted Genotype
Case $O\{(A,B)\}$ - The genotype has no missing allele and is heterozygous.	
1. $F\{homo \neq B\}$ AND $M\{B \in (\cdot, \cdot)\}$	$O\{(A B)\}$
2. $F\{homo \neq A\}$ AND $M\{A \in (\cdot, \cdot)\}$	$O\{(B A)\}$
3. $F\{A \in (\cdot, \cdot)\}$ AND $M\{homo \neq A\}$	$O\{(A B)\}$
4. $F\{B \in (\cdot, \cdot)\}$ AND $M\{homo \neq B\}$	$O\{(B A)\}$
Case $O\{(A,0)\}$ - The genotype has one missing allele.	
5. $F\{homo=A\}$ AND $M\{\sim homo\}$	$O\{(A 0)\}$
6. $F\{\sim homo\}$ AND $M\{homo=A\}$	$O\{(0 A)\}$
7. $F\{homo=A\}$ AND $M\{homo\}$	$O\{(A a^m)\}$
8. $F\{homo\}$ AND $M\{homo=A\}$	$O\{(a^f A)\}$
9. $F\{homo \neq A\}$ AND $M\{\sim homo \text{ and } A \in (\cdot, \cdot)\}$	$O\{(a^f A)\}$
10. $F\{\sim homo \text{ and } A \in (\cdot, \cdot)\}$ AND $M\{homo \neq A\}$	$O\{(A a^m)\}$

Condition on genotypes	Resulted Genotype
Case $O\{(A 0)\}$ - The paternal allele has been specified but the maternal allele is not known.	
11. $M\{homo\}$	$O\{(A a^m)\}$
Case $O\{(0 A)\}$ - The maternal allele has been specified but the paternal allele is not known.	
12. $F\{homo\}$	$O\{(a^f A)\}$
Case $O\{(0,0)\}$ - The genotype has two missing alleles.	
13. $F\{homo\}$ AND $M\{homo\}$	$O\{(a^f a^m)\}$
14. $F\{homo\}$ AND $M\{\sim homo\}$	$O\{(a^f 0)\}$
15. $F\{\sim homo\}$ AND $M\{homo\}$	$O\{(0 a^m)\}$

5.3 Detection of Looped Nodes

A pedigree can possibly have loops (the looped pedigree with less than 30 members usually has only one loop). If the given pedigree contains a loop, some computational phases can not be computed directly. This problem should be handled by fixing the looped nodes with specific computational values. So it is necessary that the looped nodes must be specified first.

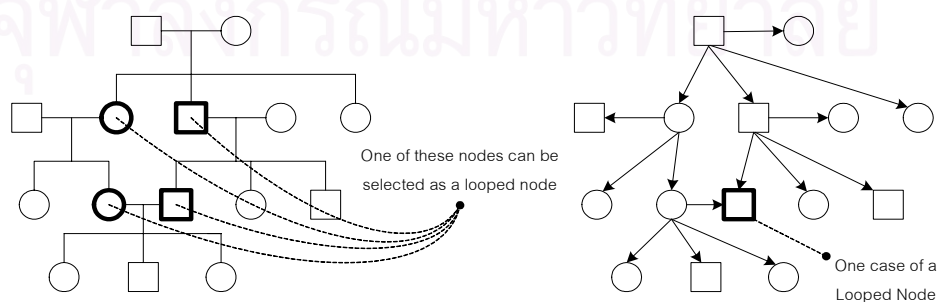


Figure 5.2 The looped pedigree and the corresponding looped nodes.

A particular loop in a pedigree may have more choices of node to be assigned as a looped node. But for our computational purpose, we need only one looped node per loop. Detecting the looped node is a first task to be performed before sending the looped pedigree to the inference process. The algorithm used for detecting the looped nodes is the algorithm for building the directed acyclic graph corresponding to the looped pedigree. The nodes that have the in-degree equal to two will be identified as the looped nodes. The resulted looped nodes may vary, depending on the order or step of a graph building process.

The Algorithm for Detecting the Looped Nodes

Part 1: Set the flags for each node to present that each node has not been checked yet. Then, randomly select the starting node from the first generation of the pedigree. Next, set this starting node to be a parent node and perform part 2 with this node.

Part 2: Consider a specific node. If this node is a parent node, perform part A. Otherwise (this node is an offspring node), perform part B.

Part A: Consider a specific parent node.

1. Consider the mate node. If it has not been checked then set it to be 'checked'. Otherwise (it has already been checked), if it has not been set as the looped node then set it to be the looped node.

2. Consider each offspring node. If it has not been checked then set it to be 'checked'. Otherwise (it has already been checked), if it has not been set as the looped node then set it to be the looped node.

3. If the mate node has not been set as the looped node and it has parents, then, set it to be an offspring node and perform part 2 with this mate node.

4. Consider each offspring node. If it has not been set as the looped node and it has some offspring, then, set it to be a parent node and perform part 2 with this offspring node.

Part B: Consider a specific offspring node.

1. Consider each parent node (father node and mother node). If it has not been checked then set it to be 'checked'. Otherwise (it has already been checked), if it has not been set as the looped node then set it to be the looped node.

2. Consider each sibling node (the brother or sister of this node). If it has not been checked then set it to be 'checked'. Otherwise (it has already been checked), if it has not been set as the looped node then set it to be the looped node.

3. Consider each parent node. If it has not been set as the looped node and it has parents, then, set it to be an offspring node and perform part 2 with this mate node.

4. Consider each sibling node. If it has not been set as the looped node and it has some offspring, then, set it to be a parent node and perform part 2 with this offspring node.

After the algorithm for detecting the looped nodes has been performed, all the looped nodes will be specified (one looped node per each loop). These looped nodes will be used to separate the looped pedigree into the corresponding non-looped pedigrees.

5.4 The Pedigree Tree and the Sub-Rooted Nodes

In some computational phases, we must perform computations by traveling along each node in the pedigree. To do this, we transform the pedigree into a tree structure. For the case of non-looped pedigree, the representation of a pedigree as a directed acyclic graph is already a tree structure. But for the case of looped pedigree, it cannot be directly transformed to a tree structure. In this case, the looped nodes must be duplicated into two identical nodes in order to preserve the property of a tree structure as in Figure 5.3. The detail of breaking the loops by duplicating the looped nodes will be considered in the later section.

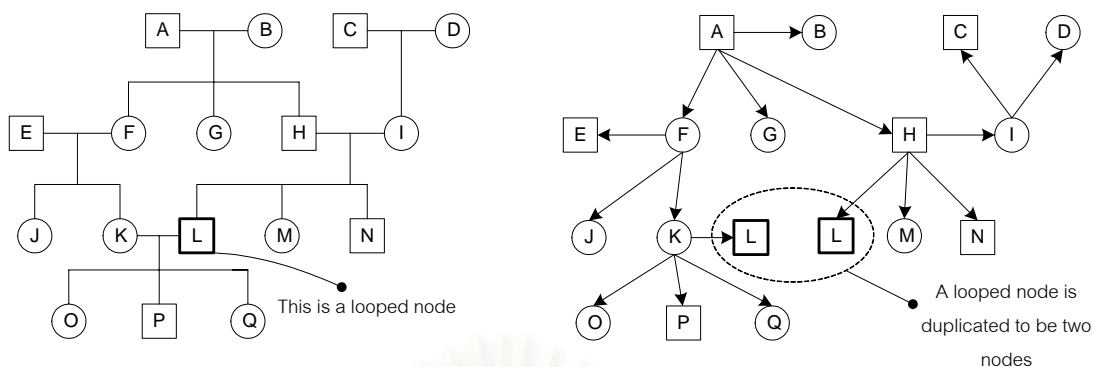


Figure 5.3 The looped pedigree and its corresponding tree structure.

To traverse a pedigree tree, we will go through the sub-rooted nodes. Here, the term “sub-rooted node” refers to the node that is not a branch node of the tree. When we perform the operation on a particular family, the sub-rooted nodes will be used as the reference nodes for each family. Figure 5.4 shows the example of the sub-rooted nodes.

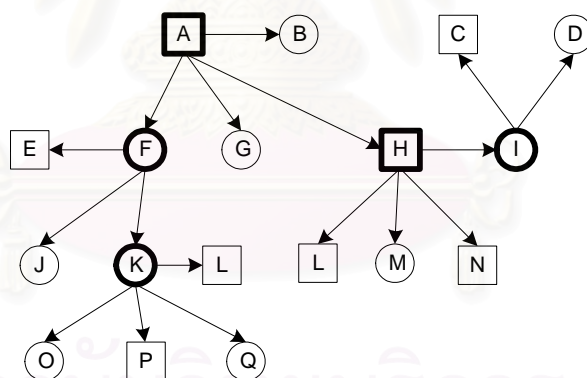


Figure 5.4 The example for the sub-rooted nodes of a pedigree. In this pedigree, node A, F, K, H, and I are sub-rooted nodes.

5.5 Elimination of the Haplotype Configurations

O’Connell and Weeks (1999) proposed the optimal algorithm for automatic genotype elimination which was used to eliminate all genotypes that led to a Mendelian inconsistency. Their algorithm is the extension of the original Lange-Goradia

algorithm which is not optimal for a pedigree with loops. O'Connell and Weeks improved the Lange-Goradia algorithm so that it finds the optimal solution for a pedigree with loops. Their idea is to break each loop by duplicating a looped-breaker node (one looped node of a particular loop) so that the pedigrees corresponding to each possible genotype of the looped-breaker nodes will be created as a non-looped pedigree and, then, separately performing the calculation for each non-looped pedigree. The obtained results are merged together.

Elimination of the haplotype configurations is the process that is used to eliminate all haplotype configurations leading to Mendelian inconsistency. Our algorithm for elimination of the haplotype configurations is based on the algorithm of O'Connell and Weeks but has an extension for handling the data containing some Mendelian inconsistent alleles. In fact, in the algorithm of O'Connell and Weeks, the genotype is considered as an ordered genotype which is identical to our term, haplotype configuration.

5.5.1 The Algorithm for Eliminating the Haplotype Configurations for a Non-looped Pedigree

Consider at one particular locus, the algorithm for eliminating the haplotype configurations for the non-looped pedigree can be described as follows.

Part 1: List all possible haplotype configurations for each member in the pedigree. This includes setting of all possible values for the missing alleles. For example, let's consider the genotype $g=(1,0)$ in which the possible values of the missing allele can be either 1 or 2. The possible haplotype configurations of this genotype can be listed as $h \in \{(1|1), (1|2), (2|1)\}$. Next, perform part 2.

Part 2: For each family (normally, consider in bottom-up order), perform the following steps. Then, perform part 3.

1. Consider each father-mother haplotype configuration pair.
 - a. If each offspring has at least one consistent haplotype configuration (the haplotype configuration that can inherit from this father-mother haplotype configuration pair) in the list, then save the haplotype configurations of the

parents from this father-mother haplotype configuration pair and also save those consistent haplotype configurations of each offspring.

b. Otherwise, take no action (there are some offspring that has none of the consistent haplotype configuration with this father-mother haplotype configuration pair).

2. For each member of this family, exclude the haplotype configurations that have not been saved during step 1. If there are no haplotype configurations have been saved for this family, it means that there are some Mendelian inconsistent alleles in this family. For this case, no haplotype configurations should be excluded. So keep all haplotype configurations of each member in the family.

Part 3: Repeat part 2 until no more haplotype configurations can be excluded.

After the elimination algorithm above has been performed, there should be only those haplotype configurations that can possibly be assigned in any particular ways to make the whole pedigree consistent. But if there are some Mendelian inconsistent alleles, there may be some invalid haplotype configurations left in the eliminated haplotype configuration set.

5.5.2 The Problem of the Looped Pedigree

The concept of the haplotype elimination algorithm in Section 5.5.1 can be stated as follows. For a particular parent, if it is assigned with a valid haplotypes configuration, the other members in its family must be assigned with some haplotype configurations so that all haplotype configurations are consistent. Using this concept for a looped pedigree may not result in the optimal solution. For example, let's consider the looped pedigree in Figure 5.5. The current haplotype configuration lists of each member are shown in the figure. If we perform the elimination process on this pedigree using the algorithm above, there will be no haplotype configurations to be excluded. Let's consider the case that B is assigned with (1|1). Following, both C and D will be assigned with (1|1) in order to be consistent. But when both C and D are assigned with (1|1), the haplotype configurations of the trio C, D, and F will be inconsistent. This means that, in

fact, the haplotype configuration (1|1) of the member B should be eliminated. The algorithm above fails to eliminate the haplotype configuration (1|1) of B because it checks the consistency for each family independently but not check the consistency of all families simultaneously. So, as in this example, it cannot know that C and D must not be assigned as (1|1), simultaneously.

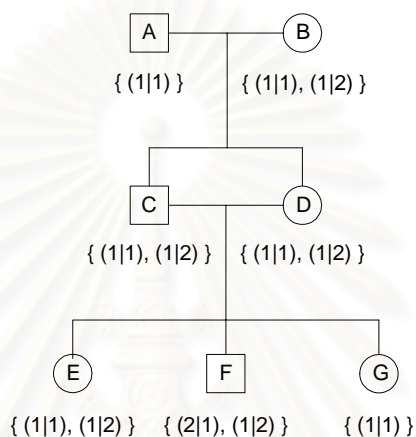


Figure 5.5 The example looped pedigree with the sets of possible haplotype configurations of each member.

To consider this problem in a larger pedigree, let's see Figure 5.6. Assume that the initial haplotype configuration lists of both F and G are $\{(1|1), (1|2)\}$.

First, consider the pedigree with no loops in Figure 5.6. The elimination process is performed as follows.

Step 1: Find the valid haplotype configurations for the family of A, B, E, and F. After this step, assume that the haplotype configuration (1|1) and (1|2) of F are still valid.

Step 2: Find the valid haplotype configurations for the family of F, G, J, K, and L. After this step, assume that G has a valid haplotype configuration (1|1) corresponding to the haplotype configuration (1|2) of F (the haplotype configuration of the parents are considered as a pair, each one from the father and mother), and also,

has a valid haplotype configuration (1|2) corresponding to the haplotype configurations (1|1) of F.

Step 3: Find the valid haplotype configurations for the family of C, D, G, H, and I. After this step, assume that the haplotype configuration (1|1) and (1|2) of G are still valid.

Let's consider the result. If the family of C, D, G, H, and I are assigned with some valid haplotype configurations that forces G to be assigned with the haplotype configuration (1|1), F will be forced to be assigned with haplotype configuration (1|2). Consequently, the member A, B, E, J, K, and L could be also assigned with some valid haplotype configurations. In general, the elimination process for the pedigree with no loops must result in the optimal solution.

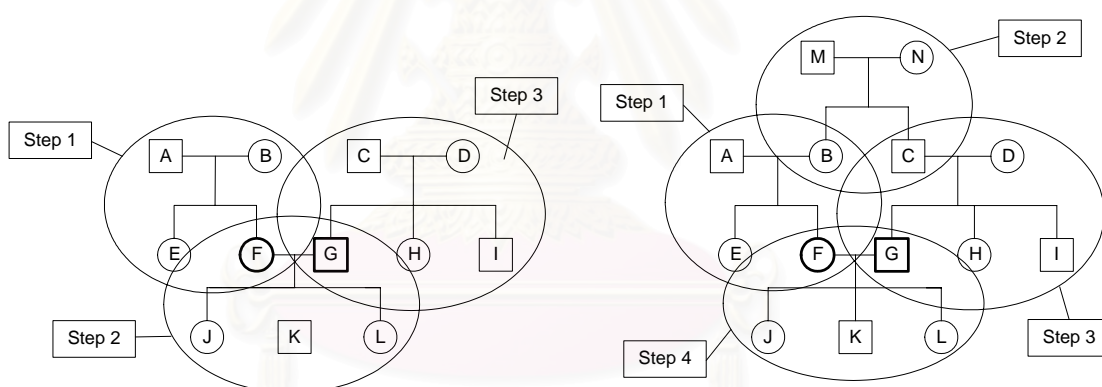


Figure 5.6 The performing steps of the haplotype configuration elimination process for the non-looped pedigree (left) and the pedigree with one loop (right).

Now, let's consider the case of a looped pedigree in Figure 5.6. Assume that the initial haplotype configuration lists of both F and G are $\{ (1|1), (1|2) \}$. The elimination process is performed as follows.

Step 1: Find the valid haplotype configurations for the family of A, B, E, and F. After this step, assume that the haplotype configuration (1|1) and (1|2) of F are still valid and there exists some valid haplotype configurations for B.

Step 2: Find the valid haplotype configurations for the family of M, N, B, and C corresponding to the valid haplotype configurations of B. After this step, assume that C has some valid haplotype configurations.

Step 3: Find the valid haplotype configurations for the family of C, D, G, H, and I corresponding to the valid haplotype configurations of C. After the third step, assume that G has valid haplotype configurations as (1|1) and (1|2). And assume that M has one valid haplotype configuration as (1|1) which is valid only if F is assigned with (1|1) and G is assigned with (1|1).

Step 4: Find the valid haplotype configurations for the family of F, G, J, K, and L corresponding to the valid haplotype configurations of G. After this step, assume that J, K, and L have some valid haplotype configurations corresponding to the haplotype configuration pair of its parents which are the pair of F(1|1)/G(1|2) and the pair of F(1|2)/G(1,1). This means that the haplotype configurations, (1|1) and (1|2), of both F and G are still valid.

There is a problem at the valid haplotype configuration (1|1) of M which is assumed to be valid only if F is assigned with (1|1) and G is assigned with (1|1). But there is no valid haplotype configuration of the offspring of F and G that is valid with the haplotype configuration pair F(1|1)/G(1|1). This means that the haplotype configuration (1|1) of M should be eliminated. For a looped pedigree, this problem can occur when some valid haplotype configurations in one particular family require the haplotype configuration pairs in the other family that are not valid for that family.

5.5.3 The Elimination Process

To avoid the problem from the loop, the original looped pedigree will be separated into non-looped pedigrees. In each non-looped pedigree, the looped nodes will be assigned with one fixed haplotype configuration (see the example in Figure 5.7). The number of the corresponding non-looped pedigrees will be equal to the possible number of the combinations of all possible haplotype configurations of each looped

node. So, the algorithm above (in Section 5.5.1) will be performed at each non-looped pedigree. After that, the results from each non-looped pedigree will be merged together to get the final result for the original looped pedigree.

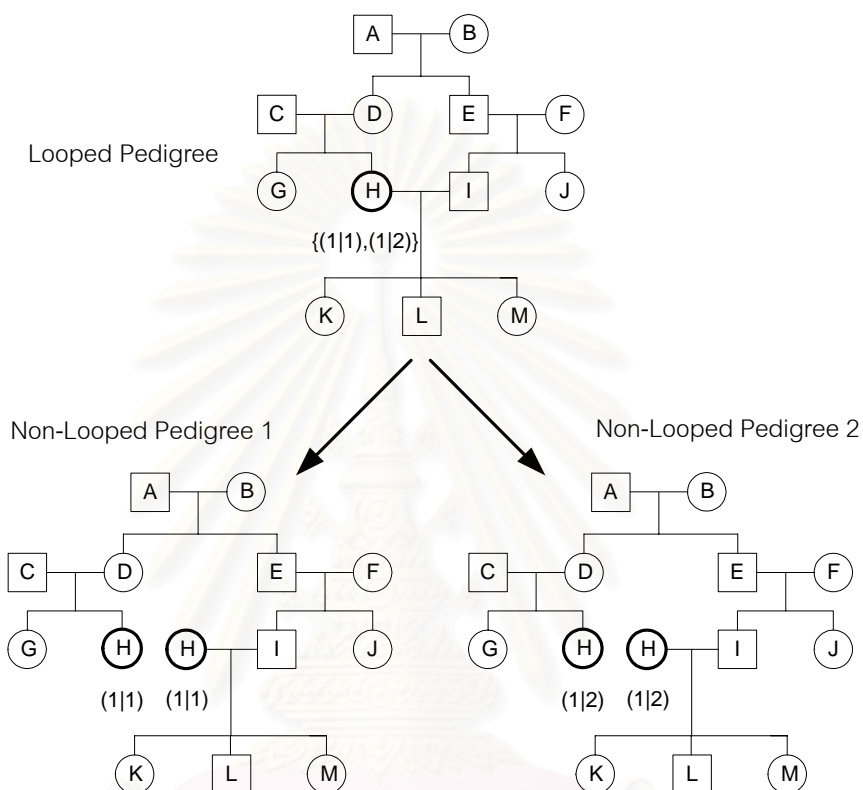


Figure 5.7 The separation of the looped pedigree into non-looped pedigrees.

When perform the elimination process at each non-looped pedigree, some families may have no valid haplotype configurations which may cause from the occurrence of Mendelian inconsistent alleles or the invalid of the haplotype configuration of the looped node. Assume that a given looped pedigree has two corresponding non-looped pedigrees. In the first non-looped pedigree, there is one family that has no valid haplotype configurations, and so, none of its haplotype configurations are eliminated. In the second non-looped pedigree, some of the haplotype configurations of this family are eliminated. After merging process, none of the haplotype configurations of this family are eliminated. In fact, the haplotype configurations of this family should be eliminated.

Then, it is necessary that the elimination process must be performed again. We will perform the elimination process in the original looped pedigree at first and, then, separately perform for each non-looped pedigrees again. Finally, the results from each non-looped pedigree will be merged together into the original looped pedigree as the final result.

5.6 Finding the Set of Pedigree Haplotype Configurations with Minimum Number of Inconsistent Alleles

In order to find the pedigree haplotype configurations that have minimum number of recombinants, first, we must find the set of all possible pedigree haplotype configurations that are consistent with the Mendelian law. But in this work, the pedigree can contain some inconsistent alleles which make it impossible to find consistent haplotype configurations for some loci. Then, the problem will be relaxed to finding the set of all possible pedigree haplotype configurations that have minimum number of inconsistent alleles. This method is based on the assumption that the inconsistent alleles usually occur rarely in the observed pedigrees.

The pedigree haplotype configuration refers to one set of the haplotype configurations of all members in the pedigree at one particular locus. The consistency of the haplotype configurations in one locus does not depend on the configurations of the other loci. Therefore, the consistency of the haplotype configurations in a pedigree can be considered separately by each locus at a time. Finding the set of all possible pedigree haplotype configurations that have minimum number of inconsistent alleles is performed one locus at a time.

The method that is used for finding the set of all possible pedigree haplotype configurations that have minimum number of inconsistent alleles is a dynamic programming. The concept is to divide the problem into smaller problems as subpedigrees. The consistency will be considered at the unit of a family. To perform a dynamic programming by subpedigree, the pedigree must not contain loops. Then, if

the pedigree has loops, it will be separately considered by the corresponding non-looped pedigrees which can be created by the same method as described in Section 5.5. At first, the pedigree will be transformed into a tree structure and the computation of each family will be considered at each sub-rooted node. A detailed description of the algorithm for finding the set of all possible pedigree haplotype configurations that have minimum number of inconsistent alleles at one particular locus is given below.

Firstly, assume that the pedigree has already been transformed into a tree structure and the sub-rooted nodes have been specified already. If the pedigree has loops then it will be separately considered by each corresponding non-looped pedigree. And the final result will be computed by merging from the results of each non-looped pedigree. The computation will be performed from the smallest subpedigrees and extend to the larger subpedigrees until reaching the root of the pedigree. The process of computing the minimum number of inconsistent alleles will be done at each sub-rooted node by postorder traversing in the pedigree tree. And the process of enumerating all optimal pedigree haplotype configurations will be done at each sub-rooted node by preorder traversing in the pedigree tree. See Figure 5.8 for the example of the preorder and postorder traversals

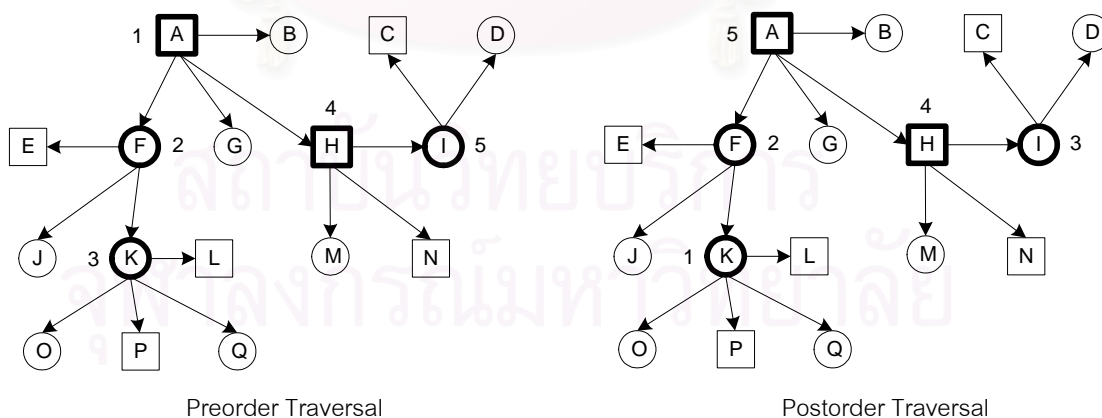


Figure 5.8 The pedigrees and their corresponding sub-rooted nodes. The highlighted nodes are the sub-rooted nodes with the numbers that represent the traversal orders.

We denote some notations used in the algorithm as follows.

R_i denotes the sub-rooted node of index i where $i \in \{1, 2, \dots, m\}$ and m is the number of all sub-rooted nodes in the pedigree.

$h_{i,t}$ denotes the haplotype configuration of index t of R_i where $t \in \{1, 2, \dots, p_i\}$ and p_i is the number of all possible haplotype configurations of R_i .

C denotes a pedigree haplotype configuration of a particular locus. The pedigree haplotype configuration keeps the specification of the family haplotype configurations for each sub-rooted node in the pedigree.

The term “family of R_i ” refers to the family in which each member in the family connects to R_i by a directed edge that has the beginning side at R_i (R_i may be a parent or an offspring of the family).

The term “optimal” is used in the meaning that the subpedigree of the current considered sub-rooted node with a specific configuration has minimum number of inconsistent alleles.

Using the notations above, the algorithm for finding the set of all possible pedigree haplotype configurations that have minimum number of inconsistent alleles at one particular locus can be described as follows.

The Algorithm for Finding the Set of All Possible Pedigree Haplotype Configurations with Minimum Number of Inconsistent Alleles

Part A: Beginning at the rooted node of the pedigree, perform part B with this rooted node. Then, perform part C.

Part B: Consider a specific sub-rooted node R_i .

1. If there are any sub-rooted nodes in the family of R_i , perform part B for each of these sub-rooted nodes.

2. For each haplotype configuration ($h_{i,t}$) of R_i , specify the minimum number of inconsistent alleles from the optimal pedigree haplotype configurations of the subpedigree in which R_i is the root and has the haplotype configuration as $h_{i,t}$ (more detail of this process will be described later).

Part C: Perform backtracking to get all optimal pedigree haplotype configurations. The backtracking process can be described as follows.

1. Remove the haplotype configurations that are not associated with any optimal configurations by performing part 1-A. This process will be done by preorder traversing on each sub-rooted node in the pedigree tree.

Part 1-A: At the rooted node of a pedigree, remove the haplotype configurations of this rooted node that are not the optimal haplotype configurations. Then, perform part 1-B for each of these optimal haplotype configurations.

Part 1-B: Consider a specific haplotype configuration $h_{i,t}$ of a specific sub-rooted node R_i .

1. Remove the family haplotype configurations associated with $h_{i,t}$ that are not the optimal family haplotype configurations for $h_{i,t}$.

2. For each optimal family haplotype configuration, save the corresponding haplotype configurations for each sub-rooted node in the family of R_i .

3. For each sub-rooted node in the family of R_i , exclude the haplotype configurations that are not saved in step 2 above, then, perform part 1-B for each haplotype configuration of this sub-rooted node.

(If there is no sub-rooted node in the family of R_i , ignore step 2 and 3 above.)

2. Enumerate all optimal pedigree haplotype configurations by performing path 2-A. This process will be done by preorder traversing on each sub-rooted node in the pedigree tree. Firstly, the list of sub-rooted nodes will be specified and each node will have the index following the traversing order (preorder traversal). At each sub-rooted node, the branching paths used for enumeration are generated from the haplotype configurations and their associated family haplotype configurations. Each family haplotype configuration will be used for assigning the haplotype configuration of each member in the family.

Part 2-A: At the rooted node of a pedigree (the sub-rooted node with the first index), initialize the pedigree haplotype configuration C for each haplotype

configuration of this rooted node and perform part 2-B for each of these haplotype configurations with their pedigree haplotype configuration C (now there are no assignments in C).

Part 2-B: Consider a specific haplotype configuration $h_{i,t}$ of a specific sub-rooted node R_i and a pedigree haplotype configuration C . Perform part 2-C for each family haplotype configuration associated with $h_{i,t}$.

Part 2-C: Consider a specific family haplotype configuration of a specific sub-rooted node R_i and a pedigree haplotype configuration C . Assign this family haplotype configuration with this sub-rooted node R_i in C . Then perform step a or b below.

a. If i is the last index in the list of the sub-rooted nodes, it means that all sub-rooted nodes in C has already been assigned with a specific family haplotype configuration. In this case save C in the list of optimal pedigree haplotype configurations.

b. Otherwise, perform part 2-B for the sub-rooted node of the next index with its haplotype configuration corresponding to the associated family haplotype configuration that has been previously specified in C (by the preorder traversing, it is certain that this associated family haplotype configuration has been specified already in C) and with this current pedigree haplotype configuration C .

For a particular sub-rooted node R_i with a specific family haplotype configuration $f_{i,t,c}$, the minimum number of inconsistent alleles of the subpedigree in which R_i is the root and has the family haplotype configuration as $f_{i,t,c}$ can be computed as follows.

$$\begin{aligned} \text{MinNumF}(R_i, f_{i,t,c}) = & \sum_{s \in U_i} \text{MinNumH}(R_s, \text{Hap}(R_s, f_{i,t,c})) + \\ & \sum_{1 \leq l \leq q_i} \text{NumTrio}(\text{Hap}(F_i, f_{i,t,c}), \text{Hap}(M_i, f_{i,t,c}), \text{Hap}(O_{i,l}, f_{i,t,c})) \end{aligned}$$

$f_{i,t,c}$ denotes a family haplotype configuration of index c where $c \in \{1, 2, \dots, d_{i,t}\}$ and $d_{i,t}$ is the number of all possible family haplotype configurations associated with $h_{i,t}$.

R_s denotes the sub-rooted node of index s .

U_i denotes the set of the indices of the sub-rooted nodes in the family of R_i (excluding the node R_i).

q_i denotes the number of the offspring in the family of R_i .

F_i denotes the father node in the family of R_i

M_i denotes the mother node in the family of R_i

$O_{i,l}$ denotes the offspring node of index l in the family of R_i

The function $Hap(R_s, f_{i,t,c})$ returns the haplotype configuration of R_s corresponding to the family haplotype $f_{i,t,c}$.

The function $NumTrio(h^F, h^M, h^O)$ returns the number of inconsistent alleles computed from the haplotype configurations of each member in the trio composed of the father (h^F), the mother (h^M), and the offspring (h^O). The function $MinNumH(R_s, h_{s,t})$ returns the minimum number of inconsistent alleles of the subpedigree in which R_s is the root and has the specific haplotype configuration as $h_{s,t}$. This function can be computed as follows.

$$MinNumH(R_s, h_{s,t}) = \min_c (MinNumF(R_s, f_{s,t,c}))$$

Consider the pedigree in Figure 5.9. Each sub-rooted node is assigned with the index obtained from the postorder traversing. On the left, two families are shown with the possible haplotype configurations of each node. From this pedigree, the example of the computation in part B of the algorithm above can be described as follows.

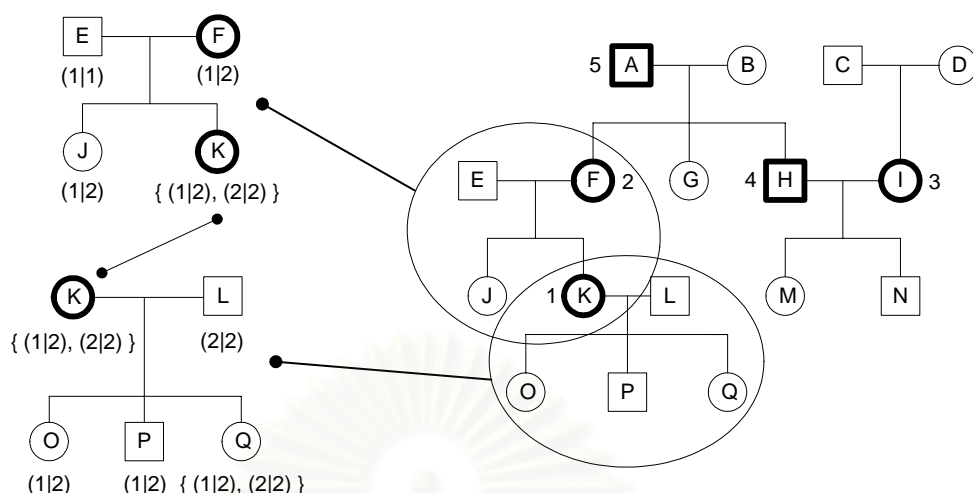


Figure 5.9 The pedigree used as an example for the algorithm of finding the minimum number of inconsistent alleles.

First, we will consider at the sub-rooted node K which is the node with the index 1. All possible haplotype configurations of each node in the family of node K are shown in the left of Figure 5.9. The number of inconsistent alleles for each haplotype configuration of node K can be computed as follows.

We use the notation $X(a|b)$ to represent the haplotype configuration $(a|b)$ of node X .

$$\begin{aligned} \text{MinNumF}(R_1, f_{1,1,1}) &= \text{NumTrio}(L(2|2), K(1|2), O(1|2)) + \\ &\quad \text{NumTrio}(L(2|2), K(1|2), P(1|2)) + \\ &\quad \text{NumTrio}(L(2|2), K(1|2), Q(1|2)) \\ &= 0 + 0 + 0 = 0 \end{aligned}$$

$$\begin{aligned} \text{MinNumF}(R_1, f_{1,1,2}) &= \text{NumTrio}(L(2|2), K(1|2), O(1|2)) + \\ &\quad \text{NumTrio}(L(2|2), K(1|2), P(1|2)) + \\ &\quad \text{NumTrio}(L(2|2), K(1|2), Q(2|2)) \\ &= 0 + 0 + 1 = 1 \end{aligned}$$

$$\begin{aligned}
\text{MinNumF}(R_1, f_{1,2,1}) &= \text{NumTrio}(L(2|2), K(2|2), O(1|2)) + \\
&\quad \text{NumTrio}(L(2|2), K(2|2), P(1|2)) + \\
&\quad \text{NumTrio}(L(2|2), K(2|2), Q(1|2)) \\
&= 1 + 1 + 1 = 3
\end{aligned}$$

$$\begin{aligned}
\text{MinNumF}(R_1, f_{1,2,2}) &= \text{NumTrio}(L(2|2), K(2|2), O(1|2)) + \\
&\quad \text{NumTrio}(L(2|2), K(2|2), P(1|2)) + \\
&\quad \text{NumTrio}(L(2|2), K(2|2), Q(2|2)) \\
&= 1 + 1 + 0 = 2
\end{aligned}$$

The minimum number of inconsistent alleles of node K corresponding to the haplotype configuration (1|2) can be computed as follows.

$$\begin{aligned}
\text{MinNumH}(R_1, h_{1,1}) &= \min_c(\text{MinNumF}(R_1, f_{1,1,c})) \\
&= \min(0, 1) = 0
\end{aligned}$$

The minimum number of inconsistent alleles of node K corresponding to the haplotype configuration (2|2) can be computed as follows.

$$\begin{aligned}
\text{MinNumH}(R_1, h_{1,2}) &= \min_c(\text{MinNumF}(R_1, f_{1,2,c})) \\
&= \min(3, 2) = 2
\end{aligned}$$

Next, we consider at the sub rooted node F. The index of this node is 2 and all possible haplotype configurations of each node in the family of node F are shown on the left of Figure 5.9. The number of inconsistent alleles for each haplotype configuration of node F can be computed as follows.

$$\begin{aligned}
\text{MinNumF}(R_2, f_{2,1,1}) &= \text{MinNumH}(R_1, h_{1,1}) + \\
&\quad \text{NumTrio}(E(1|1), F(1|2), J(1|2)) + \\
&\quad \text{NumTrio}(E(1|1), F(1|2), K(1|2)) \\
&= 0 + 0 + 0 = 0
\end{aligned}$$

$$\begin{aligned}
\text{MinNum}F(R_2, f_{2,1,1}) &= \text{MinNum}H(R_1, h_{1,2}) + \\
&\quad \text{NumTrio}(E(1|1), F(1|2), J(1|2)) + \\
&\quad \text{NumTrio}(E(1|1), F(1|2), K(2|2)) \\
&= 2 + 0 + 1 = 3
\end{aligned}$$

The minimum number of inconsistent alleles of node F corresponding to the haplotype configuration (1|2) can be computed as follows.

$$\begin{aligned}
\text{MinNum}H(R_2, h_{2,1}) &= \min_c(\text{MinNum}F(R_2, f_{2,1,c})) \\
&= \min(0, 3) = 0
\end{aligned}$$

For the other sub-rooted nodes, the computation will be performed in the similar way. Finally, the process will finish at the rooted node A.

5.7 Local Grandparent Inference by a Mendelian Law

After performing the processes in Sections 5.2 to 5.6 above, there will be many loci in which the paternal and maternal alleles have been specified. If we know the paternal or maternal alleles, the grandparent corresponding to these specified alleles may be directly inferred. Local grandparent inference refers to the process of inferring the grandparent of a specific allele in the way that does not consider a whole pedigree simultaneously. This means that, by using just the information of the parents or the grandparent values of the adjacent loci, the specific grandparent value that will lead to the minimum number of recombinants could be directly inferred. The algorithm for inferring the grandparent for each paternal or maternal allele can be described as follows.

The Algorithm for Inferring the Grandparents

Step 1: For each paternal or maternal allele that is not a Mendelian inconsistent allele and whose haplotype configuration in the associated parent's locus is known heterozygous, perform step a or b below.

a. For the paternal allele, if it is identical to the paternal allele of the father's locus then set its grandparent to be 'grandfather' (with the value 1) otherwise if it

is identical to the maternal allele of the father's locus then set its grandparent to be 'grandmother' (with the value 2).

b. For the maternal allele, if it is identical to the paternal allele of the mother's locus then set its grandparent to be 'grandfather' (with the value 1) otherwise if it is identical to the maternal allele of the mother's locus then set its grandparent to be 'grandmother' (with the value 2).

Step 2: Perform forward checking from the first locus to the last locus by considering only the alleles that have been specified; and are Mendelian inconsistent or their associated parents' loci are homozygous. For each of these specified alleles, perform step a or b below.

a. For the paternal allele, if the grandparent of the previous adjacent paternal allele has been specified, then set its grandparent to be the same as the grandparent of the previous adjacent paternal allele.

b. For the maternal allele, if the grandparent of the previous adjacent maternal allele has been specified, then set its grandparent to be the same as the grandparent of the previous adjacent maternal allele.

Step 3: Perform backward checking from the last locus to the first locus by considering only the alleles that have been specified; and are Mendelian inconsistent or their associated parents' loci are homozygous. For each of these specified alleles, perform step a or b below.

a. For the paternal allele, if the grandparent of the next adjacent paternal allele has been specified, then set its grandparent to be the same as the grandparent of the next adjacent paternal allele.

b. For the maternal allele, if the grandparent of the next adjacent maternal allele has been specified, then set its grandparent to be the same as the grandparent of the next adjacent maternal allele.

After performing local grandparent inference, some grandparents will be specified. There may be some haplotype vectors in which all alleles have been specified but all of their grandparents have not been specified. This case usually occurs for the

data that have much of homozygous loci. For each haplotype vector matching this case, all of the alleles in that haplotype vector will be assigned with one identical grandparent value (by setting all values to be only “grandfather” or “grandmother”).

The algorithm for the local grandparent inference above has a specific point to be considered. By considering the pedigree haplotype configurations of all loci simultaneously, there may be many forms of the pedigree grandparent configurations of all loci that result in minimum number of recombinants (recombination points). This means that we can set the grandparents in many different ways in order to make the whole pedigree optimal in the number of recombinants. The processes in steps 2 and 3 are performed as a heuristic method by immediately setting the grandparent with the value equal to the grandparent of the adjacent locus, in order to avoid the recombination event.

There may be many possible optimal forms of the pedigree grandparent configuration for one particular pedigree haplotype configuration. Although this heuristic algorithm may neglect some of these possible forms, it does not exclude any pedigree haplotype configurations.

5.8 Finding the Set of All Possible Pedigree Grandparent Configurations

Each pedigree haplotype configuration can have many possible pedigree grandparent configurations corresponding to it (note that, each pedigree haplotype configuration or pedigree grandparent configuration referred to here, are the configurations of one locus). The pedigree grandparent configurations will be found after the set of pedigree haplotype configurations with minimum number of inconsistent alleles have been determined.

5.8.1 Finding the Set of All Possible Family Grandparent Configurations

The family grandparent configuration is the specification of both paternal and maternal grandparent on the specific locus of each offspring in the family. The

process of finding the set of all possible family grandparent configurations is performed at each family corresponding to each sub-rooted nodes. At each family, the set of all possible family grandparent configurations corresponding to each family haplotype configuration will be determined. Figure 5.10 shows the example of determining all possible family grandparent configurations corresponding to a particular family haplotype configuration.

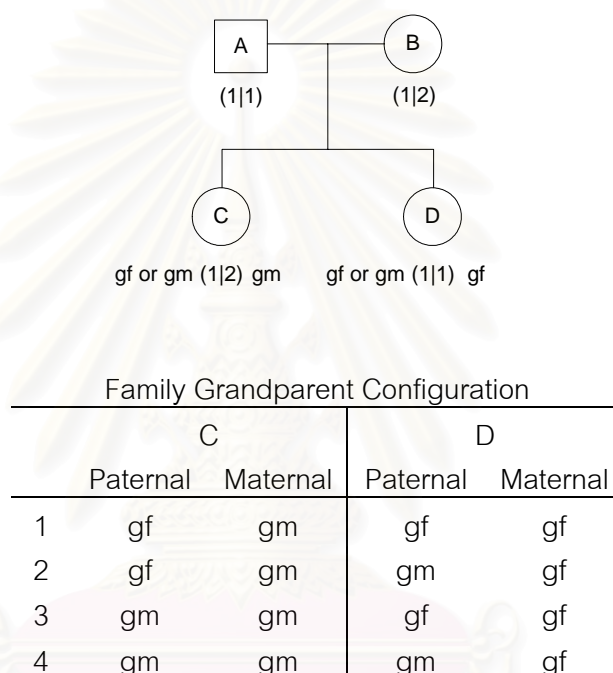


Figure 5.10 The example of possible family grandparent configurations corresponding to one family haplotype configuration (here, 'gm' = grandmother and 'gf' = grandfather).

At each allele, the possible grandparents are determined by considering its parental locus. If the given allele is identical to the paternal allele of the parental locus, one possible grandparent could be a grandfather. And if the considered allele is identical to the maternal allele of the parental locus, one possible grandparent could be a grandmother. If the considering allele is not identical to both paternal and maternal allele of the parental locus, it mean that this allele is a Mendelian inconsistent allele. For

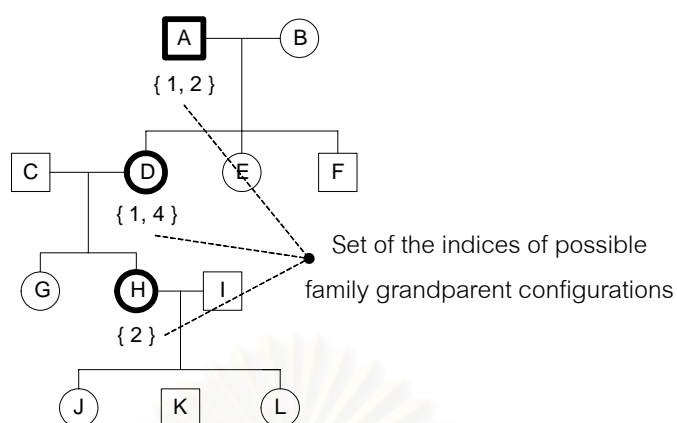
the case of a Mendelian inconsistent allele, the possible grandparent values will be both grandfather and grandmother.

5.8.2 Enumerating All Possible Pedigree Grandparent Configurations

The pedigree grandparent configuration refers to a specification of the family grandparent configurations for all sub-rooted nodes in the pedigree at one particular locus. From a particular pedigree grandparent configuration, we can specify all grandparent configurations for each member in the pedigree except those members that do not have parents (called founder).

In the previous section, we have described the process of finding all pedigree haplotype configurations that have minimum number of inconsistent alleles. From these pedigree haplotype configurations, we need to enumerate all corresponding pedigree grandparent configurations to be used in the later inference process. Each pedigree haplotype configuration has the corresponding set of possible pedigree grandparent configurations. The process of enumerating all pedigree grandparent configurations will be performed by considering at each pedigree haplotype configuration and then merging all results together.

At a particular pedigree haplotype configuration, assume that the sets of possible family grandparent configurations corresponding to the family haplotype configurations of each sub-rooted node are already determined. Each of the possible pedigree grandparent configurations corresponding to a particular pedigree haplotype configuration can be enumerated from each combination of the family grandparent configurations of all sub-rooted nodes. Figure 5.11 shows an example of this enumeration.



Pedigree Grandparent Configuration			
	A	D	H
1	1	1	2
2	1	4	2
3	2	1	2
4	2	4	2

Figure 5.11 The example of possible pedigree grandparent configurations corresponding to one pedigree haplotype configuration in which the possible family grandparent configurations for each sub-rooted node are defined as in the figure.

5.9 Finding the Grandparent Solutions with Minimum Number of Recombinants by a Dynamic Programming

A particular grandparent solution refers to the specification of the pedigree grandparent configurations of all loci in which the number of recombinants is minimum. Determining all grandparent solutions is a combinatorial optimization problem. The method used to solve this problem is a dynamic programming. Using a dynamic programming, we solve the problem from a smallest subproblem and then gradually extend the size of the subproblem until reaching the original size of the problem. For the problem of finding the grandparent solutions, the size of the problem is determined by the number of loci (haplotype length). The set of all possible pedigree grandparent configurations for each locus will be used for the computation in a dynamic

programming. The dynamic programming algorithm for finding the grandparent solutions that have minimum number of recombinants is described as follows.

The dynamic programming algorithm begins with the problem of two loci, the first and second loci (locus 1 and locus 2). Next, the computation between the locus 2 and locus 3 will be performed using the result from the first step. Then, the next consecutive locus will be considered using the information from the previous step. The similar computation will continue on the other loci until reaching the last locus. The problem of finding grandparent solutions can be considered as a network form (similar to the shortest path problem described in Chapter 3) in which each node in the network represents the pedigree grandparent configuration of each locus. The equivalent network of this problem is shown in Figure 5.12.

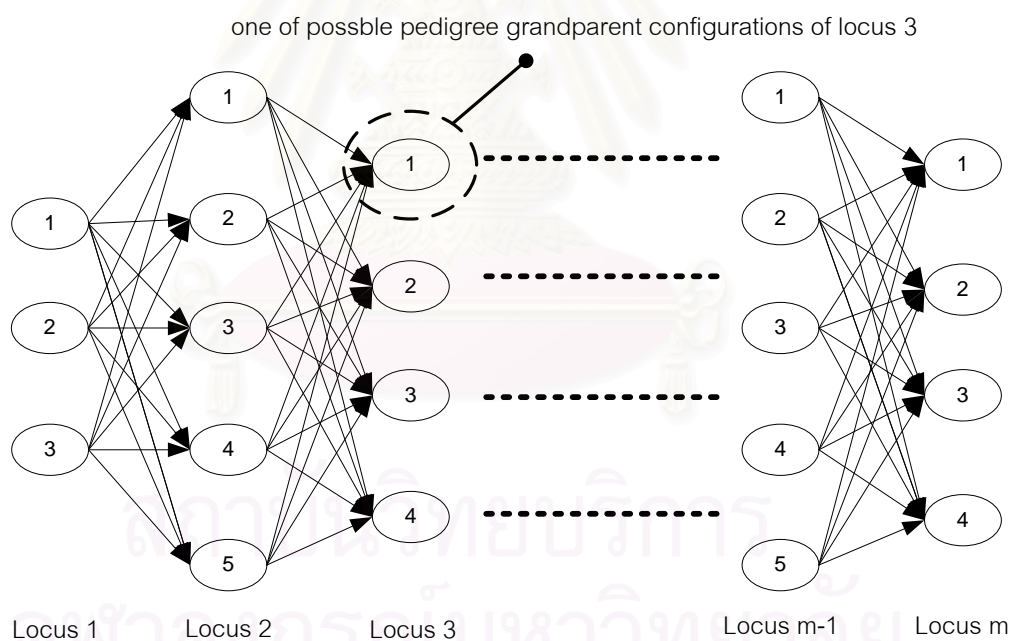


Figure 5.12 The network representing the problem of finding grandparent solutions that have minimum number of recombinants.

The recurrent function of the dynamic programming algorithm, for the problem of finding grandparent solutions that have minimum number of recombinants, can be formulated as follows.

$$MinNum(i, t) = \min_s (MinNum(i-1, s) + RecNum(w_{i-1,s}, w_{i,t})) , s \in \{1, 2, \dots, k_i\}$$

where i denotes the locus index, $i = 2, \dots, m$, and m is the number of all loci. t and s denote the indices of the pedigree grandparent configuration regarding to each locus. k_i is the number of all possible pedigree grandparent configurations of locus i .

$MinNum(i, t)$ is a function that returns the minimum number of recombinants that can possibly be for the problem of i loci (locus 1 to i). Locus i is assigned with the pedigree grandparent configuration of index t . For the problem of one locus (only the first locus), we define $MinNum(1, t) = 0$ for all values of index t .

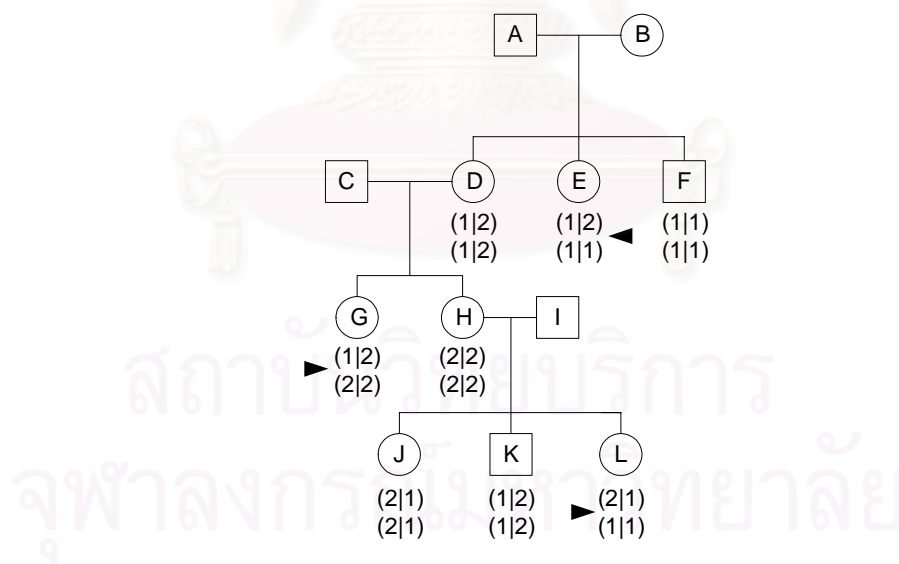


Figure 5.13 The pedigree with the assignment of grandparent configurations at two adjacent loci of each non-founder member. From these configurations, the number of recombination points is 3 as indicated by the arrows.

$RecNum(w_{i-1,s}, w_{i,t})$ is a function that returns the number of recombinants (recombination points) between two pedigree grandparent configurations ($w_{i-1,s}$ and $w_{i,t}$) of two adjacent loci, where $w_{i,t}$ denotes the pedigree grandparent configuration of index t at locus i . An example for the computation of the function $RecNum(...)$ is shown in Figure 5.13.

The possible grandparent solutions of this problem may be from only one solution to many million solutions, depending on the input pedigree data. To get the solutions, the function $MinNum(i, t)$ will be computed for all nodes. After the computations for all nodes in the last locus have been performed, the optimal (minimal) number of recombinants for the solutions can be computed by the following equation.

$$MinRecNum = \min_s (MinNum(m, s)), \quad s \in \{1, 2, \dots, k_m\}$$

where $MinRecNum$ is the optimal (minimal) number of recombinants (recombination points) for the solutions. m is the index of the last locus and k_m is the number of all possible pedigree grandparent configurations of the last locus.

When the optimal nodes in the last locus are determined, the optimal paths in the network could be determined by the backtracking algorithm. One path represents one grandparent solution which is one specification of the pedigree grandparent configurations of all loci.

Based on the network representation of this problem in Figure 5.12, the dynamic programming algorithm for determining all grandparent solutions can be described as follows.

We denote some notations as follows.

$L_{i,t}$ denotes the list of nodes in locus $i-1$ having the value of the term $MinNum(i-1, s) + RecNum(w_{i-1,s}, w_{i,t})$ in the function $MinNum(i, t)$ equal to value of $MinNum(i, t)$.

Y denotes a grandparent solutions which keeps the specification of the pedigree grandparent configurations for all loci.

The Algorithm for Finding the Grandparent Solutions

Step 1: Initialize a bound number to be 0.

Step 2: Perform the following steps (a and b) for each locus (with index i), by beginning at the second locus and continue on the next locus until finishing at the last locus.

a. For each node (a specific pedigree grandparent configuration with index t) of a specific locus i , compute $MinNum(i, t)$ and save the list of nodes in locus $i-1$ that make the value of the term $MinNum(i-1, s) + RecNum(w_{i-1,s}, w_{i,t})$ in the function $MinNum(i, t)$ equal to value of $MinNum(i, t)$ (this list is denoted as $L_{i,t}$).

b. Remove the nodes that have the value of $MinNum(i, t)$ greater than the bound number. If all nodes have been removed then increment the bound number to be equal to the minimum value of $MinNum(i, t)$ and go back to begin step 2 again.

Step 3: Perform backtracking to get all optimal grandparent solutions. This backtracking process is the process of enumerating all optimal grandparent solutions by preorder traversing on each node in the network tree (here the direction of a network graph will be changed to be from the last locus to the first locus). At each node, the branching paths used for enumeration are generated from the list $L_{i,t}$. The backtracking process begins by performing part A below.

Part A: At the last locus, initialize the grandparent solution Y (now, there is no assignment in Y) for each optimal node (the nodes with minimum value of $MinNum(...)$). Then, perform part B for each of these optimal nodes.

Part B: Consider a specific node t of a locus i with a grandparent solution Y . Assign the pedigree grandparent configuration of index t with the locus i in Y . Then, perform step a or b below.

a. If i is equal to 1, it means that each of all loci in Y has already been assigned with a specific pedigree grandparent configuration. In this case, save this grandparent solution Y to the list of grandparent solutions.

b. Otherwise, perform part B for each node in the list $L_{i,t}$ (these nodes are in the locus $i-1$) with the current grandparent solution Y .

A dynamic programming is, in general, an algorithm that mainly reduces the redundancy of the computations. For this problem, it works quite well in reducing many of computations. But as from the characteristic of a computing time of this problem that increases exponentially, there is still a demand for reducing as much computing time as possible. By considering the algorithm above, there are still some redundancies in the computation process. Now, let's consider the computation of the function $RecNum(w_{i-1,s}, w_{i,t})$. To count the number of recombination points between two pedigree grandparent configurations of two adjacent loci, we must compare each two grandparent configuration from two loci on every member in a pedigree. Let's consider the computation of the function $RecNum(w_{1,1}, w_{2,1})$ and $RecNum(w_{1,2}, w_{2,1})$. If the pedigree grandparent configuration $w_{1,1}$ is nearly similar to $w_{1,2}$ and assume that they are different by only one grandparent configuration, it should mean that the computation for $RecNum(w_{1,1}, w_{2,1})$ and $RecNum(w_{1,2}, w_{2,1})$ are nearly similar, and thus, there occur some redundancies of computation. So as from this point, we have developed the method that can reduce some of these redundancies.

In our method, the function $RecNum(w_{i-1,s}, w_{i,t})$ is computed by considering at each family grandparent configuration corresponding to a particular pedigree grandparent configuration. Counting the number of recombination points will be considered at a unit of family, not a unit of member. So we can compute the number of recombination points between two family grandparent configurations of two loci at first. To compute the function $RecNum(w_{i-1,s}, w_{i,t})$, we can use the pre-computed values so that many redundancies could be reduced.

5.10 Getting the Haplotype Solutions by Mapping from the Grandparent Solutions

After the set of grandparent solutions with minimum number of recombinants are determined, the corresponding set of the haplotype solutions could be determined. A particular haplotype solution refers to the specification of the pedigree haplotype configurations of all loci in which the number of recombinants is minimum. One grandparent solution refers to the specification of pedigree grandparent configurations for all loci. At each locus, the pedigree grandparent configuration can be mapped back to its corresponding set of pedigree haplotype configurations (in the process of enumerating the pedigree grandparent configurations from the pedigree haplotype configurations in Section 5.8, each pedigree grandparent configuration has kept a list of its corresponding pedigree haplotype configurations). Therefore, each grandparent solution can be mapped to the haplotype solutions by enumerating from the combination of each corresponding pedigree haplotype configuration at each locus. One grandparent solution can be mapped to many haplotype solutions. Note that the number of resulted haplotype solutions may be very large, especially for the long haplotypes, due to the nature of a combinatorial problem.

CHAPTER VI

EXPERIMENTAL RESULTS

For the problem of the occurrence of Mendelian inconsistent alleles, we have constituted the procedure to handle this problem based on the assumption which states that the preferable haplotype solutions should have minimum number of inconsistent alleles. In this work, we left the consideration of the reliability of our method for the problem of inconsistent alleles with only a concordance of this assumption. So, there is no explicit experiment concerning with the problem of inconsistent allele.

Mainly, the experiment was conducted for examining the efficiency of this method with respect to the computing time and size of data. The efficiency of this method was compared with the dynamic programming method proposed by Doi, Li, and Jiang (2003). To compare our method with their method, we used their program called "PedPhase" which has two implementations of dynamic programming methods, the member-based and locus-based algorithms. The results from our implemented program and from PedPhase were compared in our experiment.

Other than the efficiency testing, we also tested for the effect of each parameter to the computing time, the accuracy and the number of solutions. These tests do not mainly concern with our objective, but they are used to assess the correctness and also make some observations on our method.

Since the scope of this work does not concern with the real data, the input pedigree data for our experiment were generated from our simulation program (see Appendix A). In the simulation program, the characteristic of the pedigree data can be controlled by particular genetic parameters. The simulation program can generate the pedigree with and without loops. We can specify all important parameters that were used in our experiment. The parameters used in our experiment are the number of generations, the number of members in a pedigree, the number of loci in a particular haplotype (haplotype length), the number of different alleles in each locus, the rate of

missing alleles, the number of recombination events, and the number of Mendelian inconsistent alleles. At each tested case, one parameter was used as the control parameter while the others were fixed with the specific values.

Our experiment was performed on a Pentium IV PC with 2.4 GHz CPU and 256 MB RAM. Each result shown in our experiment was an average from 30 runs from different input data. All tested data in this experiment are the biallelic data (SNPs data) in which each locus has only two possible values of alleles. As for the reasons of time consuming and the limit number of memory (RAM), in some cases, we restricted to use the results from only those input data that are feasible to compute. The feasible input data are defined as the data that result in the number of limited parameters not exceed the maximum limit. Here, the limited parameters are the number of the possible pedigree haplotype configurations, the possible pedigree grandparent configurations, the computing cases in a dynamic programming module at the phase of finding optimal grandparent solutions (the multiply of the number of nodes between two adjacent loci), the grandparent solutions, and the haplotype solutions. The maximum limits of these parameters should be set by specific constants that appropriated to the testing PC. For our experiment, the default maximum values for the limited parameters at each locus were set as follows. The maximum number of the possible pedigree haplotype configurations, the possible pedigree grandparent configurations, and the grandparent solutions were set as 200,000. The maximum number of haplotype solutions was set as 1,000,000. And the maximum number of the computing cases in a dynamic programming module at the phase of finding optimal grandparent solutions was set as 150,000,000.

6.1 Comparing the Computing Time with PedPhase

PedPhase is a program that contains the functions for inferring haplotypes from genotypes for each member in a general pedigree. It was developed by Doi, Li, and Jiang (2003) and is used as the software for solving the minimum-

recombinant haplotype configuration (MRHC) problem. There are five algorithms in PedPhase that can be used to solve MRHC problem. But we only interested in two of these algorithms, the member-based and locus-based dynamic programming algorithms.

For the member-based dynamic programming algorithm, it assumes that the input pedigree is small and performs dynamic programming on the marker loci in each member of the pedigree simultaneously. The algorithm works for any input pedigree (without or with mating loops) and has a running time linear in the number of marker loci. It could be useful as a subroutine for solving MRHC on small pedigrees, nuclear families from a large input pedigree or independent nuclear families from a population data.

For the locus-based dynamic programming algorithm, it assumes that the number of marker loci is bounded by a small constant and performs dynamic programming on the members of the input pedigree. It works only for pedigrees without mating loops. The algorithm takes advantage of the tree structure of the input pedigree and has a running time linear in the size of the pedigree.

Both algorithms above also assume that the data do not contain missing alleles and all alleles are consistent with a Mendelian law. Thus, the input data for this test were generated so that there were no missing and inconsistent alleles. And also there is no loop in the input pedigree.

These two algorithms were used for evaluating the efficiency in the computing time of our method, by comparing the computing time of our method with the time from these two algorithms. As for the simplicity, we will refer to the member-based method as the dynamic programming by locus (DP-by-Locus) and refer to the locus-based method as the dynamic programming by member (DP-by-Member). Here, the number of members and the number of loci were used as parameters for examining the efficiency in the computing time. Since PedPhase returns only one solution for each input data, thus we excluded the time for enumerating all solutions from our method. We

just set our method to get only one solution in order to compare the computing time with these two algorithms of PedPhase.

6.1.1 Comparing by the Number of Loci

To compare the computing time by varying the number of loci (haplotype length), the number of members will be fixed. Since the DP-by-Locus of PedPhase can work with only a very small size pedigree (by our observation, not over 9 members), so we compared our method with two dynamic programming algorithms separately, using the different numbers of members. For the case of DP-by-Locus, we set the number of members as 7 from a pedigree with 3 generations. And for the case of DP-by-Member, we set the numbers of members from 3 different values as 15, 20, and 25 from a non-looped pedigree with 3, 4, and 4 generations respectively.

The other parameters were set as follows. There was no recombinant. And there were no missing and inconsistent alleles. Note that, to compare between two methods at each case, the same input data were used for both methods. The results are shown in Table 6.1, Figure 6.1, Table 6.2, and Figure 6.2.

By comparing with the DP-by-Member, the results show that the computing time of our method quite linearly increases by the number of loci while the computing time of the DP-by-Member dramatically increases when the number of loci is above 15. These results show that the DP-by-Member is appropriated for only the data with small number of loci while our method still fairly works with more number of loci. And it can be inferred from these results when working with the moderate-size pedigree (15-25 members) and the haplotype length is more than 20 loci, our method clearly tends to outperform the DP-by-Member (has less computing time).

By comparing with the DP-by-Locus, the computing times of our method are apparently less than those of the DP-by-Locus for all cases. And also, the computing times of our method for different numbers of loci seems to be constant. This is because when the number of members is small (in this case, we used only 7 members), the time

used in the dynamic programming process will be very small compared to the time used in other pre-processes which are not depend much on the number of loci.

Table 6.1 The computing times varying by the number of loci between our proposed method and the DP-by-Member of PedPhase (by fixing the number of members as 15, 20, and 25).

#Loci	Time (sec)							
	5	10	13	15	17	18	19	20
#Members = 15								
Proposed Method	0.40	1.17	1.33	1.60	3.47	4.45	6.61	6.87
DP-by-Member	0.09	0.12	0.37	4.02	7.63	21.30	49.82	81.0
#Members = 20								
Proposed Method	0.85	22.93	24.12	28.25	33.33	34.50	41.42	63.70
DP-by-Member	0.10	0.13	0.46	17.43	41.05	74.42	201.0	349.8
#Members = 25								
Proposed Method	13.32	39.77	53.87	91.67	97.66	116.7	121.6	127.3
DP-by-Member	0.11	0.33	0.55	10.32	11.52	49.66	167.5	358.6

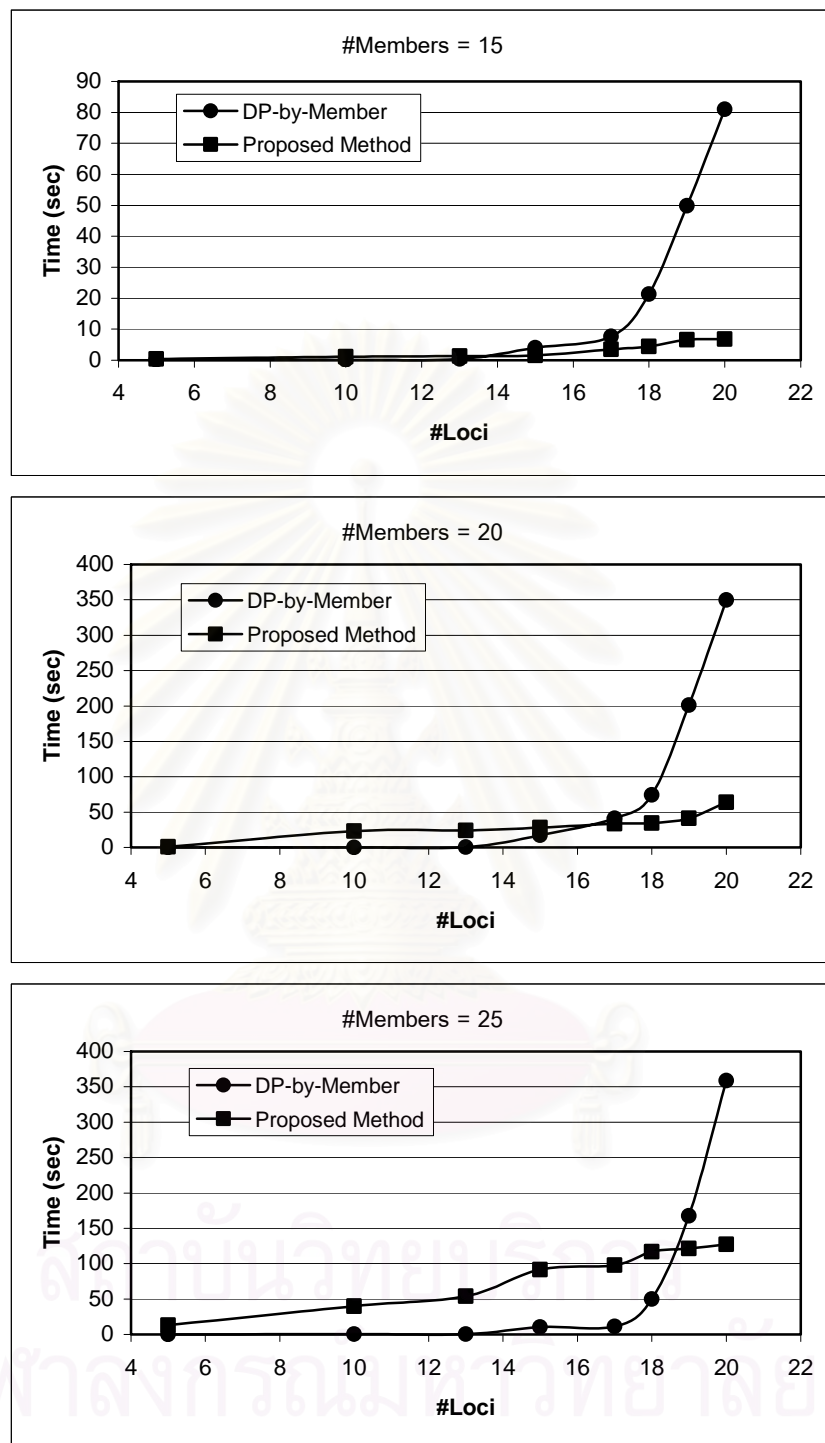


Figure 6.1 The computing times varying by the number of loci between our proposed method and the DP-by-Member of PedPhase (by fixing the number of members as 15, 20, and 25).

Table 6.2 The computing times varying by the number of loci between our proposed method and the DP-by-Locus of PedPhase (#Members = 7).

#Loci	Time (sec)				
	5	10	15	20	25
Proposed Method	0.242	0.243	0.260	0.264	0.273
DP-by-Locus	0.836	2.03	3.265	5.962	6.599

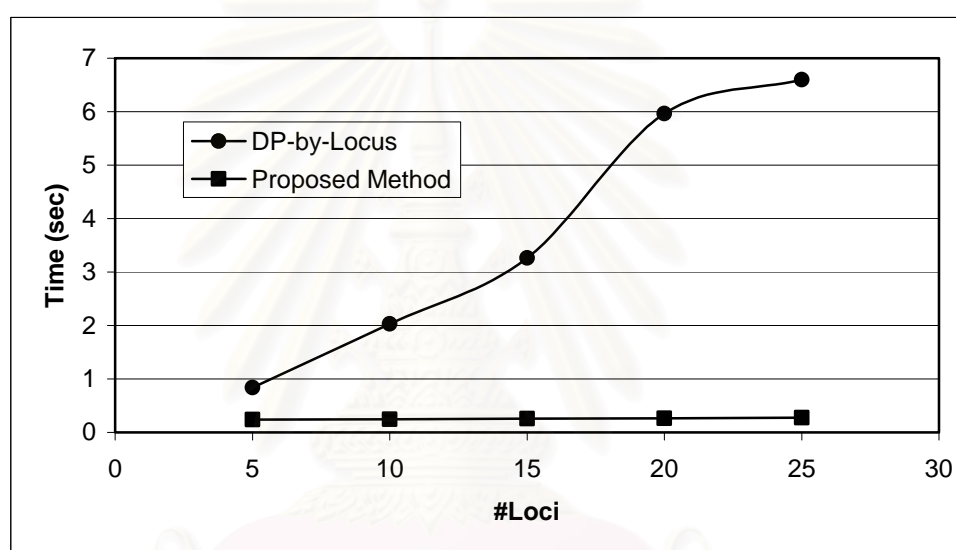


Figure 6.2 The computing times varying by the number of loci between our proposed method and the DP-by-Locus of PedPhase (#Members = 7).

6.1.2 Comparing by the Number of Members

In this case, the computing times were tested from different numbers of members. The parameters were set as follows. The pedigree had no loop. The number of loci was 10. There was no recombinant. There were no missing and inconsistent alleles. The number of generations was varied by the size of a pedigree. We used 2 generations for the pedigrees with 5 and 6 members (only one family), 3 generations for

the pedigrees with 7-14 members, and 4 generations for the pedigree with 16-28 members. The result is shown in Table 6.3 and Figure 6.3 below.

Table 6.3 The computing times varying by the number of members between our proposed method and two dynamic programming algorithms of PedPhase.

#Members	DP-by-Locus	DP-by-Member	Proposed Method
5	0.108	0.116	0.244
6	1.340	0.090	0.255
7	2.258	0.095	0.249
8	35.206	0.149	0.270
9	167.087	0.094	0.262
10	-	0.123	0.322
12	-	0.098	0.385
14	-	0.104	1.247
16	-	0.095	1.711
18	-	0.119	7.583
20	-	0.021	23.792
24	-	0.163	60.594
28	-	0.349	142.057

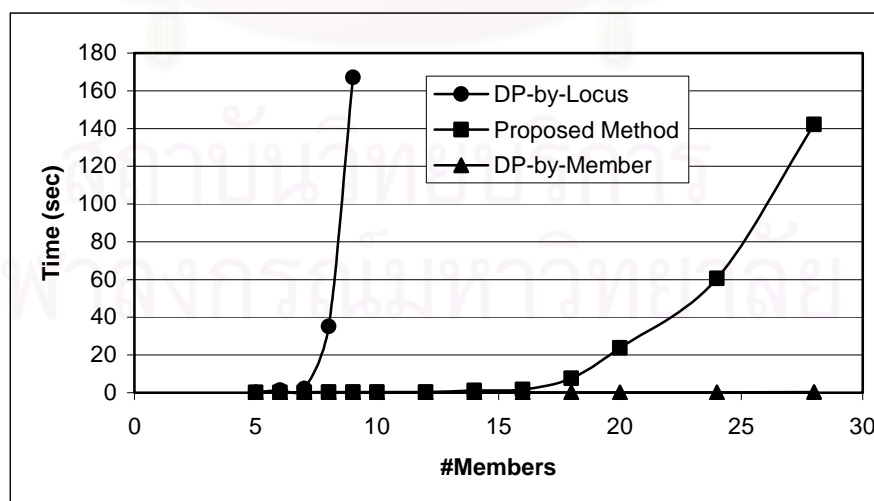


Figure 6.3 The computing times varying by the number of members between our proposed method and two dynamic programming algorithms of PedPhase.

Note that for the case of DP-by-Locus, the program PedPhase rejected all our input data that have number of members more than 9. Then there were no results from these cases. And for the case that the number of members is 9, the resulted time was an average from only 10 input data because most tested cases were rejected by PedPhase (not feasible to compute). The average computing time from these 10 feasible input data would be considered as an approximated lower bound of this case (the real average time would be very much more than this).

From the result above, we can see that the computing time of the DP-by-Locus dramatically increases when the number of members is more than 8. The computing time of our method is more than the DP-by-Member for all cases. And when the number of members is more than 20, our method seems to dramatically increase in the computing time. This apparently shows that the DP-by-Member performs better than our method when the size of pedigree is large. It is because the algorithm of DP-by-Member, in general, has a computing time linear in the number of members while our method has a computing time exponential in the number of members.

6.2 The Effect of each Parameter to the Computing Time

To examine the effect of each parameter to the computing time, we set our program to enumerate only one solution in order to avoid the effect of a solution sizes. The number of solutions varies very much on different input data and when the number of solutions is very large, the phase of enumerating the solutions would consume very much more time than the optimization phase. Since the computing time that we are interested is only the time of the optimization phase then the time for enumerating all solutions should be ignored.

6.2.1. Testing by the number of members

In this case, our program was tested with different numbers of members. The parameters were set as follows. The number of loci was 10. There was no

recombinant. There were no missing and inconsistent alleles. The number of generations was varied by the number of members. We used 2 generations for the pedigrees with 5 members (only one family), 3 generations for the pedigrees with 10-15 members, and 4 generations for the pedigree with 20-30 members. The result is shown in Table 6.4 and Figure 6.4.

Table 6.4 The computing time varying by the number of members.

#Members	5	10	15	20	25	30
Time (sec)	0.016	0.061	5.320	17.076	56.577	110.615

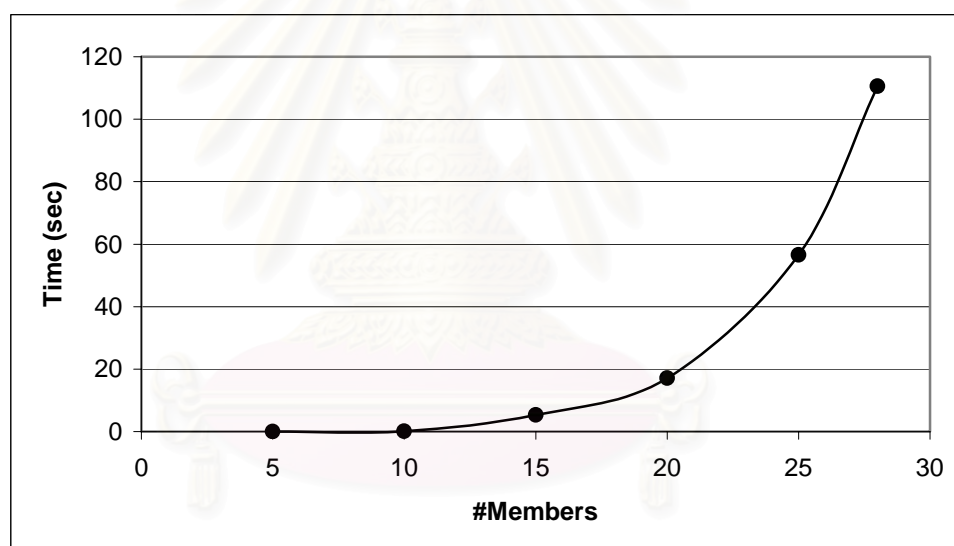


Figure 6.4 The computing time varying by the number of members.

Note that in the case of 30 members, it took very long time for the computation (could be many hours). Thus, we just used the average time from only the feasible input data which should be considered as an approximated lower bound of this case. The result apparently shows that the computing time exponentially increases by the number of members.

6.2.2 Testing by the number of loci

In this case, our program was tested with different numbers of loci. The parameters were set as follows. The number of members was 15 from the pedigree with 3 generations. There was no recombinant as well as no missing and inconsistent alleles. The result is shown in Table 6.5 and Figure 6.5.

Table 6.5 The computing time varying by the number of loci.

#Loci	5	10	20	40	80
Time (sec)	0.179	1.307	3.099	6.801	19.063

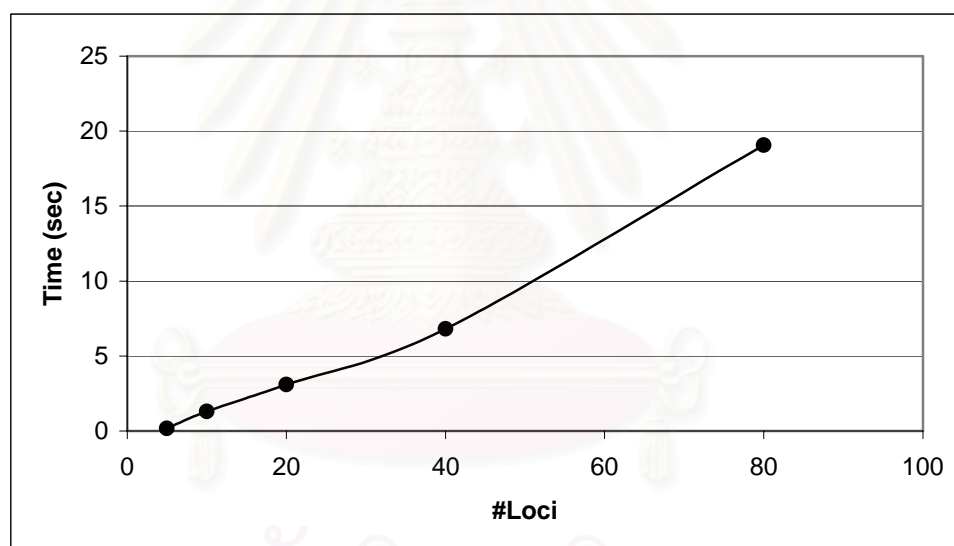


Figure 6.5 The computing time varying by the number of loci.

We can see from the result that the computing time increases quite linearly by the number of loci. This result corresponds to the concept of our method which perform dynamic programming by locus, thus, it should have a computation time linear in the number of loci.

6.2.3 Testing by the Rate of Missing Alleles

In this case, our program was tested with different rate of missing alleles. The parameters were set as follows. The number of members was 15 from the pedigree with 3 generations. The number of loci was 10. There was no recombinant and no inconsistent allele. The result is shown in Table 6.6 and Figure 6.6.

Table 6.6 The computing time varying by the rate of missing alleles.

%Missing	0	5	10	15	20
Time (sec)	0.778	33.832	77.346	163.064	394.119

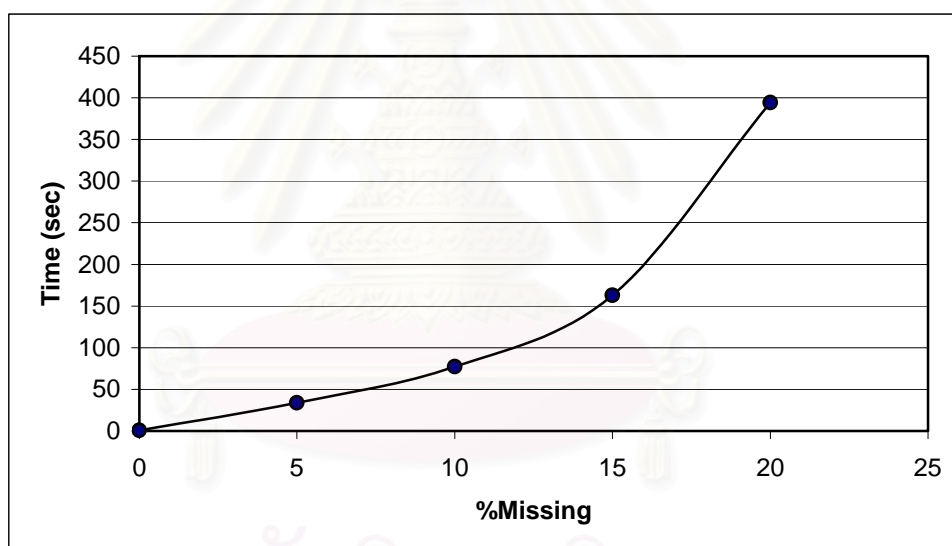


Figure 6.6 The computing time varying by the rate of missing alleles.

Note that in the case that the rate of missing alleles is 20%, it took very long time for the computation (could be many hours). Thus, we just used the average time from only the feasible input data which should be considered as an approximated lower bound. The result shows that the rate of missing alleles directly affects the

computing time. And as from the graph in Figure 6.6, we can see that the time exponentially increases by the rate of missing alleles.

6.3 The Effect of each Parameter to the Accuracy

Here, the accuracy refers to the percent of correct inferred alleles which can be computed by comparing the haplotype solution with the true haplotype data. As from the scope of our work, we ignore the direct testing for the accuracy of our method. This testing is just mainly used for considering the effect of each parameter to the accuracy. In most cases, solving the MRHC problem usually results in many solutions. The number of solutions may range from a few to many millions. Indeed, we aim our method to be just a pre-filtering process for the haplotype inference problem. And then, in this work, there is no considering about the selection of the best solution from all possible solutions. In order to examine the effect of each parameter to the accuracy, we will approximate the accuracy by averaging from all solutions. In this experiment, we limited the number of solutions used to compute for the accuracy as 100,000 solutions. This is because getting the large number of solutions is very time consuming and only 100,000 solutions should be enough for approximating of the accuracy.

For this experiment, the default settings for each parameter are set as follows. The number of members was 15 from the pedigree with 3 generations. The number of loci was 10. There was no recombinant as well as no missing and inconsistent alleles. In each tested case, the parameters were set from the default values excepted for one specific parameter that was a control parameter. The results from each tested case are described as follows.

6.3.1 Testing for the Rate of Local-Inferred Alleles

In the local inference phase, we perform direct inference according to the Mendelian law. During this process, some alleles could be directly inferred as the paternal or maternal alleles and also some missing alleles could be imputed. Here, we

will call these alleles as local-inferred alleles. The alleles directly inferred by this process, in general, are quite reliable. If all alleles in the input data can be directly inferred, the solution should be 100% correct. If there are more directly inferred alleles, there will be more reliable alleles and also there will be fewer alleles to be inferred in the optimization phase. So the number of the local-inferred alleles directly affects the accuracy of the final solutions.

Normally, the rate of local-inferred alleles could vary by the number of homozygous loci and the structure of the pedigree. But in this experiment, we will not concern in more details of these factors. Here, the rate of local-inferred alleles would be considered for only the default case of parameter settings defined above, in order to use for analyzing the results from the accuracy testing below. From the experiment, we found that the rate of local-inferred alleles for the default case is about 79% (this is not the rate of exactly true alleles).

6.3.2 Testing by the Number of Members

In this case, the accuracy was tested by varying the number of members. We used 2 generations for the pedigrees with 5 members (only one family), 3 generations for the pedigrees with 10-15 members, and 4 generations for the pedigree with 20-25 members. The result is shown in Table 6.7. This result shows that the accuracy seems to be higher for the pedigree with more number of members. This is because the more information on a pedigree could increase the degree of inference, and so, inferring on a pedigree with more number of members should have more accuracy than those with less number of members. However, there may be other factors that can affect the accuracy and can cause the decreasing of accuracy in a larger pedigree. One of these factors could be the structure of a pedigree which we ignored to consider here. Hence the result here may not consistent for all cases as can be seen at the case of 25 members.

Table 6.7 The accuracy varying by the number of members.

#Members	5	10	15	20	25
%Correct	89.3	90.6	91.6	93.0	91.4

6.3.3 Testing by the Number of loci

In this case, the accuracy was tested by varying the number of loci. The result apparently shows that the number of loci has no effect on the accuracy (see Table 6.8).

Table 6.8 The accuracy varying by the number of loci.

#Loci	5	10	20	40
%Correct	91.3	90.5	91.7	91.9

6.3.4 Testing by the Rate of Missing Alleles

In this case, the accuracy was tested by varying the rate of missing alleles. The result apparently shows that the rate of missing alleles decreases the accuracy by a linear manner (see Table 6.9 and Figure 6.7).

Table 6.9 The accuracy varying by the rate of missing alleles.

%Missing	0	5	10	15	20
%Correct	91.3	87.7	84.6	81.0	77.5

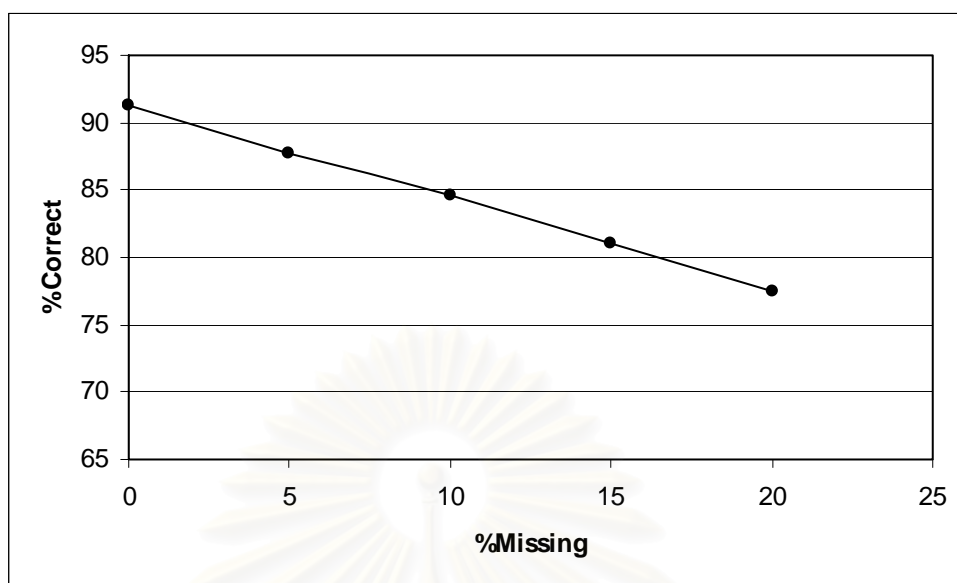


Figure 6.7 The accuracy varying by the rate of missing alleles.

6.3.5 Testing by the Number of Recombination Points

In this case, the accuracy was tested by varying the number of recombination points. From the result (see Table 6.10), the number of recombination points do not show any effect on the accuracy.

Table 6.10 The accuracy varying by the number of recombination points.

#Recombination	0	1	2	4	8	16
%Correct	91.6	92.2	91.3	91.5	91.6	91.1

6.3.6 Testing by the Rate of Homozygous Loci

In this case, the accuracy was tested by varying the rate of homozygous loci. The locus that is homozygous can be directly inferred without any consideration. So if there are many homozygous loci, the rest alleles to be inferred will be less, and thus

result in high accurate solutions. This is corresponding to our experimental result as shown in Table 6.11.

Table 6.11 The accuracy varying by the rate of homozygous loci

%Homozygous	Default (Avg. 55.8%)	41-60% (Avg. 50.7%)	61-80% (Avg. 70.2%)	81-100% (Avg. 91.7%)
%Correct	92.1	90.6	94.1	98.2

* Note that for the case of Default, it refers to the case that all parameters are set as the default setting. The rate of homozygous loci for the default setting is about 56%.

6.4 The Effect of Each Parameter to the Number of Solutions

Indeed, the solution should be represented by the haplotype configurations with their corresponding grandparent configurations. One haplotype solution usually has many corresponding grandparent configurations that result in the minimum number of recombinants. But for the reason of simplicity and limitation of memory, the solutions here will just refer to only the haplotype solutions. Although it is not difficult to determine all corresponding optimal grandparent configurations for a particular known haplotype solution, selecting the best grandparent configurations is out of the scope of this work. Thus, we will ignore these corresponding optimal grandparent configurations.

For the cases that there are a large number of solutions, there may not be enough memory to keep all solutions. In our experiment, we set the maximum number of haplotype solutions as 1,000,000. This maximum number is sufficient for examining the effect of each parameter to the number of solutions.

The default settings for each parameter are set as follows. The number of members was 15 from the pedigree with 3 generations. The number of loci was 10. There was no recombinant as well as no missing and inconsistent alleles. In each tested

case, the parameters were set from the default values excepted for one specific parameter that was a control parameter. The results from each tested case are described as follows.

6.4.1 Testing by the Number of Members

In this case, the number of solutions was tested by varying the number of members. We used 2 generations for the pedigrees with 5 members (only one family), 3 generations for the pedigrees with 10-15 members, and 4 generations for the pedigree with 20 members. The result is shown in Table 6.12 and Figure 6.8. This result shows that the number of solutions exponentially increases by the number of members.

Table 6.12 The number of solutions varying by the number of members.

#Members	5	10	15	20
#Solutions	19	4,280	22,963	113,283

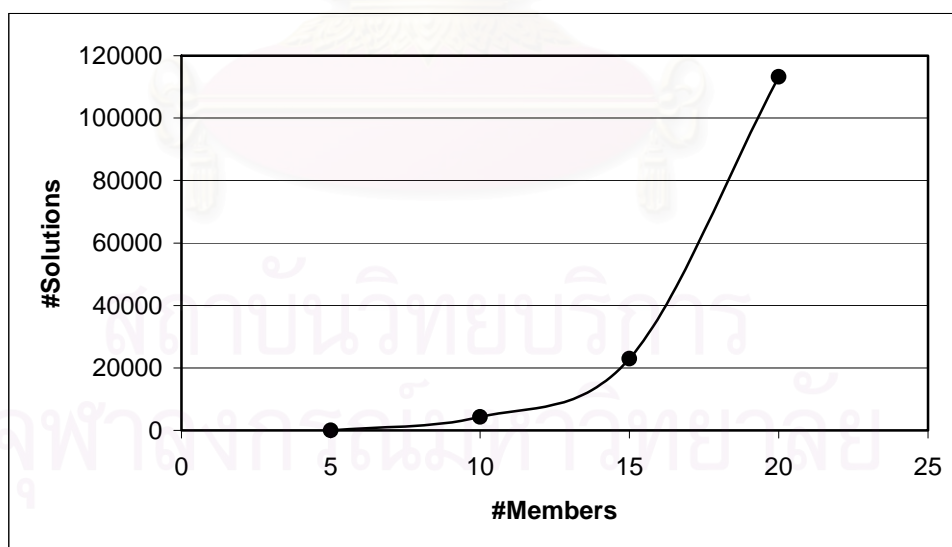


Figure 6.8 The number of solutions varying by the number of members.

6.4.2 Testing by the Number of loci

In this case, the number of solutions was tested by varying the number of loci. We used only 3 different numbers of loci with the maximum as 15 loci. Here, we must ignore the other higher numbers of loci because there was not enough memory to compute. For the case of 15 loci, there were many cases that result in the number of solutions exceeding the maximum limit. So, the result of this case would be considered as the lower bound, not a real average value. As from the result shown in Table 6.13 and Figure 6.9, we can roughly conclude that the number of solutions exponentially increases by the number of loci.

Table 6.13 The number of solutions varying by the number of loci.

#Loci	5	10	15
#Solutions	14,693	27,788	93,046

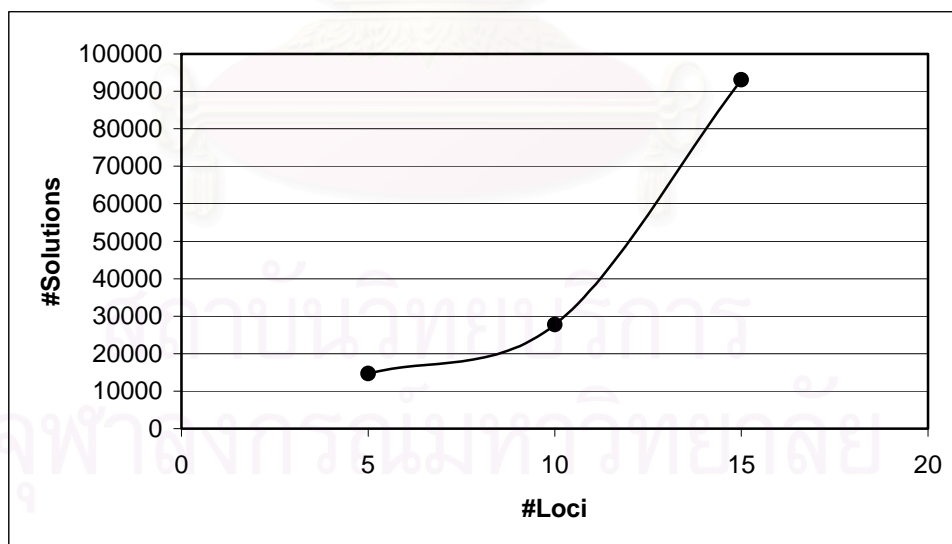


Figure 6.9 The number of solutions varying by the number of loci.

6.4.3 Testing by the Rate of Missing Alleles

In this case, the number of solutions was tested by varying the rate of missing alleles. For the case of 10%, there were many cases that result in the number of solutions exceeding the maximum limit. So, we used only the feasible input data and the result of this case would be considered as the lower bound, not a real average value. The result is shown in Table 6.14 and Figure 6.10. This result shows that the number of solution increases when the rate of missing alleles increases.

Table 6.14 The number of solutions varying by the rate of missing alleles.

%Missing	0	5	10
#Solutions	23,591	82,589	167,101

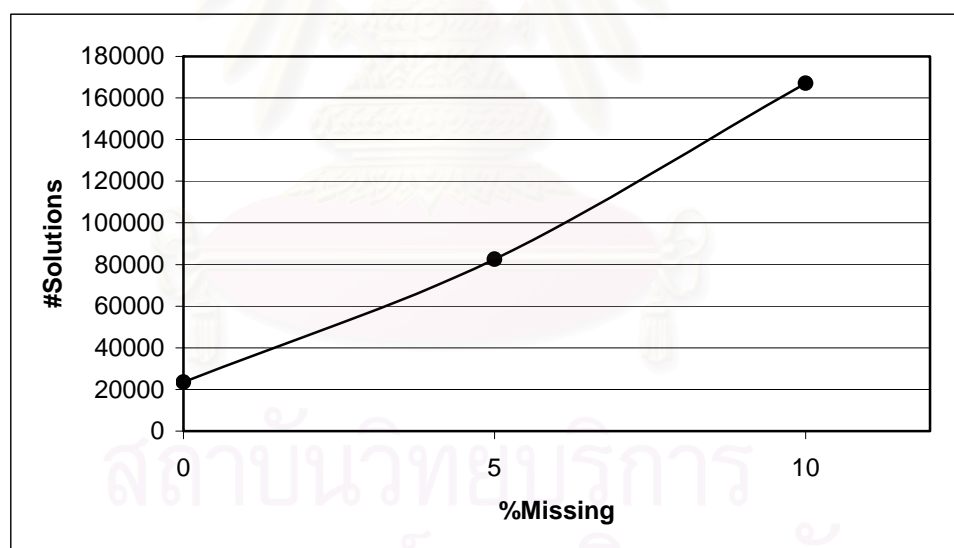


Figure 6.10 The number of solutions varying by the rate of missing alleles.

6.5 Evaluation

The efficiency has been tested by comparing our method with PedPhase. Comparing our method with the DP-by-Locus of PedPhase, we found that

our method used less computing time, and also, the DP-by-Locus could not work with the pedigree with a number of members more than 9 while our method could fairly work for all cases. Comparing with the DP-by-Member of PedPhase, our method used more computing time when the number of loci is small. But with more number of loci (upper than 15), the computing time of the DP-by-Member apparently increased exponentially. In contrast, the computing time of our method still linearly increased by the number of loci. At the number of loci upper than 20, our method used very less computing time than the DP-by-Member methods. We can clearly conclude from this testing that when the number of members is about 15-25 and the number of loci is more than 20, our method will outperform both dynamic programming algorithms in PedPhase.

In the inference process, the main procedure that consumes most of computation time is the phase of finding the minimum-recombinant haplotype configuration. Note that, in our method, all of the phases before the phase of finding minimum-recombinant pedigree grandparent configurations uses the computing time less than a second. The optimization procedure for finding minimum-recombinant haplotype configuration has been proved to be NP hard problem (Li and Jiang, 2003), which means that the computing time should exponentially increase by the size of data. But, by using a dynamic programming by locus, solving this optimization procedure could conceptually have a computing time linear in the number of loci but still exponential by the number of members. Our method, which used a dynamic programming by locus in this procedure, has shown the corresponding result with this concept. Thus, this shows the consistent of our implementation to the concept of the method we used.

The rate of missing alleles apparently shows the effect to the computing time, the accuracy, and the number of solutions. The results show that the occurrence of missing alleles would increase the computing time, decrease the accuracy, and increase the number of solutions. The reason is that the occurrence of missing alleles produces more number of possible haplotype configurations to be processed, and thus, require more computing time. And also it could result in more possible cases of the

optimal haplotype solution, and thus, lead to higher possibility for the occurrence of incorrect haplotype configurations.

By considering our inference process, we can see that the parameters that affect the number of possible haplotype or grandparent configurations directly affect the number of solutions. The results from the experiment clearly show the effects of the number of members, the number of loci, and the rate of missing alleles on the number of solutions. Increasing in the values of these parameters would lead to the increasing of the number of solutions. Since all of these parameters directly relate to the number of possible haplotype or grandparent configurations in the inference process, thus, these results infer the correctness of our implementation for the aspect of the solution number.

Due to the fact that the problem of minimum-recombinant haplotype inference is NP-hard which mean that the computing time is exponential in the size of data, thus, our method could be used for limit size of data. Our method should be practical used for the pedigree of size not over than 25-30 members. And although our method has computing time linearly in the number of loci, there is still a limitation in the number of loci. From the experiment, we have seen that the number of solutions exponentially increase by the number of loci. When there are large number of solutions (about a million up), the memory would be insufficient for keeping the solutions. And so, this will cause the limitation on the number of loci.

Although the computing time which is one of the limitations of this method exponentially increases by the size of data, but it does not cause any critical breaks of the program. In contrast, the requirement of memory for keeping the possible cases of the haplotype and grandparent configurations during the computation process can cause a critical break of the program when the use of memory exceed the maximum limit. The size of pedigree, the number of loci and the rate of missing alleles are the main parameters that relate exponentially to the required size of the memory. Thus, for a

particular input data, these parameters should be considered before running the program in order to avoid the problem of insufficient memory.



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER VII

CONCLUSION

7.1 Conclusion

This work mainly intends to improve the current dynamic programming method for the minimum-recombinant haplotype inference problem so that it works efficiently with respect to the computing time and size of data (to use for long haplotypes), and also, able to work with the data containing some missing alleles or a few of Mendelian inconsistent alleles. This improvement is an extending of the limitations in the current methods in order to make the computational methods for the problem of haplotype inference on a pedigree more practical in the real-life usages.

We have solved the problem of the Mendelian inconsistent alleles by assume that the optimal haplotype configurations must have minimum number of inconsistent alleles. We have organized the local inference rules so that it can be able to cope with inconsistent genotypes. And also, we have developed the method for enumerating the possible haplotype configurations that have minimum number of inconsistent alleles. The imputation of missing alleles is also incorporated with these local inference rules and the process of enumerating the possible haplotype configurations.

For the phase of finding the minimum-recombinant haplotype configurations, the current algorithm used the pedigree haplotype/grandparent configuration (one configuration represent by a pedigree haplotype configuration with one of its corresponding pedigree grandparent configurations) as a computing case in a dynamic programming procedure. But in our algorithm, we have made an improvement by using the pedigree grandparent configuration instead. In our observation, this can reduce the number of computing cases by about 40%. We have reduced some redundant computations in this dynamic programming phase by perform pre-

computation for the computing of the recombination points in each family. And also, we have used the bounding technique to exclude some invalid nodes in the dynamic programming process. As from the experimental results, we can clearly conclude that these improvements cause our method to outperform both dynamic programming algorithms in PedPhase for a case of inferring long haplotypes (more than 20 loci) in a moderate-size pedigree (15-25 members).

Other than the efficiency testing, we also tested for the effect of each parameter to the computing time, the accuracy, and the number of solutions. Most of these testing apparently show the reasonable results which indicate the correctness of our method.

7.2 Summary of Contributions

Our method can be extended the current methods of haplotype inference based on minimum recombination principle to be able to work efficiently with long haplotypes (more than 20 loci) in a moderate-size pedigree (15-25 members) and be able to handle the data that contain some missing alleles or a few of Mendelian inconsistent alleles. We have made some improvements that can reduce the computing time in the phase of finding the minimum-recombinant haplotype configuration compared with the original work proposed by Li and Jiang (2003). And also, this extension could be considered as the relaxation from the strictly consistent inference to be a maximum consistent inference, which is useful for the case that there are a few of inconsistent alleles occur in the pedigree data.

7.3 Further Works

Although our method can be fairly used with some practical sizes of the input data (with long haplotypes and moderate size of pedigrees); however, it is occasionally very time consuming and often faces the problem of insufficient memory.

Also, inferring haplotype using only the assumption of minimum recombinants often results in very large number solutions. So, it is clearly that we need more additional criterions or hypotheses that can exclude more invalid cases of haplotype or grandparent configurations during the computation process, and also, can reduce the number of solutions. And the intensive consideration of how to set the maximum limit for each parameter in the computational process in order to avoid the problem of insufficient memory is also necessary for further investigation. Furthermore, we have seen in the experimental results that the DP-by-Member algorithm can work very well for any practical pedigree size with a haplotype length not over than 20 loci. Since the long haplotypes in practical data may not be much longer than 20 loci, thus, it would be interesting if we can improve the DP-by-Member algorithm so that it can work with the haplotypes of about 20-30 loci in length.

Finding the haplotype configurations with minimum number of recombinants can also be very useful for detecting the presence of genotyping errors in pedigrees. Generally, the genotyping errors are detected by checking for the occurrence of genotypes that is not consistent with Mendelian inheritance. And for the errors that do not cause inconsistencies in Mendelian inheritance, the technique that is typically used is to detect the presence of an excess of recombination events among closely spaced markers. By applying any particular algorithm used for solving the MRHC problem, if no zero-recombinant pedigree haplotype configuration exists, the algorithm can find pedigree haplotype configurations with the smallest number of recombinants and identify any haplotypes that contain obligate double recombinants. If there are any double recombinants occurring in a data, it means there must be some genotyping errors. However, identifying for a particular allele that is exactly an error allele (for both consistent and inconsistent errors) is still a difficult task. In this work, we have developed an inference algorithm which allows the occurrence of a few inconsistent alleles that may caused from the genotyping errors. But our algorithm does not directly investigate in the detection of the error alleles. So it would be interesting to

take more investigation in applying our method for detecting the exact error alleles in the genotyping data.

7.4 Recommendation

From the experiment, the characteristic of input data have great significant effect on the computing time, the memory usage, the accuracy, and the number of solutions. The investigation of the natural characteristic of real data is important for measuring the reliability and the feasibility of our method for the real-life usages. But in the scope of this work, we only perform experiment on the simulated data because it hardly to collect many of the real pedigree data. So, if possible, it is highly recommend for testing this method with the real data. The experimental results from the real data should be better used to evaluate the truly feasibility of this method than testing on only the simulated data.



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

REFERENCES

- Aceto, L., Hansen, J. A., Ingolfsdottir, A., Johnsen, J., and Knudsen, J. (2003). The Complexity of Checking Consistency of Pedigree Information and Related Problems. Proceedings of the 8th Italian Conference on Theoretical Computer Science (ICTCS'03), pp. 174-187.
- Bonizzoni, P., Vedova, G. D., Dondi, R., and Li, J. (2003). The haplotyping problem: An overview of computational models and solutions. Journal of Computer Science and Technology 18: 675-688.
- Daly, M. J., Rioux, J. D., Schaffner, S. F., Hudson, T. J., and Lander, E. S. (2001). High-resolution haplotype structure in the human genome. Nature Genetics 29: 229-232.
- Doi, K., Li, J., and Jiang, T. (2003). Minimum recombinant haplotype configuration on tree pedigrees. Proceedings of the 3rd Annual Workshop on Algorithms in Bioinformatics (WABI'03): 339-353.
- Douglas, J. A., Boehnke, M., Gillanders, E., Trent, J. M., and Gruber, S. B. (2001). Experimentally-derived haplotypes substantially increase the efficiency of linkage disequilibrium studies. Nature Genetics 28: 361-364.
- Eskin, E., Halperin, E., and Karp, R. M. (2003). Efficient reconstruction of haplotype structure via perfect phylogeny. Bioinformatics and Computational Biology 1: 1-20.
- Fallin, D., and Schork, N. J. (2000). Accuracy of haplotype frequency estimation for biallelic loci, via the expectation-maximization algorithm for unphased diploid genotype data. American Journal of Human Genetics 67: 947-959.
- Gabriel, S. B., Schaffner, S. F., Nguyen, H., Moore, J. M., Roy, J., Blumenstiel, B., Higgins, J., DeFelice, M., Lochner, A., Faggart, M., Liu-Cordero, S. N., Rotimi, C., Adeyemo, A., Cooper, R., Ward, R., Lander, E. S., Daly, M. J., Altshule, D. (2002). The structure of haplotype blocks in the human genome. Science 296: 2225-2229.

- Gordon, D., Heath, S. C., and Ott, J. (1999). True pedigree errors more frequent than apparent errors for single nucleotide polymorphisms. Human Heredity 49: 64-70.
- Gusfield, D. (2001). Inference of haplotypes from samples of diploid populations: complexity and algorithms. Journal of Computational Biology 8: 305-323.
- Halliburton, R. (2004). Introduction to population genetics. Upper Saddle River, NJ: Pearson Prentice Hall.
- Jenkins, S., and Gibson, N. (2002). High-throughput SNP genotyping. Comparative and Functional Genomics 3: 57-66.
- Kang, H., Qin, Z. S., Niu, T., and Liu, J. S. (2004). Incorporating genotyping uncertainty in haplotype inference for single-nucleotide polymorphisms. American Journal of Human Genetics 74: 495-510.
- Li, J., and Jiang, T. (2003). Efficient inference of haplotypes from genotypes on a pedigree. Journal of Bioinformatics and Computational Biology 1: 41-69.
- Li, J., and Jiang, T. (2004). An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming. Proceedings of the 8th Annual Conference on Research in Computational Molecular Biology (RECOMB'04): 101-110.
- Lin, S., and Speed, T. P. (1997). An algorithm for haplotype analysis. Journal of Computational Biology 4: 535-46.
- MichalatosBeloin, S., Tishkoff, S. A., Bentley, K. L., Kidd, K. K., and Ruano, G. (1996). Molecular haplotyping of genetic markers 10 kb apart by allelic-specific long-range PCR. Nucleic Acids Research 24: 4841-4843.
- O'Connell, J. R., and Weeks, D. E. (1998). PedCheck: A Program for Identification of Genotype Incompatibilities in Linkage Analysis. American Journal of Human Genetics 63: 259-266.
- O'Connell, J. R., and Weeks, D. E. 1999. An optimal algorithm for automatic genotype elimination. American Journal of Human Genetics 65: 1733-1740.

- Patil, N., Berno, A. J., Hinds, D. A., Barrett, W. A., Doshi, J. M., Hacker, C. R., Kautzer, C. R., Lee, D. H., Marjoribanks, C., McDonough, D. P., Nguyen, B. T. N., Norris, M. C., Sheehan, J. B., Shen, N., Stern, D., Stokowski, R. P., Thomas, D. J., Trulson, M. O., Vyas, K. R., Frazer, K. A., Fodor, S. P. A., and Cox, D. R. (2001). Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. Science 294: 1719-1723.
- Peltonen, L., Palotie, A., and Lange, K. (2000). Use of population isolates for mapping complex traits. Nature Reviews Genetics 1: 182-190.
- Qian, D., and Beckmann, L. (2002). Minimum-recombinant haplotyping in pedigrees. American Journal of Human Genetics 70: 1434-1445.
- Sebastiani, P., Lazarus, R., Weiss, S. T., Kunkel, L. M., Kohane, I. S., and Ramoni, M. F. (2003). Minimal haplotype tagging. PNAS 100: 9900-9905.
- Sobel, E., Papp, J. C., and Lange, K. (2002). Detection and Integration of Genotyping Errors in Statistical Genetics. American Journal of Human Genetics 70: 496-508.
- Stephens, M., Donnelly, P. (2003). A comparison of bayesian methods for haplotype reconstruction from population genotype data. American Journal of Human Genetics 73: 1162-9.
- Stephens, M., Smith, N. J., and Donnelly, P. (2001). A new statistical method for haplotype reconstruction from population data. American Journal of Human Genetics 68: 978-989.
- Tapadar, P., Ghosh, S., and Majumder P. P. (2000). Haplotyping in pedigrees via a genetic algorithm. Human Heredity 50: 43-56.
- The international HapMap consortium. (2003). The international HapMap project. Nature 426: 789-796.
- The International SNP Map Working Group. (2001). A map of human genome sequence variation containing 1.42 million single nucleotide polymorphisms. Nature 409: 928-933.
- Winston, W. L. (2004). Operations research: applications and algorithms. 4th edition. Belmont, CA: Thomson Brooks/Cole.

Zhoa, H., Pfeiffer, R., and Gail, M. H. (2003). Haplotype analysis in population genetics and association studies. Pharmacogenomics 4: 171-178.



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย



APPENDICES

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

APPENDIX A

SIMULATING THE PEDIGREE DATA

The data used in this work is in the form of a pedigree with the genotype and haplotype information for each individual. For this work, it is not feasible to collect a real data so that they are adequate to our experiment. So we have to use a simulated data instead. In order to simulate the pedigree data, we must concern with the concept of population genetics. We just use a simple population genetic model based on the basic principle of the Mendelian genetics to construct a genealogy with the genotype and haplotype information. And after a large genealogy of many generations has been created, the pedigree of any interested size could be sampled from this genealogy.

In our simulation model, we assume that the population evolves from generation to generation in a discrete time. Firstly, the initial generation will be created with a specific population size. Then the mating will occur randomly in a population and the offspring would be randomly created from each couple. During the creation of each offspring, the haplotype information of the parents is transferred to the offspring. In the transferring process, the point mutations and recombination events could possibly occur, which will cause the variation of the haplotypes. After all offspring of a current generation has been created, these offspring will be considered as a new population for a successive generation. The details for each parameter used in this simulation model are described as follows.

The Parameters for Simulating a Genealogy

1. Initial Size of the Population (size of the first generation)
2. Sex Rate – a ratio between the males and the females (used at the creation of the offspring)
3. Mating Rate – a ratio between the mating couples and the singles
4. Breeding Rate – the frequency of each number of the offspring in each family

5. Number of Alleles – the number of different alleles per locus
5. Haplotype Length – the number of loci per haplotype
6. Mutation Rate – the rate that one point mutation can occur per locus
7. Recombination Rate – the rate that one or more recombination events can occur per locus

The Parameters for Creating a Pedigree

1. Loop – used for selecting whether the pedigree has loop or not
2. Number of generations – the number of generations in a pedigree
3. Number of members – the number of members in a pedigree
4. Missing Rate – the rate of missing alleles in the genotype data (the alleles that have not been specified)
5. Number of Inconsistent Alleles – the number of inconsistent alleles in a genotype data which will be randomly set by the specific number

Using above parameters, we can control various characteristics of the input data which will be used in the experiment. The simulated pedigree data from the simulation program will have both genotype and haplotype information. The genotype data will be used as the input data and the haplotype data will be used for checking the accuracy. The example of the pedigree data generated from our program can be seen in Figure 4.5 and Figure 4.6 in Chapter 4 (the diagram pictures are manually created).

There are some parameters that were set with the constant values throughout the experiment. These parameters are set as follows.

1. The Initial size of the population is set as 100.
2. The sex rate is set as 0.5 (rate of the male offspring equal to the female offspring).
3. The mating rate is set as 0.8 (there are about 20% of population that do not have mates).

4. The breeding rate is set as follow. The possible numbers of the offspring that each couple can breed are 0, 1, 2, 3, and 4. The probabilities for each number of the offspring are set by the frequency ratio as 1:4:12:10:8 respectively.

To create a pedigree from p generations, we will simulate the genealogy with 10 generations at first. Next, we will simulate p successive generations. The specific pedigree will be created by sampling one couple in generation 11 and getting all of its successors (and also some of the parents of these successors). This method will result in the pedigree with a random size of members. To create a pedigree with a specific size of members, we will repeat the creation of the pedigree as described above until we have a pedigree with a specific size of members.

In the experiment, there is the case that uses the rate of homozygous loci as a control parameter. We control the rate of homozygous loci by varying the mutation rate. The haplotypes in the initial generation will be set with the same values for all members and the successive generations will be generated with the occurrence of the mutation events corresponding to the specific mutation rate. The resulted pedigree will have a number of homozygous loci related to the mutation rate.

APPENDIX B

USING THE PROGRAMS

In this work, we developed two programs to implement our proposed method. One is used for simulating a pedigree data and another one is used for solving the MRHC problem. These two programs can be executed using a command prompt in a DOS mode on Windows. The parameters for each program can be adjusted by editing the configuration file. The details of these two programs are described as follows.

1. The Program for Simulating a Pedigree Data

This program is used to create the pedigree data using the method described in an appendix A. Each parameter can be specified by editing the value in the configuration file ("config.txt"). One can simulate the pedigree data by typing the command "simulate" in the current directory that contains the program "simulate.exe" and the configuration file "config.txt". The pedigree data generated from this program have three different file formats as follows.

1. ped<x>.dat – a binary file that keeps a pedigree data in our specific data structure.
2. ped<x>.txt – a text file that represents a pedigree data in our specific representation.
3. pedphase<x>.txt – a text file that represents a pedigree data in the format similar to the input file of the program PedPhase.

Note that, <x> represents the number (ID) of a pedigree. An example of the pedigree data that consists of 15 members in 3 generations in a file format "ped<x>.txt" is shown below.

An Example of the Pedigree Data File

#Members : 15

Generation : 1

Gen 1 : 1)

ID : 1 Father : 0 Mother : 0 Mate : 2

Offspring : 5, 6, 7, 8,

Father Grand : 2222222222

Father Hap : 2121211212

Mother Hap : 1222121122

Mother Grand : 1111111111

Gen 1 : 2)

ID : 2 Father : 0 Mother : 0 Mate : 1

Offspring : 5, 6, 7, 8,

Father Grand : 2222222222

Father Hap : 2112211222

Mother Hap : 2212111212

Mother Grand : 1111111111

Generation : 2

Gen 2 : 1)

ID : 3 Father : 0 Mother : 0 Mate : 7

Offspring : 12, 13, 14, 15,
 Father Grand : 2222222222
 Father Hap : 1121222112
 Mother Hap : 2112211222
 Mother Grand : 1111111111

Gen 2 : 2)
 ID : 4 Father : 0 Mother : 0 Mate : 5
 Offspring : 9, 10, 11,
 Father Grand : 1111111222
 Father Hap : 1222121112
 Mother Hap : 2112211211
 Mother Grand : 2222222222

Gen 2 : 3)
 ID : 5 Father : 2 Mother : 1 Mate : 4
 Offspring : 9, 10, 11,
 Father Grand : 2222222222
 Father Hap : 2212111212
 Mother Hap : 2121211212
 Mother Grand : 1111111112

Gen 2 : 4)
 ID : 6 Father : 2 Mother : 1 Mate : 0
 Offspring :
 Father Grand : 1111111111
 Father Hap : 2112211222
 Mother Hap : 1222121122
 Mother Grand : 2222222222

Gen 2 : 5)

ID : 7 Father : 2 Mother : 1 Mate : 3

Offspring : 12, 13, 14, 15,

Father Grand : 2222222222

Father Hap : 2212111212

Mother Hap : 1222121122

Mother Grand : 2222222222

Gen 2 : 6)

ID : 8 Father : 2 Mother : 1 Mate : 0

Offspring :

Father Grand : 1111111111

Father Hap : 2112211222

Mother Hap : 2121211212

Mother Grand : 1111111111

Generation : 3

Gen 3 : 1)

ID : 9 Father : 4 Mother : 5 Mate : 0

Offspring :

Father Grand : 2222222222

Father Hap : 2112211211

Mother Hap : 2212111212

Mother Grand : 1111111111

Gen 3 : 2)

ID : 10 Father : 4 Mother : 5 Mate : 0

Offspring :

Father Grand : 1111111222

Father Hap : 1222121211

Mother Hap : 2121211212

Mother Grand : 2222222222

Gen 3 : 3)

ID : 11 Father : 4 Mother : 5 Mate : 0

Offspring :

Father Grand : 2222222222

Father Hap : 2112211211

Mother Hap : 2122111212

Mother Grand : 2221111111

Gen 3 : 4)

ID : 12 Father : 7 Mother : 3 Mate : 0

Offspring :

Father Grand : 2222222222

Father Hap : 1222121122

Mother Hap : 1121222112

Mother Grand : 1111111112

Gen 3 : 5)

ID : 13 Father : 7 Mother : 3 Mate : 0

Offspring :

Father Grand : 1111111111

Father Hap : 2212111212

Mother Hap : 2112211222

Mother Grand : 2222222222

Gen 3 : 6)

ID : 14 Father : 7 Mother : 3 Mate : 0

Offspring :

Father Grand : 2222222222

Father Hap : 1222121122

Mother Hap : 2112211222

Mother Grand : 2222222222

Gen 3 : 7)

ID : 15 Father : 7 Mother : 3 Mate : 0

Offspring :

Father Grand : 1111111111

Father Hap : 2212111212

Mother Hap : 1121222112

Mother Grand : 1111111111

2. The Program for Solving the MRHC Problem

This program is used for finding the haplotype solutions from the input pedigree data. Each parameter can be specified by editing the value in the configuration file ("config.txt"). One can execute this program by typing the command "mrhc <InputFile>" in the current directory that contains the program "mrhc.exe" and the configuration file "config.txt". Note that, this program will generate the output file of only one solution. The output solution will be kept in the file "solution.txt" and "result.log".

The input file for this program is in a format similar to the input file format of the program PedPhase. The input file can be prepared by using any file editors such as Notepad on Windows. The structure of the input file is simple. Each line represents one member. Different fields are separated using Tab. Each line consists of the following fields:

FamilyID MemberID FatherID MotherID Gender AffectionStatus LiabilityClass
 Allele11 Allele12 Allele21 Allele22....

FamilyID and MemberID are natural numbers (in our program, FamilyID is ignored). FatherID and MotherID are the current member's parents' MemberID and are set to 0 if the current member does not have parent information available in the pedigree. For Gender, 1 stands for male and 2 stands for female. AffectionStatus and LiabilityClass are reserved fields and are ignored in our program. Allele numbers (Allele11 Allele12 Allele21 Allele22 ...) are non-negative integers where 0 stands for missing value.

Note that, in our program, each member that has a mate must have the offspring and each offspring must have both father and mother. The order of MemberIDs must be ascending by the generation (from the first generation to the last generation). An example of the input file and the corresponding output files are shown below.

An Example of the Input File

0	1	0	0	2	0	0	2	1	1	2	2
	2	1	2	2	1	1	2	1	1	2	1
	1	2	2	2							
0	2	0	0	1	0	0	2	2	1	2	1
	1	2	2	2	1	1	1	1	1	2	2
	2	1	2	2							
0	3	0	0	2	0	0	1	2	1	1	2
	1	1	2	2	2	2	1	2	1	1	2
	1	2	2	2							
0	4	0	0	1	0	0	1	2	2	1	2
	1	2	2	1	2	2	1	1	1	1	2
	1	1	2	1							

0	5	2	1	2	0	0	2	2	2	1	1
	2	2	1	1	2	1	1	1	1	2	2
	1	1	2	2							
0	6	2	1	1	0	0	2	1	1	2	1
	2	2	2	2	1	1	2	1	1	2	1
	2	2	2	2							
0	7	2	1	1	0	0	2	1	2	2	1
	2	2	2	1	1	1	2	1	1	2	1
	1	2	2	2							
0	8	2	1	1	0	0	2	2	1	1	1
	2	2	1	2	2	1	1	1	1	2	2
	2	1	2	2							
0	9	4	5	2	0	0	2	2	1	2	1
	1	2	2	2	1	1	1	1	1	2	2
	1	1	1	2							
0	10	4	5	1	0	0	1	2	2	1	2
	2	2	1	1	2	2	1	1	1	2	2
	1	1	1	2							
0	11	4	5	2	0	0	2	2	1	1	1
	2	2	2	2	1	1	1	1	1	2	2
	1	1	1	2							
0	12	7	3	2	0	0	1	1	2	1	2
	2	2	1	1	2	2	2	1	2	1	1
	2	1	2	2							
0	13	7	3	1	0	0	2	2	2	1	1
	1	2	2	1	2	1	1	1	1	2	2
	1	2	2	2							
0	14	7	3	1	0	0	1	2	2	1	2
	1	2	2	1	2	2	1	1	1	1	2
	2	2	2	2							

0	15	7	3	1	0	0	2	1	2	1	1
	2	2	1	1	2	1	2	1	2	2	1
	1	1	2	2							

An Example of the Output File

#Members : 15

Generation : 1

Gen 1 : 1)

ID : 1 Father : 0 Mother : 0 Mate : 2

Offspring : 5, 6, 7, 8,

Father Grand : 0000000000

Father Hap : 2121211212

Mother Hap : 1222121122

Mother Grand : 0000000000

Gen 1 : 2)

ID : 2 Father : 0 Mother : 0 Mate : 1

Offspring : 5, 6, 7, 8,

Father Grand : 0000000000

Father Hap : 2212111212

Mother Hap : 2112211222

Mother Grand : 0000000000

Generation : 2

Gen 2 : 1)

ID : 3 Father : 0 Mother : 0 Mate : 7

Offspring : 12, 13, 14, 15,

Father Grand : 0000000000

Father Hap : 1121222112

Mother Hap : 2112211222

Mother Grand : 0000000000

Gen 2 : 2)

ID : 4 Father : 0 Mother : 0 Mate : 5

Offspring : 9, 10, 11,

Father Grand : 0000000000

Father Hap : 1222121112

Mother Hap : 2112211211

Mother Grand : 0000000000

Gen 2 : 3)

ID : 5 Father : 2 Mother : 1 Mate : 4

Offspring : 9, 10, 11,

Father Grand : 1111111111

Father Hap : 2212111212

Mother Hap : 2121211212

Mother Grand : 1111111111

Gen 2 : 4)

ID : 6 Father : 2 Mother : 1 Mate : 0

Offspring :

Father Grand : 2222222222

Father Hap : 2112211222

Mother Hap : 1222121122

Mother Grand : 2222222222

Gen 2 : 5)

ID : 7 Father : 2 Mother : 1 Mate : 3

Offspring : 12, 13, 14, 15,

Father Grand : 1111111111

Father Hap : 2212111212

Mother Hap : 1222121122

Mother Grand : 2222222222

Gen 2 : 6)

ID : 8 Father : 2 Mother : 1 Mate : 0

Offspring :

Father Grand : 2222222222

Father Hap : 2112211222

Mother Hap : 2121211212

Mother Grand : 1111111111

Generation : 3

Gen 3 : 1)

ID : 9 Father : 4 Mother : 5 Mate : 0

Offspring :

Father Grand : 2222222222

Father Hap : 2112211211

Mother Hap : 2212111212

Mother Grand : 1111111111

Gen 3 : 2)

ID : 10 Father : 4 Mother : 5 Mate : 0

Offspring :

Father Grand : 1111111222

Father Hap : 1222121211

Mother Hap : 2121211212

Mother Grand : 2222222222

Gen 3 : 3)

ID : 11 Father : 4 Mother : 5 Mate : 0

Offspring :

Father Grand : 2222222222

Father Hap : 2112211211

Mother Hap : 2122111212

Mother Grand : 2221111111

Gen 3 : 4)

ID : 12 Father : 7 Mother : 3 Mate : 0

Offspring :

Father Grand : 2222222222

Father Hap : 1222121122

Mother Hap : 1121222112

Mother Grand : 1111111111

Gen 3 : 5)

ID : 13 Father : 7 Mother : 3 Mate : 0

Offspring :

Father Grand : 1111111111

Father Hap : 2212111212

Mother Hap : 2112211222

Mother Grand : 2222222222

Gen 3 : 6)

ID : 14 Father : 7 Mother : 3 Mate : 0

Offspring :

Father Grand : 2222222222

Father Hap : 1222121122

Mother Hap : 2112211222

Mother Grand : 2222222222

Gen 3 : 7)

ID : 15 Father : 7 Mother : 3 Mate : 0

Offspring :

Father Grand : 1111111111

Father Hap : 2212111212

Mother Hap : 1121222112

Mother Grand : 1111111111

An Example of the Output File from the Program PedPhase

0	1	0	0	2	0	0	2 1	1 2	2 2	1 2	2 1
	1 2	1 1	2 1	1 2	2 2	0000000000	0000000000				
0	2	0	0	1	0	0	2 2	1 2	1 1	2 2	2 1
	1 1	1 1	2 2	2 1	2 2	0000000000	0000000000				
0	3	0	0	2	0	0	1 2	1 1	2 1	1 2	2 2
	2 1	2 1	1 2	1 2	2 2	0000000000	0000000000				

0	4	0	0	1	0	0	1 2	2 1	2 1	2 2	1 2
	2 1	1 1	1 2	1 1	2 1	0000000000	0000000000				
0	5	2	1	2	0	0	2 2	2 1	1 2	2 1	1 2
	1 1	1 1	2 2	1 1	2 2	1111111111	0000000000				
0	6	2	1	1	0	0	2 1	1 2	1 2	2 2	2 1
	1 2	1 1	2 1	2 2	2 2	0000000000	1111111111				
0	7	2	1	1	0	0	2 1	2 2	1 2	2 2	1 1
	1 2	1 1	2 1	1 2	2 2	1111111111	1111111111				
0	8	2	1	1	0	0	2 2	1 1	1 2	2 1	2 2
	1 1	1 1	2 2	2 1	2 2	0000000000	0000000000				
0	9	4	5	2	0	0	2 2	1 2	1 1	2 2	2 1
	1 1	1 1	2 2	1 1	1 2	1111111111	0000000000				
0	10	4	5	1	0	0	1 2	2 1	2 2	2 1	1 2
	2 1	1 1	2 2	1 1	1 2	0000000111	1111111111				
0	11	4	5	2	0	0	2 2	1 1	1 2	2 2	2 1
	1 1	1 1	2 2	1 1	1 2	1111111111	1110000000				
0	12	7	3	2	0	0	1 1	2 1	2 2	2 1	1 2
	2 2	1 2	1 1	2 1	2 2	1111111111	0000000000				
0	13	7	3	1	0	0	2 2	2 1	1 1	2 2	1 2
	1 1	1 1	2 2	1 2	2 2	0000000000	1111111111				
0	14	7	3	1	0	0	1 2	2 1	2 1	2 2	1 2
	2 1	1 1	1 2	2 2	2 2	1111111111	1111111111				
0	15	7	3	1	0	0	2 1	2 1	1 2	2 1	1 2
	1 2	1 2	2 1	1 1	2 2	0000000000	0000000000				

VITAE

Mr. Amares Kotcharat was born in Phitsanulok in September 8, 1977. He received a bachelor's degree in Computer Engineer from Faculty of Engineer, Chulalongkorn University in 1999.



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย