

Algorithms for Haplotype Inference from Genotypes in the Presence of Recombination

Syedur Rahman
Wolfson College
University of Oxford

Supervised by Dr. Rune Lyngsoe,
Prof. Jotun Hein and Prof. Tom Melham
Oxford University Computing Laboratory and Dept. of Statistics

A dissertation submitted for the degree of
Master of Science in Computer Science

September 2006

Abstract

The haplotype inference (HI) problem is defined as the problem of inferring $2n$ haplotype pairs from n observed genotype vectors. The inference of haplotype information from genotype data (the latter of which is more readily available) is very useful in researching genes affecting health, disease and responses to drugs and environmental factors

The PPH or the Perfect Phylogeny Haplotype model assumes that inferred haplotypes from a sample can be derived using a single tree, i.e. a perfect phylogeny. However, there are biological events such as recombination that violate this model. Stochastic methods on the other hand can infer haplotypes despite recombination but they can be time consuming and their inferences often depend on the initial state randomly chosen during a run. The aim of this project was to analyse previous models and solutions to the haplotype inference problem and engineer algorithms that would infer haplotypes from genotypes in the presence of recombination and scale better in terms of time complexity.

One engineered algorithm, Disjoint Phylogeny Region Haplotyping (DPRH) uses the LPPH program (which performs PPH inference in linear time) to find disjoint regions of perfect phylogeny encompassing adjacent sites from a given sample of genotypes. Two versions of the algorithm, DPRH1 and DPRH2 use methods based on local greedy and pure parsimony approaches respectively to combine the regional solutions into complete inferred haplotypes.

Another algorithm Overlapping Phylogeny Region Haplotyping (OPRH) uses LPPH to find all maximal regions (that may include overlaps) of perfect phylogeny given a sample of genotypes. The algorithm then exploits the overlaps between these regions to help combine the partial PPH solutions for each region into the complete inferred haplotypes.

These algorithms were implemented in MATLAB and then along with two popular HI methods, fastPHASE and Haplotyper they were tested using datasets of varying population sizes, numbers of sites and recombination rates simulated with ms and their speeds and accuracies are compared. All three engineered algorithms seemed to usually operate slightly faster than Haplotyper and ten times faster than fastPHASE. OPRH failed in terms of accuracy for datasets of almost all configurations while DPRH1 gave decent inferences only when recombination rates were very low. DPRH2 on the other hand had levels of accuracy comparable to that of the established methods for samples produced with low to medium rates of recombination and ran with an empirical complexity of $O(m\alpha(n))$, where n is the number of individuals with m sites each and α is a very slowly growing function.

CONTENTS

1	INTRODUCTION	1
1.1	Haplotype Inference	1
1.2	Aims of this project	1
1.3	The MSc and this project	1
1.4	Structure of this document	2
2	BACKGROUND AND LITERATURE REVIEW	4
2.1	Preliminary Concepts and Haplotype Inference	4
2.2	Mutation, Recombination and Phylogenies	7
2.3	Perfect Phylogeny Haplotyping	11
2.3.1	LPPH Solution to Haplotype Inference	11
2.3.2	Near-Perfect Phylogeny Haplotyping	12
2.4	Clark's Algorithm and Maximum Resolution	12
2.5	Haplotype Inference by Pure Parsimony	13
2.6	Statistical Methods - PHASE and Haplotyper	13
3	DESCRIPTION OF PROPOSED METHODS	14
3.1	Motivations and Use of PPH	14
3.2	Disjoint Phylogeny Region Haplotyping (DPRH)	14
3.2.1	Aligning Regional Solutions via DPRH1	17
3.2.2	Aligning Regional Solutions via DPRH2	19
3.3	Overlapping Phylogeny Region Haplotyping (OPRH)	20
4	IMPLEMENTATION OF ENGINEERED ALGORITHMS	23
4.1	Development Environment	23
4.2	Details of all functions	24
4.3	Integration of LPPH	27
4.4	Disjoint Phylogeny Region Haplotyping	28
4.4.1	Implementation of DPRH1	29
4.4.2	Implementation of DPRH2	30
4.5	Overlapping Phylogeny Region Haplotyping	31
4.6	Computing Accuracy of Inferences	33
5	TESTING AND EVALUATION	37
5.1	The Test Plan and its Implementation	37
5.2	Evaluation of Accuracy	39
5.2.1	Varying Recombination Rates	39
5.2.2	Varying Number of Individuals	40
5.2.3	Varying Number of Sites	41
5.3	Evaluation of Speed	42
5.3.1	Varying Number of Individuals	43
5.3.2	Varying Number of Sites	44
5.4	Summary of Evaluation	44
6	CONCLUSION	46
6.1	Summary of Dissertation	46
6.2	Research Results	46
6.3	Further Work	47
	BIBLIOGRAPHY	48
	APPENDIX A - CODE LISTING	50
	APPENDIX B - SAMPLE RUNS OF IMPLEMENTATION	65
	APPENDIX C - RESULTS FROM EVALUATION	70

LIST OF FIGURES

Figure 2.1: Point Mutations: Transitions and Transversions	7
Figure 2.2: Recombination between two sequences	8
Figure 2.3: Haplotypes inherited by a child via recombination	8
Figure 2.4: Phylogeny showing evolution of six species	9
Figure 2.5: Phylogeny with proper time axis	9
Figure 2.6: A phylogeny satisfying the infinite sites model	10
Figure 2.7: Effect of recombination on evolutionary history	10
Figure 2.8: Phylogenies of parts of a sequence subject to recombination	11
Figure 3.1: Evolutionary histories of individuals and their haplotypes	15
Figure 3.2: Disjoint Phylogeny Region Haplotyping (DPRH) - An example	16
Figure 3.3: Overlapping Phylogeny Region Haplotyping (OPRH) - An example	21
Figure 5.1: Accuracy evaluation across varying recombination rates	40
Figure 5.2: Accuracy evaluation across varying number of individuals	41
Figure 5.3: Accuracy evaluation across varying number of sites	42
Figure 5.4: Speed evaluation across varying recombination rates	43
Figure 5.5: Speed evaluation across varying number of individuals	43
Figure 5.6: Speed evaluation across varying number of sites	44

LIST OF TABLES

Table 1.1: Timetable for activities related to this project	2
Table 2.1: Haplotypes, SNPs and their binary coding	4
Table 2.2: Genotype coding for haplotypes of individuals shown in Table 2.1	5
Table 2.3: Genotypes and all possible haplotype pairs	7
Table 3.1: Possible haplotype pairs and their frequencies in DPRH1 - An example	17
Table 3.2: Table showing regional solutions and overlaps from Figure 3.3	22
Table 4.1: Parameters and descriptions of all implemented functions	25
Table 4.2: Accuracy measures and variables returned by the evaluation function	33
Table 5.1: Classification of number of individuals, sites and recombination rates	38

Word Count: 18,201 words excluding appendices

1 Introduction

1.1 *Haplotype Inference*

Briefly the haplotype inference (HI) problem can be defined as the problem of inferring the pair of haplotype sequences of an individual from his/her genotype sequence, which is the more readily available combined information on the haplotype pair.

Haplotype information is considered quite useful for researchers in finding genes affecting health, disease and responses to drugs and environmental factors. The international HapMap project [13] is a multi-national effort in the field attempting to identify and catalogue genetic similarities and differences in human beings for such purposes.

1.2 *Aims of this project*

Several methods have been put forward that attempt to solve the haplotype inference problem. Dan Gusfield introduced and solved a perfect phylogeny solution (which assumes that the polymorphisms in a set of sequences can be explained via branches in a phylogenetic tree) to the HI problem in 2002 [9] and later with Zhentao Ding developed a linear time solution to it [5]. However, this algorithm is unlikely to produce solutions for diploid organisms such as humans where recombination occurs during reproduction thereby violating the perfect phylogeny model.

Most HI methods that can account for recombination are either incomplete or NP-hard [9]. Others use heuristics and statistical methods to infer haplotypes reasonably accurately but can still be very time consuming and their accuracies are often determined by the random initial state chosen by the expectation maximisation algorithm they use.

The aim of this project was to design and implement new algorithms that:

- Infer haplotypes from genotypes in the presence of recombination
- Scale better in terms of time complexity compared to current solutions

In order to evaluate the engineered algorithms, a thorough evaluation of their performance was planned. This would include testing these algorithms and a few currently popular statistical methods with multiple datasets of different configurations and comparing their performances in terms of accuracy and time.

1.3 *The MSc and this project*

This dissertation was written in order to satisfy the requirements of the M.Sc. in Computer Science at the Oxford University Computing Laboratory (OUCL). According to the M.Sc. handbook, the project is supposed to “demonstrate in the dissertation an appreciation of the role of methods studied in the course” [20].

This project can be regarded as one in Bioinformatics. Therefore the author was primarily supervised by Dr. Rune Lyngsoe and Professor Jotun Hein from the Oxford University Department of Statistics (DoS). Professor Tom Melham acted as the supervisor from the OUCL end. A project proposal was written and submitted to the OUCL academic administrator at the end of April 2006, after which the author officially started work on the project. This document, i.e. the dissertation based on the project was due on 1st September. This gave the author four months to work on the project. However owing to the assessment in the “Requirements Engineering” course, other than light reading the author was able to properly start work on the project in late May. A presentation on the proposal and the project’s progress was made to the head and other academic staff at the OUCL in June 2006 and at the Oxford

Centre for Gene Function in July 2006. The final timetable for the activities related to this project is shown in Table 1.1.

From	Until	Days	Activity
23 May 2006	06 June 2006	15	Literature Review
07 June 2006	24 June 2006	18	Design of Algorithms
25 June 2006	02 July 2006	8	Implementation I
03 July 2006	11 July 2006	9	Short Break
12 July 2006	25 July 2006	14	Implementation II
26 July 2006	10 August 2006	16	Testing/Evaluation
11 August 2006	25 August 2006	15	Writing Up

Table 1.1: Timetable for activities related to this project

The author initially gained interest in bioinformatics owing to taking the “Bioinformatics and Computational Biology” course taught by members of the DoS as an optional course for M.Sc. students at the OUCL and other departments. The course offered students with some basic background in genetics and covered computational topics in the field such as the construction of phylogenies, sequence alignment etc. This course was instrumental in the author’s decision to pursue a project in the field. Other courses such as “Intelligent Systems” helped the author better understand the Bayesian processes used by currently popular haplotype inference programs. The material from “Introduction to Specification” was used by the author to present many of the mathematical concepts related to HI and the algorithms engineered in formal notation which would be better understood by readers of this document. “Object Oriented Programming” helped the author gain more discipline in his programming and documentation even though no object oriented programming was conducted as part of this project. Synopses and details of these courses can be found at OUCL’s website, www.comlab.ox.ac.uk [21].

1.4 Structure of this document

The dissertation for this project is structured into six chapters. Chapter 1 has so far been a very brief introduction to haplotype inference and the purpose of this project. The structure of the remainder of this document is as follows:

Chapter 2 titled “Background and Literature Review” describes haplotype inference and the genetics behind it thoroughly to the reader and it formulates it as a computation problem. Past and current HI methods are also introduced and discussed briefly.

Chapter 3 titled “Description of Proposed Methods” describes the algorithms that were engineered by the author. Diagrams and simple examples of sequences are used to explain the operation of these algorithms and the motivations behind them.

Chapter 4 titled “Implementation of Engineered Algorithms” gives a comprehensive description of the implementations of the algorithm described earlier. A brief justification of the chosen development environment is also provided. Pseudocode and example sequences are also provided to explain the more complicated mathematical operations associated with the implementation.

Chapter 5 titled “Testing and Evaluation” introduces the reader to a thorough test plan, which includes a description of the datasets to be used and how they are obtained. It also introduces the statistical HI methods that are being tested along with the algorithms developed. The chapter ends with a comparison of performances in terms of accuracy and time across methods illustrated via graphs and tables.

Chapter 1: Introduction

Chapter 6 entitled “Conclusion” gives summaries of the preceding chapters and discusses the results obtained by this project. It includes suggestions to improve the engineered algorithms’ performances, many of which were not possible merely owing to time restrictions associated with the MSc project, and then goes on to conclude this dissertation.

Appendix A gives annotated code for all the functions written in MATLAB as part of this project.

Appendix B shows screen outputs of runs of the implemented algorithms with datasets of various configurations.

Appendix C is simply a large table containing all the results obtained from the evaluation conducted as part of this project which is explained in Chapter 5.

2 Background and Literature Review

Even though this is a project in bioinformatics, according to the MSc guidelines [20] it must be assumed that the dissertation will be read by a computer scientist who may or may not have any knowledge of genetics. Thus, the first section of this chapter introduces the reader to the basic concepts in genetics that contribute to the haplotype inference problem. Once this is done, the HI problem is analysed and discussed exhaustively and is formulated as a mathematical/computational problem that computer scientists can relate better to.

The Perfect Phylogeny Haplotype model which is crucial to the development of the algorithms (described in Chapter 3) is also covered comprehensively. Other models and particular methods of haplotype inference are also discussed briefly.

2.1 Preliminary Concepts and Haplotype Inference

DNA, Haplotypes and Genotypes

Deoxyribonucleic acid (**DNA**) is a nucleic acid - usually in the form of a double helix - that contains the genetic instructions specifying the biological development of all cellular forms of life, and many viruses [18]. DNA is normally packed in the form of one or more macromolecules called **chromosomes**. Each chromosome appears as a pair in individuals and contains a number of **sites**. Sites are encoded by the possible bases present in the particular **nucleotide**, which are adenine (abbreviated **A**), cytosine (**C**), guanine (**G**) and thymine (**T**) in the case of DNA. Among the nucleotides, A and G are from a family of compounds called **purines** whereas C and T are **pyrimidines**.

Haplotypes can be defined as the genetic constitution of an individual chromosome. In diploid organisms such as human beings, each individual has two copies of each chromosome which may not be identical [23]. We refer to **haplotype data** as information from each copy of the chromosome and **genotype data** as information combined from both copies for an individual.

The regions of interest when studying populations are often sites of SNPs. A **SNP** or a **Single Nucleotide Polymorphism** is a single nucleotide or site where exactly two out of the four possible nucleotides (A, C, G and T) occur in the vast majority of the population. Therefore haplotype information for an individual can be written as a binary string of 0's and 1's.

Table 2.1 shows a simple coding of haplotypes for four individuals with just five SNPs each. Therefore there are eight sequences in total (a pair of haplotypes from each individual) and each sequence is five characters long.

Haplotype	Sequence in Nucleotides	Coding Used	Sequence in Binary
<i>Ind₁Hap₁</i>	AGACC	Site1: A-0, C-1 Site2: G-0, T-1 Site3: A-0, G-1 Site4: A-0, C-1 Site5: C-0, T-1	00010
<i>Ind₁Hap₂</i>	CGACC		10010
<i>Ind₂Hap₁</i>	AGAAC		00000
<i>Ind₂Hap₂</i>	CGAAT		10001
<i>Ind₃Hap₁</i>	ATGAT		01101
<i>Ind₃Hap₂</i>	AAGCC		00110
<i>Ind₄Hap₁</i>	ATGAT		01101
<i>Ind₄Hap₂</i>	AAACC		00010

Table 2.1: Haplotypes, SNPs and their binary coding

Chapter 2: Background and Literature Review

For the sequences shown in Table 2.1, only A and C usually occur at site 1, G and T at site 2, A and G at site 3, A and C at site 4 and C and T at site 5. Since different pairs of bases occur at different sites, different nucleotide-to-binary codings are used at different sites, e.g. A and C refer to 0 and 1 respectively in site 1 whereas G and T refer to 0 and 1 respectively in site 2. Please note that only one possible binary coding of the haplotype information is shown. Others are possible but it is always convenient to code bases 0 and 1 in accordance with the alphabetical order of the bases present in the particular site or SNP.

Therefore each individual in diploid organisms will have a pair of haplotype sequences i.e. two strings of 0's and 1's.

Let us assume, x is an individual with the following haplotype pair.

	Sites	1	2	3	4	5	6	7	8
Ind_xHap_1 :		0	1	0	0	1	1	1	0
Ind_xHap_2 :		0	0	0	1	0	1	1	0

Sites 1, 3, 6, 7 and 8 are called **homozygous sites** since they contain the same nucleotide in both sequences for individual x .

That is, the site y is homozygous for individual x if $Ind_xHap_1(y) = Ind_xHap_2(y)$

Sites 2, 4 and 5 are called **heterozygous sites** since they do not contain the same nucleotide in both sequences for individual x .

That is, the site y is heterozygous for individual x if $Ind_xHap_1(y) \neq Ind_xHap_2(y)$

The genotype information for an individual is coded such that a 0 represents a homozygous site where both nucleotides are 0 and a 1 represents a homozygous site where both nucleotides are 1. A 2 on the other hand represents a heterozygous site where one nucleotide is a 1 and the other is a 0 (in any order).

With the haplotypes given earlier, individual x 's genotype sequence would be as follows:

	Sites	1	2	3	4	5	6	7	8
Ind_xHap_1 :		0	1	0	0	1	1	1	0
Ind_xHap_2 :		0	0	0	1	0	1	1	0
Ind_xGen :		0	2	0	2	2	1	1	0

The following table shows the genotype coding for the four individuals shown earlier in Table 2.1

Haplotype	Sequence in Nucleotides	Coding Used	Sequence in Binary	Genotypes
Ind_1Hap_1	AGACC	Site1: A-0, C-1 Site2: G-0, T-1 Site3: A-0, G-1 Site4: A-0, C-1 Site5: C-0, T-1	00010	20010
Ind_1Hap_2	CGACC		10010	
Ind_2Hap_1	AGAAC		00000	20002
Ind_2Hap_2	CGAAT		10001	
Ind_3Hap_1	ATGAT		01101	02122
Ind_3Hap_2	AAGCC		00110	
Ind_4Hap_1	ATGAT		01101	02222
Ind_4Hap_2	AAACC		00010	

Table 2.2: Genotype coding for haplotypes of individuals shown in Table 2.1

Therefore, haplotype sequences for an individual can be thought of as a pair of binary vectors whereas genotype information as a vector on the alphabet $\{0, 1, 2\}$. The genotype data is more readily available however the haplotype information is more crucial in studying variations in populations for

the purposes of disease mapping, inferring population histories, etc. and this is where the haplotype inference problem arises.

Haplotype Inference

The **haplotype inference (HI)** problem is the problem of inferring $2n$ haplotype pairs from n observed genotype vectors i.e. there are n individuals in the sample [23]. As mentioned in Chapter 1, information from haplotypes can be quite useful for researchers but it is not as readily available as genotype data. Haplotype information can be instrumental in finding genes affecting health, disease and responses to drugs and environmental factors.

Let us assume, the genotypes for a sample of individuals is given as an $n \times m$ matrix G (where m is the number of sites and n is the number of individuals). Then for an individual x , the row $G(x, 1..m)$ represents his genotype information and $Ind_xHap_1(1..m)$ and $Ind_xHap_2(1..m)$ represent x 's pair of haplotype sequences to be inferred.

Therefore we can formulate the haplotype inference problem as:

- (1) $G(x, y)=0 \rightarrow Ind_xHap_1(y)=0 \wedge Ind_xHap_2(y)=0$
- (2) $G(x, y)=1 \rightarrow Ind_xHap_1(y)=1 \wedge Ind_xHap_2(y)=1$
- (3) $G(x, y)=2 \rightarrow (Ind_xHap_1(y)=1 \wedge Ind_xHap_2(y)=0) \vee (Ind_xHap_1(y)=0 \wedge Ind_xHap_2(y)=1)$

It is this non-determinism in (3) that is the core of the haplotype inference problem.

In table 2.2, $G = \begin{pmatrix} 20010 \\ 20002 \\ 02122 \\ 02222 \end{pmatrix}$

Individual 1 has just one heterozygous site. By using the HI inference equation, we can deduce that
 either $Ind_1Hap_1 = 10010$ and $Ind_1Hap_2 = 00010$
 or $Ind_1Hap_1 = 00010$ and $Ind_1Hap_2 = 10010$

However when inferring haplotypes for an individual, the order of the sequences in the pair is not taken into account. So in this case there is only one possibility with regard to the haplotype pair that can be inferred i.e. $\{00010, 10010\}$. Therefore genotypes of individuals with just one heterozygous site can be inferred to just one possible haplotype pair. Table 2.3 shows the possible inferences for all four individuals from Table 2.2.

Genotypes Sequence	Htrz. Sites	Possible Inferences of Haplotype Pairs	No of Solutions
20010	1	{00010, 10010}	1
20002	2	{00001, 10000} {00000, 10001}	2
02122	3	{00100, 01111} {00110, 01101} {00101, 01110} {00111, 01100}	4

02222	4	{00000, 01111} {00010, 01101} {00001, 01110} {00011, 01100} {00100, 01011} {00110, 01001} {00101, 01010} {00111, 01000}	8
-------	---	--	---

Table 2.3: Genotypes and all possible haplotype pairs

It can be deduced that the number of possible inferences increases exponentially with respect to the number of heterozygous sites present in an individual's genotype sequence. For an individual with x heterozygous sites the number of possible solutions is 2^{x-1} .

2.2 Mutation, Recombination and Phylogenies

During reproduction, the offspring inherits the genetic information from the parent(s). There are two events that may occur during this transfer of information: mutation and recombination

Mutation

Mutations can be described simply as changes to the genetic material. The causes of mutation include “copying errors in the genetic material during cell division and by exposure to radiation, chemicals (mutagens), or viruses, or can occur deliberately under cellular control during processes such as meiosis or hypermutation” [27]. What makes mutations significant is that natural selection removes the less favourable mutations while the more favourable ones tend to accumulate [28].

Mutations can be **insertion** and **deletion mutations** (where one or more nucleotides are added or deleted) and **point mutations**. For the purposes of this project, we are only concerned with point mutations during which a single nucleotide is exchanged for another. Mutations where purine is exchanged for a purine ($A \leftrightarrow G$) or a pyrimidine is exchanged for a pyrimidine ($C \leftrightarrow T$) are called **transitions**. Mutations involving purines and pyrimidines ($A/G \leftrightarrow C/T$) are called **transversions**.



Figure 2.1: Point Mutations: Transitions and Transversions

Figure 2.1 shows two point mutations. *seqM2* is derived from *seqM1* via a transition ($A \rightarrow G$) occurring at site 3 and *seqM4* is derived from *seqM3* via a transversion ($T \rightarrow G$) at site 7.

Recombination

Genetic **recombination** or **cross-over** is the transmission-genetic process by which an offspring inherits a combination of the genetic material of its two parents. In other words, it can be described as “the physical breakage and exchange of segments of DNA during reproduction” [28].

Figure 2.2 shows two sequences *seqR1* and *seqR2* which recombine to form *seqR3*. In this example *seqR3* inherits the first five sites from *seqR1* and the last three sites from *seqR2*.

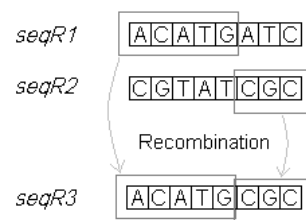


Figure 2.2: Recombination between two sequences

Figure 2.3 shows how segments of the pair of haplotypes of one parent combine to produce one of the child's haplotypes during *meiosis* (cell division in sexually reproducing organisms).

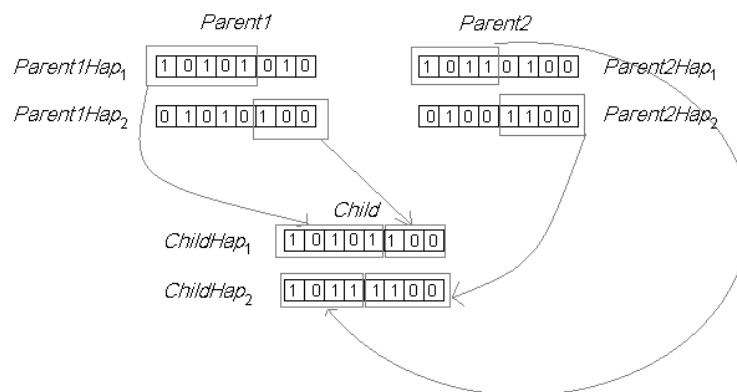


Figure 2.3: Haplotypes inherited by a child via recombination

Each haplotype inherited by the child is a recombination of the haplotypes of its parents. In Figure 2.3, *Child* is an offspring of *Parent1* and *Parent2*. In this example, *ChildHap₁* is entirely derived from the haplotypes of *Parent1* and is a combination of the first five sites of *Parent1Hap₁* and the last three sites of *Parent1Hap₂*, whereas *ChildHap₂* is entirely derived from *Parent2*'s haplotypes and is a combination of the first four sites *Parent2Hap₁* and the last four sites *Parent2Hap₂*.

Phylogenies and Evolutionary Histories

A **phylogeny** is a tree showing the evolutionary interrelationships among various species or other entities that are believed to have a common ancestor [15]. Figure 2.4 shows a very simple example showing the evolutionary history of six species.

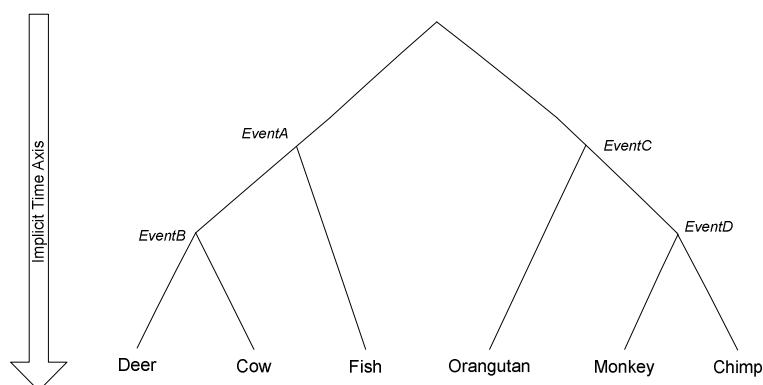


Figure 2.4: Phylogeny showing evolution of six species

Figure taken from [15]

A time axis is implicit down a path but it is not defined across paths. E.g. it is obvious that *EventB* where Deer and Cow diverged from a common ancestor occurred after *EventA* i.e. where Cow and Deer's common ancestor and Fish split from the root ancestor. Similarly one can tell that Monkey and Chimp diverged from a common ancestor (*EventD*) after Orangutan and this ancestor diverged from the root. It is not possible however to tell whether *EventC* occurred before or after *EventA*. On the other hand, even though *EventC* seems higher than *EventB* one still can not tell which occurred first, since it is possible with a proper time axis (Figure 2.5), the path leading to *EventC* took longer than the path leading to *EventB* (via *EventA*)

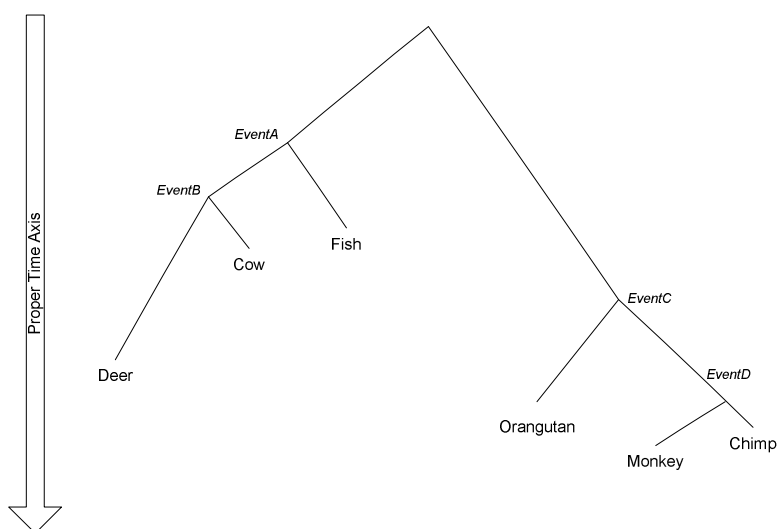


Figure 2.5: Phylogeny with proper time axis

Infinite Sites Model

The **infinite sites model**, proposed by Tomoko Ohta and Motoo Kimura in 1971 [19], assumes that each mutation occurs at a new site. The basic concept behind the infinite sites model is that with an infinite number of sites, the possibility of a mutation occurring at the same site twice is zero. Under this model, a site can label at most one branch (representing one mutation) in the tree representing the evolutionary history of a set of sequences. Many haplotype inference techniques are based on this assumption.

Figure 2.6 shows an illustration of the infinite sites model using four sequences (of individuals or species) $s1$, $s2$, $s3$ and $s4$ each with three sites. Each site changes as new edges are formed on the tree. $s1$ and $s2$ share a common ancestor $a_{1,2}$ with a mutation between $a_{1,2}$ and $s2$ at site 1. $s3$ and $s4$ share a common ancestor $a_{3,4}$ with a mutation between $a_{3,4}$ and $s4$ at site 2. Therefore, $a_{1,2}$ and $a_{3,4}$ share a common ancestor at the root $a_{1,2,3,4}$ (with a mutation between $a_{1,2,3,4}$ and $a_{3,4}$ at site 3).

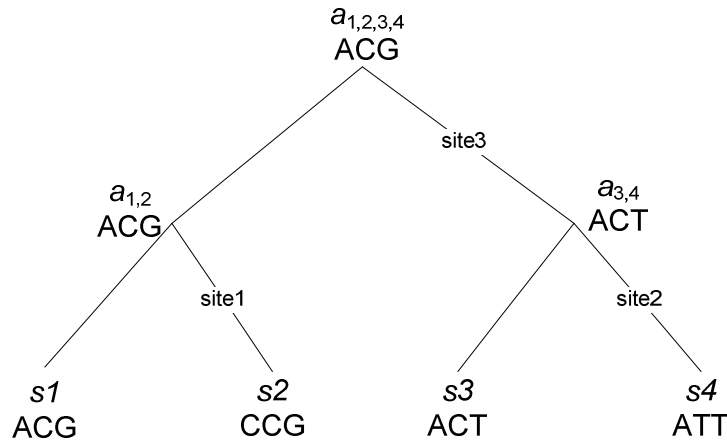


Figure 2.6: A phylogeny satisfying the infinite sites model

Evolutionary Histories and Recombination

In case of diploid organisms where each individual has two parents, a phylogeny can not be used to represent the history of sequences that contain recombination events. The graph in Figure 2.7 (which is not a phylogenetic tree) shows the evolutionary history of sequences $s1$, $s2$, $s3$ and $s4$ owing to a recombination event at rp .

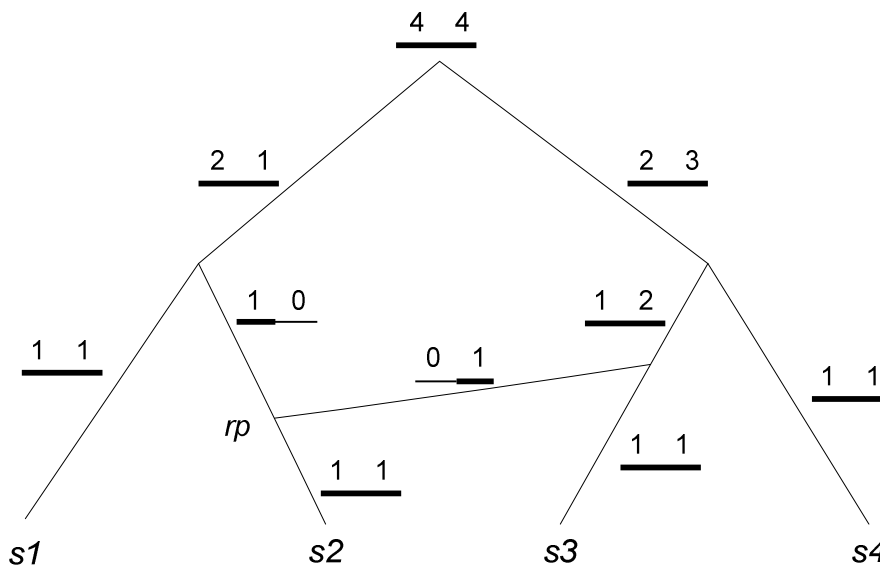


Figure 2.7: Effect of recombination on evolutionary history

Figure taken from [14]

Genetic material that is passed on to descendants is shown by thick lines and those that do not have any descendants in the sample are shown by thin lines. Let us assume the sequence is split into halves and the number on top of the genetic material on each half shows the number of descendants it has.

Ancestral Recombination Graphs or **ARGs** are often used instead to show the evolutionary histories of sequences that have been subject to recombination. Figure 2.7 is in fact an ARG where $s3$ and $s4$ share a common maternal and paternal ancestor thus the recombination is not evident. However, one of $s2$'s parents shares a common ancestor with $s1$ and the other parent shares it with $s3$. Let us assume the left half of $s2$'s genetic material is inherited from the ancestor it shares with $s1$ and the right half from the ancestor with $s3$. This recombination event at rp , splits the sequence into two parts.

The history of the left part of the sequence is described by a different phylogeny as compared to the history of the right part of the sequence. The history of the left part can be described by following the left going edge upward as shown in Figure 2.8a and the right part can be explained by following the right going edge upward as shown in Figure 2.8b.

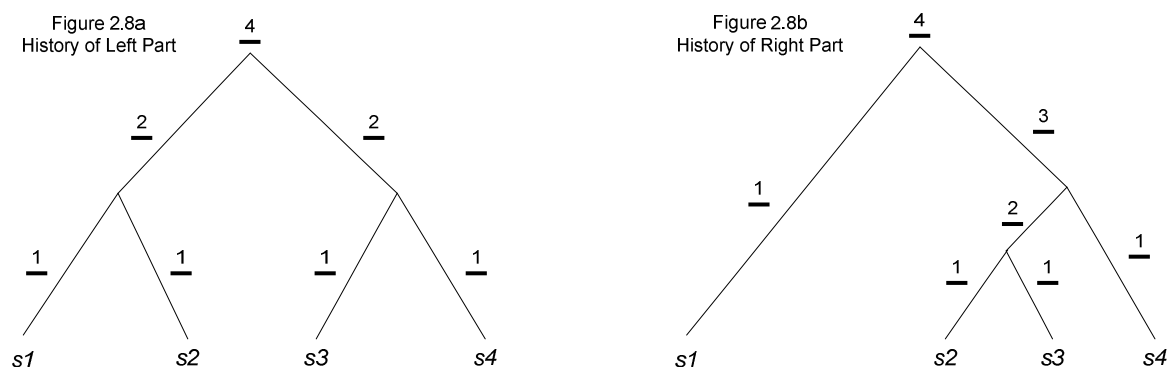


Figure 2.8: Phylogenies of parts of a sequence subject to recombination

2.3 Perfect Phylogeny Haplotyping

Perfect Phylogeny Haplotyping (PPH) is a method of haplotype inference based on a model which assumes that all the haplotypes in a sample are derived from a perfect phylogeny.

Dan Gusfield first introduced this model and formulated a PPH solution to haplotype inference in 2002 [9]. The solution is based on the **coalescent model** of haplotype evolution which uses the infinite sites assumption and states that in the absence of recombination the evolutionary history of $2n$ haplotypes can be displayed as a rooted tree with $2n$ leaves (with some ancestral sequence as the root) and where each of the m sites labels exactly one edge of the tree (representing a mutation) and the resulting sequences appear at its leaves.

Gusfield initially put forth a $O(nm\alpha(nm))$ where α is the extremely slowly growing inverse Ackerman function which reduced the PPH problem to a graph realisation problem [9]. Several programs such as GPPH [2], DPPH [1] and BPPH [3] were written the following year all of which had worst case running times of $O(nm^2)$.

2.3.1 LPPH Solution to Haplotype Inference

Finding a linear deterministic solution to the PPH problem remained an open problem despite a series of papers and interest in the field until Ding, Filkov and Gusfield suggested “a practical, deterministic linear-time algorithm based on a simple data-structure and simple operations on it” [5] in 2005.

The **LPPH** [5] or linear PPH algorithm uses a directed rooted graph called a “**shadow tree**” as its primary data structure. As the matrix of genotype vectors is processed by the algorithm new edges are

added to the shadow tree and information about old edges is updated. A shadow tree can be thought of as a description of a collection of trees. Individuals may be related to each other by many possible trees however there may be only one shadow tree describing their association. That is, each possible phylogeny is contained in (or described by) this shadow tree. Once the shadow tree has been created, the haplotypes for each individual can be easily inferred using his/her evolutionary relationships to others in the relevant phylogenies.

2.3.2 Near-Perfect Phylogeny Haplotyping

Generally in case of diploid organisms, most solutions will not fit the PPH model since each individual has two parents and recombination occurs during reproduction. A “**near-perfect phylogeny**” approach adopted by E. Eskin, E. Halperin and R.M. Karp led to the development of the **HAP** program [6].

Near Perfect Phylogeny is based on the assumption that most of the common haplotypes in a sample will fit a perfect phylogeny and therefore calculated modifications to the dataset haplotypes could be inferred using PPH [6]. HAP splits the genotypes into regions of a fixed number of adjacent sites. For each region, it determines a subset of the haplotypes that conform to a perfect phylogeny. These haplotypes are inferred using PPH and their solutions along with maximum-likelihood techniques are used to infer the rest of the haplotypes.

Algorithms that solve the HI problem given a sample where individuals do not conform to a perfect phylogeny are often referred to as **Imperfect Phylogeny Haplotyping (IPPH)** algorithms.

2.4 Clark’s Algorithm and Maximum Resolution

Clark’s method, put forward in 1990 [4], starts off by selecting genotype vectors that have only one possible inference (i.e. vectors with zero or one heterozygous sites) and attempts to resolve the remaining haplotypes using these initial resolved haplotypes.

Given a previously resolved vector R , Clark’s method uses the following rule to resolve a new vector NR from an ambiguous (unresolved) vector A :

“Suppose A is an ambiguous genotype vector with h ambiguous (heterozygous) sites and R is a resolved vector that is a haplotype in one of the 2^{h-1} potential resolutions of vector A . Then infer that A is the conflation of one copy of resolved vector R and another (uniquely determined) resolved vector NR . All of the ambiguous positions in A are set in NR to the opposite of the entry in R . Once inferred, vector NR is added to the set of known resolved vectors, and vector A is removed from the set of ambiguous vectors.” [11]

So for example, if one takes $A = 10122$ as an ambiguous vector with an already resolved vector $R = 10110$ where R may be an initial resolved haplotype or one resolved by previous application of the inference rule, one can infer NR as 10101.

There may however be several choices for R when attempting to infer any A and this will not only lead to a different inferences for NR but also can constrain future choices for other inferences. The “wrong” choice may as a result leave many ambiguous vectors that can not be resolved at all.

Clark recommended that the execution which resolves most genotypes must be chosen. The determination of this choice is called the **Maximum Resolution** problem, which is NP-hard. The **consensus approach** which is a stochastic method is often used to solve the MR problem. Using this approach the algorithm is run a large number of times (e.g. 10000 runs) and the runs which inferred the fewest distinct haplotypes are selected (e.g. 100 of them). From among these runs, the haplotype pair that was most commonly used to describe each genotype is recorded as its final inferred pair [11].

2.5 Haplotype Inference by Pure Parsimony

The **pure parsimony approach** to haplotype inference (**HIPP**) is based on finding a solution that minimises the total number of distinct haplotypes inferred. This is a very sensible constraint since owing to low mutation and recombination rates the number of distinct haplotypes observed in a natural population is much smaller than the number of possible haplotypes [11].

The **TIP formulation** [10] is used to describe a conceptual integer-linear-programming solution to the Pure Parsimony problem. This solution infers the fewest number of distinct haplotypes but it has a theoretical worst case complexity that is exponential with respect to the number of genotypes. Empirically the solution is only appropriate for moderate sized datasets. Because of the large number of constraints this formulation generates, the **RTIP formulation** is often used. This is a more practical formulation which is based on reducing the constraints and variables in the formulation which do not affect the final solution [10].

As far as computational results go, the Pure Parsimony approach was found practical for sample sizes of up to 50 individuals and 30 SNPs and with moderate recombination rates, 80-90% of the haplotypes are usually inferred correctly in a matter of seconds to minutes. However with higher recombination rates and larger data sets the approach may take hours to reach solutions [11].

2.6 Statistical Methods - PHASE and Haplotyper

Population geneticists and statisticians have used maximum likelihood to estimate haplotype frequencies and to infer haplotypes. Under the reasonable assumption of random mating, the likelihood function associated with a sample of individuals can be easily formulated. The objective is determining the maximum value of this likelihood function. From this one can obtain estimates of haplotype frequencies underlying the observed genotype frequencies from which the most likely haplotype pairs of any ambiguous genotype can be inferred [26].

Bayesian inference techniques are often used to solve the haplotype inference problem using the likelihood approach. One such algorithm, **PHASE** [24] derives the Bayesian-prior from the infinite sites model and then uses the Gibbs sampler to calculate the posterior distribution which is used subsequently to infer the haplotype pairs. PHASE also attempts to exploit the pure parsimony criterion in order to increase accuracy of inferences.

Haplotyper [17] is another such method that uses the Dirichlet distribution for sampling along with a model of inheritance where parents and children may be independent of each other [25]. Unlike PHASE, an explicit population model is not used but Haplotyper also uses the Gibbs sampler to calculate the posterior distribution in order to finally derive the haplotype pairs.

Such methods are all stochastic and each execution of the program may result in different solutions since the derivations are dependent on the initial configuration which is randomly selected. With multiple possible solutions many researchers have suggested the use of consensus [7] to resolve any ambiguities in solutions but this does not guarantee the veracity of the solutions from this approach.

3 Description of Proposed Methods

This chapter starts off with the motivations and the basic idea behind the algorithms to be engineered. It then goes on to thoroughly describe the algorithms and their operation using very simple examples of a few genotypes.

3.1 Motivations and Use of PPH

Most algorithms of imperfectly phylogeny haplotyping use stochastic methods and as discussed earlier in Section 2.6 may result in different solutions in different runs of the program. Furthermore these methods can prove to be very time consuming if a large number of iterations is made in the maximum likelihood algorithm in order to derive an accurate solution.

LPPH discussed in Section 2.3.1 on the other hand runs in linear time, i.e. $O(mn)$ where n is the number of individuals and m is the number of sites, however it fails completely in inferring haplotypes that include recombination events.

In Chapter 1, the aims of this project were laid out which were to design, implement and evaluate new algorithms that:

- Infers haplotypes from genotypes in the presence of recombination
- Scales better in terms of time complexity compared to current solutions

It can be observed that even though recombination prevents sequences of individuals within a sample from conforming to a perfect phylogeny, there may be regions in a sequence (e.g. between ranges of contiguous sites) that fit the perfect phylogeny model. Two algorithms were developed with exploiting these perfect phylogeny regions as the main motivation behind them. They are:

- **Disjoint Phylogeny Region Haplotyping (DPRH)**
- **Overlapping Phylogeny Region Haplotyping (OPRH)**

Both algorithms infer haplotypes by first finding PPH regions in the sample, inferring partial solutions and then putting them together to form complete haplotype solutions. The approach may seem similar to but is not inspired by the HAP method [6] described in Section 2.3.2. Near-perfect phylogeny methods such as HAP work with fixed length regions culling individuals that do not conform to a perfect phylogeny solution in the region whereas DPRH and OPRH first find maximal regions of phylogenies using all individuals and then use those regions to infer haplotypes. Moreover, methods such as HAP use maximum-likelihood to infer any missing information whereas DPRH and OPRH would not use any stochastic methods at all.

DPRH and OPRH would both use the LPPH program [5] repeatedly on parts of the input genotypes to find regions of perfect phylogeny and to infer their partial solutions. Please note that explicit permission was taken from Zhentao Ding and Dan Gusfield, the developers of LPPH, for using the LPPH executable as a black box for the purposes of this project.

3.2 Disjoint Phylogeny Region Haplotyping (DPRH)

The DPRH algorithm solves the IPPH problem by finding maximum disjoint regions of perfect phylogeny (i.e. without overlaps) and uses their partial solutions to reconstruct the complete haplotype pairs for each individual.

Owing to recombination, the only way to represent the evolutionary history of a group of individuals is via an ancestral recombination graph rather than a phylogeny (See Section 2.2). Figure 3.1a shows

Chapter 3: Description of Proposed Methods

an example where the haplotypes of individuals are broken into two regions *A* and *B*, owing to the recombination at point *rp*. This graph is very similar to that in Figure 2.7 in Chapter 2, where thick lines indicate genetic material that is passed on and thin lines indicate those that are not. Numbers above each part of a sequence give the number of descendants it has. *ind1* shares ancestors with *ind2* for the left part of its genetic material and shares with *ind3* for the right part of its genetic material.

Let us assume that the three individuals in Figure 3.1a have 30 sites each (i.e. $n=3$ and $m=20$), region *A* encompasses the first 10 sites and region *B* the last 10. Figure 3.1b shows the phylogeny one would derive using PPH on region *A* whereas Figure 3.1c shows the phylogeny one would derive from region *B*. One can also obtain haplotype inferences using the phylogenies in each region. It is interesting to note that across a single recombination event (in this case *rp*), the phylogenies of the adjacent PPH regions (*A* and *B*) differ by a single SPR (sub-tree pruning and regrafting) operation [15].

Figure 3.1a:
Evolutionary History of Individuals

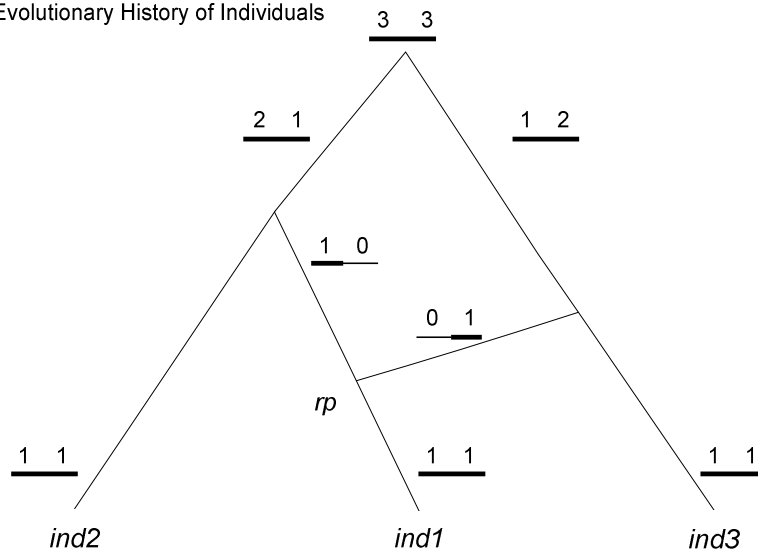


Figure 3.1b
Phylogeny for Region A
(left part i.e. sites 1-10)

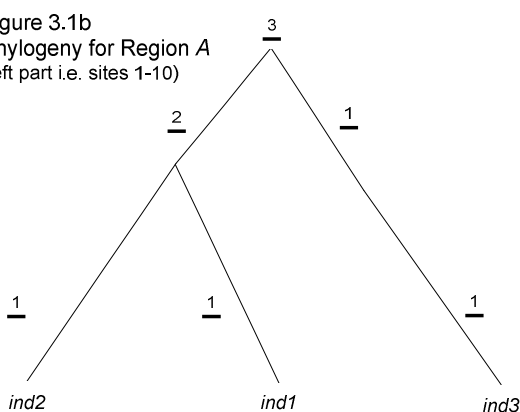


Figure 3.1c
Phylogeny for Region B
(right part i.e. sites 11-20)

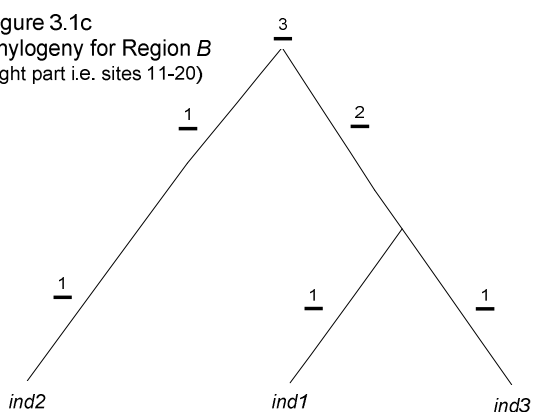


Figure 3.1: Evolutionary histories of individuals and their haplotypes

The LPPH program [5] can be used to find such regions of phylogeny and infer regional haplotype solutions for each of them. For each maximal region the algorithm infers heterozygous sites quite

accurately (usually 98-100%). However this gives rise to the problem of piecing these regional solutions together.

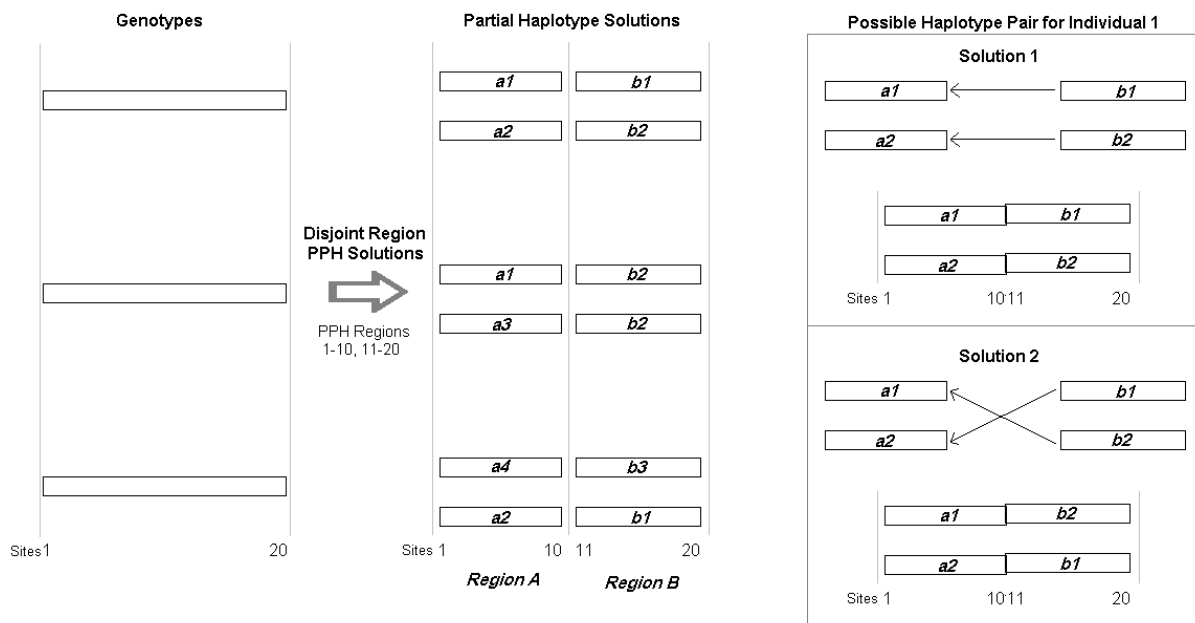


Figure 3.2: Disjoint Phylogeny Region Haplotyping (DPRH) - An example

Let us assume in Figure 3.2 that through the application of LPPH it is deduced that sites 1-10 (region A) conform to a perfect phylogeny whereas sites 11-20 (region B) conforms to a different phylogeny as shown in Figure 3.1.

For individual 1, the pair $a1$ and $a2$ have been inferred via LPPH for region A and the pair $b1$ and $b2$ for region B. There are two possible ways of aligning the partial regional solutions together. $a1$ appended with $b1$ and $a2$ appended with $b2$ is one possible haplotype pair for individual 1 while $a1$ appended with $b2$ and $a2$ appended with $b1$ is the other possible pair. The two solutions can be written using standard Z notation as $\{a1\bar{b}1, a2\bar{b}2\}$ and $\{a1\bar{b}2, a2\bar{b}1\}$.*

It can be easily deduced that for r PPH regions over a sequence, there can be 2^{r-1} possible ways of aligning the partial solutions to form the complete pair for an individual. This greatly reduces the complexity of the haplotype inference problem which presents 2^{h-1} possible solutions for h heterozygous sites. With moderate recombination rates there should be a far fewer number of PPH regions than heterozygous sites in a sequence, i.e. $r \ll h$.

An expectation maximisation method using the Gibbs sampler was considered to find the best alignment between regions. However this was not implemented since the objective of this method was to avoid stochastic methods that may be quite time consuming.

Two sub-methods of aligning partial solutions of disjoint phylogeny regions together were considered. These are described later as **DPRH1** and **DPRH2**. Both algorithms however will use the same technique to explore regions of phylogeny in the dataset.

* $p\bar{q}$ refers to the concatenation of sequence p with sequence q

Using LPPH to find PPH Regions in DPRH

The LPPH program [5] takes a file of genotypes as input and writes an output file with inferred haplotype pairs and the shadow tree (Section 2.3) if a PPH solution exists. Whether a perfect phylogeny over a region or segment of sites exists can be ascertained by writing the specific range of sites in the genotypes for all individuals onto the input file and checking whether the output file with solutions is generated.

Regions of disjoint phylogeny can be found by setting a *start* site and a *stop* site. These would both be initialised as 1. If a PPH solution exists between the *start* site and the *stop* site then the *stop* site is moved one site to the right. Otherwise it can be assumed the end of the PPH region has been reached. A new region and its regional solution are recorded as that between *start* and *stop*-1. Following the discovery of each PPH region, the *start* site is set as the last *stop* site and the *stop* site is incremented and the method continues until $stop = m$ (where m is the number of sites).

For the example in Figure 3.2 (where for 20 sites a perfect phylogeny exists at sites 1-10 and 11-20), the method would start at site 1 and keep running LPPH on regions 1-1, 1-2, 1-3, 1-4.... until it reaches 1-11 where a perfect phylogeny would not be found. The algorithm would then take down the last PPH solution i.e. from 1-10 as a disjoint PPH region in the sample. It would then run LPPH on regions 11-12, 11-13... until it reaches 11-20. Since $m = 20$, the algorithm would take down 11-20 and its solution as the second PPH region in the sample.

Please note that binary search techniques were considered rather than scanning across the sites in a linear way. However, with moderate-high recombination rates each PPH region would be only a few sites and binary search technique would in fact be more time consuming. This is because the LPPH program is $O(nm)$. Therefore, if a binary search is conducted, the method would run LPPH with datasets of dimensions $n \times m$, $n \times m/2$, $n \times m/4$ and so on which might be quite time consuming for a large number of sites. The linear search on the other hand would run LPPH with datasets of dimensions $n \times 1$, $n \times 2$, $n \times 3$ and so on which would take much less time for a large number of sites and small PPH regions.

3.2.1 Aligning Regional Solutions via DPRH1

Once the regions of partial phylogeny and their solutions have been discovered, a very simple approach of piecing these regional solutions together is used by **DPRH1**. With the example in Figure 3.2, all the individuals in the sample are analysed while considering the best alignment between regions *A* and *B*. All possible haplotype pairs are enumerated and their frequencies are taken down (Table 3.1). e.g. for individual one, two pairs are possible from the four possible haplotype sequences $a1^{\wedge}b1$, $a1^{\wedge}b2$, $a2^{\wedge}b1$ and $a2^{\wedge}b2$.

Individual	Possible haplotypes	Possible pairs
1	$a1^{\wedge}b1$ $a1^{\wedge}b2$ $a2^{\wedge}b1$ $a2^{\wedge}b2$	$a1^{\wedge}b1$ & $a2^{\wedge}b2$ OR $a1^{\wedge}b2$ & $a2^{\wedge}b1$
2	$a1^{\wedge}b2$ $a3^{\wedge}b2$ $a1^{\wedge}b2$ $a3^{\wedge}b2$	$a1^{\wedge}b2$ & $a3^{\wedge}b2$.
3	$a4^{\wedge}b3$ $a2^{\wedge}b1$ $a4^{\wedge}b1$ $a2^{\wedge}b3$	$a4^{\wedge}b3$ & $a2^{\wedge}b1$ OR $a4^{\wedge}b1$ & $a2^{\wedge}b3$

Haplotypes	Frequency
$a1^{\wedge}b1$	1
$a1^{\wedge}b2$	3
$a2^{\wedge}b1$	2
$a2^{\wedge}b2$	1
$a2^{\wedge}b3$	1
$a3^{\wedge}b2$	2
$a4^{\wedge}b1$	1
$a4^{\wedge}b3$	1

Table 3.1: Possible haplotype pairs and their frequencies in DPRH1 - An example

Chapter 3: Description of Proposed Methods

The next step is to look at each individual's possible pair and compute which pair has the highest frequency. This pair is chosen as the inferred pair between the two regions.

For individual 1, the possible haplotype pairs are

Solution1: $a1^b1$ & $a2^b2$

Solution2: $a1^b2$ & $a2^b1$

Solution1: $freq(a1^b1) + freq(a2^b2) = 1 + 1 = 2$

Solution2: $freq(a1^b2) + freq(a2^b1) = 3 + 2 = 5$

Therefore *Solution2* is the preferred pair for individual 1.

For individual 2,

Solution1: $a1^b2$ & $a3^b2$

There is just one possible solution which is chosen as the inferred pair.

For individual 3:

Solution1: $a4^b3$ & $a2^b1$

Solution2: $a4^b1$ & $a2^b3$

Solution1: $freq(a4^b3) + freq(a2^b1) = 1 + 2 = 3$

Solution2: $freq(a4^b1) + freq(a2^b3) = 1 + 1 = 2$

Therefore *Solution1* is the preferred pair for individual 3.

The final solution with haplotype pairs for each individual via DPRH1 is:

Individual 1: $a1^b2$ & $a2^b1$

Individual 2: $a1^b2$ & $a3^b2$

Individual 3: $a4^b3$ & $a2^b1$

For each subsequent region, the previous region is used like this to align the regional solutions properly.

Complexity of DPRH1

The worst case running time of this algorithm is $O(nm^2)$. The following is a brief justification of this statement.

When finding perfect phylogeny regions DPRH must run LPPH which is $O(nm)$ repeatedly starting with the first site and scanning left to right. The *start* and *stop* variables need to be maintained. Through the iterations $start + stop \leq 2m$ is true. Owing to the manner in which the algorithm scans across sites and updates these values, there can not be more than $2m$ iterations where each iteration is $O(nm)$. This gives an upper bound to the complexity as $O(2m \times nm)$ i.e. $O(nm^2)$. In the worst case LPPH will be run on datasets of dimensions $n \times 1$ and $n \times 2$ and so on until $n \times m$, giving the first part of the algorithm a worst case running time of:

$$O(n + 2n + 3n + 4n + \dots + nm)$$

$$= O(n\{1 + 2 + 3 + \dots + m\})$$

$$= O(n\{m(m-1)/2\})$$

$$= O(n\{m^2 - m\})$$

$$= O(nm^2)$$

The disjoint phylogeny finding part of DPRH can therefore be considered $O(nm^2)$.

The alignment of each region is $O(\{q+r\}n)$ where r is the number of sites in the region and q is that in the previous region. All values of r and q add up to less than $2m$ since there are no overlaps between regions. So the alignment part of the algorithm is $O(nm)$.

As a result the worst case time complexity of DPRH1 is $O(nm^2 + nm)$, i.e. $O(nm^2)$.

3.2.2 Aligning Regional Solutions via DPRH2

DPRH2 uses a similar method of using frequencies of possible pairs to align regional solutions. However instead of comparing each region to the previous one (when forming the possible solutions shown in table 3.1), DPRH2 compares it to the portion of the haplotype already constructed (i.e. concatenation of the already aligned regions).

For example, let us assume the PPH regions in a dataset are 1-10, 11-15, 16-20 and 21-25. DPRH2 will first leave 1-10 as it is. It will then try to align 11-15 by comparing it to 1-10 exactly as DPRH1 would. However after that it will try to align 16-20 by comparing it to the solution of 1-15, and then align 21-25 by comparing it to region 1-20.

Let us assume that the partial solutions have been put in the matrix H , but solutions across regions have not been aligned. Therefore H is a matrix with dimensions n (number of individuals) \times m (number of sites) \times 2 (two haplotypes for each individual)

When filling out the possible haplotypes table, for each individual x DPRH1 only used the regional solutions from the previous and current pair. So for adjacent regions $\{i, j\}^*$ and $\{p, q\}$, the table would have four entries for individual x , i.e. $a1^b1$, $a1^b2$, $a2^b1$ and $a2^b2$ where:

$$\begin{aligned} a1 &= H(x, i..j, 1) \\ a2 &= H(x, i..j, 2) \\ b1 &= H(x, p..q, 1) \\ b2 &= H(x, p..q, 2) \end{aligned}$$

DPRH2 on the other hand will use the partial solutions of all the previous regional upto $\{i, j\}$ to align $\{p, q\}$ and fill out the table for each individual x with entries $a1^b1$, $a1^b2$, $a2^b1$ and $a2^b2$ where:

$$\begin{aligned} a1 &= H(x, 1..j, 1) \\ a2 &= H(x, 1..j, 2) \\ b1 &= H(x, p..q, 1) \\ b2 &= H(x, p..q, 2) \end{aligned}$$

DPRH2 and Pure Parsimony

This algorithm infers haplotypes attempting to achieve parsimony since the most likely pairs from all the combinations of pairs are selected. However there is no guarantee that DPRH2 will produce the fewest number of distinct haplotypes since it is unknown whether the partial solutions are 100% accurate and sometimes regions are too small to infer anything useful from the regional solution. E.g. a region with 1 heterozygous site for an individual can be inferred as 0&1 and 1&0 which are both 100% right since the haplotype pair is $\{1, 0\}$. A 2 heterozygous site region has two solutions $\{10, 01\}$, $\{11, 00\}$.

A pure parsimony approach that took all combinations of alignments would be of exponential complexity. This is because each sequence would have 2^{r-1} solutions where r is the number of perfect

* $\{a, b\}$ is a set with two elements representing a pair with a and b as its elements. PPH regions can be identified as pairs/sets where the smaller element is the starting site and the larger one is the ending site.

phylogeny regions. Such an algorithm would be $O(e^r)$ but r can never exceed the number of sites m , therefore it would also be $O(e^m)$

Complexity of DPRH2

The worst case running time of DPRH2 is also $O(nm^2)$.

The PPH region finding part of DPRH2 is exactly the same as that of DPRH1 and is $O(nm^2)$.

The alignment of each region however is $O(\{p+r\}n)$ where r is the number of sites in the current region and p is the number of sites in all the previous regions put together. The sum of all values of p upper bounded by m^2 , since in the worst case where each region contains a single site $\Sigma p = 1 + 2 + 3 + \dots + m-1 = (m-1)(m-2)/2$. Therefore the alignment part of DPRH is $O(nm^2)$.

As a result the worst case time complexity of DPRH2 is $O(nm^2 + nm^2)$, i.e. $O(nm^2)$ which is the same as DPRH1. Even though the complexities of the two algorithms are similar, DPRH1 should run much faster than DPRH2 owing to the fact the second half of the algorithm is much less computationally complex than that of DPRH2.

3.3 Overlapping Phylogeny Region Haplotyping (OPRH)

A second method of IPPH using regions of phylogeny was suggested by the author's supervisor Dr Rune Lyngsoe. This method, which was later named Overlapping Phylogeny Region Haplotyping (OPRH), analyses maximal regions of perfect phylogeny (i.e. allowing overlaps) rather than disjoint regions. The maximal regions can be found quite easily using LPPH and the inferences from these regions can be stored.

Once again, aligning or rather combining the regional solutions remains the major issue. As with disjoint regions, for x PPH regions over a sequence, there can be 2^{x-1} possible ways of aligning the partial solutions to form the complete pair for an individual.

The basic concept behind OPRH is to use the overlapping part between regions for each individual to decide which way the regional sequences should be aligned. If the overlapping region's match for a possible pair (concatenation of current and previous inferences) is greater than the other pair, the first pair will be chosen as the solution for the two regions.

Using LPPH to find regions for OPRH

A similar method of using *start* and *stop* sites (as in DPRH) is used to find overlapping regions of perfect phylogeny in OPRH. The algorithm begins with the *start* and *stop* sites initialised at 1. If a PPH solution exists (checked using LPPH) between the *start* site and the *stop* site then the *stop* site is moved one site to the right. Otherwise it can be assumed the end of the PPH region has been reached. This region and its solution is recorded as between *start* and *stop*-1. Then the start site is set as *start*+1 and the process continues till *stop* = m (where m is the number of sites).

In the example in Figure 3.3 (where the regions to be discovered are 1-10 and 7-20), the method starts off by running LPPH on regions 1-1, 1-2, 1-3, 1-4... until it reaches 1-11 and finds no PPH solution. It takes down 1-10 and its solution as a region of perfect phylogeny. It then tries running LPPH on 2-11, 3-11, 4-11, 5-11 and 6-11 all of which fail. However 2-10, 3-10, 4-10, 5-10 and 6-10 are not recorded as PPH regions since they are subsets of the first region 1-10. The method runs LPPH on 7-11 successfully and then continues 7-12, 7-13 until 7-20. Since $m=20$, 7-20 is taken down as the last PPH region.

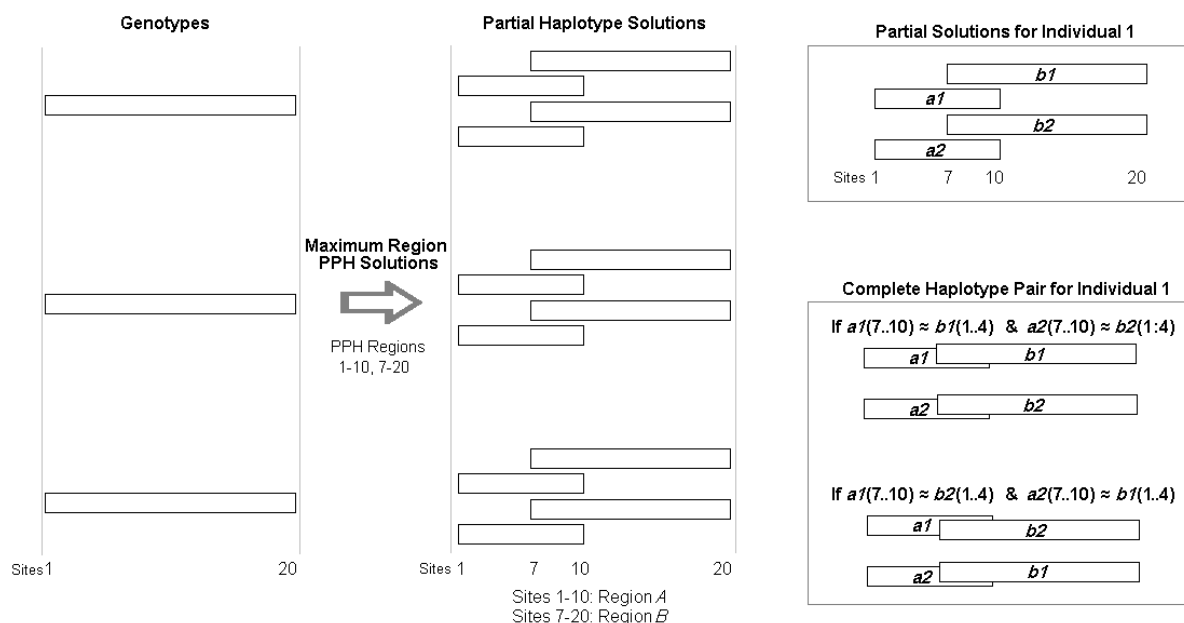


Figure 3.3: Overlapping Phylogeny Region Haplotyping (OPRH) - An example

Sequence Alignment via OPRH

As mentioned earlier, the overlap between adjacent regions are used by OPRH to piece regional solutions together. In Figure 3.3, sites 1-10 and 7-20 are two PPH regions *A* and *B* respectively that overlap between sites 7 and 10. Assuming an evolutionary history similar to that in Figure 3.1, one can assume sites 1-6 conform to the phylogeny in Figure 3.1b, sites 11-20 conform to the phylogeny in Figure 3.1c and sites 7-10 conform to both.

Coming back to Figure 3.3, for individual 1 the inferred pair for region *A* is *a1* and *a2* and that for region *B* is *b1* and *b2*. The overlapping regions are *a1*(7..10), *b1*(1..4), *a2*(7..10) and *b2*(1..4).

Using OPRH, the possible ways of concatenating the regional solutions to form the haplotype pair between Region *A-B* for individual 1 are*:

Solution1: a1(1..7)~b1(8..20) & a1(1..7)~b1(8..20)

Solution1 is true if $a1(7..10) \approx b1(1..4)$ & $a2(7..10) \approx b2(1..4)$

Solution2: a1(1..7)~b2(8..20) & a2(1..7)~b1(8..20)

Solution2 is true if $a1(7..10) \approx b2(1..4)$ & $a2(7..10) \approx b1(1..4)$

* Please note in the given solutions, the overlapping part from region *B* overwrites that from region *A*. One could also have the solutions where the converse happens i.e. *a1*(1..10)~*b1*(11..20) & *a1*(1..10)~*b1*(11..20) and *a1*(1..10)~*b2*(11..20) & *a2*(1..10)~*b1*(11..20). The former is chosen in this case however since region *B* (14 sites) is larger than region *A* (10 sites) and thus the regional solution from *B* is more reliable.

Sequence	Site no													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
<i>a1</i>	1	0	0	1	0	1	0	1	1	0				
<i>a2</i>	1	0	1	0	1	1	0	0	0	1				
<i>b1</i>	0	1	0	1	1	0	1	0	1	1	1	0	0	1
<i>b2</i>	0	1	1	1	1	0	1	0	1	1	0	0	1	1
<i>Overlap</i>														
<i>a1(7..10)</i>	0	1	1	0										
<i>a2(7..10)</i>	0	0	0	1										
<i>b1(1..4)</i>	0	1	0	1										
<i>b2(1..4)</i>	0	1	1	1										

Table 3.2: Table showing regional solutions and overlaps from Figure 3.3

One can deduce the following using Table 3.2:

Solution1: *a1(7..10)* and *b1(1..4)* are 50% equal whereas *a2(7..10)* and *b2(1..4)* are 50% equal.

Solution2: *a1(7..10)* and *b2(1..4)* are 75% equal whereas *a2(7..10)* and *b1(1..4)* are 75% equal.

Therefore it is obvious that *Solution2* is more likely than *Solution1*. Similarly for each subsequent PPH region, this matching of overlapping part between itself and the previous region is conducted to select the alignment of the pairs.

There will be cases however whether there is no overlap between regions or the overlapping region for an individual contains no heterozygous sites. Under such circumstance, the DPRH1 (as opposed to the slower DPRH2) approach is taken where the possible haplotype pairs between the regions is enumerated for all individuals and the most frequent one is chosen for each of them.

Complexity of OPRH

The worst case running time of OPRH is also $O(nm^2)$. The following is a brief justification of this statement.

The complexity explanation is very similar to that of DPRH. *start* and *stop* sites are maintained where $start+stop \leq 2m$. Because of the way OPRH scans from left to right until it reaches the end of the sequences, there can be no more than $2m$ iterations. Each iteration is a run of LPPH and is upperbounded by $O(nm)$. This makes the algorithm $O(nm^2)$. In the worst case, LPPH will be executed with datasets of dimensions $n \times 1$, $n \times 2$ and so on until $n \times m$, where the *start* and *stop* site are moved one step at a time from 1 to m . This makes the complexity of this part of the algorithm $O(n + 2n + 3n + 4n + \dots + nm) = O(nm^2)$. The disjoint phylogeny finding part of OPRH can therefore be considered $O(nm^2)$.

The alignment of each region where there is an overlap is $O(\{q+r\}n)$, where r is the number of sites in the region and q is that in the previous region, since only the overlaps between regions needs to be considered which is bound by the size of the individual regions. The sum of values of r and q is bound by m^2 . So the alignment part of the algorithm is $O(nm^2)$.

As a result the worst case time complexity of OPRH is $O(nm^2 + nm^2)$, i.e. $O(nm^2)$.

Please note that even though the complexity of OPRH is the same as that of DPRH1, on average OPRH should take longer than DPRH1 since it uses the same tables as DPRH1 to enumerate possible pairs between regions when there is no overlap or heterozygous sites in the region.

4 Implementation of Engineered Algorithms

This chapter explains the implementation of the algorithms described in Chapter 3. First of all however, a brief justification is given behind the choice of MATLAB as the development environment. Then the implementation of each algorithm is explained in great detail using variable names identical to those in the code found in Appendix A. This is done such that the reader can reference between the code and this report with ease.

Pseudo-code is often provided to explain complex matrix operations which can not be described very clearly using text. In order to maintain a strong understanding between the report and the implementation, MATLAB notations for matrix indices and operations are often used in the explanations as well as pseudo-codes. Some MATLAB notations that readers may be unfamiliar with are described here:

- $x:y$ refers to the range $x..y$ (inclusive).
- $:$ refers to the entire range, e.g. $A(5, :)$ to all the cells in the 5th row of matrix A .
- $[p\ q]$ refers to $p\tilde{q}$ i.e. the concatenation of sequence p with sequence q .
- $=$ refers to assignment. e.g. $x=2$ means that x is set to 2.
- $==$ refers to equality. e.g. $x==2$ is true if and only if x is equal to 2.

All variable and function names are italicised. Variable names for matrices and lists (e.g. G or $SolList$) start with an uppercase character whereas those for scalars and single sequences start with a lowercase character (e.g. n or aI)

4.1 Development Environment

MATLAB and C were the development environments mainly considered for implementing the algorithms proposed in Chapter 3. The following is a discussion of the pros and cons of each environment which led to MATLAB being chosen to implement and evaluate the DPRH and OPRH algorithms.

Advantages of MATLAB over C

- MATLAB is an interpreted language while C is compiled. This means that code and data can be updated while the program is executing, making MATLAB ideal for research and algorithm engineering.
- MATLAB is specifically intended for numerical computation while C is a general purpose language. MATLAB operates on matrices rather than scalars as its primary data structure. C can also use matrices via custom packages but their sizes are restricted on declaration. Furthermore, MATLAB can perform calculations involving multiple elements of a matrix with a simple (visually obvious) command whereas C would require complex FOR loops to do the same. Such matrix manipulations are essential when working with haplotypes and genotypes.
- MATLAB allows for easy exchange of data between itself and spreadsheet applications such as Excel which was an advantage to the author since the latter was to be used to store results and draw graphs in order to evaluate the performance of the HI methods.
- Unlike C, MATLAB does not require the user to declare or predefine every variable with a type and a size before its first use. This gives the developer a great degree of flexibility which however requires an equal degree of programming discipline.

- Many developers spend large quantities of time going through books and web resources while attempting to use complicated functions or downloaded libraries in C. MATLAB on the other hand comes with very comprehensive documentation of all features of the language.

Advantages of C over MATLAB

- The fact that MATLAB is an interpreted language makes it rather slow. This would be a great disadvantage when trying to compare speeds of DPRH and OPRH against popular methods which are already in compiled form. This was a sacrifice that had to be made since MATLAB's other features allow very fast development, debugging and evaluation, a characteristic that was essential for this project (which the author was allotted just four months for).
- As mentioned earlier, the fact that variables are not declared or defined may lead to many errors (such as typos when spelling variable names) go undetected and give unpredictable results. For a disciplined programmer however this would not be an issue at all.
- With free open-source compilers such as gcc available for C, one does not have to pay for developing in it. MATLAB however is a commercial product which costs \$100-150. This was not an issue for the development of this project since the licenses purchased by the Oxford University Computing Laboratory (OUCL) allows its students to use MATLAB at the OUCL software labs.

Details of Development Environment Used

MATLAB The Language of Technical Computing
The MathWorks, Inc.
Version 7.0.0.19901 (R14)
May 06, 2004

4.2 Details of all functions

MATLAB is a function-based language and a number of functions were used in the implementation of the DPRH and OPRH algorithms. The modularity in the development process was crucial since the algorithms shared a number of common operations.

Table 4.1 shows a list of all functions used, details of their input and output parameters and a brief description of their purpose. The table is shown early in the chapter such that the user can understand the purpose of certain functions when they are used in the implementation of the DPRH and OPRH algorithms in later sections. Please see Appendix A for the annotated code of all functions written.

Table 4.1: Parameters and descriptions of all implemented functions*

<i>Function</i>	<i>Input(s)</i>	<i>Output(s)</i>	<i>Description</i>
<i>runDPRH1</i>	G	H	This is the implementation of the DPRH1 algorithm designed in Section 3.2.1. It takes in a $n \times m$ matrix of genotypes G , infers haplotypes via DPRH1 and returns them in a $n \times m \times 2$ matrix H .
<i>runDPRH2</i>	G	H	This is the implementation of the DPRH2 algorithm designed in Section 3.2.2. It takes in a $n \times m$ matrix of genotypes G , infers haplotypes via DPRH2 and returns them in a $n \times m \times 2$ matrix H .
<i>runOPRH</i>	G	H	This is the implementation of the OPRH algorithm designed in Section 3.3. It takes in a $n \times m$ matrix of genotypes G , infers haplotypes via OPRH and returns them in a $n \times m \times 2$ matrix H .
<i>runLPPH</i>	G	H	This runs the LPPH executable [5] on the $n \times m$ matrix of genotypes G , and returns the inferred haplotypes in a $n \times m \times 2$ matrix H if a PPH solution is found. Otherwise the function returns H as -1.
<i>leafcount</i>	gc	l	Returns the leaf count of gc which is a column of a genotype matrix. The leaf count l is the number of 2's in the column plus twice the number of 1's. Used by runLPPH.
<i>rightmost1</i>	gr	p	Returns p , the column position of the rightmost one in gr which is a row of a genotype matrix. Used by runLPPH.
<i>findDPRs</i>	G	H Regions	This function is used by <i>runDPRH1</i> and <i>runDPRH2</i> to find the disjoint phylogeny regions in genotypes input as a $n \times m$ matrix G . It also returns the regional solutions in a $n \times m \times 2$ matrix H which is later aligned by DPRH1 or 2.
<i>file2hap</i>	<i>filename</i>	H	This method opens the file with the path <i>filename</i> , and reads its contents into a $n \times m \times 2$ matrix H . The file must be written in text format (with only 0s and 1s) where every pair of rows represents the haplotypes for an individual. Therefore n is half the number of rows in <i>filename</i> and m is the number of columns in <i>filename</i> .
<i>hap2gen</i>	H	G	Given a $n \times m \times 2$ matrix of haplotype pairs H , this function returns its genotypes as a $n \times m$ matrix G , using the rules described in Section 2.1
<i>displayHaps</i>	H		Displays the haplotypes in H in a more readable format to the user. An individual's two haplotypes are displayed in consecutive rows whereas haplotypes of different individuals are separated by blank lines.

* Please note that variable names containing scalars which start with lower case characters (e.g. n , m etc.) may have any value. For example, m in *runLPPH* does not have to be the number of sites in the entire dataset of genotypes being inferred when the program is attempting find PPH solutions for a subset of the sites (i.e. a region).

Function	Input(s)	Output(s)	Description
<i>file2gen</i>	<i>filename</i>	<i>G</i>	This method opens the file with the path <i>filename</i> , and reads its contents into a $n \times m$ matrix <i>G</i> . The file must be written in text format (with only 0s, 1s and 2s) where every rows representing the genotype for an individual. Therefore <i>n</i> is the number of rows in <i>filename</i> and <i>m</i> is the number of columns in <i>filename</i> .
<i>displayGens</i>	<i>G</i>		Displays the genotypes in <i>G</i> on screen with each individual's genotype on a different row.
<i>compareHap</i>	<i>h1</i> <i>h2</i>	<i>allIdn</i> <i>htrzIdn</i>	Compares how identical two haplotype pairs <i>h1</i> and <i>h2</i> are and returns the proportion of all sites that are identical as <i>allIdn</i> and the proportion of heterozygous sites that are identical as <i>htrzIdn</i> . Please note that <i>h1</i> and <i>h2</i> must be of identical dimensions of the form $1 \times m \times 2$.
<i>evalHInf</i>	<i>InfDH</i> <i>RealH</i> <i>dispInfo</i>	<i>idnRatio</i> <i>htrzAcc</i> <i>swchErr</i> <i>crctHaps</i> <i>avgIndAcc</i> <i>stdIndAcc</i>	This methods evaluates the accuracy of a haplotype inference given the original haplotype pairs <i>RealH</i> and the inferred haplotype pairs <i>InfDH</i> in matrices of $n \times m \times 2$ dimensions. It returns measures for all the accuracy criteria described in Section 4.6. If the parameter <i>dispInfo</i> > 0, all this information is displayed on screen as well as being returned as output parameters.
<i>rows</i>	<i>M</i>	<i>r</i>	Returns the number of rows a 2D matrix <i>M</i> has.
<i>swap</i>	<i>X</i> <i>Y</i>	<i>B</i> <i>A</i>	Returns the swapped version of the input variables to aid in swapping operations. Eg. to swap <i>X</i> and <i>Y</i> , one can use $[X, Y] = \text{swap}(X, Y)$
<i>n2str</i>	<i>n</i> <i>x</i>	<i>str</i>	Returns a string <i>str</i> which is the string representation of <i>n</i> right indented using <i>x</i> characters.
<i>txt2gen</i>	<i>G</i>	<i>textG</i>	Returns a genotype matrix <i>G</i> of dimensions $n \times m$, when the genotypes are input in text format as <i>textG</i> , where each row of 0s, 1s and 2s, represent an individual's genotype.
<i>txt2hap</i>	<i>H</i>	<i>textH</i>	Returns a haplotype matrix <i>H</i> of dimensions $n \times m \times 2$, when the haplotypes are input in text format as <i>textH</i> , where pairs of rows containing 0s and 1s represent the haplotype pair for each individual.
<i>equalRat</i>	<i>X</i> <i>Y</i>	<i>ratio</i>	Returns the proportion of corresponding elements in two matrices (of identical dimensions) that are equal. E.g. $\text{equalRat}([6 \ 5], [2 \ 5])$ would return <i>ratio</i> as 0.5.
<i>freq</i>	<i>M</i> <i>x</i>	<i>f</i>	Returns the number of rows <i>f</i> in a 2D matrix <i>M</i> that are equal to a vector or 1D matrix <i>x</i> . Please note that if <i>M</i> is a $r \times c$ matrix, then <i>x</i> must be of dimensions $1 \times c$ for <i>freq</i> to execute properly.

4.3 Integration of LPPH

As mentioned in Section 3.1 explicit permission was taken by the author from LPPH's developers [5] to use their executable as a black-box. The LPPH program was available as a Unix executable "lpph" which takes the input filename as a parameter and writes an output file 'output.pph' if a PPH solution is found. The input file is a text file where each individual's genotype sequence (of 0s, 1s and 2s) appears on a different line.

The output file contains information regarding the execution of lpph. It includes the shadow tree (see Section 2.3.1) followed by the inferred haplotype pairs.

One complication associated with the lpph executable was that it takes a dataset of genotypes as input and sorts the columns (by decreasing leaf count which is the number of 2's in a column plus twice the number of 1's) and rows (with the position of rightmost 1 entry in each row decreasingly) before processing it [5]. These operations are performed by functions *leftcount.m* and *rightmost1.m*. The haplotype inferences are unfortunately inferences of the sorted dataset rather than the original genotypes. Therefore in order to infer haplotypes correctly, one would have to find a way of "unsort"-ing the final inferences from lpph such that they correspond with the input genotypes.

Since the algorithms engineered require multiple runs of LPPH, a function *runLPPH.m* was written to help interface between them and lpph. This function takes a genotype matrix G of dimensions $n \times m$ where n is the number of individuals and m is the number of sites as input. It then sorts columns and rows of G according to the rules of lpph mentioned in the last paragraph to give matrix $LpphG$. While performing the sorting operation, *runLPPH.m* keeps track of the position in G of each cell in $LpphG$ using matrices *rowOrigs* and *colOrigs* each of dimensions $n \times m$. Therefore *rowOrigs*(x, y) returns the row number $LpphG(x, y)$ originated from in G whereas *colOrigs*(x, y) returns the column number. This matrix is later used to "unsort" lpph's solution into the real inference for G .

runLPPH.m writes $LpphG$ in text format onto a local file 'temp.pph' and executes lpph with the filename as the parameter. *runLPPH.m* then checks whether the output file 'output.pph' was created. If it was not, then a perfect phylogeny solution is not found and the function returns -1.

If 'output.pph' has been created, *runLPPH.m* goes through the file and looks for the inferred haplotype pairs in the file. These are then written to a 3 dimensional matrix $LpphH$, of dimensions n (number of individuals) \times m (number of sites) \times 2 (since there is a pair of haplotypes for each individual). $LpphH$ however has the haplotypes sorted according leaf-count and rightmost 1 entry. A matrix H of dimensions $(n, m, 2)$ is created and its contents are filled using *rowOrigs*, *colOrigs* and $LpphH$ such that it contains the unsorted real haplotype inferences for G . This is done by taking each site of an individual (x, y) in $LpphH$ and copying it to the location in H indicated by *rowOrigs*(x, y) and *colOrigs*(x, y).

```

For all individuals x
  For all sites y
    xyRow = rowOrigs(x, y)
    xyCol = colOrigs(x, y)
    H(xyRow, xyCol, 1) = LpphH(x, y, 1)
    H(xyRow, xyCol, 2) = LpphH(x, y, 2)
  End for
End for

```

The final matrix H is returned as the output of *runLPPH.m* if a PPH solution exists. Otherwise -1 is returned.

4.4 Disjoint Phylogeny Region Haplotyping

Versions 1 and 2 of Disjoint Phylogeny Region Haplotyping described in Section 3.2 are implemented using the functions *runDPRH1.m* and *runDPRH2.m* respectively. The first part of both algorithms that find the maximal disjoint PPH regions are identical. They only differ in the alignment of their regional solutions.

runDPRH1.m and *runDPRH2.m* also have the same input and output parameters. They take a matrix G as input as the genotype matrix and return a matrix H with inferred pairs as the output. The number of individuals in the sample n is equal to the rows in G and the number of sites m is equal to the columns in G .

Since the disjoint phylogeny region finding parts of both algorithms are similar, a function *findDPRs.m* was written that takes a matrix G (genotypes) as input. It outputs a matrix *Regions* which gives the boundaries of each disjoint region and H , a matrix of haplotypes with regional solutions that have not been aligned yet.

The matrix H of dimensions n , m and 2 is initialised with 0s as *findDPRs.m* commences execution and is populated as the algorithm progresses.

As explained in Section 3.2.1, *start* and *stop* site variables are used to scan across the sites in G finding regions of perfect phylogeny. As each region is found, *start* is set as the last *stop* site, allowing the algorithm to find maximal disjoint PPH regions i.e. without any overlaps. The boundaries of each region are stored in a $r \times 2$ matrix called *Regions* where r is the number of PPH regions*. For each region, the PPH solution is copied on to the appropriate sites in H . Therefore all partial solutions are stored in H , parts of which must be aligned properly later.

Pseudocode for finding disjoint phylogeny regions

Function *findDPRs(G)*

```

n = rows(G)
m = columns(G)

H = new n*m*2 matrix

start = 1
stop = 1
Regions = []

While stop <= m
    PPHSolution = runLPPH(g(1:n, start:stop))

    If there is a PPH Solution
        stop = stop + 1
        LastSolution = PPHSolution
    Else
        Add {start, stop} to Regions
        H(1:n, start:stop-1) = LastSolution
        start = stop
    End if
End While

Return (Regions, H)

```

* Since DPRH only deals with disjoint regions only the end site of each region could be stored and the start site could be computed as the end site of the previous region plus 1. This was not done merely to keep things simple and to give the author a generic way to maintain region boundaries to use between DPRH and OPRH.

4.4.1 Implementation of DPRH1

Once *runDPRs.m* has returned *Regions* and the unaligned *H*, the proper alignment of sequences for each individual must be determined for each pair of adjacent regions $\{i, j\}$ and $\{p, q\}$. This is done by DPRH1 (Section 3.2.1) via the following steps:

First all four possible concatenations of sequences between the two regions for each individual *x*, i.e. $[H(x, i:j, 1) H(x, p:q, 1)]$, $[H(x, i:j, 2) H(x, p:q, 2)]$, $[H(x, i:j, 1) H(x, p:q, 2)]$ and $[H(x, i:j, 2) H(x, p:q, 1)]$ are entered into a matrix called *PossSeqs*.

Then for each individual *x*, whichever of the alignments seems to appear more frequently in *PossSeqs* is selected as the correct one.

Please note that since all partial solutions were initially copied into *H* the default alignment will be:

$$[H(x, i:j, 1) H(x, p:q, 1)] \text{ and } [H(x, i:j, 2) H(x, p:q, 2)],$$

therefore if the sum of frequencies of $[H(x, i:j, 1) H(x, p:q, 2)]$ and $[H(x, i:j, 2) H(x, p:q, 1)]$ exceed that the default pair, the latter part of the pairs must be flipped. Otherwise, *H* can be left unchanged.

DPRH1 is implemented using the *runDPRH1.m* function which uses *runDPRs.m* to find disjoint phylogeny regions and then uses the method described to piece partial solutions together. Ideally one would open a file and read the genotypes into an array *G* using *file2gen.m*, before running *runDPRH1(G)*.

Pseudocode for DPRH1

Function *H* = *runDPRH1(G)*

(Regions, H) = *findDPRs(G)*

For each $\{p, q\}$ in *Regions* starting from the 2nd

$\{i, j\}$ = previous region

PossSeqs = ()

 For each individual *x*

a1 = *H(x, i:j, 1)*

a2 = *H(x, i:j, 2)*

b1 = *H(x, p:q, 1)*

b2 = *H(x, p:q, 2)*

 Add $[a1\ b1]$, $[a2\ b2]$, $[a1\ b2]$ and $[a2\ b1]$ to *PossSeqs*

 End for

 For each individual *x*

 If $\text{freq}(\text{PossSeqs}, [a1\ b2]) + \text{freq}(\text{PossSeqs}, [a2\ b1])$

 > $\text{freq}(\text{PossSeqs}, [a1\ b1]) + \text{freq}(\text{PossSeqs}, [a2\ b2])$

H(x, j, 1) = $[a1\ b2]$

H(x, j, 2) = $[a2\ b1]$

 End if

 End for

End for

Return *H*

4.4.2 Implementation of DPRH2

As explained in Section 3.2.2 DPRH2 uses a similar method of alignment. Instead of comparing each region to the previous region, it compares to the portion of the haplotype already constructed (i.e. concatenation of the already aligned regions). First of all however DPRH2 uses *findDPRs.m* to compute the disjoint regions and the unaligned H matrix.

Let us consider an example dataset with 25 sites where $Regions = \begin{pmatrix} 1 & 10 \\ 11 & 15 \\ 16 & 20 \\ 21 & 25 \end{pmatrix}$

For all individuals DPRH2 will first leave sites 1-10 as it is. Then for each individual it will try to align 11-15 by comparing it to 1-10. After that it will try to align 16-20 by comparing it to region 1-15, and then align 21-25 by comparing it to region 1-20.

DPRH2 is implemented using the *runDPRH2.m* function which uses *runDPRs.m* to find disjoint phylogeny regions and then uses the method described to piece partial solutions together. The pseudocode for *runDPRH2.m* is exactly the same as that for *runDPRH1.m* except all the i 's are replaced with 1's.

This means that all possible concatenation of sequences between the regions $\{p \ q\}$ and the previously inferred haplotypes for each individual x , i.e.:

$[H(x, 1:j, 1) \ H(x, p:q, 1)]$, $[H(x, 1:j, 2) \ H(x, p:q, 2)]$, $[H(x, 1:j, 1) \ H(x, p:q, 2)]$ and $[H(x, 1:j, 2) \ H(x, p:q, 1)]$ are entered into a matrix called *PossHaps*.

Then for each individual x , whichever of the alignments seems to appear more frequently in *PossHaps* is selected as the correct one.

Please note that since all partial solutions were initially copied into H the default alignment will be:

$$[H(x, 1:j, 1) \ H(x, p:q, 1)] \text{ and } [H(x, 1:j, 2) \ H(x, p:q, 2)],$$

therefore if the sum of frequencies of $[H(x, 1:j, 1) \ H(x, p:q, 2)]$ and $[H(x, 1:j, 2) \ H(x, p:q, 1)]$ exceed that of the default alignment, the latter part of the pairs must be flipped. Otherwise, the H can be left unchanged.

Since the length of each entry in *PossHaps* will be quite long, it is very likely that for samples with a large number of sites, when scanning through the latter regions each alignment entry in *PossHaps* may be unique. So, when aligning for individual x the algorithm may end up with:

$$\begin{aligned} \text{freq}(\text{PossHaps}, [H(x, 1:j, 1) \ H(x, p:q, 1)]) &= 1 \\ \text{freq}(\text{PossHaps}, [H(x, 1:j, 2) \ H(x, p:q, 2)]) &= 1 \\ \text{freq}(\text{PossHaps}, [H(x, 1:j, 1) \ H(x, p:q, 2)]) &= 1 \\ \text{freq}(\text{PossHaps}, [H(x, 1:j, 2) \ H(x, p:q, 1)]) &= 1 \end{aligned}$$

In such circumstances DPRH2 will not have enough information to align the regional solutions with any reliability. To help DPRH2 in situations such as these, the *PossSeqs* matrix from DPRH1 is always maintained and used to align regional solutions if the frequencies of all the possible sequences for an individual in *PossHaps* are equal.

This does not change the complexity of DPRH2 since the alignment part of DPRH1 is $O(nm)$ whereas that of DPRH2 is $O(nm^2)$. However this means that DPRH2 will always take longer than DPRH1 to infer haplotypes.

4.5 Overlapping Phylogeny Region Haplotyping

The Overlapping Phylogeny Region Haplotyping algorithm described in Section 3.3 is implemented using the function *runOPRH.m*. Like the DPRH functions, it takes a matrix G as input as the genotype matrix and returns a matrix H of inferred haplotype pairs. The number of individuals in the sample n is equal to the rows in G and the number of sites m is equal to the columns in G .

Finding Maximal PPH Regions

runOPRH.m also uses *start* and *stop* sites to keep track of its position as it scans across the sites of G but it updates them differently from *findDPRs.m* in order to find regions that may include overlaps. As a region of perfect phylogeny is found, the *start* site is incremented, allowing the discovery of maximal regions (and not just disjoint ones). The boundaries of each region are stored in a $r \times 2$ matrix called *Regions* where r is the number of PPH regions and the partial solution for the region is stored in a list of matrices called *SolList*.

Alignment of Regional Solutions

The first regional solution is simply copied onto H since its own alignment does not matter. After that for each pair of adjacent regions $\{i, j\}$ and $\{p, q\}$, the proper alignment of sequences for each individual x must be determined. Let us assume that:

$a1$ and $a2$ are the partial solutions for individual x in region $\{i, j\}$

$b1$ and $b2$ are the partial solutions for individual x in region $\{p, q\}$

$a1o$ and $a2o$ are the suffixes of $a1$ and $a2$ respectively that overlap onto region $\{p, q\}$

$b1o$ and $b2o$ are the prefixes of $b1$ and $b2$ respectively that overlap onto region $\{i, j\}$

$a1$ is aligned with $b1$ and $a2$ is aligned with $b2$, if and only if their overlapping regions agree, that is:

$$\text{equalRat}(a1o, b1o) + \text{equalRat}(a2o, b2o) > \text{equalRat}(a2o, b1o) + \text{equalRat}(a1o, b2o)$$

This is done by simply copying the first haplotype of the x th row of the regional solution H_{pq} for $\{p, q\}$ for both its haplotypes from *SolList* i.e. $H_{pq}(x, :, 1)$ or $b1$ on to the first haplotype at H at row x and columns p to q i.e. $H(x, p:q, 1)$ and the same with the second that is:*

$$H(x, p:q, 1) = b1 = H_{pq}(x, :, 1)$$

$$H(x, p:q, 2) = b2 = H_{pq}(x, :, 2)$$

Otherwise, $a1$ is aligned with $b2$ and $a2$ is aligned with $b1$

This is done by copying the first haplotype in the x th row of the partial solution H_{pq} for $\{p, q\}$ i.e. $H_{pq}(x, :, 1)$ or $b1$ onto the second haplotype in H at row x and columns p to q i.e. $H(x, p:q, 2)$ and vice versa i.e.

$$H(x, p:q, 1) = b1 = H_{pq}(x, :, 2)$$

$$H(x, p:q, 2) = b2 = H_{pq}(x, :, 1)$$

However there will be cases where there is no overlap between regions e.g. $\{2, 5\}$ and $\{6, 10\}$, or the overlapping regions for an individual have no heterozygous sites and this method of alignment exploiting overlaps can not be used. Under these circumstances, the same method of alignment as that of the disjoint PPH method DPRH1 is used to combine the partial solutions across the regions for each individual. For the sake of simplicity this part of the algorithm is not explicitly described in the following pseudocode:

* Please note that this method of copying the aligned solution of the second region on to H means the overlap part of the previous region is overwritten. i.e. for this straight alignment $b1o$ replaces $a1o$ and $a2o$ replaces $b2o$. This is not considered a problem since ideally these values should be equal for the best alignment. It was initially considered to use a consensus approach on sites that were part of overlaps between three regions to help align solutions better. However, runs of the phylogeny finding part of OPRH showed this to be a rarity and the idea was therefore not implemented since it would have added to the complexity of the algorithm.

Pseudocode for OPRH

Function $H = \text{runOPRH}(G)$

```

n = rows(G)
m = columns(G)

start = 1
stop = 1
regions = []
PPHSolution = -1
SolList = {}

While stop <= m

    PPHSolution = runLPPH(g(1:n, start:stop))

    If there is a PPH Solution
        stop = stop + 1
        Store LastSolution = PPHSolution
    Else
        If stop of last region /= stop-1
            Add {start, stop} to regions
            Add LastSolution to SolList
            start = start+1
        End if
    End if

End while

H = new n*m*2 matrix
Let {a, b} be the first in Regions
H(:, a:b, :) = PPH solution for {a b} from SolList

For each {p, q} in Regions starting from the 2nd

    {i, j} = previous region
    Hij = H(:, i:j, :)
    Hpq = PPH solution for region {p, q} from SolList

    For each individual x

        a1 = Hij(x, i:j, 1)
        a2 = Hij(x, i:j, 2)
        b1 = Hpq(x, p:q, 1)
        b1 = Hpq(x, p:q, 2)
        a1o = Hij(x, p-i+1:j-i+1, 1)
        a2o = Hij(x, p-i+1:j-i+1, 2)
        b1o = Hpq(x, p-i+1:j-i+1, 1)
        b2o = Hpq(x, p-i+1:j-i+1, 2)

        If equalRat(a1o, b1o) + equalRat(a2o, b2o)
            > equalRat(a1o, b2o) + equalRat(a2o, b1o)
                H(x, p:q, 1) = [a1 b1]
                H(x, p:q, 2) = [a1 b1]
            Else
                H(x, p:q, 1) = [a1 b2]
                H(x, q:q, 2) = [a2 b1]
            End

    End For

End For

End for

Return H

```

4.6 Computing Accuracy of Inferences

Several measures of accuracy were used to compare the final inferred *InfdH* matrix and the original matrix *RealH* which the genotype *G* was derived from. The function *evalHInf.m* takes in *RealH* and *InfdH* as parameters and outputs the accuracy measures explained here. The accuracy measures returned and their output variable names are shown in Table 4.2. The function takes in a third parameter *dispInfo*. This can be thought of as a flag, which if set to true *evalHInf.m* displays with headings all this information along with accuracies for each individual on screen. Please see Appendix B, for examples of screen outputs obtained from *evalHInf.m* run with *dispInfo*=true.

<i>Accuracy Measure</i>	<i>Variable Returned</i>
Ratio of Identical Sites	<i>idnRatio</i>
Heterozygous Sites Accurately Inferred	<i>htrzAcc</i>
Average Switch Error	<i>swchErr</i>
Correctly Inferred Haplotypes	<i>crctHaps</i>
Heterozygous Sites Accurately Inferred/Individual - average and standard deviation	<i>avgIndAcc</i> <i>stdIndAcc</i>

Table 4.2: Accuracy measures and variables returned by the evaluation function

Ratio of Identical Sites

A very simple way of comparing haplotypes in *InfdH* would be to compare the entirety of matrices *InfdH* and *realH*. This can be done very easily in MATLAB using *EqualMat* = (*InfdH*==*RealH*), which would produce a matrix *EqualMat* of the same dimensions as *InfdH* and *RealH*. Each cell of *EqualMat* would contain a 1 if the corresponding cells in *InfdH* and *RealH* are equal and a 0 otherwise. The ratio of equality between *InfdH* and *RealH* can be computed by dividing the sum of all cells in *EqualMat* by the product of its dimensions i.e. $n \times m \times 2$.

One must bear in mind that the order of the haplotypes in each pair for an individual does not matter and therefore such direct *InfdH* to *RealH* comparisons can make sound inference seem inaccurate. Consider the simple example with just one individual and four sites where:

$$\begin{array}{ll} \text{InfdH}(1, :, 1) & = 1 \ 0 \ 0 \ 1 \\ \text{InfdH}(1, :, 2) & = 0 \ 0 \ 1 \ 1 \end{array} \qquad \begin{array}{ll} \text{RealH}(1, :, 1) & = 0 \ 0 \ 1 \ 1 \\ \text{RealH}(1, :, 2) & = 1 \ 0 \ 0 \ 1 \end{array}$$

After *EqualMat* = (*InfdH*==*RealH*)

$$\begin{array}{l} \text{EqualMat}(1, :, 1) = 0 \ 1 \ 0 \ 1 \\ \text{EqualMat}(1, :, 2) = 0 \ 1 \ 0 \ 1 \end{array}$$

$$\begin{aligned} \text{idnRatio} &= \text{Sum}(\text{EqualMat}) / (n \times m \times 2) \\ &= (0+1+0+1+0+1+0+1) / (1 \times 4 \times 2) \\ &= 4/8 = 0.50 = 50\% \end{aligned}$$

Please note this is exactly the method used by the function *equalRat.m* which takes in two matrices as parameters and returns the ratio of accuracy between them.

In this case, by looking at the real and inferred pairs, one can tell that they are 100% similar. The fact that the orientation within the pair is not the same is not relevant to the accuracy of haplotype inference. Therefore a different algorithm (pseudo-code shown below) was used that for each

individual, found out which inferred haplotype is closest to which real haplotype between the pair and then performed a comparison accordingly.

Pseudocode

```

EqualMat = new n*m*2 matrix

For each individual x

    If equalRat(InfdH(1, :, 1), RealH(1, :, 1))
      > equalRat(InfdH(1, :, 1), RealH(1, :, 2))

        EqualMat(x, :, :) = InfdH(x, :, :) == RealH(x, :, :)

    Else

        EqualMat(x, :, 1) = InfdH(x, :, 1) == RealH(x, :, 2)
        EqualMat(x, :, 2) = InfdH(x, :, 2) == RealH(x, :, 1)

    End if

End for

Accuracy = Sum(EqualMat) / (n*m*2)

```

In the example given earlier, the first inferred haplotype is closest to the second real haplotype and the second inferred haplotype is closest to the first real haplotype and thus the updated algorithm will compute the percentage of identical sites as follows

$$\begin{aligned}
 \text{EqualMat}(1, :, 1) &= \text{InfdH}(1, :, 1) == \text{RealH}(1, :, 2) = 1 \ 1 \ 1 \ 1 \\
 \text{EqualMat}(1, :, 2) &= \text{InfdH}(1, :, 2) == \text{RealH}(1, :, 1) = 1 \ 1 \ 1 \ 1 \\
 \\
 \text{idnRatio} &= \text{Sum}(\text{EqualMat}) / (n \times m \times 2) \\
 &= (1+1+1+1+1+1+1+1) / (1 \times 4 \times 2) \\
 &= 8/8 = 1.00 = 100\%
 \end{aligned}$$

Heterozygous Sites Accurately Inferred

The vast majority of the sites in any genotype matrix are homozygous and thus the previous measure may make the uninformed researcher or reader feel rather confident about the results with very high accuracies. A better measure of accuracy was considered as the ratio of heterozygous sites accurately inferred. The number of zeroes in *EqualMat* gives twice the number of heterozygous sites in *G* that have been inaccurately inferred (since each inaccurate inference leads to an error in each of the two haplotypes of the individual). The number of heterozygous sites accurately inferred can be very easily computed from this and this can be divided by the number of heterozygous sites in *G*. Please note that homozygous sites are always accurately inferred since there is no ambiguity there.

In MATLAB, the cell locations of entries in a matrix *M* of a certain value *x* can be found by the command `find(M==x)`, whereas `length(find(M==x))` would give the number of entries. Therefore, the ratio of heterozygous sites accurately inferred can be found very simply by:

```

correctHtrzs = length(find(EqualMat==1))/2
numberOfHtrzs = length(find(G==2))
htrzsAcc = correctHtrzs / numberOfHtrzs

```

Average Switch Error

Another measure of accuracy of haplotype inference is the switch error. This can be defined as the number of flips that need to be performed between the haplotypes of the inferred pair such that it is equal to the original pair. Let us consider the example:

	Sites	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>RealH</i> ($x, :, 1$)		1	0	0	1	1	0	1	0	1	0	1	1	0
<i>InfDH</i> ($x, :, 1$)		1	0	0	1	1	0	0	1	0	0	1	1	0
<i>RealH</i> ($x, :, 2$)		0	0	1	1	1	0	0	1	0	1	1	1	1
<i>InfDH</i> ($x, :, 2$)		0	0	1	1	1	0	1	0	1	1	1	1	1

One can observe that using two flips between the haplotypes in *InfDH*, one can obtain *RealH*. If a flip is performed at site i , all the values of sites between i and m are flipped across the haplotypes of the pair.

The first flip is performed the first inconsistency between *RealH* and *InfDH* is observed. This happens at site 7. Therefore all values between the haplotypes in *InfDH* for x between sites 7 and 13 are swapped between the haplotypes of x 's pair.

	Sites	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>RealH</i> ($x, :, 1$)		1	0	0	1	1	0	1	0	1	0	1	1	0
<i>InfDH</i> ($x, :, 1$) ₁		1	0	0	1	1	0	1	0	1	1	1	1	1
<i>RealH</i> ($x, :, 2$)		0	0	1	1	1	0	0	1	0	1	1	1	1
<i>InfDH</i> ($x, :, 2$) ₁		0	0	1	1	1	0	0	1	0	0	1	1	0

The next inconsistency occurs at site 10 where another flip is performed.

	Sites	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>RealH</i> ($x, :, 1$)		1	0	0	1	1	0	1	0	1	0	1	1	0
<i>InfDH</i> ($x, :, 1$) ₂		1	0	0	1	1	0	1	0	1	0	1	1	0
<i>RealH</i> ($x, :, 2$)		0	0	1	1	1	0	0	1	0	1	1	1	1
<i>InfDH</i> ($x, :, 2$) ₂		0	0	1	1	1	0	0	1	0	1	1	1	1

The sequences have been aligned perfectly with 2 flips. This is the switch error for individual x . The switch error is calculated for each individual and their average is outputted as the average switch error.

The switch error can also be found computationally by first taking x 's inferred and real haplotypes and filtering out the homozygous sites.

	Sites	1	3	5	7	8	9	10	13
	Columns	1	2	3	4	5	6	7	8
<i>RealH</i> _{htrz} ($x, :, 1$)		1	0	1	1	0	1	0	0
<i>InfDH</i> _{htrz} ($x, :, 1$)		1	0	1	0	1	0	0	0
<i>RealH</i> _{htrz} ($x, :, 2$)		0	1	1	0	1	0	1	1
<i>InfDH</i> _{htrz} ($x, :, 2$)		0	1	1	1	0	1	1	1

An equality analysis between the remaining sites is then performed as in:

$$EqualMat(x, :, :) = InfDH_{htrz}(x, :, :) == RealH_{htrz}(x, :, :)$$

This gives:

	Columns	1	2	3	4	5	6	7	8
<i>EqualMat(x, :, 1)</i>		1	1	1	0	0	0	1	1
<i>EqualMat(x, :, 2)</i>		1	1	1	0	0	0	1	1

The number of flips can be counted by the number of times *EqualMat(x, :, 1)* or *EqualMat(x, :, 2)* changes value. In this case, this happens twice (at columns 4 and 7 representing the flips at sites 7 and 10) and therefore the number of flips or the switch error for x is 2.

Haplotypes Correctly Inferred

Another method often used to measure the accuracy of haplotype inference algorithms is the percentage of haplotypes correctly inferred completely. This can be done simply by checking for each individual x whether the pair in *RealH(x, :, :)* is exactly equal to the pair in *InfdH(x, :, :)*, in any order of haplotypes. This divided by the number of individuals n gives the ratio of haplotypes correctly inferred.

Although widely used, this is not the best measure of accuracy since it only gives credit to fully correct haplotypes. With this measure, a method that infers a few haplotypes completely correctly and the rest quite erratically would seem significantly better than one that infers all haplotypes quite well in terms of sites accurately inferred but very few completely correctly. Moreover, the number of haplotypes inferred completely correctly decreases very quickly as the number of sites is increased and eventually this measure becomes too small to indicate any value for accuracy.

Average Heterozygous Accuracy/Individual

Other methods of measuring accuracy include finding the average heterozygous accuracy per individual. The ratio of heterozygous sites accurately inferred in each individual can be very easily computed following the same method as that for computing the ratio of heterozygous sites accurately for the entire sample. A function *compareHap.m* is written for the explicit purpose of doing this for an individuals pairs. The ratio for each individual are averaged to give a reliable accuracy measure. The *standard deviation* of these values is also noted to observe how erratically the algorithms perform across individuals.

5 Testing and Evaluation

5.1 The Test Plan and its Implementation

As mentioned in 4.6, several criteria can be used to evaluate the accuracy of the haplotype inference methods. The aim of this project was to analyse and evaluate the performance of the implemented algorithms against algorithms currently in use.

Algorithms to compare against

The following methods were chosen to compare performances with implementations of the DPRH and OPRH algorithms against:

- **Haplotyper** [17]
- **FastPHASE** [22]

The operation of Haplotyper is briefly described in Section 2.6. FastPHASE is made by the same developer and is based on the same statistical methods as PHASE [24] which is also described in Section 2.6.

FastPHASE as the name suggests can be thought of as a faster version of PHASE and shares Matthew Stephens as one of the developers. Evaluation of fastPHASE which was electronically published just a few months back in February 2006 suggests that it is only slightly less accurate than PHASE but runs in a fraction of the time [22].

The author also considered to test HAP [6] described in Section 2.3.2 which would have been very interesting to analyse since HAP like the developed DPRH and OPRH algorithms attempts to solve the imperfect phylogeny haplotype (IPPH) problem exploiting phylogenetic relationships in parts of the dataset. Unfortunately, a download version of HAP is not available. Researchers can use HAP to infer haplotypes via a web interface and obtain the results via email. This was simply not feasible for the evaluation purposes of this project especially since a large number of tests and time measures were required.

Test data used for evaluation

Because of the lack of availability of real haplotype data, the **ms** program [16] was used to produce simulated data for this section. **ms** can be used to produce haplotype sequences with the following parameters*

<i>s</i>	number of sequences, i.e. number of individuals $\times 2$. i.e. $n = s/2$
<i>m</i>	number of sites
<i>r</i>	recombination rate

The algorithms were to be tested with varying number of individuals, sites and recombination rates. 10 datasets were generated for each of the combinations/configurations of the parameters with their possible values being from the following sets:

<i>Number of individuals</i>	<i>N</i>	{10, 25, 50, 75, 100}
<i>Number of sites</i>	<i>M</i>	{10, 25, 50, 75, 100, 125, 150, 175, 200}
<i>Recombination rate</i>	<i>R</i>	{10, 25, 50, 100}

Therefore one can think of a configuration *c* to be a element with a number of individuals from *N*, a number of sites from *M* and a recombination rate from *R*. The set of all configurations *C* can be formalised as:

* **ms** uses a number of other parameters which are not relevant to generating datasets for this project

$$C : N \times M \times R \wedge \forall n \in N, \forall m \in M, \forall r \in R \bullet (n, m, r) \in C$$

A small program was written in java that simulated 10 sets of data with each combination (n, m, r) using ms and stored each such set in filenames of the format:

$$n-m-r\#i.txt$$

where i indicates that it is the i -th run with that configuration. E.g. filename “50-100-25#4.txt” indicates the file is the 4th dataset with haplotype sequences of 50 individuals (i.e. 100 sequences with a pair for each individual), with 100 sites each and was simulated with a recombination rate of 25.

The set of filenames F can be formalised as:

$$I = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

$$F : N \times M \times R \times I \wedge \forall n \in N, \forall m \in M, \forall r \in R, \forall i \in I \bullet (n, m, r, i) \in F$$

or $F : C \times I \wedge \forall c \in C, \forall i \in I \bullet (c, i) \in F$

The author’s initial aim was to test for upto 200 individuals but the public release of Haplotyper only allowed a maximum of 100 individuals while fastPHASE took over an hour to infer haplotypes for datasets with 200 individuals and it was impractical to run multiple configurations on it with such huge datasets.

For the purposes of these simulations and their evaluation, one can consider $N < 50$ and $M < 75$ as small data-sets in terms of individuals and sites respectively, whereas $N > 50$ and $M > 75$ can be considered as large datasets. Recombination rates of 10 can be considered low, 25 as medium and 50-100 as high. This is summarised in Table 5.1

Number of Individuals	<i>n</i>	Small	{10, 25}
		Medium	{50}
		Large	{75, 100}
Number of Sites	<i>m</i>	Small	{10, 25, 50}
		Medium	{75, 100, 125}
		Large	{150, 175, 200}
Recombination Rate	<i>r</i>	Low	{10}
		Medium	{25}
		High	{50, 100}

Table 5.1: Classification of number of individuals, sites and recombination rates

Testing Mechanism

To conduct tests for each configuration, the data from the $n-m-r\#i.txt$ file would have to be read into a matrix called *RealH*. The function *file2hap.m* (Section 4.2) was written for this purpose. The genotypes G would then be derived from *RealH* using *hap2gen.m*. Implementations of DPRH1, DPRH2 and OPRH simply accepted G as the input and returned *InfdH* as the inferred haplotypes.

Unfortunately, fastPHASE and Haplotyper do not accept genotypes in the same file format as that generated by executions of ms. Two functions, *runFPHASE.m* and *runHTyper.m* were written to solve this problem. The functions accept G as input, run fastPHASE or Haplotyper respectively on the dataset and returned H as the inferred haplotypes. This means these functions had the same interfaces (input and output parameters) as *runDPRH1.m*, *runDPRH2.m* and *runOPRH.m*.

runFPHASE.m first takes the input matrix G and write it to a file called temp.pph in the format that fastPHASE accepts genotypes as input. It then executse fastPHASE via a shell and then opens the output file generated. It then goes through the output file and retrieves only the inferred sequences and

writes them onto the matrix H which it returns as its output. *runHTypeper.m* performs similar operations for inferring haplotypes using the Haplotyper executable.

The function *evalHInf.m* (Section 4.6) can be run with *RealH* and H as parameters which outputs the several criteria of accuracy mentioned earlier.

A module *testAlgs.m* was written to perform a comprehensive tests of the algorithms given the ms generated datasets. It infers haplotypes for five datasets of each configuration (n, m, r) using the methods DPRH1, DPRH2, OPRH, fastPHASE and Haplotyper. For each run (i.e. each dataset for a configuration) it takes down the time taken to execute each method and all the accuracy measures and writes them onto an excel spreadsheet file “results.xls”.

For each configuration it also takes down the average time taken and accuracy measures for each of the methods across the five datasets of the configurations and writes them onto a file called “resultsSum.xls”. This file is crucial for analysis of the performance of the algorithms since the values are more reliable when averaged across a number of runs.

The contents of the “resultsSum.xls” file can be found in Appendix C.

5.2 Evaluation of Accuracy

From among the accuracy criteria, the average accuracy of heterozygous sites inferred per individual (output as *avgIndAcc* by *evalHInf.m*) was chosen as the best way of evaluating the performance of individuals. The accuracy of heterozygous sites inferred throughout the sample was also considered but these two measures are highly correlated.

The switch error gave very little information since it showed a very strong correlation with the number of sites inferred and values of switch error/ m were simply too small to draw proper conclusions from. Section 4.6 includes a detailed discussion regarding why other measures such identity ratio and proportion of correctly inferred haplotypes should not be used in the evaluation of inferences.

With three parameters for each configuration (n, m, r). It is very difficult to represent the results using two dimensions. Therefore, the author chose to break down the analysis and its conclusions into the following categories:

- Runs with varying recombination rates (r) and fixed number of individuals (n) and number of sites (m)
- Runs with varying number of individuals (n) and fixed number of sites (m) and recombination rates (r)
- Runs with varying number of sites (m) and fixed number of individuals (n) and recombination rates (r)

5.2.1 Varying Recombination Rates

Datasets with a small number of individuals and number of sites give too little information to infer haplotypes properly from. As a result all the methods gave more or less equal results but their performances decreased rapidly with increasing recombination rates as shown in Figure 5.1a.

Figure 5.1b, shows the graph for runs with a small number of individuals and a large number of sites. OPRH seems to perform quite badly compared to the others initially however they all seem to converge to similar levels of low accuracy with increasing r . This is because there is very little information with a small number of sites to infer haplotypes with Regional PPH or Bayesian techniques.

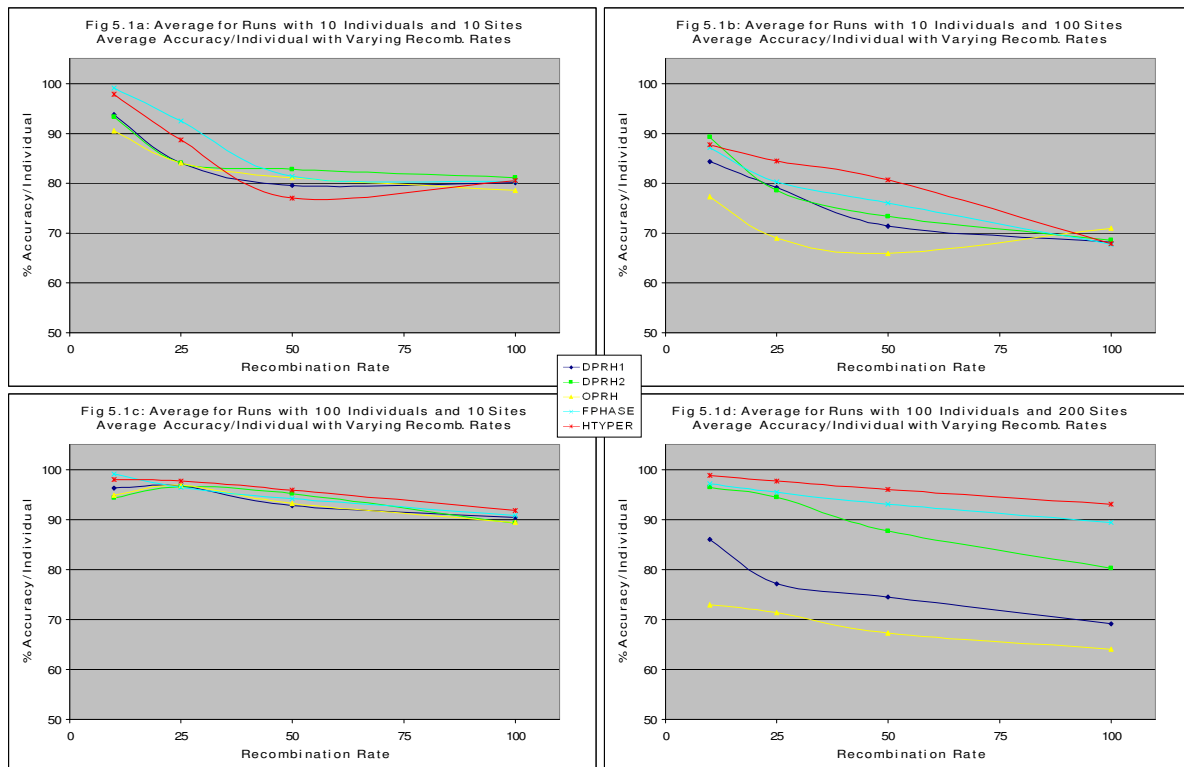


Figure 5.1: Accuracy evaluation across varying recombination rates

On the other hand, with a large number of individuals and a small number of sites, all methods seem to perform more or less similarly (Figure 5.1c). With large datasets, Haplotyper and FastPHASE seem to perform much better than DPRH1 and OPRH regardless of the recombination rate (Figure 5.1d). DPRH2's performance however is very comparable to that Haplotyper and FastPHASE for low-medium recombination rates. The same is true for experiments with moderate sized data sets in terms of individuals and sites.

5.2.2 Varying Number of Individuals

With low recombination rates, DPRH2 is as good as FastPHASE and Haplotyper in terms of accuracy with DPRH1 and OPRH slightly behind (Figure 5.2a). However as the number of sites is increased, the gap between DPRH2 and the established duo widens slightly with DPRH1 and especially OPRH lagging behind always as shown in Figure 5.2b.

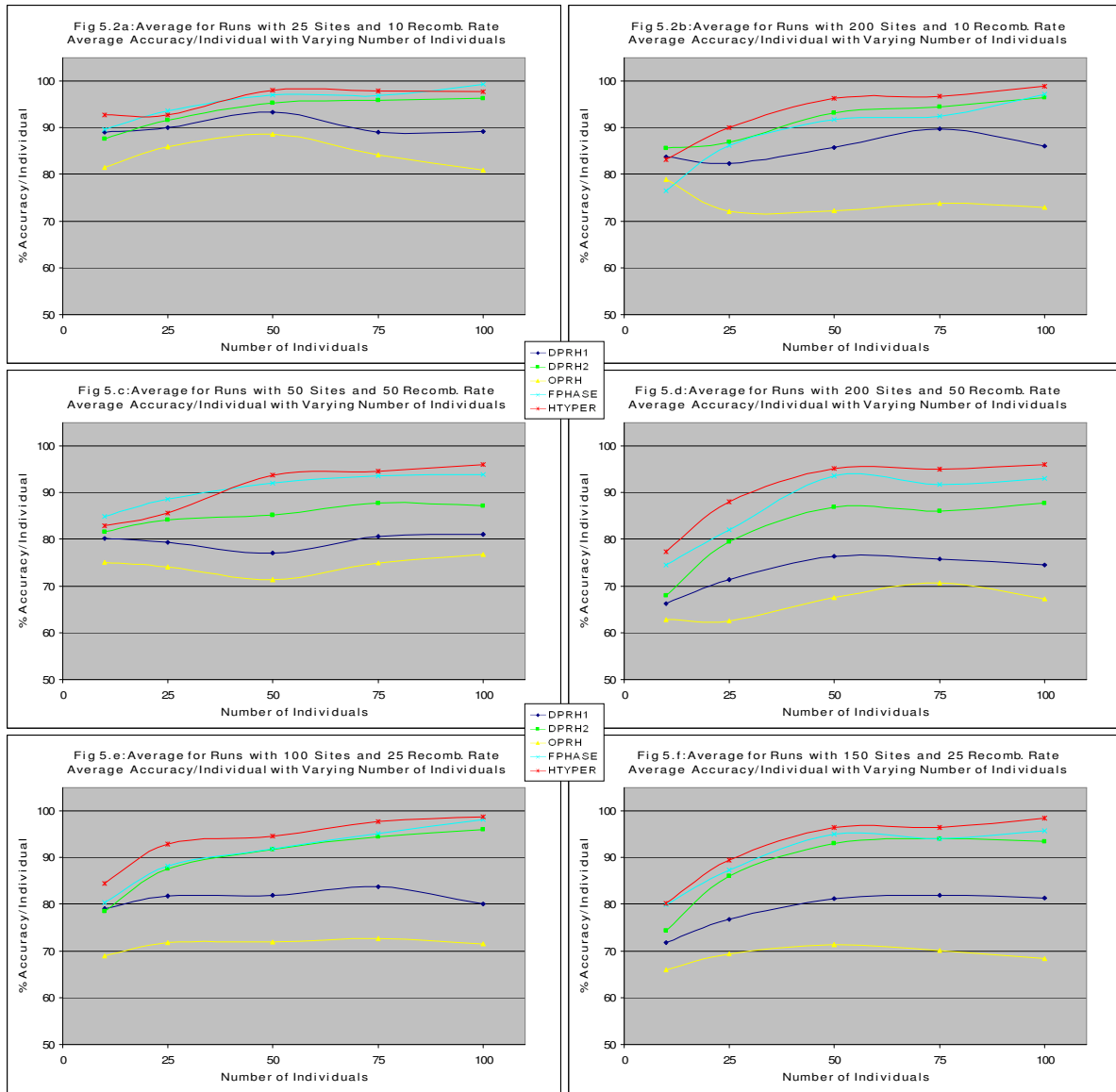


Figure 5.2: Accuracy evaluation across varying number of individuals

With small recombination rates, all the methods other than OPRH seem to work quite well, however with larger rates the gap between the established methods and the engineered ones becomes obvious. Among the latter, only DPRH2 gives results with any credibility at all (Figure 5.2c-d)

Regardless of the number of sites DPRH2 is almost as good as fastPHASE and Haplotyper in terms of accuracy (Figure 5.2e-f).

DPRH1 and OPRH however have proved quite disappointing throughout and are therefore not talked about for the rest of the accuracy evaluation section.

5.2.3 Varying Number of Sites

As mentioned earlier, the algorithms seem to prove to be quite similar in terms of performance for a small number of individuals and low recombination rates. This is once again shown in Figure 5.3a. For higher recombination rates, all methods have the same level of accuracy for a small number of sites but that of the engineered methods decreases slowly with increasing sites (Figure 5.3b).

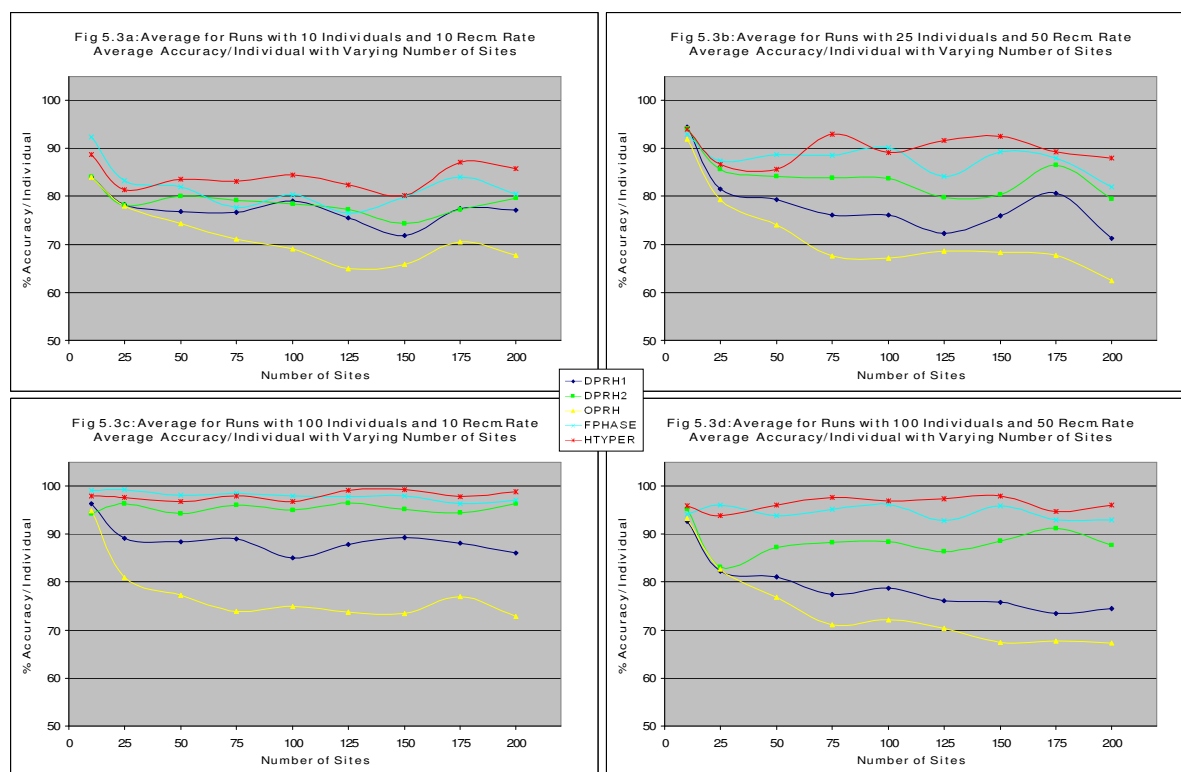


Figure 5.3: Accuracy evaluation across varying number of sites

DPRH2 seems to perform as well as the established ones for low recombination rates even for large sample sizes (Figure 5.3c) but this is no longer the case if the recombination rate is increased (Figure 5.3d).

5.3 Evaluation of Speed

Execution time of each of the methods was recorded by *testAlgs.m* in seconds to 2 decimal places. To save time, the set of configurations in *C* (Section 5.1) for DPRH1, DPRH2, OPRH, Haplotyper and fastPHASE was broken up and run on six Sun Solaris machines simultaneously, all of which had the following configuration:

Intel Pentium 4
2.6 GHz Processor
1024 MB RAM

DPRH1, DPRH2 and OPRH have the disadvantage of running *lpph* at least m times for a inferring a $n \times m$ dataset of genotypes, and each of these runs involves writing an input file and reading from an output file. This combined with the fact that they are being run from an interpreter driven platform (MATLAB) puts them at a great disadvantage and these factors must be kept in mind while evaluating their speeds.

Varying recombination rates seem to affect the time very slightly. The time for Halotyper, DPRH1, DPRH2 and OPRH increase very slowly with increasing recombination rates as shown in Figure 5.4.

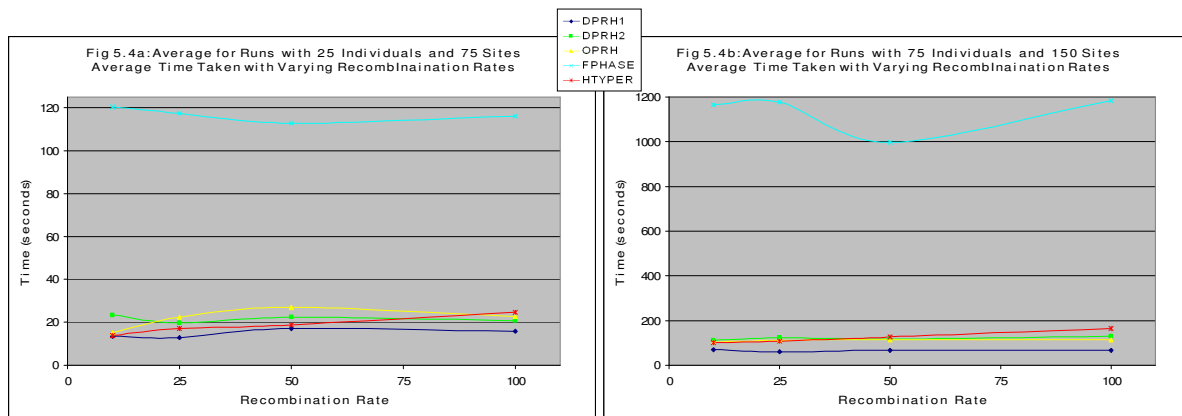


Figure 5.4: Speed evaluation across varying recombination rates

Since the recombination rates did not affect speeds much, the times taken for the algorithms to execute was not analysed in this report across varying rates. Instead, only the following categories of runs were used to evaluate the performance of algorithms:

- Runs with varying number of individuals n and fixed number of sites m
- Runs with varying number of sites m and fixed number of individuals n

5.3.1 Varying Number of Individuals

Regardless of the number of sites (Figure 5.4a-b), FastPHASE seems to take much more time than others. The other methods all seem to take similar amounts of time with DPRH1 always being the fastest.

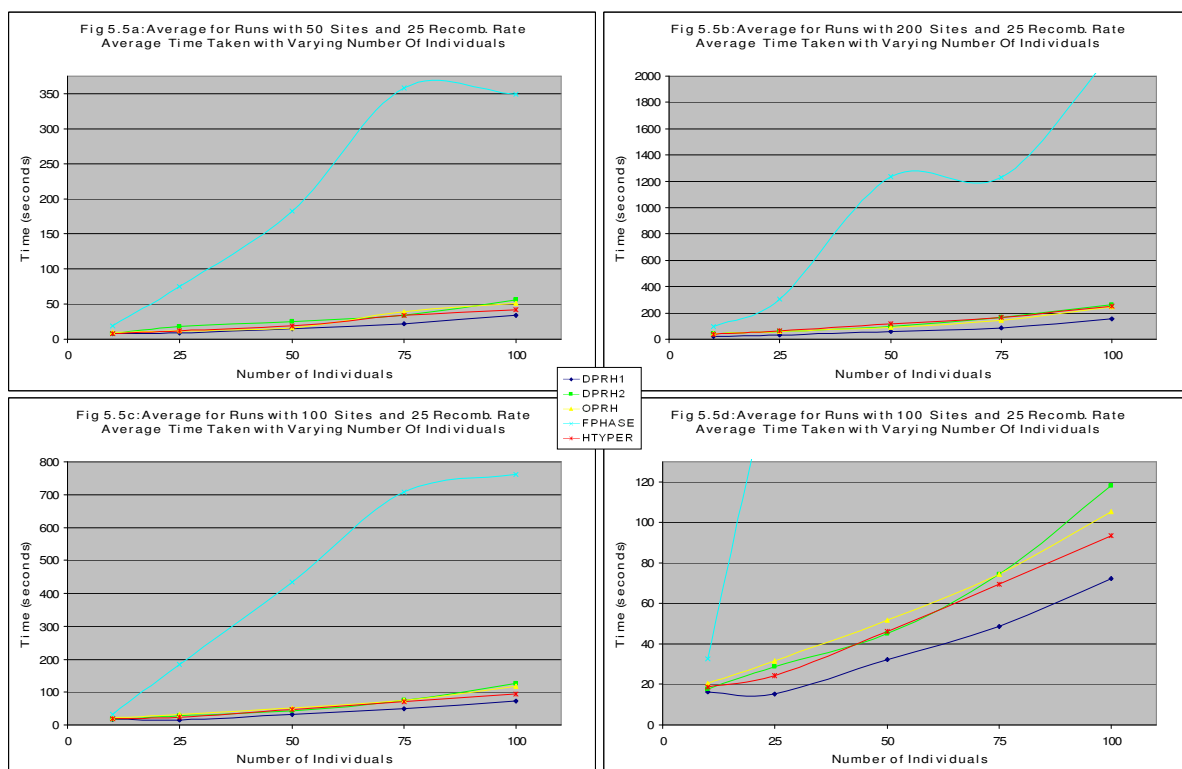


Figure 5.5: Speed evaluation across varying number of individuals

Figure 5.4c shows the same thing with fastPHASE taking vast amounts of time. The others appear to have a linear relationship between time taken and number of individuals. A closer look (Fig 5.4d)

confirms this for DPRH1, OPRH and Haplotyper. For DPRH2, time seems to be related to slowly increasing function of the number of individuals.

5.3.2 Varying Number of Sites

The results show that fastPHASE takes much longer time than all others (Figure 5.21) and the relationship seems exponential in nature. A closer look at the runs show that the all other methods have linear relationships with respect to sites (Figure 5.5b). For moderate sized data sets, DPRH1 seems to be the fastest. OPRH and DPRH2 are just behind it but still perform better than Haplotyper. The same is true for small datasets as shown in Figure 5.5c. However for a large number of individuals, Haplotyper runs a bit faster than DPRH2 for a large number of sites i.e. $m > 150$.

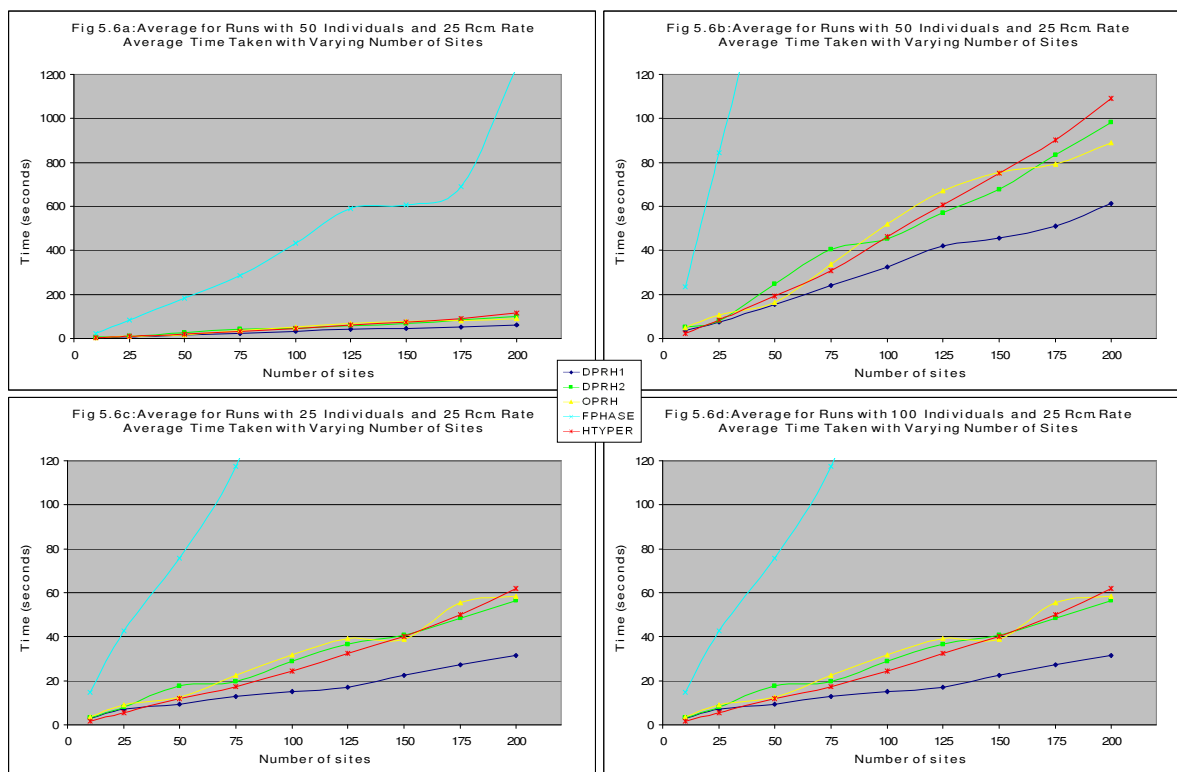


Figure 5.6: Speed evaluation across varying number of sites

Summary of Evaluation

The following is list of observations made regarding the performance of the engineered and published algorithms via the evaluation conducted as part of this project

Accuracy Evaluation

- The performances of all methods deteriorate with increasing recombination rate. However even for high rates, Haplotyper and fastPHASE still maintain decent (although decreasing) accuracy rates. DPRH2 is almost as good as them when recombination rates are low-medium but performs poorly with anything higher.
- DPRH1 and especially OPRH perform quite poorly compared to the other methods for almost all configurations unless they are of low recombination rates coupled with small sample sizes and/or number of sites. This is simply because with small datasets there is very little to infer haplotypes using and so the other methods perform poorly.

- All methods seem to perform better with more individuals in the dataset. This is because the HI inference techniques used involve using the individuals to find phylogenetic relationships or statistical patterns both of which are more reliable with larger datasets.
- Unless the number of sites is very small, the number of sites does not really affect the accuracy of any of the methods.

Speed Evaluation

- fastPHASE takes considerable more time compared to all the other methods whereas DPRH1 takes the least amount of time for all configurations of datasets.
- DPRH1, OPRH, FastPHASE and Haplotyper have linear relationships between time and number of individuals, whereas this is a slowly increasing function for DPRH2. In fact, both DPRH1 and DPRH2 are slowly increasing functions of n however it is more evident in case of DPRH2 since the alignment part of this algorithm takes more time than DPRH1, whereas they share the phylogeny finding part. The alignment algorithm is in fact $O(mn^2)$ unless a very efficient hashtable is used to lookup the frequencies of sequences in *PossSeqs* or *PossHaps* but in this implementation they are both lists.
- FastPHASE has an exponential relationship between time and number of sites. DPRH1, DPRH2, OPRH and Haplotyper on the other hand have linear relationships.
- DPRH2 and OPRH usually take less time than Haplotyper.
- DPRH2 seems to take more time than Haplotyper only for a large number of sites and a large number of individuals.
- One must bear in mind that DPRH1, DPRH2 and OPRH could run faster if they were compiled rather than interpreted and if they encapsulated the LPPH program in their code rather than use it as a black-box.

6 Conclusion

This chapter starts off by summarising the previous chapters in this dissertation. It also gives a brief evaluation of the research results obtained. It concludes with suggestions of future work that may be carried out to improve the algorithms engineered.

6.1 Summary of Dissertation

Chapter 1 titled “Introduction” gave a brief introduction to haplotype inference. It also described the main aims of this project and its association to the M.Sc. in Computer Science at the Oxford University Computing Laboratory.

Chapter 2 titled “Background and Literature Review” introduced the reader to basic concepts in genetics related to haplotype inference. It covered summaries of all literature researched by the author in order to achieve the objectives laid out in Chapter 1.

Chapter 3 titled “Description of Proposed Methods” explained the operations and complexities of the algorithms engineered by the author (DRPH1, DPRH2 and OPRH) to infer haplotypes from genotypes in the presence of recombination.

Chapter 4 titled “Implementation of Engineered Algorithms” covered the implementation details of all functions written in MATLAB for the algorithms engineered in Chapter 3. It also provided a brief justification for choosing MATLAB as the development environment.

Chapter 5 titled “Testing and Evaluation” described a comprehensive test plan and used its results to evaluate the performance of the three engineered algorithms against established HI programs fastPHASE and Haplotyper in terms of speed and accuracy.

6.2 Research Results

The observations made from the runs in Chapter 5 have been summarised here. They can be considered the final conclusions that can be drawn from this project regarding the engineered HI methods:

- DPRH1 - The implementation method seems to run very fast with an empirical complexity of only $O(mn)$. However its results are only reliable for genotypes produced at very low recombination rates.
- DPRH2 - This method seemed to perform the best among the engineered ones in terms of accuracy. It was comparable to fastPHASE and Haplotyper for datasets low-medium recombination rates. It operated slightly slower than DPRH1 at an empirical complexity of $O(m\alpha(n))$ where α is a slowly growing function.
- OPRH - The implementation of OPRH gave very disappointing results in terms of accuracy for almost all datasets. This is simply because there is too little information in overlaps between regions of perfect phylogeny to be used reliably for the alignment of partial solutions. The empirical complexity of OPRH was $O(mn)$ but it always executed slower than DPRH1.

6.3 Further Work

Due to the time constraints mentioned in Chapter 1, many possible enhancements to algorithms had to be left out. These are briefly described in this section.

- The main concern with regards to the evaluation was the fact that LPPH was being used as a black box executable for which the DPRH and OPRH implementations had to repeatedly write input files and read outputs files for finding regions of phylogeny. Furthermore, these algorithms were run interpreted which put them at a significant disadvantage to compiled programs. If the source code for LPPH were to be obtained and incorporated into compiled versions of the DPRH and OPRH implementations, their speeds could be better evaluated. The author believes that this would prove the engineered algorithms much faster than other IPPH methods currently in use.
- An improvement can be made to the DPRH1 and DPRH2 implementations. Hashtables can be used rather than lists to maintain the frequencies of occurring subsequences when aligning regional solutions. With an efficient implementation of hashtables, the algorithms can be expected to adhere to an empirical complexity of $O(nm)$ rather than $O(m\alpha(n))$.
- One measure of evaluation left out from Chapter 5's test plan was that concerning the pure parsimony criterion. The number of distinct haplotypes divided by the number of genotypes is a good measure of how well an algorithm satisfies this criterion and a function for determining this value from the inferred haplotypes could be written very easily in MATLAB. Unfortunately, the author was not initially concerned about pure parsimony and only considered evaluating algorithms with regards to it after the entire test plan had been executed (which took over a week) and only the relevant data was recorded which was insufficient to perform parsimony measures.
- Once the regional solutions are inferred using the DPRH algorithms, the Gibbs Sampler can be used to find the best possible alignment of sequences for each individual. Since the number of regions even for datasets with high recombination rates is smaller than the number of heterozygous sites, this approach would produce solutions faster than current stochastic methods and more accurate than the DPRH algorithms.
- The DPRH and OPRH algorithms can be combined to give heuristics for better alignment of regional solutions. For example, if DPRH1 or DPRH2 suggests an alignment *sol1* with a degree of confidence x (perhaps calculated by the number of times a solution appears in *PossHaps* divided by n) and OPRH suggests the same with a degree of confidence y (perhaps calculated by the percentage match in the overlapping part of *sol1*), *sol1* is chosen as the correct solution if

$$a.x_{sol1} + b.y_{sol1} > a.x_{sol2} + b.y_{sol2}$$

where a and b are constants

- The LPPH program can be used to obtain the shadow tree for regions of perfect phylogeny in a sample of genotypes. The phylogenies of adjacent regions should differ by a single subtree pruning or regrafting operation assuming a recombination event has taken place. These relationships between adjacent phylogenies can be used to update a tree structure maintained throughout the operation of a new HI method, which can be used to infer haplotypes despite recombination. However this method might prove to be very computationally complex and heuristics may have to be used in order to obtain solutions in reasonable time.

Bibliography

- [1] V. Bafna, D. Gusfield, G. Lancia, and S. Yooseph: “*Haplotyping as perfect phylogeny: A direct approach*”, *Journal of Computational Biology*, 10:323-340, 2003.
- [2] R.H. Chung and D. Gusfield: “*Empirical exploration of perfect phylogeny haplotyping and haplotypers*”, *Proceedings of the 9th International Conference on Computing and Combinatorics COCOON03*. *Spring Lecture Notes in Computer Science*, Vol. 2697, 5-19, 2003.
- [3] R.H. Chung and D. Gusfield: “*Perfect phylogeny haplotyper: Haplotype inferral using a tree model*”, *Bioinformatics*, 19(6):780-781, 2003.
- [4] A. Clark: “*Inference of haplotypes from PCR-amplified samples of diploid populations*”. *Molecular Biology and Evolution*, 7:111-122, 1990.
- [5] Z. Ding, V. Filkov, and D. Gusfield: “*A linear-time algorithm for the perfect phylogeny haplotyping problem*”. *Proceedings of the Ninth Annual International Conference on Computational Biology (RECOMB)*, 2005.
- [6] E. Eskin, E. Halperin, and R. M. Karp: “*Efficient reconstruction of haplotype structure via perfect phylogeny*”, *Journal of Bioinformatics and Computational Biology*, 1:1-20, 2003.
- [7] S. M. Fullerton, A. V. Buchanan, V. A. Sonpar, S. L. Taylor, J. D. Smith, C. S. Carlson, V. Salomaa, J. H. Stengard, E. Boerwinkle, A. G. Clark, D. A. Nickerson, and K. M. Weiss: “*The effects of scale: variation in the apoA1/c3/a4/a5 gene cluster*”, *Human Genetics*, 115:36-56, 2004.
- [8] D. Gusfield: “*Inference of haplotypes from samples of diploid populations: complexity and algorithms*” *Journal of Computational Biology*, 8(3):305-323, 2001.
- [9] D. Gusfield: “*Haplotyping as perfect phylogeny: Conceptual framework and efficient Solutions*” (Extended Abstract). In *Proc. of RECOMB*, pages 166-175, 2002.
- [10] D. Gusfield: “*Haplotype inference by pure parsimony*”, In E. Chavez R. Baeza-Yates and M. Crochemore, editors, *14th Annual Symposium on Combinatorial Pattern Matching (CPM'03)*, volume 2676, pages 144-155. *Springer LNCS*, 2003.
- [11] D. Gusfield and S.H. Orzack: “*Haplotype Inference*”, *CRC Handbook on Bioinformatics* (S. Aluru Editor), 2005.
- [12] E. Halperin and E. Eskin: “*Haplotype reconstruction from genotype data using imperfect phylogeny*”, *Bioinformatics*, 20:1842-1849, 2004.
- [13] International HapMap Project: “*Information: About the Project*”, <http://www.hapmap.org/>, 2006.
- [14] J.J. Hein: “*A Heuristic Method to Reconstruct the History of Sequences Subject to Recombination*”, *J.Mol.Evol.* 20.402-411, 1993.
- [15] P.S. Ho: “*A combinatorial approach to predicting haplotypes from genotypes in the presence of recombination*”, *MSc Dissertation*, Oxford University Computing Laboratory, 2005.
- [16] R.R. Hudson: “*Generating samples under a Wright-Fisher neutral model of genetic variation*”, *Bioinformatics* 18:337-338, 2003.
- [17] T. Niu, Z. Qin, X. Xu, and J.S. Liu: “*Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms*”, *American Journal of Human Genetics*, 70:157-169, 2002.
- [18] L.E. Orgel: “*The origin of life on the earth*”, *Scientific American* 271(4) (Oct): 76-83, 1994.

Bibliography

- [19] T. Ohta and M. Kimura: “*On the constancy of the evolutionary rate of cistrons*”, *Molecular Biology and Evolution*, 1:18-25, 1971.
- [20] OUCL: “*M.Sc. in Computer Science, Course Handbook, 2005-06 Edition*”, Oxford University Computing Laboratory, 2005.
- [21] OUCL: “*Current Students, Course Materials 2005-2006*”, Available from <http://web.comlab.ox.ac.uk/internal/courses/materials05-06/>, Oxford University Computing Laboratory, 2006.
- [22] P. Scheet and M. Stephens: “*A fast and flexible statistical model for large-scale population genotype data: applications to inferring missing genotypes and haplotypic phase*” *Am J Hum Genet* 78: 629-644, 2006.
- [23] Y.S. Song, Y. Wu & D. Gusfield: “*Algorithms for Imperfect Phylogeny Haplotyping (IPPH) with a Single Homoplasy or Recombination Event*”, *Proceedings of Workshop on Algorithms in Bioinformatics 2005, Lecture Notes in Computer Science 3692*, 152-164, 2005.
- [24] M. Stephens, N. Smith, and P. Donnelly: “*A new statistical method for haplotype reconstruction from population data*”, *American Journal of Human Genetics*, 68:978-989, 2001.
- [25] M. Stephens and P. Donnelly: “*A comparison of Bayesian methods for haplotype reconstruction from population genotype data*”, *American Journal of Human Genetics*, 73:1162-1169, 2003.
- [26] B. S. Weir: “*Genetic Data Analysis II*”, Sinauer Associates 1996.
- [27] Wikipedia: “*Mutation*”, <http://www.wikipedia.org>, Wikipedia The Free Encyclopaedia, 2006.
- [28] Wikipedia: “*Reproduction*”, <http://www.wikipedia.org>, Wikipedia The Free Encyclopaedia, 2006.

Appendix A - Code Listing

CONTENTS

RUNDPRH1 - performs DPRH1 haplotype inference	50
RUNDPRH2 - performs DPRH2 haplotype inference	51
RUNOPRH - performs OPRH haplotype inference	51
RUNLPPH - performs haplotype inference using the LPPH method	53
FINDDPRS - finds disjoint PPH regions and partial solutions for genotypes	54
COMPAREHAP - compares two haplotype pairs	55
COMPTRZ - compares htrz sites of two sequences	55
DISPLAYGENS - displays genotypes in a given matrix	55
DISPLAYHAPS - displays haplotypes in a given matrix	56
EQUALRAT - compares two matrices	56
EVALHINF - evaluates haplotypes inferred	56
FREQ - finds frequency of a row in a matrix	57
FILE2GEN - read genotypes from a text file into a matrix	58
FILE2HAP - reads haplotypes from a text file into a matrix	58
HAP2GEN - gets genotypes from haplotypes	59
LEAFCOUNT - calculates leafcount	59
N2STR - converts a number to a string	59
RIGHTMOST1 - returns the position of the rightmost 1	59
ROWS - gives number of rows	60
SWAP - helps swap two values	60
TXT2GEN - read genotypes from text matrix to a genotype matrix	60
TXT2HAP - read haplotypes from text matrix to a haplotype matrix	60
RUNFPHASE - performs haplotype inference via the fastPHASE method	61
RUNHTYPER - performs haplotype inference via the Haplotyper method	62
TESTALGS - tests and evaluates HI algorithms	63

LISTING

```
% RUNDPRH1 - performs DPRH1 haplotype inference
%
% This is the implementation of the DPRH1 algorithm designed. It takes in a
% nxm matrix of genotypes G, infers haplotypes via DPRH1 and returns them
% in a nxmx2 matrix H.
%
% Usage:   H = runDPRH1(G)
%
% Arguments:
%   G - matrix of genotypes containing 0s, 1s and 2s of dimensions nxm
%       where n is the no individuals and m is the no of sites
%
% Returns:
%   H - matrix of haplotypes containing 0s and 1s of dimensions nxmx2
%       where n is the no of individuals and m is the no of sites in G
%
function H = runDPRH1(G)

[n m] = size(G);
disp(' ');
disp('Inferring haplotypes from genotypes using DPRH1');
disp([num2str(n) ' individuals with ' num2str(m) ' sites each'])
% uses findDPRs to get disjoint regions & unaligned regional solutions as H
[H Regions] = findDPRs(G);

% alignment of regional solutions
for k = 2:rows(Regions)

    i = Regions(k-1, 1);
    j = Regions(k-1, 2);
    p = Regions(k, 1);
    q = Regions(k, 2);

    PossSeqs = [];
    % PossSeqs contains all possible sequences that may be formed with
    % partial solutions across two adjacent regions
    for x = 1:n
        PossSeqs = [PossSeqs; H(x, i:j, 1) H(x, p:q, 1)];
        PossSeqs = [PossSeqs; H(x, i:j, 2) H(x, p:q, 2)];
        PossSeqs = [PossSeqs; H(x, i:j, 1) H(x, p:q, 2)];
        PossSeqs = [PossSeqs; H(x, i:j, 2) H(x, p:q, 1)];
    end

    for x = 1:n
        if freq(PossSeqs, [H(x, i:j, 1) H(x, p:q, 2)])+freq(PossSeqs, [H(x, i:j, 2) H(x, p:q, 1)]) > freq(PossSeqs, [H(x, i:j, 1) H(x, p:q, 1)])+freq(PossSeqs, [H(x, i:j, 2) H(x, p:q, 2)])
            % if the frequency of the default alignment is less than that
            % of the other, then the latter part of the solution is flipped
            [H(x, p:q, 1) H(x, p:q, 2)] = swap(H(x, p:q, 1), H(x, p:q, 2));
        end
    end
end
```

Please note that the entire source code and all test results are not available in this online version.

Appendix A - Code Listing

```

end

end
disp(' ');
disp(' ');
disp('Haplotypes Inferred!');
disp(' ');
disp(' ');

return

% RUNDPRH2 - performs DPRH2 haplotype inference
%
% This is the implementation of the DPRH2 algorithm designed. It takes in a
% nxm matrix of genotypes G, infers haplotypes via DPRH2 and returns them
% in a nxmx2 matrix H.
%
% Usage: H = runDPRH2(G)
%
% Arguments:
% G - matrix of genotypes containing 0s, 1s and 2s of dimensions nxm
% where n is the no individuals and m is the no of sites
%
% Returns:
% H - matrix of haplotypes containing 0s and 1s of dimensions nxmx2
% where n is the no of individuals and m is the no of sites in G
%
function H = runDPRH2(G)

[n m] = size(G);
disp(' ');
disp('Inferring haplotypes from genotypes using DPRH2');
disp([num2str(n) ' individuals with ' num2str(m) ' sites each'])
[H Regions] = findDPRs(G);

for k = 2:rows(Regions)

    i = Regions(k-1, 1);
    j = Regions(k-1, 2);
    p = Regions(k, 1);
    q = Regions(k, 2);

    % PossHaps contains all possible sequences that may be formed with
    % already aligned solutions and solutions from current region (p, q)
    PossHaps = [];
    for x = 1:n
        PossHaps = [PossHaps; H(x, 1:j, 1) H(x, p:q, 1)];
        PossHaps = [PossHaps; H(x, 1:j, 2) H(x, p:q, 2)];
        PossHaps = [PossHaps; H(x, 1:j, 1) H(x, p:q, 2)];
        PossHaps = [PossHaps; H(x, 1:j, 2) H(x, p:q, 1)];
    end

    % PossSeqs contains all possible sequences that may be formed with
    % partial solutions across two adjacent regions (i, j) and (p, q)
    PossSeqs = [];
    for x = 1:n
        PossSeqs = [PossSeqs; H(x, i:j, 1) H(x, p:q, 1)];
        PossSeqs = [PossSeqs; H(x, i:j, 2) H(x, p:q, 2)];
        PossSeqs = [PossSeqs; H(x, i:j, 1) H(x, p:q, 2)];
        PossSeqs = [PossSeqs; H(x, i:j, 2) H(x, p:q, 1)];
    end

    for x = 1:n

        % comparing frequencies for alignment in PossHaps
        cmp = int8(freq(PossHaps, [H(x, 1:j, 1) H(x, p:q, 2)])+freq(PossHaps, [H(x, 1:j, 2)
H(x, p:q, 1)])) - int8(freq(PossHaps, [H(x, 1:j, 1) H(x, p:q, 1)])+freq(PossHaps, [H(x, 1:j,
2) H(x, p:q, 2)]));
        if cmp>0
            % if default alignment's frequency is less than the other then
            % the (p, q)'s regional solution is flipped
            [H(x, p:q, 1) H(x, p:q, 2)] = swap(H(x, p:q, 1), H(x, p:q, 2));
        end
        if cmp==0
            % if the PossHaps comparison is inconclusive then the DPRH1
            % method involving adjacent regional alignment is used
            if freq(PossSeqs, [H(x, i:j, 1) H(x, p:q, 2)])+freq(PossSeqs, [H(x, i:j, 2) H(x,
p:q, 1)])>freq(PossSeqs, [H(x, i:j, 1) H(x, p:q, 1)])+freq(PossSeqs, [H(x, i:j, 2) H(x, p:q,
2)])
                [H(x, p:q, 1) H(x, p:q, 2)] = swap(H(x, p:q, 1), H(x, p:q, 2));
            end
        end
    end

end

end

disp(' ');
disp(' ');
disp('Haplotypes Inferred via DPRH2');
disp(' ');disp(' ');

return

% RUNOPRH - performs OPRH haplotype inference
%

```

Appendix B - Sample Runs of Implementation

This section shows screen outputs from sample runs of the implementations of DPRH1, DPRH2 and OPRH. The programs were tested using simulated datasets created by ms (Section 5.1). Text followed by '>>'s are commands entered by the user to run the tests whereas text followed by '%'s are comments. For some inferences input files are shown as images. Please note that most of the runs involve loading from haplotype files so that the inferences can be compared to real haplotypes later.

```
>>
>> % RUNNING DPRH1 WITH 10 INDIVIDUALS,
>> % 25 SITES AND 10 RECOMBINATION RATE
>>
>> file = 'mssims/10-25-10#1.txt';
>> % selecting haplotype file to test
>> RealH = file2hap(file);
>> % reading real haplotypes from file
>> displayHaps(RealH);
>> % displaying real haplotypes
```



Haplotypes of 10 individuals with 25 sites each

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0
0 0 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 1
1 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1
```

```
>> G = hap2gen(RealH); % converts real haplotypes to genotypes
>> displayGens(G); % displays converted genotypes
```

Genotypes of 10 individuals with 25 sites each

```
0 0 0 0 0 2 0 0 0 0 2 2 0 0 2 2 2 2 0 0 0 0 0 0 0 1
0 0 2 0 0 0 0 0 0 0 0 0 0 0 2 2 0 2 0 0 0 0 0 0 0 1
0 0 0 0 0 2 0 0 2 2 0 0 2 2 0 2 2 0 1 0 0 0 0 2 2 2
0 0 2 2 0 0 2 2 2 2 0 0 0 0 0 0 2 2 0 0 2 0 0 0 1
0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
2 2 0 2 2 0 2 2 2 0 0 0 0 0 0 2 2 0 0 0 0 0 2 2 2
2 2 0 2 0 0 2 2 2 0 0 0 0 0 2 2 0 1 2 2 0 2 0 0 1
2 2 0 2 0 0 2 2 2 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 1
2 2 0 2 0 0 2 2 2 0 0 0 0 0 0 0 2 0 2 0 2 0 0 1
0 0 2 0 0 0 0 0 0 0 0 0 0 0 2 2 0 2 0 0 0 0 0 0 1
```

```
>> InfDH = runDPRH1(G); % infers haplotypes from genotypes via DPRH1
```

Inferring haplotypes from genotypes using DPRH1

Appendix B - Sample Runs of Implementation

10 individuals with 25 sites each

Disjoint regions of perfect phylogeny exist between sites:

```

  1    16
 17    24
 25    25

```

Haplotypes Inferred!

```
>> displayHaps(InfDH);           % displays haplotypes inferred by DPRH1
```

Haplotypes of 10 individuals with 25 sites each

```

0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 1 0 1 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1

0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1

0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1

0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

1 1 0 1 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1

1 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

1 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1

```

```
>> evalHInf(InfDH, RealH, true); % compares inferred and real haplotypes
```

Analysing inference for 10 individuals with 25 sites each

Ratio of Identical Sites between Real and Inferred Haplotypes:
95.2%

Heterozygous Sites Accurately Inferred throughout Sample:
83.78%

Average Switch Error/Individual:
0.8

Correctly Inferred Haplotypes:
50%

Heterozygous Sites Accurately Inferred/Individual:
57% 100% 89% 50% 100% 75% 92% 100% 100% 100%

Average: 86.3% Std Dev: 10.8%

>>

```

>>
>> % RUNNING DPRH2 WITH 50 INDIVIDUALS, 10 SITES AND 25 RECOMB. RATE
>> file = 'mssims/50-10-25#1.txt';           % selecting haplotype file to test
>> RealH = file2hap(file);                   % reads real haplotypes from file
>> G = hap2gen(RealH);                       % converts real haplotypes to genotypes
>> InfDH = runDPRH2(G);                      % infers haplotypes from genotypes via DPRH2

```

Inferring haplotypes from genotypes using DPRH2
50 individuals with 10 sites each

Appendix B - Sample Runs of Implementation

Disjoint regions of perfect phylogeny exist between sites:

```
1      3
4      6
7     10
```

Haplotypes Inferred via DPRH2

```
>> evalHInf(InfDH, RealH, true); % compares inferred and real haplotypes
```

Analysing inference for 50 individuals with 10 sites each

```
Ratio of Identical Sites between Real and Inferred Haplotypes:
98.8%
```

```
Heterozygous Sites Accurately Inferred throughout Sample:
96.2%
```

```
Average Switch Error/Individual:
0.16
```

```
Correctly Inferred Haplotypes:
88%
```

Heterozygous Sites Accurately Inferred/Individual:

```
100% 100% 100% 83% 100% 100% 100% 100% 100% 100%
100% 100% 100% 100% 83% 100% 83% 100% 75% 100%
100% 100% 100% 100% 83% 100% 100% 100% 100% 100%
75% 100% 100% 100% 100% 100% 100% 100% 100% 100%
100% 100% 100% 100% 100% 100% 100% 100% 100% 100%
```

```
Average: 97.64%      Std Dev: 4.38%
```

```
>>
```

```
>>
```

```
>> % RUNNING OPRH WITH 25 INDIVIDUALS, 25 SITES AND 25 RECOMB. RATE
```

```
>>
```

```
>> file = 'mssims/25-25-25#1.txt';      % selecting haplotype file to test
>> RealH = file2hap(file);              % reading real haplotypes from file
>> G = hap2gen(RealH);                  % converts real haplotypes to genotypes
>> InfDH = runOPRH(G);                  % infers haplotypes from genotypes via OPRH
```

Inferring haplotypes from genotypes using OPRH
25 individuals with 25 sites each

Maximal regions of perfect phylogeny exist between sites:

```
1      4
4      7
6      9
7     14
12     23
14     25
```

Haplotypes Inferred!

```
>> evalHInf(InfDH, RealH, true);      % compares inferred and real haplotypes
```

Analysing inference for 25 individuals with 25 sites each

```
Ratio of Identical Sites between Real and Inferred Haplotypes:
95.52%
```

```
Heterozygous Sites Accurately Inferred throughout Sample:
80.69%
```

```
Average Switch Error/Individual:
0.92
```

```
Correctly Inferred Haplotypes:
40%
```

Appendix B - Sample Runs of Implementation

```
Heterozygous Sites Accurately Inferred/Individual:
100% 100% 86% 67% 100% 50% 67% 71% 80% 50%
73% 67% 100% 100% 100% 88% 73% 100% 100% 80%
90% 89% 100% 100% 55%
```

```
Average: 83.44% Std Dev: 10.2%
```

```
>>
>>
```

```
>>
>> % RUNNING DPRH2 WITH 100 INDIVIDUALS, 200 SITES AND 25 RECMB. RATE
>>
>> file = 'mssims/100-200-25#1.txt'; % selecting haplotype file to test
>> RealH = file2hap(file); % reading real haplotypes from file
>> G = hap2gen(RealH); % converts real haplotypes to genotypes
>> InfDH = runDPRH2(G); % infers haplotypes from genotypes via DPRH2
```

```
Inferring haplotypes from genotypes using DPRH2
100 individuals with 200 sites each
```

```
Disjoint regions of perfect phylogeny exist between sites:
```

```
1 13
14 21
22 31
32 51
52 58
59 67
68 84
85 95
96 109
110 113
114 123
124 134
135 146
147 153
154 159
160 175
176 190
191 195
196 200
```

```
Haplotypes Inferred via DPRH2
```

```
>> evalHInf(InfDH, RealH, true); % compares inferred and real haplotypes
```

```
Analysing inference for 100 individuals with 200 sites each
```

```
Ratio of Identical Sites between Real and Inferred Haplotypes:
98.19%
```

```
Heterozygous Sites Accurately Inferred throughout Sample:
88.78%
```

```
Average Switch Error/Individual:
1.61
```

```
Correctly Inferred Haplotypes:
39%
```

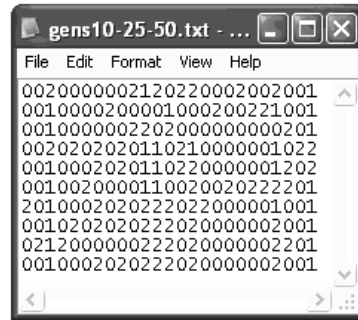
```
Heterozygous Sites Accurately Inferred/Individual:
100% 50% 100% 100% 59% 88% 100% 100% 97% 100%
88% 78% 100% 56% 66% 55% 67% 74% 83% 62%
77% 100% 82% 100% 97% 100% 100% 51% 100% 100%
83% 98% 100% 100% 70% 97% 72% 76% 100% 94%
67% 100% 100% 54% 98% 100% 100% 96% 100% 82%
96% 100% 100% 82% 93% 100% 88% 97% 87% 100%
97% 69% 96% 100% 81% 100% 100% 97% 100% 84%
85% 100% 98% 95% 100% 57% 97% 93% 100% 100%
79% 91% 96% 100% 100% 87% 100% 97% 98% 77%
100% 100% 79% 93% 58% 65% 76% 97% 100% 63%
```

```
Average: 88.65% Std Dev: 9.73%
```

```
>>
```

Appendix B - Sample Runs of Implementation

```
>> % RUNNING OPRH WITH 10 INDIVIDUALS,
>> 25 SITES AND 50 RECOMBINATION. RATE
>> % Loading from a file of genotypes
>>
>> file = 'gens10-25-50.txt';
>> % selecting genotypes file to infer from
>> G = file2gen(file);
>> % read genotypes from file
>> displayGens(G)
>> % displays genotypes read from file
```



Genotypes of 10 individuals with 25 sites each

```
0 0 2 0 0 0 0 0 0 2 1 2 0 2 2 0 0 0 2 0 0 2 0 0 1
0 0 1 0 0 0 0 2 0 0 0 0 1 0 0 0 2 0 0 2 2 1 0 0 1
0 0 1 0 0 0 0 0 0 2 2 0 2 0 0 0 0 0 0 0 0 0 2 0 1
0 0 2 0 2 0 2 0 2 0 1 1 0 2 1 0 0 0 0 0 0 1 0 2 2
0 0 1 0 0 0 2 0 2 0 1 1 0 2 2 0 0 0 0 0 0 1 2 0 2
0 0 1 0 0 2 0 0 0 0 1 1 0 0 2 0 0 2 0 2 2 2 2 0 1
2 0 1 0 0 0 2 0 2 0 2 2 2 2 0 2 2 0 0 0 0 1 0 0 1
0 0 1 0 2 0 2 0 2 0 2 2 2 0 2 0 0 0 0 0 2 0 0 1
0 2 1 2 0 0 0 0 0 2 2 2 2 0 2 0 0 0 0 0 2 2 0 1
0 0 1 0 0 0 2 0 2 0 2 2 2 0 2 0 0 0 0 0 2 0 0 1
```

```
>> H = runOPRH(G); % infers haplotypes from genotypes via OPRH
```

Inferring haplotypes from genotypes using OPRH
10 individuals with 25 sites each

Maximal regions of perfect phylogeny exist between sites:

```
1 10
4 14
10 19
14 21
16 22
23 25
```

Haplotypes Inferred!

```
>> displayHaps(H) % display haplotypes inferred
```

Haplotypes of 10 individuals with 25 sites each

```
0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 1 0 0 1
0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1

0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 1 1 0 0 1
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1

0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1

0 0 1 0 1 0 1 0 1 0 1 1 0 1 1 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 1 1

0 0 1 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 1

0 0 1 0 0 1 0 0 0 0 1 1 0 0 1 0 0 1 0 1 1 1 0 0 1
0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1

1 0 1 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 0 0 1 0 0 1
0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1

0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1

0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 0 1

0 0 1 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1
```

```
>>
```

Appendix C - Results from Evaluation

This section merely contains a large table with all the results obtained from the evaluation of the engineered as well as the established algorithms. Please refer to Chapter 5, for a description of the test plan and evaluation conducted as part of this project. The following is a key for column headers used in the table:

idnRatio Ratio of Identical Sites %
htrzAcc Heterozygous Sites Accurately Inferred %
swchErr Average Switch Error/Individual
crctHaps Correctly Inferred Haplotypes %
avgIndAcc Heterozygous Sites Accurately Inferred/Individual %
stdIndAcc - average and standard deviation

All data from file "resultsSum.xls"

Algorithm	n	m	r	t	idn	htrz	swch	crct	avgInd	stdInd	Algorithm	n	m	r	t	idn	htrz	swch	crct	avgInd	stdInd
	inds	sites	rate	secs	Ratio	Acc	Err	Haps	Acc	Acc		inds	sites	rate	secs	Ratio	Acc	Err	Haps	Acc	Acc
DPRH1	10	10	10	0.9	96.0	89.9	0.40	80.0	93.8	6.0	DPRH1	10	100	100	16.2	90.8	68.8	6.50	0.0	68.2	9.2
DPRH2	10	10	10	0.9	96.4	90.8	0.36	80.0	93.4	5.4	DPRH2	10	100	100	17.0	91.0	69.3	6.37	0.0	68.6	9.4
OPRH	10	10	10	1.2	94.4	84.1	0.38	62.0	90.5	10.0	OPRH	10	100	100	20.7	91.2	69.8	6.40	0.0	71.0	8.0
FPHASE	10	10	10	3.6	99.6	98.8	0.04	96.0	99.1	1.8	FPHASE	10	100	100	39.5	90.7	67.7	5.07	3.3	67.7	11.0
HTYPER	10	10	10	1.3	99.0	97.0	0.08	94.0	97.8	3.2	HTYPER	10	100	100	21.8	90.3	67.6	6.70	0.0	67.9	9.4
DPRH1	10	10	25	0.8	93.0	77.9	0.96	52.0	84.0	10.7	DPRH1	10	125	10	17.0	92.8	70.9	4.97	20.0	75.9	10.8
DPRH2	10	10	25	0.9	93.0	77.9	0.96	52.0	84.0	10.7	DPRH2	10	125	10	19.6	93.0	72.0	4.83	26.7	79.1	10.5
OPRH	10	10	25	1.1	94.0	80.9	0.68	52.0	84.1	10.8	OPRH	10	125	10	26.6	93.0	71.2	5.37	16.7	75.9	9.5
FPHASE	10	10	25	3.8	97.0	91.1	0.36	72.0	92.4	7.2	FPHASE	10	125	10	54.4	95.2	80.2	3.07	23.3	84.0	9.6
HTYPER	10	10	25	1.0	95.2	85.2	0.48	68.0	88.7	8.6	HTYPER	10	125	10	19.5	96.3	84.1	4.62	17.6	84.6	8.5
DPRH1	10	10	50	1.0	92.4	73.7	0.78	42.0	79.5	10.9	DPRH1	10	125	25	20.1	93.0	74.3	5.30	3.3	75.5	10.6
DPRH2	10	10	50	1.0	93.6	77.1	0.66	50.0	82.7	11.4	DPRH2	10	125	25	19.2	93.7	76.2	5.07	10.0	77.3	8.9
OPRH	10	10	50	1.3	93.2	76.6	0.68	48.0	81.0	9.9	OPRH	10	125	25	22.3	90.2	64.6	6.53	0.0	65.0	8.2
FPHASE	10	10	50	5.1	93.2	76.2	0.72	52.0	81.3	11.7	FPHASE	10	125	25	81.0	93.0	76.0	4.53	10.0	76.5	9.9
HTYPER	10	10	50	0.9	91.6	70.9	1.02	38.0	77.0	11.0	HTYPER	10	125	25	20.1	96.0	84.3	4.90	22.4	82.5	9.7
DPRH1	10	10	100	0.9	93.0	75.5	0.84	42.0	80.1	10.8	DPRH1	10	125	50	15.9	91.5	70.4	4.43	3.3	70.2	8.8
DPRH2	10	10	100	1.0	93.0	75.4	0.86	44.0	81.0	11.0	DPRH2	10	125	50	21.2	91.9	71.8	4.67	0.0	71.7	8.9
OPRH	10	10	100	1.3	92.4	74.1	0.94	38.0	78.6	10.5	OPRH	10	125	50	22.9	88.6	60.1	7.00	0.0	60.7	7.5
FPHASE	10	10	100	3.5	92.6	74.5	0.88	44.0	80.4	11.0	FPHASE	10	125	50	56.6	93.8	78.5	3.63	13.3	76.9	9.9
HTYPER	10	10	100	0.9	92.6	74.2	0.96	46.0	80.5	11.2	HTYPER	10	125	50	22.3	95.1	82.3	5.71	34.4	82.9	10.1
DPRH1	10	25	10	2.4	94.9	78.8	1.06	48.0	89.1	11.3	DPRH1	10	125	100	13.6	88.5	63.4	9.40	0.0	62.5	8.3
DPRH2	10	25	10	2.1	95.4	81.6	0.98	54.0	87.6	11.1	DPRH2	10	125	100	16.4	88.5	63.6	9.53	0.0	62.8	8.1
OPRH	10	25	10	3.3	93.6	74.3	1.44	42.0	81.5	10.5	OPRH	10	125	100	28.4	86.8	58.4	10.10	0.0	58.0	6.2
FPHASE	10	25	10	12.1	96.2	84.8	0.80	64.0	89.7	8.3	FPHASE	10	125	100	43.9	90.1	68.9	7.17	0.0	69.1	8.6
HTYPER	10	25	10	3.1	97.4	90.0	0.82	64.0	92.7	8.2	HTYPER	10	125	100	24.7	92.6	72.8	6.99	1.6	72.7	10.1
DPRH1	10	25	25	3.3	93.4	73.3	1.72	30.0	78.3	10.4	DPRH1	10	150	10	18.5	91.9	71.3	7.47	6.7	72.4	9.8
DPRH2	10	25	25	3.1	93.2	72.7	1.70	32.0	78.1	10.6	DPRH2	10	150	10	28.9	92.0	71.8	7.47	10.0	72.8	9.8
OPRH	10	25	25	4.9	93.8	74.3	1.72	32.0	78.0	10.1	OPRH	10	150	10	34.4	91.4	69.2	8.27	0.0	67.8	9.3
FPHASE	10	25	25	10.7	95.4	80.8	1.36	38.0	83.3	9.1	FPHASE	10	150	10	59.3	92.2	72.5	6.60	6.7	72.9	9.6
HTYPER	10	25	25	3.3	95.1	78.8	1.72	32.0	81.4	9.0	HTYPER	10	150	10	22.7	94.8	80.8	7.03	3.3	80.3	8.0
DPRH1	10	25	50	3.8	92.4	73.8	1.92	16.0	74.8	9.2	DPRH1	10	150	25	20.8	90.6	68.3	6.57	10.0	71.8	9.3
DPRH2	10	25	50	3.1	93.0	76.2	1.88	22.0	77.3	9.2	DPRH2	10	150	25	23.5	91.1	69.6	6.63	10.0	74.4	10.3
OPRH	10	25	50	6.3	92.1	72.2	2.16	16.0	74.2	8.3	OPRH	10	150	25	28.7	89.3	63.3	8.63	3.3	65.9	8.0
FPHASE	10	25	50	15.5	92.2	72.9	1.52	22.0	74.9	10.1	FPHASE	10	150	25	56.2	93.9	73.7	4.10	16.7	79.9	10.3
HTYPER	10	25	50	3.8	91.6	71.1	1.88	14.0	69.3	9.0	HTYPER	10	150	25	21.1	94.4	82.0	5.80	20.0	80.3	9.8
DPRH1	10	25	100	3.3	91.8	71.4	1.92	16.0	72.4	9.4	DPRH1	10	150	50	20.3	92.0	73.7	6.23	0.0	73.6	8.3
DPRH2	10	25	100	3.2	92.9	75.2	1.74	20.0	76.7	9.7	DPRH2	10	150	50	27.1	92.1	74.2	5.93	3.3	74.9	9.3
OPRH	10	25	100	5.2	92.0	71.9	1.84	12.0	73.1	9.6	OPRH	10	150	50	29.9	89.9	66.9	8.33	0.0	65.8	8.5
FPHASE	10	25	100	14.6	92.3	73.1	1.84	22.0	75.9	9.5	FPHASE	10	150	50	53.9	93.0	77.1	5.10	3.3	75.6	9.4
HTYPER	10	25	100	4.1	91.6	70.7	2.14	20.0	72.8	9.7	HTYPER	10	150	50	23.9	95.4	84.8	4.83	40.0	84.1	9.5
DPRH1	10	50	10	6.4	95.0	82.8	1.96	40.0	84.6	10.5	DPRH1	10	150	100	20.1	88.3	60.2	8.70	0.0	60.4	7.3
DPRH2	10	50	10	6.4	95.5	84.7	1.84	48.0	86.2	10.6	DPRH2	10	150	100	21.6	88.7	61.6	8.67	0.0	61.7	6.5
OPRH	10	50	10	9.0	92.9	76.0	2.20	16.0	76.5	9.5	OPRH	10	150	100	31.8	88.5	61.0	11.63	0.0	61.1	5.8
FPHASE	10	50	10	21.6	97.3	90.7	1.08	48.0	90.0	8.1	FPHASE	10	150	100	55.9	92.2	73.7	6.40	3.3	74.4	8.5
HTYPER	10	50	10	8.0	97.9	92.6	1.30	48.0	91.7	6.9	HTYPER	10	150	100	27.5	93.0	76.2	6.90	3.3	75.5	10.0
DPRH1	10	50	25	8.1	93.4	76.1	2.82	20.0	76.8	10.1	DPRH1	10	175	10	26.8	95.7	80.6	6.40	13.3	77.7	9.9
DPRH2	10	50	25	8.2	93.8	77.5	2.76	30.0	80.0	10.2	DPRH2	10	175	10	36.4	95.7	80.9	6.50	13.3	78.0	9.7
OPRH	10	50	25	10.5	91.6	70.4	3.86	18.0	74.3	9.4	OPRH	10	175	10	41.9	93.5	71.7	6.10	6.7	73.2	8.0
FPHASE	10	50	25	19.5	94.5	79.9	2.10	30.0	81.9	9.9	FPHASE	10	175	10	59.8	95.5	80.3	5.77	6.7	79.8	9.6
HTYPER	10	50	25	7.9	94.4	80.5	2.06	44.0	83.5	9.6	HTYPER	10	175	10	25.5	95.8	81.7	6.63	6.7	78.6	10.3
DPRH1	10	50	50	8.0	93.7	76.9	2.66	26.0	80.2	9.9	DPRH1	10	175	25	23.0	93.0	73.0	6.03	13.3	77.4	9.2
DPRH2	10	50	50	8.9	94.3	78.9	2.60	30.0	81.6	9.6	DPRH2	10	175	25	30.6	93.0	73.0	6.13	13.3	77.3	9.2
OPRH	10	50	50	10.9	92.0	71.2	2.90	18.0	75.1	9.3	OPRH	10	175	25	40.1	92.1	70.4	8.03	3.3	70.5	8.2
FPHASE	10	50	50	21.6	94.9	81.8	1.96	34.0	84.9	9.9	FPHASE	10	175	25	62.5	96.0	83.9	4.67	3.3	84.1	9.2
HTYPER	10	50	50	8.0	94.6	80.1	2.50	38.0	82.9	10.6	HTYPER	10	175	25	26.3	96.4	85.3	4.43	10.0	87.1	8.9
DPRH1	10	50	100	7.9	90.7	64.6	4.50	8.0	66.3	8.9	DPRH1	10	175	50	21.8	91.0	68.2	8.57	3.3	67.2	9.9
DPRH2	10	50	100	9.1	90.7	66.7	4.52	10.0	68.6	9.5	DPRH2	10	175	50	54.9	91.1	68.4	8.63	3.3	67.4	9.7
OPRH	10	50	100	11.1	89.9	63.6	5.10	2.0	64.4	8.7	OPRH	10	175	50	38.5	90.4	64.7	10.47	3.3	65.3	7.9
FPHASE	10	50	100	20.9	92.0	71.2	3.44	10.0	73.2	10.3	FPHASE	10	175	50	60.7	93.2	74.0	6.80	10.0	75.2	10.0
HTYPER	10	50	100	9.2	93.0	74.3	3.70	14.0	76.1	10.4	HTYPER	10	175	50	31.0	93.7	76.3	7.80	3.3	76.6	9.9
DPRH1	10	75	10	9.7	94.1	79.0	2.90	32.0	82.9	10.5	DPRH1	10	175	100	24.0	90.0	61.9	11.70	0.0	62.1	8.5
DPRH2	10	75	10	13.8	94.4	80.0	2.94	34.0	81.1	10.0	DPRH2	10	175	100	49.9	89.9	61.6	11.70	0.0	61.6	8.0
OPRH	10	75	10	13.1	91.7	71.0	3.50	8.0													