REYKJAVÍK UNIVERSITY
HÁSKÓLINN Í REYKJAVÍK

Models for solving Minimum Parsimony Haplotyping

Steinunn María Stefánsdóttir
Master of Science
June 2007

Supervisor:
Bjarni Vilhjálmur Halldórsson, Ph.D
Associate Professor

Reykjavík University - Department of Computer Science

M.Sc. Project Report

MODELS FOR SOLVING MINIMUM PARSIMONY HAPLOTYPING

by

Steinunn María Stefánsdóttir

Project report submitted to the Department of Computer Science at Reykjavík University in partial fulfillment of the requirements for the degree of **Master of Science**

June 2007

Project Committee:

Bjarni Vilhjálmur Halldórsson, Ph.D, supervisor
Associate Professor, Reykjavik University

Anna Ingólfsdóttir, Ph.D
Professor, Reykjavik University

Guðbjörn Freyr Jónsson, Ph.D
Íslensk Erfðagreining

The undersigned hereby certify that they recommend to the Department of Computer Science at Reykjavík University for acceptance this project report entitled **Models for solving Minimum Parsimony haplotyping** submitted by Steinunn María Stefánsdóttir in partial fulfillment of the requirements for the degree of **Master of Science**.

_____

Date

_____

Bjarni Vilhjálmur Halldórsson, Ph.D, supervisor
Associate Professor, Reykjavik University

_____

Anna Ingólfsdóttir, Ph.D
Professor, Reykjavik University

_____

Guðbjörn Freyr Jónsson, Ph.D
Íslensk Erfðagreining

Abstract

Single Nucleotide Polymorphisms (SNPs) are the most common form of variations in the human genome. Collection of SNP variants on a single chromosome copy are called *haplotypes*. Humans are diploid organisms, implying that they possess two nearly identical copies of each chromosome, and therefore the haplotypes come in pairs. Conflated (mixed) data from the two haplotypes is called a *genotype* of an individual. *Minimum parsimony haplotyping* (MPH) is an abstraction of haplotype finding problem arising in genetics, which tries to find the minimum set of haplotypes needed to explain a given set of genotypes. The *MPH* problem is known to be NP-Hard, meaning that finding computationally efficient solutions is unlikely in the general case. Here, we give novel efficient algorithms for sub-instances of the problem. In addition, a practical heuristic for *MPH* are implemented, solving problem instances for the general case.
Experiments are done on real genotype data from the HapMap project [10] and heuristics developed from these experiments are used to speed up the implementation. These improvements result in a algorithm that solves *MPH* several times faster than previously described methods

Acknowledgements

I would like to start of by thanking my supervisor Bjarni for his endless help and guidance, also for giving me the chance to work on this project. I also want to thank Anna Ingólfdóttir for aiding in my return to Reykjavík University and for her interest in this project. Last but not least I want to thank my boyfriend Jónas for being so patient with me during this time, and for his useful comments on regarding this project report.

Contents

List of Algorithms

Chapter 1

Introduction

More than 99% of the human DNA sequence is the same between individuals. Genetic studies focus on the variation, Single Nucleotide Polymorphisms (SNPs) are the most common form of variation in the human genome and thus are notably studied. Humans are diploid organisms, implying that they possess two nearly identical copies of each chromosome, and therefore the haplotypes come in pairs. Conflated (mixed) data from the two haplotypes is called a *genotype* of an individual. Haplotypes are valuable data since it is assumed that genetic basis of important traits can be best understood by assessing the association between the occurrence of particular haplotypes and particular traits [9]. Projects such as HapMap [10] are making a haplotype map for the human genome which could be great help to locate haplotypes that are related to particular traits. Therefore there is a great need to develop methods that can detect haplotypes in the human genome which are computationally solvable.

The problem considered here is called *Haplotype Inference* (*HI*) problem, is to infer haplotypes from genomic data, this problem is computationally expensive due to large amount of data. One approach to the *HI* problem is to find the smallest set of haplotypes to explain a set of genotypes, called the *minimum parsimony haplotyping (MPH)* first introduced by Gusfield [5]. The motivation for this approach is that there are fewer distinct haplotypes in populations than possible haplotypes because of the bottleneck effect in evolution.

There are two types of models used to solve the *HI* problem, statistical and combinatorial models. Statistical methods are mainly based on statistics assessments and analysis of genetic variation, where as the combinatorial methods focus on methods commonly used by computer scientists. *MPH* is a combinatorial and is quick and effective to solve the *HI* problem. Also solving the *HI* problem with statistical methods is not feasible because of the nature of our data, as we have a larger number of SNPs compared to individuals, therefore *MPH* was chosen for this project.

As *MPH* is a NP-Hard problem the goal of this project is to solve *MPH* for restricted cases of genotype data in polynomial time, and suggest ways to improve existing methods that solve *MPH* for general cases. The first model that we focused on are tree search models. Naive implementation of tree search algorithms can only solve the *MPH* for

small sets of genotypes, as the trees grow exponentially and tree search algorithms have exponentially worst case guarantee. For those reasons we present heuristic to improve the efficiency of the tree search. Several other methods exist that solve *MPH*, that exploit restrictions on problem instances for efficiency. Graph based models have been applied to restricted sets of the *MPH* problem, and here we review previous work on restricted problem instances and present a novel efficient graph algorithm for restrictions of problem instances. The last model that we review is a boolean satisfiability formulation [15] of the *MPH* problem that solves larger problem instances than the other two models. The satisfiability algorithm is implemented and experiments done on genotype data from the HapMap project [10]. The heuristics learnt from those experiments were used to speed up the implementation resulting in an implementation that runs several times faster than existing methods. In the end computational results for the improvements are presented, followed by discussions and conclusions.

Chapter 2

Background

2.1 SNPs

The human genome consist of 46 chromosomes that are in 23 almost identical pairs. Chromosomes are made of Deoxyribonucleic acid (DNA). DNA can be considered to be a collection of long strings, or sequences, taken from the alphabet {A,C,G,T}, where each element of the alphabet encodes one of four possible *nucleotides*. Approximately 3 billion base pair are arranged into these 23 pairs. More than 99% of the human genome is the same between populations, hence genetic studies focus on the variation. *Single Nucleotide Polymorphisms* or SNPs constitute a large class of these variations. A SNP is a single base pair position in genomic DNA at which different nucleotide variants exist in some populations, each variant is called an *allele*. For example, two sequenced DNA fragments from different individuals, AAGCCTA to AGGCCTA, contain a difference in a single nucleotide in the second position in the sequence. In human, SNPs are almost always biallelic, that is, there are two variants at the SNP site, with the most common variant referred to as the *major allele*, and the less common variant as the *minor allele*. Each variant must be represented in a significant portion of the population to be useful.

2.2 Haplotypes and Genotypes

Humans are diploid organisms, that means humans possess two nearly identical copies of each chromosome, one inherited from the father and the other inherited from the mother. Collection of SNP variants on a single chromosome copy are called *haplotype*. Thus, for a given set of SNPs, an individual possesses two haplotypes, one from each chromosome copy. A SNP site where both haplotypes have the same *allele* is called a *homozygous* site, a SNP site where the haplotypes have different *alleles* is called a *heterozygous* site. The conflated (mixed) data from the two haplotypes is called a *genotype*. See Figure 2.1 for further explanation.

Haplotypes are valuable data since it is assumed that genetic basis of important traits can be best understood by assessing the association between the occurrence of particular hap-

lotypes and particular traits (e.g disease) [9]. Projects such as HapMap [10] are making a haplotype map for the human genome which could be great help to locate haplotypes that affect health, disease, and individual responses to medications and environmental factors. Therefore it is important to develop efficient algorithms that can detect haplotypes in the human genome.

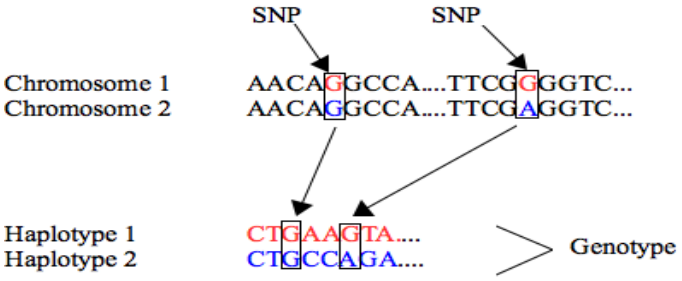

Figure 2.1: In this example two SNPs are highlighted. The first SNP site is homozygous, and the the second SNP site is heterozygous. The individual has the two haplotypes that are shown and the genotype is $CTG\{A,C\}\{A,C\}\{G,A\}\{T,G\}A$.

2.2.1 Haplotype Inference

We use the notation of Gusfield [5] were SNP (in a haplotype) are labeled 0 or 1 meaning haplotypes are represents as set of values from {0,1}. Two haplotypes h and h' represent a genotype g. For each site j in h and h' the following applies for site j in g, if h and h' both have value 0, g has value 0, if h and h' both have value 1, g has value 1 and if h and h' have opposite values, g has value 2. Thus g has value 0 and 1 in homozygous sites and value 2 in heterozygous sites. Haplotypes h and h' said to be a haplotype pair belonging to g.

For a given genotype g, resolving haplotypes h and h' by using the information above, is called *haplotype inferring*. As mentioned before, regarding the information on a heterozygous sites in a genotype g, it is not known which of the haplotype h and h' has value 1 or which one has value 0. Both possibilities are correct and that is why the number of possible haplotype pairs for each genotype depend on the number of 2's. For genotype with p number of 2's there are 2^{p-1} haplotype pair that can be inferred. See example in *Figure* 2.2 for further explanation.

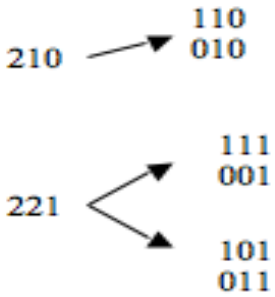

Figure 2.2: In this example two different genotypes are inferred

2.3 Minimum Parsimony Haplotyping

Minimum Parsimony Haplotyping, *MPH*, is one approach to solve the *HI* problem. *MPH* finds the the smallest set of haplotypes to explain a set of genotypes. This method was first required by Gusfield [5]. By finding the smallest set, H, of haplotypes to explain a given genotype set, G, is the same as assuming a low diversity of haplotypes for a set G. The motivation behind this method is that there are fewer distinct haplotypes in populations than possible haplotypes because of the bottleneck effect in evolution [12]. For example Patil et al. report that within short genomic regions, typically some 70-90% of the haplotypes belong to very few (2-5) common haplotypes [12].

In addition *MPH* can be used to locate *recombination hotspots* by finding parts of chromosomes that have high recombination rate. Recombination takes place during meiosis, a process involved in the formation of reproductive cells (or *gametes*) in the parents. During recombination, portions of the paternal and maternal chromosomes are exchanged. Recombination can result in haplotypes in offsprings that are different from those in the parents. The site on the chromosome where a recombination occurs is called a *recombination site*. Regions of high recombination site frequency are called *recombination hotspots*. *Recombination hotspots* can by located by finding parts in the same chromosome for a set of individuals that have high frequency of haplotypes. These *recombination hotspots* can be useful for other studies such as learning more about the recombination process and also possibly map the position of alleles that cause diseases.

2.3.1 Different methods to solve MPH

There are two main approaches for solving the *HI* problem, statistical methods and combinatorial methods. This project is focused on solving the *HI* with *Minimum Parsimony Haplotyping* that is a combinatorial method. Many methods focus on parsimonious solutions [6] and number of combinatorial methods are used to solve them.

Still there are though other methods like statistical methods that are focused on the data in biological view. Statistical methods are mainly based on researches in population genetics and statistics assessments and analysis of genetic variation. One of the statistical methods that is commonly used is Maximum Likelihood [4]. Maximum Likelihood, estimates haplotype frequencies from a underlying genotype frequencies. The method assumes HWE [1] and a uniform prior on the haplotype probabilities. It then uses the haplotype frequencies to find most probable haplotype pairs that can explain any given genotype, by finding frequencies of the haplotypes that maximize the value that certain haplotype pairs can explain a given genotype set. There are several issues that arise with this method concerning the reliability of the results. There is another method that uses more informative probabilities which is the widely used PHASE method [14]. The PHASE method is an statistical method that includes population priors. It works on genotype data at linked loci from a population sample. It has several advantages over other haplotype reconstruction methods and many use PHASE in their studies since it has very accurate results.

These different approaches have different application, the combinatorial methods are very efficient and appropriate for large datasets when the number of SNPs is large compared to the number of genotypes, as is the case for genotype set used in this project. Every method for solving the *HI* problem has their advantages. It is important to continue to use different approaches to solve *HI* problem since new things can be learnt from having a variety of approaches.

[1] Hardy-Weinberg equilibrium (HWE) is the condition that the probability of observing a genotype is equal to the product of the probabilities of observing its constituent haplotypes (see [7]).

Chapter 3

Algorithms

In this project three different models that can solve *MPH* are described, with the aim of finding the most efficient model. Reviewing different methods can help with finding what complications of the problem needs to be solved.

This chapter describes the algorithms that were constructed and improved in this project. For each method it is stated how they are used for solving *MPH* and what can be done to improve their efficiency.

3.1 Tree search model

One way to solve *Minimum Parsimony Haplotyping* is to use tree search model. Trees are frequently used in computer science to solve different kind of optimization problems and therefore there exist several methods that search for a optimal solutions. Some of these methods can be used to solve the *minimum parsimony haplotyping (MPH)*.

First we have to represent our optimization problem as a tree model. For a given genotype set, G with n number of genotypes, we construct tree T with an empty root node on depth 0. For each genotype $g_i \in G$, where $1 < i < n$, nodes are created on depth i in T that represent distinct haplotype pairs that can explain genotype g_i. *Successors* of nodes on level i are distinct haplotype pairs belonging to genotype g_{i+1}. Nodes belonging to genotype g_n in G are called *terminal nodes*.

Given the tree model for genotype set G, the next step is to find a method that finds the minimum number of haplotypes to explain G. We can use *Depth First Search (DFS)* [17] to search the tree for optimal solution. In addition we can use *Branch and Bound* [17] to limit the search space. We then suggest heuristic to improve the search even further.

Algorithms 1 and 2 are top-level description for solving *MPH* for genotype set G with *DFS* and *Branch and bound* [17]. The generation of successors will be described later.

Algorithm 1 Initialize DFS search

Input: Genotype set G
Output: Set H with of minimum number of haplotypes

1: Remove duplicate genotypes in G
2: upperBound $\leftarrow$ 2*size(G)
3: Create tree T, with empty node as root node g_0
4: $H \leftarrow$ DFS(g_0, T, upperBound)
5: **return** H

Algorithm 2 DFS search with branch and bound

Input: Haplotype pair belonging to g_i, Path in T, upperBound
Output: Set H with of minimum number of haplotypes

1: Successors $\leftarrow$ Generate all haplotype pairs belonging to g_{i+1}
2: **if** Successors $\neq \emptyset$ **then**
3: **for** each $succ \in Successors$ **do**
4: Add $succ$ as a successor to the path in T.
5: Count $\leftarrow$ number of different haplotypes in the path in T.
6: **if** count < upperBound **then**
7: Do DFS(succ, path in T, upperbound)
8: **else**
9: Go to next succ
10: **end if**
11: **end for**
12: **else**
13: **if** number of different haplotypes in path < upperBound **then**
14: upperBound $\leftarrow$ number of different haplotypes in path in T
15: Empty H
16: H $\leftarrow$ Nodes in the path in T
17: **end if**
18: **end if**
19: **return** H

All unique genotypes in G have to be considered when finding the minimum number of haplotypes needed to explain G. Therefore to find the minimum solution, we have to search the tree T until we reach a terminal node. *DFS* [17] is a recursive greedy algorithm that expands the first successor of each node until it reaches a terminal node. At the terminal node it starts backtracking up the tree towards the root node. The first time *DFS* reaches a terminal node, the number of different haplotypes in the path becomes the upper bound.

Branch and Bound is a tree pruning technique that uses a given upper bound to limit the number of nodes that needs to be constructed. Using *Branch and Bound* with *DFS* prevents it from expanding unnecessary nodes, that will always result in a less optimal solution than the one that is already found. This is achieved by bounding (See Line 6 in Algorithm 2) and by that reducing the nodes that DFS has to consider. If a more optimal solution is found, the upper bound is updated (See Line 13 in Algorithm 2). By this *DFS* is finding different combinations of haplotype sets H that can explain G, and *Branch and bound* can prevent it from considering all possible combination of H.

The complexity of the algorithm depends on the number of genotypes and the branching factor (number of successor) for each node. For a tree with branching factor b and n number of genotypes, the time complexity for *DFS* becomes O(b^n) and the space complexity is O(bn). As the number of genotypes is fixed after duplicates have been removed, the only way to reduce the complexity is to reduce the branching factor or prune the search tree. Therefore we suggest heuristic that can improve the search with pruning. The complexity of counting different genotypes (see Line 5 in Algorithm 2) is $O(mn)$, if m is the number of sites of the genotypes and n is the number of genotypes, is not considered to affect the complexity of the algorithm. Heuristic to improve the time and complexity of the algorithm will be given.

3.1.1 Branch on all possible haplotype pairs

One way to branch in the tree is to branch on all possible haplotype pairs for a given a genotype. The number of possible haplotypes pairs depends on number of heterozygous sites (2's), as mentioned before number of possible haplotype pairs for, genotype g, is 2^{p-1} where p is the number of 2's in the genotype g. *Algorithm* 3 generates successors by creating all possible haplotype pairs for each genotype g. The complexity of generating all possible haplotype pairs for a genotype g with p number of 2's is O(m2^{p-1}) where m is the number of SNP's in g. See *Algorithm* 3.

Algorithm 3 Generate all possible Haplotype pairs

Input: Genotype g
Output: Set HP of haplotype pairs.

```
Create a haplotype set H with one empty haplotype
for each site j ∈ g do
    for each haplotype k ∈ H do
        if g_j = 0 then
            Set h_kj = 0
        else if g_j = 1 then
            Set h_kj = 1
        else if g_j = 2 and number of 2's found < 1 then
            Set h_kj = 0
        else if g_j = 2 and number of 2's found ≥ 1 then
            Create two copies of h_kj, h_kj and h'_kj.
            Set h_kj = 0 and h'_kj = 1
        end if
    end for
end for
Create empty set HP
for each haplotype h ∈ H do
    Make another haplotype h' so h, h' explain g
    Create haplotype pair for h and h'
    Add haplotype pair to HP
end for
return HP
```

When constructing tree T with *DFS* and branching on all possible haplotype pairs for a given genotype set g, the space complexity is O($\sum_{i=1}^{n} 2^{(p_i-1)}$) [1] and time complexity is O($2^{\sum_{i=1}^{n}(p_i-1)}$) [2] , if n is the number of genotypes and p is the number of $2's$ for each genotype.

[1] When nodes in tree T have a different branching factor the average branching is found, in this case $\sum_{i=1}^{n} 2^{(p_i-1)}$/n

[2] Time complexity is equal to multiplication of number of nodes on each depth in T, in this case $\prod_{i=1}^{n} 2^{(p_i-1)}$ which is equal to $2^{\sum_{i=1}^{n}(p_i-1)}$

3.1.2 Branch on all possible haplotype pairs with ordered haplotype pairs

The previously described branching algorithm has high time and space complexity, this can be improved by ordering the nodes in a way where it is more likely that sub optimal upper bound is found sooner in the *DFS* and hence bigger parts of the tree can be pruned away with bounding. This can be achieved by ordering the haplotypes pairs for each successors, and for a given node hp, if possible we branch first on successors that have haplotypes in the same group as haplotypes in hp. This way we begin by branching on haplotype pairs that have common haplotypes an are more likely to be closer to the optimal solution. When constructing the tree with *DFS* and branching on common haplotypes, the space complexity is and time complexity is the same as branching on all possible haplotypes pairs.

3.1.3 Summary of Tree models

The are many ways to try to improve the time and space complexity for the *DFS* but in this project, it will not be investigated further. The tree model can also be used to branch large genotype sets into smaller ones that is more easily solved. That is why we want to investigate models that solve *MPH* in polynomial time for restricted genotype set.

3.2 Graph based model

By having pre-defined restrictions on the input genotype set, algorithms can be constructed that solve *minimum parsimony haplotyping (MPH)* in polynomial time. These restriction have to do with the number of heterozygous sites that genotypes can have, or that subset of genotypes have to share a haplotype. We first review previously described methods as a introduction to these data models. We then give novel algorithms that solve, in polynomial time, sub-instances that have not been solved before.

3.2.1 PH(k,l) bound data model

We begin by describing what pre-defined restriction are set on the input genotype set. PH(k, l) bound data models addresses the minimum parsimony haplotyping by looking at instances where the genotypes have special structure and focus on the haplotypes that they share. The input for these methods are a genotype matrix G, where rows in G represents genotypes and the columns represent a site in the genotypes. PH(k, l) sets boundaries on how many number of heterozygous sites (2's) can be at most in each row and each column in the genotype matrix G where k is the maximum number of heterozygous sites in a row and l is the maximum number of heterozygous sites in a column.

With these restriction we can construct algorithms that solve the minimum parsimony haplotyping in polynomial time. The PH(k, l) data models start by constructing Clark consistency graphs for the input matrix G, where the nodes represent genotypes and edges represents haplotypes that two genotypes share. The structure of the graph depends on the type of (k, l)-bounds; the algorithms exploit the structure of the graphs and are able to solve *MPH* polynomial time. The output is haplotype matrix H with minimum number of rows resolving G. The rows in H represent haplotypes and each column represents a site in the haplotypes.

Before describing different types of (k, l) bound data models, we introduce definitions for a the main concept in these model and the algorithm for constructing a Clark consistency graph for a genotype matrix G.

Definition 1 Two genotypes g_1 and g_2 are said to be *compatible* if there is no site j where one genotype is 0 and the other is 1. Else g_1 and g_2 are said to be *incompatible*. For example $g_1 = 210$ and $g_2 = 110$ are compatible whereas $g_1 = 210$ and $g_2 = 111$ are incompatible. If two genotypes are compatible they share a haplotype.

Compatibility can be detected by comparing SNPs sites in two genotypes, and if there is know site where one genotype is 0 and the other is 1 these genotypes are compatible.

Definition 2 Clique in a Clark consistency graph is a set of genotypes that all share a haplotype, thus are all connected.

Algorithm 4 is a top-level description on how the Clark consistency graph is created. The complexity of the algorithm for constructing a Clark consistency graph is O(g^2m) where g is the number of genotypes in the set and m is the number of sites in the genotype.

Algorithm 4 Constructing a Clark consistency graph

Input: Genotype set G
Output: Clark consistency graph, C

```
1: Create empty graph C
2: for all g ∈ G do
3:    Add node g to graph C
4:    for each node g' ∈ C do
5:       if g compatible with g' && g' ≠ g then
6:          Add an edge between g and g'
7:       end if
8:    end for
9: end for
10: Return C
```

3.2.2 PH(*,1) data model

We describe this simple data model as a motivation for further expansion to these data models. This structure is restricted to genotype sets where there can only be single heterozygous site in each column. The $*$ means that the number of heterozygous sites in each row/genotype is not restricted. The $(*, 1)$ structure means that the Clark consistency graph for this set of genotypes is a 1-sum of cliques meaning that in each clique there is one vertex that connects to another clique [13]. From this it can also be stated that the compatibility graph is *chordal* meaning that all cycles contain triangles. In every *chordal* graph there is a *simplicial* vertex v, a vertex where the neighborhood of v is a closed clique. Removing a *simplicial* vertex from a graph gives again a chordal graph. Hence if all vertex are removed from a chordal graph, one by one, all non *simplicial* vertex in a chordal graph must at some point become *simplicial* vertex since the graph is always chordal. *Simplicial* vertex is used as a starting vertex in the algorithm that finds the minimum set of haplotypes in polynomial time [13]. The algorithm always selects a *simplicial* vertex since those vertex can only be consistent with one clique and can therefore only share haplotypes with one clique in the graph. The correctness of the algorithm has been proven by Iersel *et al* [13] and the complexity of the algorithm is O(n^3m) where n is the number of genotypes and m number of sites in the genotypes. See Algorithm 5. Let h_c be the unique haplotype consistent with with any genotype in closed neighborhood clique of genotype g

Algorithm 5 PH(*,1)

Input: PH$(*, 1)$ Clark consistency graph G
Output: Haplotype set H with minimum number of haplotypes for G

while $G \neq \emptyset$ **do**
  Find g that is simplicial
  if g has no 2's **then**
    Add $h = g$ to H and remove g from G
  else if there exist haplotypes h_1, $h_2 \in H$ such that h_1, h_2 explain g **then**
    Remove g from G
  else if there exist haplotype $h_1 \in H$ such h_1, h_c explain g **then**
    Add h_c to H and remove g from G
  else if there exist haplotypes $h_1 \in H$ and $h_2 \notin H$ such that h_1, h_2 explain g **then**
    Add h_2 to H and remove g from G
  else if g is not a isolated node in G find h_1 so that h_1, h_c explain g **then**
    Add h_1 and h_c to H and remove g from G
  else
    Find h_1, h_2 that explain g, add h_1 and h_2 to H and remove g from G
  end if
end while

3.2.3 PH(*,2) data model

PH($*, 1$) data model is polynomial time solvable, but works only on very simple genotype sets. If we want to handle more complicated genotype sets we have to investigate other restriction. It has been proven that PH($4, 3$) problem is NP-Hard [16], therefore we work on a problem instances between these two models. PH($*, 2$) is a borderline open complexity problem and the border have been pushed slightly further by adding another restriction, that states how many cliques there should be in the Clark consistency graph. That are the PH($*, 2$)Cq problems were q is the number of cliques in the graph. PH($*, 2$)Cq states that genotype set G can be divided into q subsets where all genotypes in each subset share haplotype that is not the same as the haplotypes for the other subsets. Definition for a clique the PH($*, 2$)Cq instances is the following.

Definition 3 In the PH($*, 2$) graph a clique is a group of genotypes greater than two in a Clark consistency graph, that all share a haplotype.

Sharan *et al.* gave polynomial time solvable algorithm for PH($*, 2$)C1 and in this work it was expand [16]. First we give polynomial time solvable algorithm for PH($*, 2$)C2 problems instances, and used similar idea to construct a polynomial time solvable algorithm that solves PH($*, 2$)Cq problems. Both novel algorithms will be given later, but first we describe simplest PH($*, 2$)C1 problem.

3.2.3.1 PH($*, 2$)C1 data model

In the PH($*, 2$)C1 problem the Clark consistency graph must be a clique (q=1), meaning for all genotype sets G there is one haplotype that is shared by all genotypes in G and the Clark consistency graph for the genotypes is a one clique graph. One way to fulfill restriction like this is to state that every row/genotype in the input set can only contain two values from {0,1,2} and one has to be 2. If we choose 0 to be the other value we have genotypes that only contain 0's and 2's and then at least the trivial haplotype (all 0's) can be shared with all the genotypes and then every two genotypes in G share a haplotype. The trivial haplotype is all 1's if 1 is chosen as the other value. The PH($*, 2$)C1 algorithm with these constraints has been proven to be polynomial time solvable [16]. First note that these clique instance imposes some constraints on the sharing among the genotypes [16]; these constraint make it possible to create a *clique inference path* for every haplotype h that is not the trivial haplotypes and genotypes g and g' that share h (only for h and g if g does not share h with any other genotype). *Clique inference path* can be considered as a path in the PH($*, 2$)C1 Clark consistency graph or a independent path that is created outside the graph. In this project the *clique inference path* is a independent path that contains nodes that represent haplotypes and edges that represent genotypes. The path starts from some non-trivial haplotype h and genotypes g and g' that share h, and terminates when the trivial haplotype is encountered. Edges in the path that belong to haplotype haplotype h' represent genotypes in G, that h' can explain. Non-trivial haplotypes in PH($*, 2$)C1 can only be shared by two genotypes, hence each node in *clique inference path* can only have out-degree of two [16]. If the *clique inference path* is a cycle of length k it has been proven that only k haplotypes are needed to explain genotypes in the *clique inference path* [16].

It has also been proven that for a *clique inference path* of length k, that is not a cycle, most $k+1$ haplotypes are needed to explain genotypes in the path [16]. Clique inference path is the backbone behind the polynomial solvability of PH(*,2)C1. Algorithm 6 describes how the clique inference path is constructed and Algorithm 7 describes the PH(*,2)C1.

Algorithm 6 Constructing a inference path

Input: Haplotype h genotype g, and inference path IP
Output: Inference path IP

if h is trivial haplotype **then**
  return IP
else
  Create another haplotype h' where h, h' explain g
  if h' is already in IP **then**
    return IP
  end if
  Add h' to IP
  Create edge between h and h', that represent genotype g
  Find genotype g' that h' can explain
  if there is no $g' \mid\mid g' \in IP$ **then**
    return IP
  else
    Continue inference path for h' and g'
  end if
end if

Algorithm 7 PH(*,2)C1

Input: Clark consistency graph G
Output: Set H with minimum number of haplotypes

for each node $g_1 \in G$ **do**
  neighbors $\leftarrow$ Get all neighbors of g_1
  for each neighbor $g_2 \in$ neighbors **do**
    Find haplotype h that is compatible with g_1 and g_2
    if h is the trivial haplotype **then**
      Continue with next neighbor
    else
      Add h to inference path IP
      Continue a inference path IP for h, g_1 and h,g_2
      if IP is a cycle **then**
        Add all haplotypes in IP to H and remove all genotypes in IP from G
      end if
    end if
  end for
end for
if $G \neq \emptyset$ **then**
  for each node $g \in G$ **do**
    Find $h'$and h where h', h explain g
    if h and $h' \notin H$ **then**
      Add h' and h to H
    end if
  end for
end if
return H

3.2.3.2 PH($*, 2$)C2 data model

To be able to solve *MPH* for more general data set, we constructed PH($*, 2$)C2 algorithm that solves *MPH* in polynomial time. In the PH($*, 2$)C2 problem the Clark consistency graph has two cliques (q=2). In PH($*, 2$)C2 problem instances there is no one haplotype that can be shared by all the genotypes in genotype set G, there are two haplotypes, each shared by a subset of G. These haplotypes are called the clique haplotypes h_1^c and h_2^c. The genotypes in the set can contain any of the values from {0,1,2}, not only two different values as in PH($*, 2$)C1.

Definition 4 All genotype sets that fulfill the PH($*, 2$)C2 constraints can be divided into three subsets G_1,G_2 and G_{12} where

- G_{12} = {genotypes that are compatible with genotypes in G_1,G_2 and G_{12}}
- G_1 = {genotypes that are only compatible with genotypes in G_1 and G_{12}, and all share h_1^c}
- G_2 = {genotypes that are only compatible with genotypes in G_2 and G_{12}, and all share h_2^c}

The Clark consistency graph for genotype set G that fulfills the PH($*, 2$)C2 bound, has two cliques, G_1 and G_2 and nodes in G_{12} connecting the cliques. PH($*, 2$)C2 can be solved by solving PH(*,2)C1 for each clique and then combine the solutions so it will give minimum set of haplotypes H. The decision lies in choosing the optimal haplotype inferring for genotypes in G_{12}. If G_{12} is empty it means there are no nodes that connect the two cliques and there for it is just two separate PH(*,2)C1 problems that can be solved and then combined. But if G_{12} is not empty the following lemma puts constraint on the number of genotypes in it.

Lemma 1 *There can be at most two genotypes in* G_{12}

Proof According to *Definition 4*, genotypes in G_1 are incompatible with genotypes in G_2. Therefore according to *Definition 1* there must exist a site *j* where some genotypes from one clique have 0's and some genotypes in the other clique have 1's. Since genotypes in G_{12} are compatible with all genotypes in G_1 and G_2, those genotypes can neither have value 0 or value 1 at site j. Hence all genotypes in G_{12} must have value 2 at site *j*, and therefore there can only be at most two genotypes in G_{12} otherwise there would be more than two 2's in one site.

In addition the following applies for cliques G_1 and G_2 in PH($*, 2$)C2 problem instances.

Lemma 2 *Two genotypes in a clique can only share a non-clique haplotype if they have* 2 *in the same site.*

Proof According *Definition 4* all genotypes in clique share a clique haplotype h^c. A non-clique haplotype h^{nc} is haplotype that can shared by genotypes in a clique and has at least one site j where h^c and h^{nc} have different values, otherwise they would be the same haplotype. Genotypes that share h^{nc}, must have a value at site j that is compatible with both h^c and h^{nc} and the only value possible is 2. Therefore genotypes that share a non-clique haplotype must have value 2 in the same site.

Lemma 3 *Every non-clique haplotype can only be shared by two genotypes*

Proof According to the Lemma 2 a non-clique haplotype can only be shared by genotypes that have 2's in the same site so there can only be two genotypes that share the same non-clique haplotype without breaking the $(*, 2)$ bound.

Before describing the algorithm for solving PH$(*, 2)$C2 we consider how cliques in a PH$(*, 2)$C2 Clark consistency graph can be found. Lemma 1 states that in PH$(*, 2)$Cq Clark consistency graph two cliques can share at most two nodes. Hence if there are only two cliques, then there are only two nodes that are shared by the cliques and they are in set G_{12}. All other genotypes are in exactly one clique. Therefore the algorithm finds genotype g that are not in G_{12} and the neighborhood of g is declared as a clique in PH$(*, 2)$C2 Clark consistency graph. One way to find nodes that are not in G_{12} is to identify nodes that have equal or fewer number of neighbors than their neighbors, since genotype in G_{12} would always have larger number of neighbors since the are connected to genotypes in two cliques. Algorithm 8 demonstrates how cliques in a PH$(*, 2)$C2 Clark consistency graph are found.

Algorithm 8 Finding cliques in PH(*,2)C2 graph

Input: Clark consistency graph C
Output: Set of two cliques

```
1: Create set cliques for cliques
2: for each node g ∈ C do
3:    minsize ← degree of node g
4:    minode ← g
5:    neighbors ← find neighbors of g
6:    for each node g' ∈ neighbors do
7:       if g' is already in clique then
8:          continue
9:       end if
10:      neighnode size ← degree of node g'
11:      if neighnode size < minsize then
12:         minsize=neighnode, minnode=neighnode
13:      end if
14:   end for
15:   minneighbor ← find neighbors of minneighbors
16:   clique ← minode and minneighbor
17:   Add clique to cliques
18: end for
19: return cliques
```

In [16] it was proofed that in a PH($*, 2$)C1 bounded clique instances non-trivial genotypes (contain at least one heterozygous site) can only be in one clique inference cycle. Further it can be proofed that nodes in G_{12} can only be on one clique inference path that contains haplotypes from two cliques. This can be proofed by Lemma 4

Lemma 4 *In a (*,2) bounded clique instances, any node in G_{12} genotype belongs to at most one clique inference path that contains genotypes from two cliques*

Proof Assume the contrary that $g \in G_{12}$ can be explained by four haplotypes such that $g = h^a + h^b = h^c + h^d$, where h^a and h^c are compatible with genotypes in G_1 and h^b and h^d are compatible with genotypes in G_2. h^a and h^c are two distinct haplotypes, hence there must be one site j where h^a and h^c have opposite values. Then w.l.o.g. we can assume that $h^a_j = 0$ and $h^c_j = 1$, since g can be explained by either of these haplotypes, $g_j = 2$. There are genotype g' in G_1 that share h^a with g and genotype g'' that share h^c with g. Obviously g' and g'' must have different value at site j. According to *Definition 1*, g'_j and g''_j can not have opposite values, so one genotype has to have value 2 at site j. For haplotypes h^b and h^d they also have to have opposite values at site j, and since there genotypes that are compatible with both h^b and h^d it means the one genotype in G_2 must have 2 in site j for same reason described earlier. That is a contradiction, since there can only be two heterozygous value in each site because of the $(*, 2)$ constraint.

Given the constraints in Lemma 1, Lemma 4 and Algorithm 8, Algorithm 9 for solving PH(*,2)C2 can be given.

Algorithm 9 PH(*,2)C2

G_1, G_2 and G_{12} are as defined in Definition 4
Input: Clark consistency graph C
Output: Haplotype matrix H with minimum number haplotypes

Create empty set, $solutions$
if g_1 or g_2 in G_{12} can be inferred with haplotypes from cliques G_1 and G_2 in the same haplotype pair **then**
    Construct inference path for $g_i \in G_{12}$, that has haplotypes that is compatible with genotypes in both clique cliques, until the trivial haplotype is encountered in either clique.
    Solve PH($*, 2$)C1 for genotypes that are not in the other inference path
    Add solution to $solutions$
end if
for each clique c **do**
    Solve PH($*, 2$)C1 for c, with no genotypes from G_{12}. Add solution to $solutions$
    for each genotype $g_i \in G_{12}$ **do**
        Solve PH($*, 2$)C1 for c, include $g_i \in G_{12}$. Add solution to $solutions$
    end for
    if G_{12} contains two genotypes **then**
        Solve PH($*, 2$)C1 for c, and include $g_1, g_2 \in G_{12}$. Add solution to $solutions$.
    end if
end for
Create set minHap with size equal to 2*size(G)
for each $sol \in solutions$ **do**
    for each $sol' \in solutions$ **do**
        $tmpHap \leftarrow$ combine sol and sol', if the explain all genotypes in G
        if tmpMin < minHap **then**
            minHap $\leftarrow$ tmpMin
        end if
        Empty tmpMin
    end for
end for
return minHap

Theorem 1 *Parsimony can be solved in polynomial time on a PH($*, 2$)C2 instances*

Proof If we assume we have genotype set G that fulfills the PH($*, 2$)C2 clique bound. The genotype set can be divided into G_1, G_2 and G_{12} in the same manner as described in *Definition 4*.

If $G_{12} = \emptyset$ then G_1 and G_2 don't share any genotypes and the graph is just two separate cliques that can be solved in. Then PH($*, 2$)C1 solves *MPH* for each clique in in polynomial time since PH($*, 2$)C1 is polynomial solvable [16]. The haplotypes sets for each solutions are then combined as a set H of minimum number of haplotypes needed to resolve G.

If we assume that G_{12} is not empty, it can contain most two genotypes g_1 and g_2. The PH($*, 2$)C2 algorithm can then either solve PH($*, 2$)C1 for different combination of these sets of genotypes (See Lines 7 to 15 in Algorithm 9).

1. G_1
2. $G_1 \cup g_1$
3. If G_{12} has more than one genotype
   (a) $G_1 \cup g_2$
   (b) $G_1 \cup G_{12}$

The same thing is done for G_2.

PH($*, 2$)C1 finds the most parsimonious solutions for the clique that is solving for. More then one solution can give equally parsimonious solution for that clique, but one solution can be more optimal when the solutions for the two cliques are combined. This only applies when genotypes in G_{12} can be inferred with haplotypes from both cliques in the same haplotype pair.

If genotypes in G_{12} can be inferred with haplotypes from clique G_1 and G_2 in the same haplotype pair, we start by constructing the clique inference where genotypes in G_{12} are inferred with haplotypes from both cliques. We continue to construct the inference path in both direction until we encounter a trivial haplotype in either clique, or a haplotype that no other genotype shares. When reaching a trivial haplotype in either G_1 and G_2, it means for that clique all other genotypes that are not in the path, phasing them with the trivial haplotype will give the most parsimonious solution since we would only have to add one haplotype for each genotype. There is no other path that is possible for genotypes in G_{12} and for genotypes that are not in the path so it is always the most parsimonious solution for them. By constructing this path we can get a solution for each clique that is equal to a parsimonious solution that would be found for each clique if they were solved separately, but the solutions would share some haplotypes and therefore we have fewer haplotypes. We also check if there is another way to inference haplotypes by including the trivial haplotype in the path. This way we make sure we check all possible solution and are therefore guaranteed a parsimonious solution

PH($*, 2$)C1 correctness has been proven [16] and therefore we know we are getting the most parsimonious solution for all genotypes in G_1 and G_2. For genotypes in G_{12} we

have to make sure that the PH($*, 2$)C1 algorithm returns the most parsimonious solution. If genotypes in G_{12} can only be explained by haplotypes from one clique at a time, then inferring each genotype with each clique and then both together in each clique separately will find the most parsimonious solution, but if genotypes in G_{12} can be explained by haplotypes from both cliques in the same haplotype pair, then it can give more parsimonious solution to infer them in that way since we could be reusing two haplotypes. Hence if genotypes in G_{12} can be inferred in that way and Algorithm 9 does that. From Lemma 4 we known that there can only be one inference path that found for genotypes in G_{12} that has haplotypes form G_1 and G_2. Therefore we get the most parsimonious solution inferring them in that way and all possibilities for inferring genotypes in G_1 and G_2 are tried so we find the most parsimonious for them two.

Since PH($*, 2$)C1 is polynomial solvable[16] this part of the algorithm must also be polynomial solvable since it is solving PH($*, 2$)C1 in cases that are independent of each other.

The next step is to find all permutation for the PH($*, 2$)C1 solutions for G_1 and G_2 in a way that haplotypes inference for all genotypes in G are included.

Algorithm 9 then chooses the solutions with the minimum number of haplotypes to be the minimum set H that resolves G. (See Lines 17 to 25 in Algorithm 9). The smallest solution contains either two parsimonious solution for PH($*, 2$)C1 solution for G_1 and G_2 or a the most parsimonious solution for the whole graph. Therefore it can not exist a more optimal solution than the one that is chosen here. That proofs the correctness of the algorithm.

Combing these solutions takes polynomial time that depends on number of genotypes in G_{12}. This gives us two separate task that are polynomial solvable and that leads to show that PH($*, 2$)C2 is polynomial solvable and finds the minimum solution. ∎

3.2.4 Summary of Graph based models

The novel algorithms described in this chapter, bring us closer to solving the PH($*, 2$) problem. But having these restriction on the input genotype set limits the problem instances that can be solved and therefore it is also necessary to have model that solves *MPH* for the general case. In the next chapter we will describe such a model.

3.3 Boolean Satisfiability model

To be able to handle more general problem instances, and solve real life problems, we need a different model than the two previously mentioned. The boolean satisfiability model that solves *MPH* is more efficient than any other exiting models. We give a description of the boolean satisfiability model proposed by Lynce *et al.* [8], and describe the improvements we made to the model, that resulted in several times faster solution time. Boolean Satisfiability (SAT) problem is a decision problem, consisting of

- A set X of n variables $x_1, x_2, ..., x_n$
- A set C of m clauses/constraints. Where each clause consist of variables or negation of variables from X and are combined by logical *or* ($\vee$)
  $C_1 = (x_1 \vee \overline{x_2})...$
- A Conjunctive normal form *(CNF)* formula which consist of the clauses from C combined with logical *and* ($\wedge$)
  $C_1 \wedge .... \wedge C_m$ [17]

The goal of the SAT problem is to decide whether the variables in X can be given logical values (True, False) that makes the formula result in true. If there exists an assignment of logical values that makes the formula true, then the formula is said to be satisfiable. SAT is used to solve many problems in various areas of computer science, including theoretical computer science, algorithmics, artificial intelligence, hardware design and verification. Solving SAT is a NP-complete problem and was one of the first decision problem proven to be NP-complete. NP-complete problem are known to be important to many areas in science and finding polynomial solution to one NP-complete problem would mean finding solution for every other NP-complete problem. Great effort has been made to construct algorithms to solve the SAT problem and it has involved variety of different methods. In this project the SAT-problem was solved with a publicly available SAT-solver called MiniSAT [3].

3.3.1 The basic SAT Model

Lynce *et al.*[8] proposed a SAT formulation of the haplotype inference, (*HI*), problem and in this project the a SAT basic model was constructed based on their description.

The *MPH* SAT model is similar to another parsimony model that uses Integer Programming [16]. The input to the model is a genotype set G and the output is the minimum number of haplotypes needed to explain G. The SAT-based models, formulates whether there exists a set *H*, of distinct haplotypes with $r = |H|$ haplotypes such that for all genotypes $g_i \in G$ they are explained by a pair of haplotypes (can be the same haplotype) in *H*. The algorithm solves it with increasing size of *H* from lower bound *lb* to an upper bound *ub*. The algorithm terminates when it reaches a size of *H* which there exists $r = |H|$ haplotypes such that every genotype $g_i \in G$ is explained by a pair of haplotypes in H. *Algorithm* 10 summarizes the top-level operation of the SAT Model suggested by Lynce *et al. MPH*[8].

Algorithm 10 Base *MPH* SAT model

Input: Genotype set G
Output: Minimum number of haplotypes, r

```
1: G ← simplify(G)
2: ICG ← createIncompatibleGraph(G)
3: r ← calculateLowerBound(ICG)
4: ub ← 2 * size(G)
5: while r < ub do
6:    do Generate CNF form given G and r
7:    if SAT(CNF)==true then
8:       return r
9:    else
10:      r ← r + 1
11:   end if
12: end while
13: return r
```

Before the SAT formulation for *MPH* is created the *MPH* SAT model starts by simplifying the genotype set G. It removes repeated genotypes and duplicate sites, which are sites where all the genotypes have the same value. It also removes all *complimented* sites, which are two sites where the homozygous values are complemented. This simplification reduces the size of the problem resulting in fewer variables and clauses in the proposition formula. The algorithm iterates from a given lb to some ub to find the value r. A trivial value for the lb is 1 and the ub is guaranteed to be $2*n$ where n is the number of genotypes in G because each genotype can be explained by most two haplotypes.

The model needs to create all necessary variables and combine them in essential constraints/clauses to make the boolean satisfiability formula for a given G and r. The resulting formula is on *conjunctive normal form (CNF)* formula since the SAT-solver [3] that is used expects the input formula to be in *CNF* [17]. The formula is then sent to the SAT-solver [3] and if the formula is satisfiable the algorithm returns r as the least number of haplotypes, otherwise it increases r by one and creates a new formula for G and the updated r. For the basic model there are indexes i, j used for genotypes such that $1 \leq i \leq n$ and $1 \leq j \leq m$ where n is the number of genotypes and m is the number of sites in the genotype. In addition there is index k for the haplotypes such that $1 \leq k \leq r$. For a given r and each genotype g_i the model creates haplotype variables h_{ik} for all $j = 1, ..., m$ and $k = 1, ..., r$.

The model creates *selector variables* for selecting which haplotypes are used to explain genotype g_i. Since each genotype is explained by two haplotypes (can be the same haplotype), the model uses two sets of *selector variables*, a and b, of r *selector variables*, respectively s^a_{ki} and s^b_{ki}. Hence genotype g_i is explained by haplotypes h_1 and h_2 if $s^a_{1i} = 1$ and $s^b_{2i} = 1$. Obviously, g_i is explained by the same haplotypes if $s^a_{2i} = 1$ and $s^b_{1i} = 1$. The boolean satisfiability formula is created in the following way:

If site $g_{ij} = 0$, then the model requires that, for $k = 1, ..., r$:

$$(\neg h_{kj} \vee \neg s^a_{ki}) \wedge (\neg h_{kj} \vee \neg s^b_{ki}) \tag{3.1}$$

Hence, if haplotype h_k is selected to explain genotype g_i, either by s^a or s^b, then h_k has to have value 0 at site j. If site $g_{ij} = 1$, then the model requires that for, $k = 1, ..., r$:

$$(h_{kj} \vee \neg s^a_{ki}) \wedge (h_{kj} \vee \neg s^b_{ki}) \tag{3.2}$$

Hence, if haplotype h_k is selected to explain genotype g_i, either by s^a or s^b, then h_k has to have value 1 at site j

Otherwise site $g_{ij} = 2$ and then the haplotypes needed to explain g_i must have opposing values at site j. This can be done by creating a new set of variables $t_{ij} \in \{0, 1\}$ such that site j of the haplotype selected by the selector variable s^a assumes the same value as t_{ij} and the site j of the haplotype that is selected by the selector variable s^b assumes the complemented value of t_{ij}. This forces the haplotype selected by s^a to have value 1 at site j and the haplotype selected by s^b to have value 0 at site j. As a result the model require, for $k = 1, ..., r$:

$$\begin{aligned}&(h_{kj} \vee \neg t_{ij} \vee \neg s^a_{ki}) \wedge (\neg h_{kj} \vee t_{ij} \vee \neg s^a_{ki}) \quad \wedge \\ &(h_{kj} \vee t_{ij} \vee \neg s^b_{ki}) \wedge (\neg h_{kj} \vee \neg t_{ij} \vee \neg s^b_{ki})\end{aligned} \tag{3.3}$$

It is necessary to ensure that for each genotype g_i exactly one haplotype is selected by s^a and exactly one haplotype is selected by s^b. That can be done by requiring that for each k and i, the added values of s^a_{ki} equals 1 and the same for s^b_{ki}. The solution is to create a *CNF* representation of an adder circuit. This can be done by adding variables v^a_{ki} and v^b_{ki}.

v^a_{ki} are defined as follows, for $k = 1, .., r$:

$$\begin{aligned} (\neg v^a_{1i}) \quad & \wedge \\ (\neg v^a_{ki} \vee \neg s^a_{ki}) \quad & \wedge \\ v^a_{k+1i} \longleftrightarrow (v^a_{ki} \vee s^a_{ki}) \quad & \wedge \\ (v^a_{r+1i}) \quad & \end{aligned} \tag{3.4}$$

The v^a_{ki} variables are true if and only if s^a_{k-1i} is true, the model then makes sure that no other s^a_{ki} is true for that i. The last condition states that there must have been some s^a_{ki} that was true. This way it is ensured that exactly one s^a is selected for each g_i. The adder circuit for s^b is exactly the same.

Algorithm 11 is a top-level description of how the *CNF* formulation for a given G and r is created. When the formulation of the condition have been created it is sent to a SAT solver called MiniSAT [3] that checks if the formulation is satisfiable or not. If it is satisfiable it returns one solution that has boolean values for all the variables that satisfies the formulation. Retrieving the haplotypes them self would only require retrieving the boolean values given to haplotype variables. In this project the least number of haplotypes, r, is the only information considered in the *MPH* SAT Model.

Algorithm 11 Create SAT formula

Input: Genotype set G and lower bound r
Output: Boolean value for the CNF formula

```
1:  for each genotype g_i ∈ G do
2:      for each site j in g_i do
3:          if g_i==0 then
4:              Create condition for g_ij = 0
5:          else if g_i=1 then
6:              Create condition for g_ij = 1
7:          else
8:              Create condition for g_ij = 2
9:          end if
10:     end for
11:     Create Adder circuit for s^a
12:     Create Adder circuit for s^b
13: end for
14: Create condition to improve the model
```

Several key improvements were developed by Lynce *et.al* [8] to speed up the algorithm. These improvements were the following.

3.3.1.1 Lower bound

One way to calculate lower bound is by making a incompatible graph for the genotypes. After the genotype set has been simplified the model creates a incompatible graph for the genotype set where the nodes represent genotypes and nodes have an edge between them if they do not share a haplotype. It then searches for the biggest clique in the graph. Since finding a the biggest clique is NP-hard it is done with a greedy procedure that finds the node with most neighbors and adds it to a clique, and then goes through each neighbor and adds it to the clique if it is incompatible with all other nodes in the clique. The genotypes in the biggest clique do not share any haplotype, so it is know that each genotype has at least two haplotypes associated with it that no other genotype in the clique shares. Then the number of haplotypes has to be at least $lb = 2 * k - \alpha$, where k is the size of the clique and α is the number of *homozygous* haplotypes since there is only one haplotype associated with them.

3.3.1.2 h variables symmetries

Pruning the search space can be done by observing symmetries in the problem formulation. For example for two haplotypes h_1 and h_2 and the selector variables s^1_{1i}, s^a_{2i}, s^b_{1i} and s^b_{2i}. Then $s^1_{1i}s^a_{2i}s^b_{1i}s^b_{2i} = 1001$ corresponds to s^1_{1i},$s^a_{2i}s^b_{1i}s^b_{2i} = 0110$ and one of the assignment can be eliminated. Elimination of redundant assignments can be achieved by enforcing ordering of the boolean valuation to the haplotypes:

$$h_1 < h_2 < ... < h_r$$

The ordering can be achieved by CNF representation of a boolean comparator circuit between h_k and h_{k+1}, with $1 \leq k < r$ and requiring $h_k < h_{k+1}$. One possible solution is to use additional variables lt_{kj} and ltt_{kj}. The lt_{kj} and ltt_{kj} are defined as follows, for $k = 1, .., r$:

$$\begin{aligned} (\neg lt_{k0}) \quad & \wedge \\ ltt_{kj} \longleftrightarrow (\neg h_{kj} \vee h_{k+1j}) \quad & \wedge \\ lt_{kj} \longleftrightarrow (ltt_{kj} \vee ltt_{kj-1}) \quad & \wedge \\ (lt_{km}) \quad & \wedge \\ (\neg h_{kj} \vee h_{k+1j} \vee lt_{kj}) & \end{aligned} \tag{3.5}$$

3.3.1.3 s variables symmetries

It is also possible to eliminate symmetries on the s variables by requiring that only one arrangement of the s variables can be used to explain each genotype g_i. One solution for this is to require that the haplotype selected by the s^a_{ki} variables always has an index

smaller than the the haplotype selected by the s^b_{ki}. This requirement is captured by the following conditions, for $k = 1, .., r$

$$(s^a_{ki} \longleftrightarrow \bigwedge_{k_2=1}^{k-1} \neg s^b_{ki}) \wedge (s^b_{ki} \longleftrightarrow \bigwedge_{k_1=k+1}^{r} \neg s^a_{k_1 i}) \tag{3.6}$$

3.3.1.4 Constraining the s variables of incompatible genotypes

The s variables can be constrained further due to the fact that the haplotypes that can explain a given genotype g_{i_1} can not be used to explain genotype g_{i_2}, if g_{i_1} and g_{i_2} are incompatible genotypes. The model requires that for two incompatible genotype g_{i_1} and g_{i_2} and for $k = 1, ..., r$:

$$(\neg s^a_{ki1} \vee \neg s^a_{ki2}) \wedge (\neg s^a_{ki1} \vee \neg s^b_{ki2}) \wedge (\neg s^b_{ki1} \vee \neg s^a_{ki2}) \wedge (\neg s^b_{ki1} \vee \neg s^b_{ki2}) \tag{3.7}$$

Hence for incompatible genotypes g_{i_1} and g_{i_2}, if either s^a_{ki1} or s^b_{ki1} are true, meaning h_k is used for explaining genotype g_{i_1}, then haplotype h_k can not be used for explaining g_{i_2} and therefore s^a_{ki2} and s^a_{ki2} may not be true.

3.3.2 Improvements to the basic SAT Model

One reason for reconstructing the *MPH* SAT-based model was to be able to improve the model. After doing experiments on real genotype data from the HapMap project [10], observations were made that gave an ideas to improve the model even further. They were the following.

3.3.2.1 Handle unknown SNPs

In real genotype data some SNPs are unknown. In cases like that, the SNPs are labeled with a value other than {0,1,2}. In the Lynce *et al.* model genotypes sets with unknown SNP can not be solved, which is great limitation for the model. In real genotype data there are often unknown SNPs and therefore it is of great importance to handle them. In our model these unknown SNPs are handled. In the preprocessing part for the genotype set the number of unknown SNPs is reduced. For two genotypes that only differ in the sites where one of them has an unknown SNPs are merged. This is done by replacing the unknown SNPs by the known values from the other genotype. The motivation behind this is that there are fewer distinct haplotypes in populations than possible haplotypes because of bottleneck effect in evolution, hence it is more likely that two genotypes that only differ in the unknown SNPs are the same genotype. In addition, in the problem formulation the unknown SNPs are handled. Each unknown SNP can be any of the values from {0,1,2} thus in our model they are handled in a way that they don't affect the solution. In the SAT problem formulation no Constraints like 3.1, 3.2 and 3.3 are created for sites that contain unknown SNPs. The only constraints created for these sites in Constraint 3.5 and then the SAT-solver can always assign true to the those variable since there are no pervious constraints to it. Hence variables of unknown SNPs don't affect the solution

3.3.2.2 More accurate lower bound

Another way of calculating lower bound is to find the rank of the genotype matrix as suggested by Kalpakis *et al.*[11]. Kalpakis *et al.*[11] have proved that the rank of the genotype matrix is the lower bound on the size of the solution for solving *MPH* for a given genotype set. Rank of a genotype matrix G that is of size nxm is the number of linear independent rows in G and $rank(G) \leq min(n, m)$. The rank lower bound returns more accurate lower bound in most cases, though there are some cases that the greedy procedure returns a more optimal lower bound. Therefore, it is better to calculate lower bound with both procedures and return max(clique, rank) since it quick to calculate either one.

3.3.2.3 s variables symmetries

The s variable symmetry breaking suggested by Lynce *et. al* [8] requires additional variables and clauses, and makes the SAT formulation more complicated. These constraints can be simplified. Constraint 3.6 makes sure that the haplotype selected by the s^a_{ki} has an index smaller than the haplotype selected by the s^b_{ki}. The same requirement can be captured by creating simpler constraints.

$$(v^a_{ki} \vee \neg v^b_{ki}) \tag{3.8}$$

Lemma 5 *If v^a_{ki} is false then every $s^a_{k_2 i}$ is false for $k_2 < k$*

Proof According to equation 3.4 v^a_{ki} is true if and only if s^a_{k-1i} is true, and then all $v^a_{k_1 i}$ are true for $k_1 > k$, hence if v^a_{ki} is false then every $s^a_{k_2 i}$ is false for $k_2 < k$

According to *Lemma* 5 if v^a_{ki} is false then every $s^a_{k_2 i}$ is false for all $k_2 \leq k$ and the same applies to v^b_{ki}. Constraint 3.8 states that v^b_{ki} can only be true if v^a_{ki} is true. Hence if v^a_{ki} is true, $v^b_{k_1 i}$ can only be true for $k_1 \geq k$ and hence $s^b_{k_1 i}$ can only be true for $k_1 \geq k$. That way Constraint 3.8 enforces the s^a_{ki} variables to have smaller index than the s^b_{ki} variables. This reduces the number of clauses and results in a much simpler solution.

3.3.2.4 Ordering haplotypes

Lynce *et al.* [8] suggest using lexicographic order to observe the symmetries in the h variables, meaning haplotype such as 000 would considered first and 111 last. If number of genotypes in the genotype set G share a haplotype h it is likely that many of these genotypes will be explained by h. If this haplotype h is late in the lexicographic order it will be considered late as a possible haplotype for these genotype. This is not feasible if large portion of the genotypes share this haplotype. This can avoided by preprocessing the genotypes so that more common haplotypes will land early in the lexicographic order, and hence we don not have to add any new constraints to the model. In the preprocessing it is checked if the major allele is 1 at some site j. Implying that the haplotype the genotypes share is later in the order rather than if the major allele would is 0. This can be changed by flipping the SNPs in sites when the major allele is 1.

This is though not enough because it is also important to reorder SNPs sites that have minor allele 1 in a way that they are not affecting the order of the haplotypes.

Obviously flipping and reordering SNPs in a genotype set G results in a different genotype set G'. Hence the haplotype set H' needed to explain G' is different from the haplotype set H needed to explain G. It can though be proven that H and H' contain the same number of haplotypes according to Lemma 6

Lemma 6 *Flipping and reordering SNPs will result in the same minimum number of haplotypes needed to explain G' as the original G*

Proof Two genotypes g_i and g_j share a haplotype if there does not exist a site in the genotypes where one has the value 1 and the other 0. When flipping the SNPs every 0 is flipped to 1 and 1 is flipped to 0 for every genotype in G when flipping a SNPs means that for all genotypes g_i and g_j that share a haplotype there still does not exit a site ,where one has the value 1 and the other 0. For genotypes g_i and g_k that do not share haplotypes there must exist a site where one has the value 1 and the other 0 and after the SNPs have been flipped this sites still have opposite values.

When reordering the SNP, all SNPs at same site in the genotypes are moved to some other site, so for all genotypes g_i and g_j that share haplotypes and for all genotypes g_i and g_k that do not share nothing has changed except the order of the haplotypes.

Therefore the flipping and the reordering of the SNPs does not increase or decrease the number of haplotypes needed to solve G.

This the same idea described in Section 3.1.2.

3.3.2.5 Trivial genotypes

If G contains a genotype g that only has homozygous sites, then there is exactly one haplotype h that can explain g ($h = g$). It is similar for genotypes that have only one heterozygous site, then there is exactly two haplotypes that can explain that genotype. These genotypes are *trivial* because their haplotypes have to be in the minimum solution and there is no need to create constraints for these genotypes in each iteration. This genotypes can therefore be removed from the genotype set G. When solving the *MPH* for the rest of the genotypes, constraints are put on genotypes so that genotypes that are compatible with the trivial genotypes are forced to be explained by the haplotype they share. That is done by making constraints that state that the haplotypes selected to explain those genotypes have to be one or two of the haplotypes that are necessary to be apart of the minimum solution. This way these haplotypes are taken into account in the solution. If some of the haplotypes are not shared by any of the other genotypes they can be counted and added to r in the end to get the least number of haplotypes needed to explain G. This reduces the number of possibles solution for the formula, since there can be fewer haplotypes considered in the solution.

3.3.3 Summary of the SAT model for MPH

The SAT model is a satisfiability formulation of the *MPH* problem [8]. The model consist of set of constraints represented in *Conjunctive normal form* [17]. The basic model works on a general problem instances and is faster than previous models. Improvements can be made to the basic SAT model using heuristic learned from doing experiments on real genotype data, that speed up the implementation even further.

Chapter 4

Computational results

Following experiments were done on genotype data for chromosome 22, taken from the Caucasian population (CEU) in the HapMap project [10]. All experiments were run on Intel(R) Xeon(TM) CPU 3.20GHz with 2GB in RAM running on GNU/Linux

4.1 SAT algorithm

We made improvements to the basic SAT model and they are described in detail in Chapter 3.3. In this section we will demonstrate the performance gains in the SAT based model after each improvement. Each experiment shows an average running time over 1000 running times for random indexes in the chromosome. This gives a more accurate running time since the solution complexity is different between parts in the chromosome.

4.1.1 The order of haplotypes

After running the base SAT Model on real genotype data it was obvious that the solution time increases with number of individuals. Thus we made an improvement to the model that should decrease the solution time and especially for large number of individuals. Figure 4.1 shows the effect of preprocessing the genotypes in a way that the lexicographic order reflects the communality of the haplotypes, which results in significant performance gain for the model. This is because in the base model, the lexicographic order can be inefficient if haplotypes that are selected to explain many genotypes are considered late in the order and hence the model has to iterate through bigger part of the haplotype list before finding the right haplotypes. The benefit of this improvements increases as the size number of haplotypes increases. That is consistent with what Figure 4.1 that is the difference gets more significant as the number or individuals grows and hence the number of haplotypes. This improvements is 4 times faster than the base model.

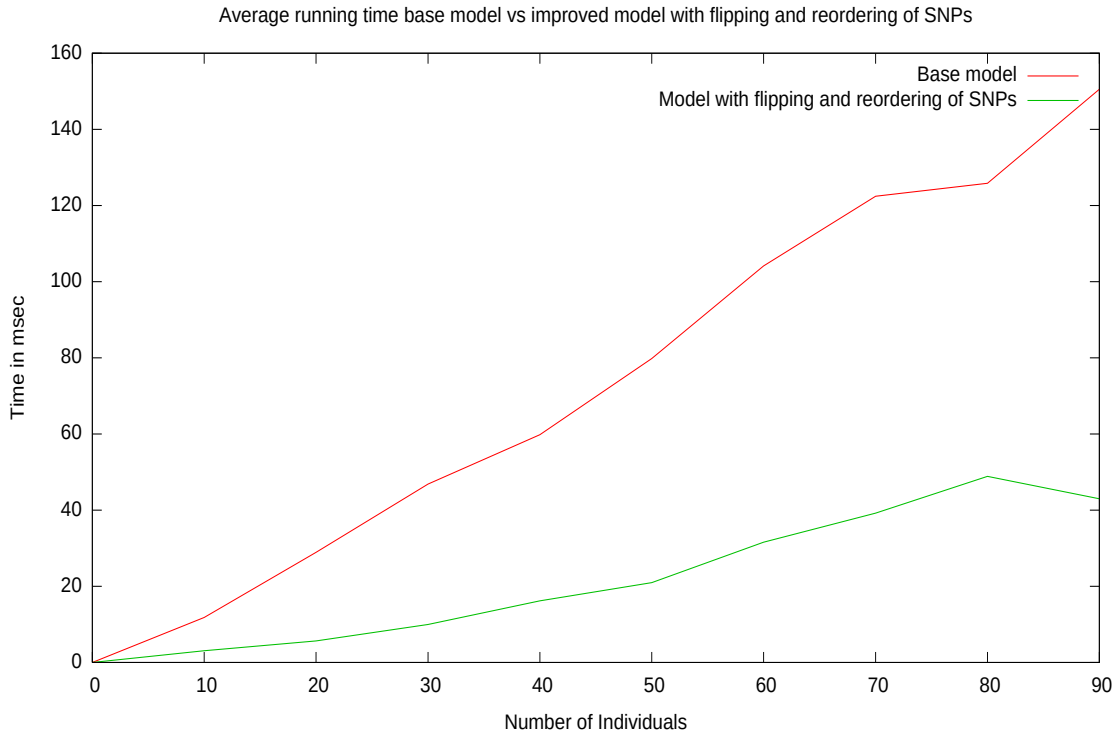

Figure 4.1: The average time over 1000 experiments with genotype length 10 SNPs

4.1.2 Trivial genotypes

From the first improvement it showed affecting the order of the haplotypes resulted in extremely fast solution, thus we decided to do another improvement that affected the haplotypes. Genotypes that have only *homozygous* sites or only one *heterozygous* can only be explained by one haplotype pair and the haplotypes that explain them have to be in the minimum solution. Therefore there is no need to include them in the problem formulation. Since it is more likely that there are fewer distinct haplotypes that explain a given genotype set it is safe to assume that some of the haplotypes belonging to the *trivial* genotypes can explain some of the genotypes that are still in the set. Therefore constraints can be added to the problem formulation that force these haplotypes to explain those genotypes that they can. This way the number of possible haplotype pairs are reduced for some genotypes. Figure 4.2 shows that this improvement also results in significant performance gain. This improvements is 3 times faster than the base model.

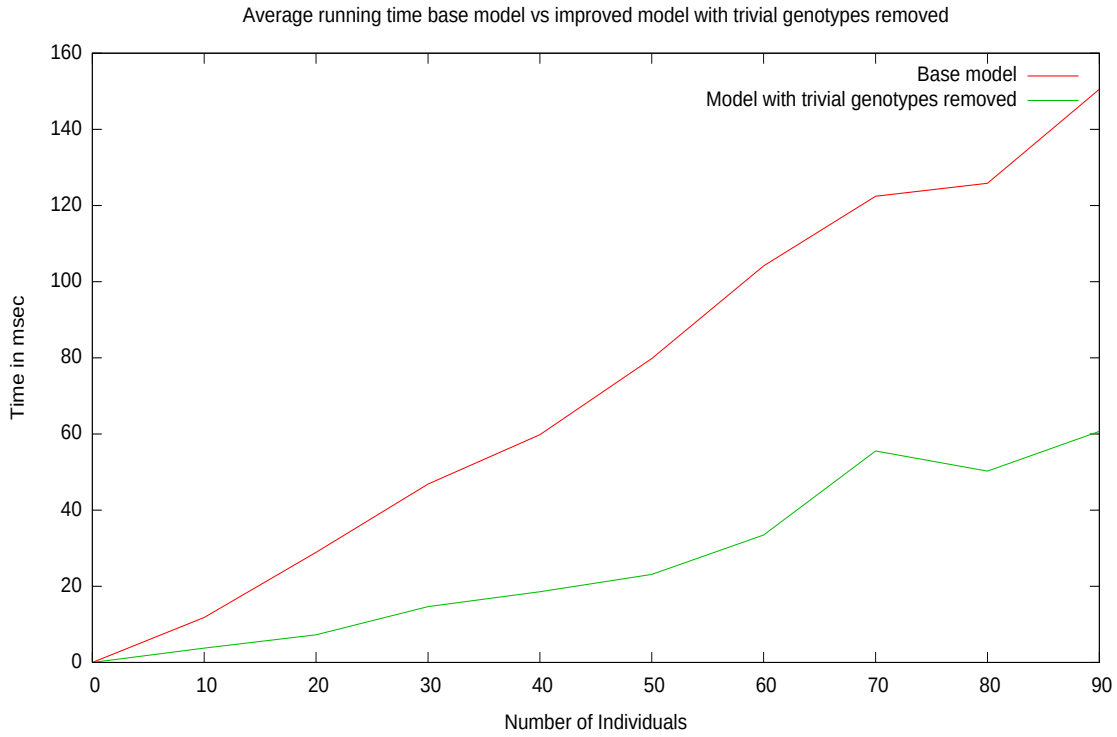

Figure 4.2: The average time over 1000 experiments with genotype length 10 SNPs

4.1.3 Combining the improvements

Individual experiments for both improvements showed performance gain. They do not preform equally well though (see Figure 4.3), which was not to be expected since the order of the haplotypes improvement benefits when the size of the solution grows but the *trivial* genotype improvement is more focused on the structure of the genotypes. But both improvements are several times faster than the base model so the rational next step was to use them simultaneously. Both these improvements were added to the basic model the result showed that the improved model was 4 times faster than the base model. See Figure 4.4

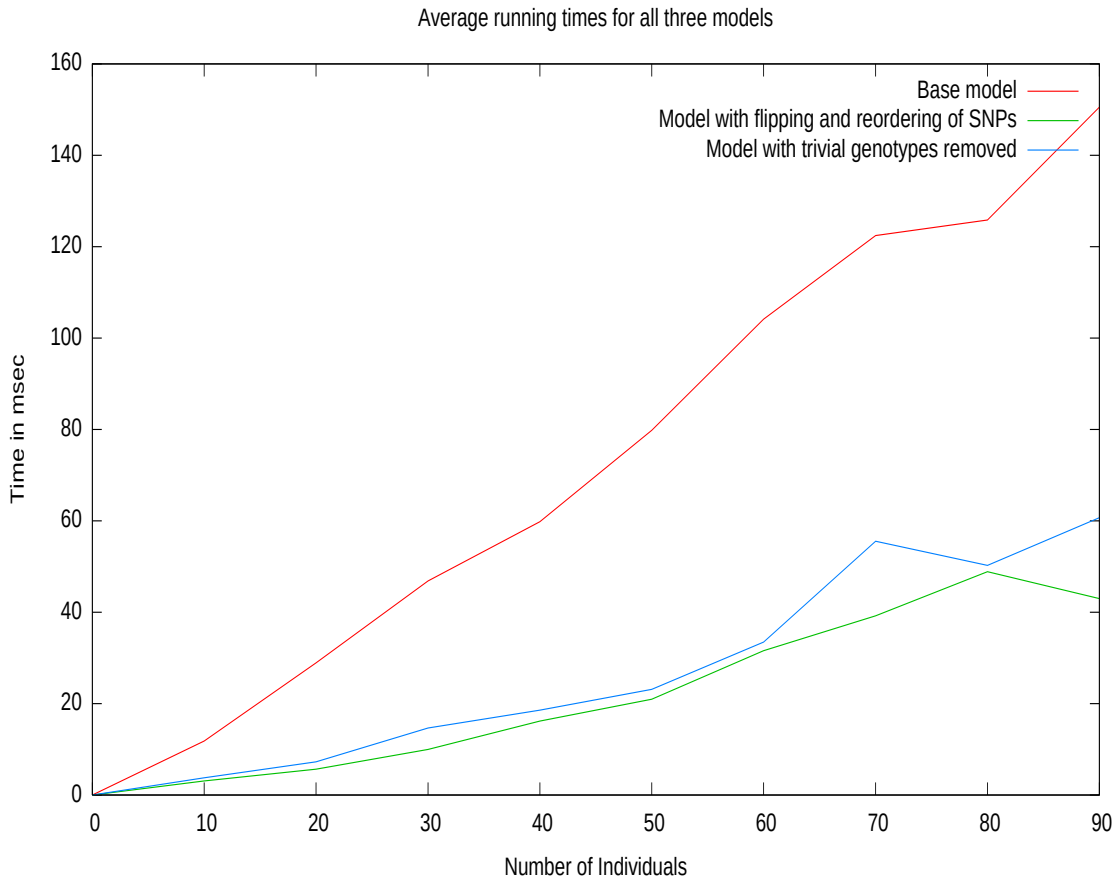

Figure 4.3: The average time over 1000 experiments with genotype length 10 SNPs

The box plot in *Figure* 4.5 shows that for the improved model quartiles and median are significant lower, even for large number of SNPs, such as 30 and 40 SNPs.

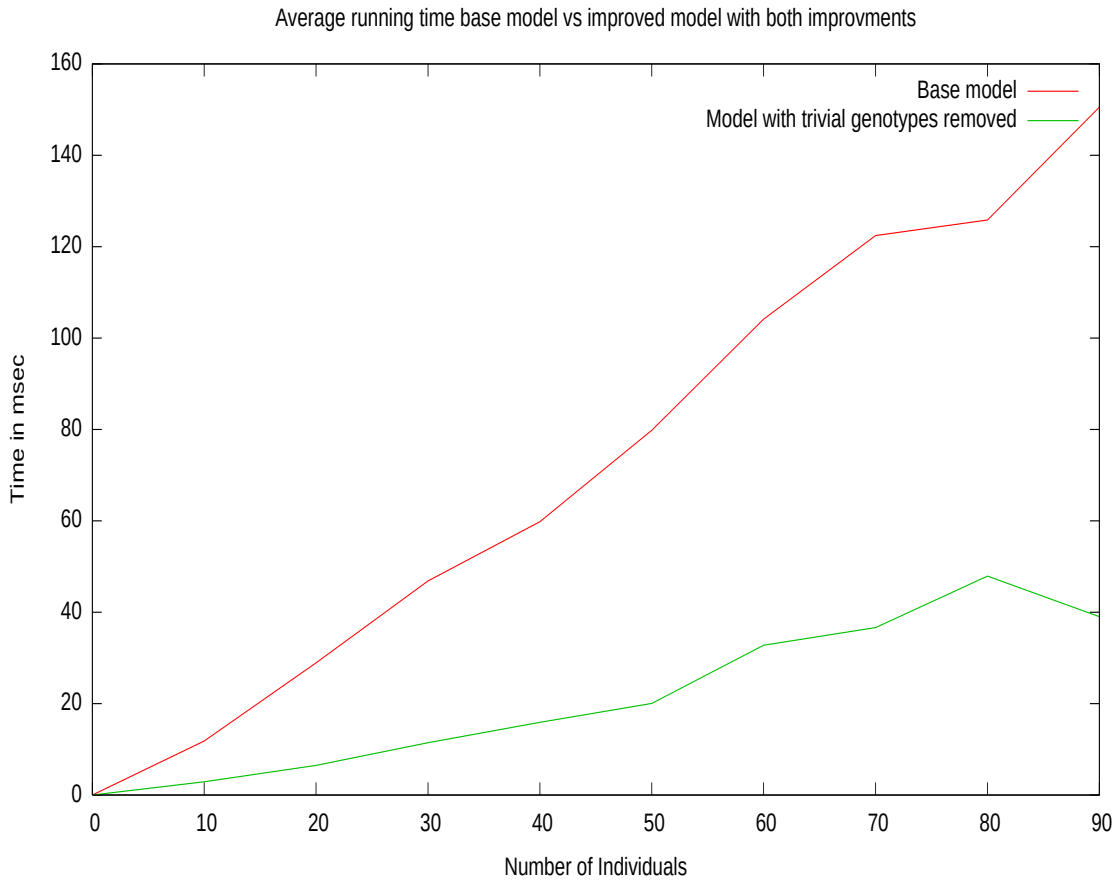

Figure 4.4: The average time over 1000 experiments with genotype length 10 SNPs

4.1.4 More accurate lower bound

In the Chapter 3.3, another method to calculate lower bound was described. In *Table* 4.1 it is shown how many times each of the two methods that gave more optimal lower bound for different number of runs. It is obvious that the calculating the lower bound by finding the rank of the matrix gives more frequently more accurate lower bound. Using greedy procedure is inconvenient for finding cliques, the order of the nodes we look at has great effect. The rank calculation is more consistent and seems to be closer to the optimal value. Although it can not be ignored that the greedy procedure is more accurate in significant number of instances, hence the best way is to calculate lower bound with both methods and return the maximum value of the two method as a lower bound.

Table 4.1: Number of times one has more accurate lower bound

Number of runs	50	100
Rank	26	45
Greedy procedure	9	23

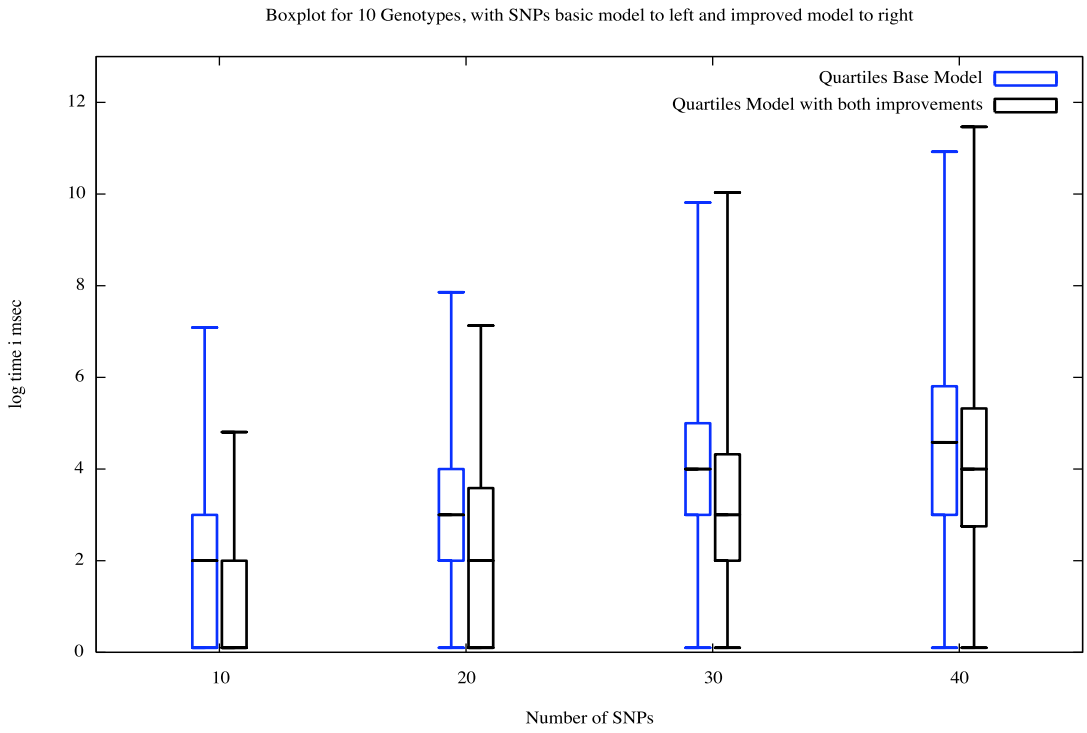

Figure 4.5: The time is in log_2 of the time

Chapter 5

Discussion

5.1 Discussion about the computational results

These result show that we can speed up the solution a lot with improvements that benefit more as the number of individual increases. That means that we have found efficient heuristic to handle the large number of individual. For the improved model it took on average 40 msec to solve *MPH* for 90 individuals and 10 SNP. The HapMap data [10] consist of 3 million SNPs and most sets contain 90 individuals. Therefore we could run our algorithm on the HapMap data in 3 hours which is extremely fast. This implies that the implementation can be use as highly efficient genetic tool, by computing recombination rate.

There is room for more improvements for the model since there are still maximum outliers with extremely high solution times. Next sections will discuss further improvements.

5.2 Future Work

5.2.1 More ideas for improvement

Further heuristics can be learned from the experiments that were done on the real genotype data. As can be seen Figure 4.5 there are still maximal outliers have high solution time. What can be observed when looking at genotype set that have high solution time is that these are sets with genotypes that have large number of heterozygous sites. Number of heterozygous sites has great affect on the complexity of the problem and therefore one solution could be to divide these complicated problems into smaller problems and solve the separately. One idea is to try to divide them into PH$(*, 2)$Cq problems where each problem contains genotypes that is not compatible with genotypes in any other problem. That way each clique can be solved independently of the genotype in the other cliques.

In addition if we want to have larger number SNPs than 40, the likelihood of finding compatible genotypes between the individuals would be come really low, meaning we could solve the problem as separate problems independent of each other. That would imply that the size of the genotypes could be increased without increasing the complexity of the solution.

5.2.2 Different models

5.2.2.1 More simpler SAT Model

The SAT-based model described in this project was similar to another parsimony model, suggest by Gusfield [5] that uses Integer Programming to solve *MPH*. Sharan *et al.* [16] came up with similar Integer Programming formulation that is more precise than the former one and eliminates some of the variables and the constraints. It would be beneficiary to implement a SAT-based model for that second formulation, since fewer variables and constraint result in a smaller problem formulation to solve.

5.2.2.2 Solving MPH with BDD

A binary decision diagram (BDD) is a data structure that is used to represent a boolean function. BDDs are a rooted, directed acyclic graph (DAG). BDDs are binary decision tree where all the redundancies have been removed. They consists of decision nodes and one or two terminal nodes labeled 0 (false) and 1 (true). Each of this decision nodes represents a Boolean variable in the Boolean function represented by BDD and have two child nodes. The two outgoing edges from the decision nodes assign 0 (false) and 1 (true) to the Boolean variable that the child nodes represent. Even though BDDs are reduced binary decision trees BDD rules can be applied to reduce them even further [1].

BDD have been known to improve the performance of a genetic linkage analysis tools which resulted in surprisingly good results [9]. Since the nature of the genetic linkage analysis is not different from the *MPH* problem there is a valid reason to assume BDD would work for the *MPH* problem also. Since BDD have not been used before to solve the *MPH* problem we need to find a way formulate the problem in way that BDD can solve it. That is not a problem since have already implement method that formulates the *MPH* problem as a SAT problem. One way to solve SAT problems to use Binary Decision Diagram (*BDD*) [2]. The *BDD* is constructed for a given SAT problem and the problem is said to be satisfiable if a *BDD* is different from the 0 terminal. *BDDs* should be more efficient to solve the SAT formulation of the *MPH* problem , than using a SAT-solver, since the success of *BDDs* is mostly due tho the fact that BDDs removes symmetries that occur in the formulation. Therefore the SAT based model could be made simpler by removing the symmetry breaking constraints mentioned in Section 3.3.1.2 and Section 3.3.1.3 since they would be unnecessary. Another possibility is to construct a different SAT formulation which is more suitable to be solved with *BDD's*. That is formulate the problem in a way that other symmetries in the problem solution can be eliminated. To be

to do that it requires more research on *BDD's* and satisfiability problems which will have to bee done in continuing work on this project.

Chapter 6

Conclusion

Each of the model discussed in this project has its advantages and disadvantages. Using a tree search model to solve *MPH* did not require complicated formulation of the problem, but is only solvable for small problem instances. The graph based solution had a more complicated formulation and solves larger problem instances, but only for restricted sets of genotypes. The last model, boolean satisfiability problem, was the most complicated model and improving it is not a trivial task since each change affects the whole formulation of the problem. The boolean satisfiability model is however able to solve *MPH* for large problem instances on real genotype data, which were several order magnitude faster than previously described methods, and still leaving room for improvements. Researching these different methods gives a good overview of what the complications of the *MPH* problem are, with each model not only giving new ideas of heuristics that can be used to improve the model but also the other models. It is therefore important to develop different models for solving *MPH*, even though they can only handle restricted instances of the problem, as every new idea can be crucial in coming up with an algorithm in the rapidly growing field of genetics, where it is important to solve as much as we can as soon as we can.

References

[1] H.R. Andersen. *An Introduction to Binary Decision Diagrams*. Technical University of Denmark, 1997.

[2] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, November 1986.

[3] N. Eén and N. Sorensson. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing*, pages 502–518, 2003.

[4] S.Gretarsdottir et al. Localization of a susceptibility gene for common forms of stroke to 5q12. *Am J Hum Genet*, 70:593–603, 2003.

[5] D. Gusfield. Haplotyping by pure parsimony. In *Proceedings of the 2003 Combinatorial Pattern Matching Conference*, pages 144–155, 2003.

[6] B. V. Halldórsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph, and S. Istrail. A survey of computational methods for determining haplotypes. In *Computational Methods for SNPs and Haplotype Inference (LNCS 2983)*, pages 26–47, 2004.

[7] D. L. Hartl and A. G. Clark. *Principles of Population Genetics*. Sinauer Associates, 1997.

[8] I.Lynce and J. Marques-Silva. Efficient haplotype inference with boolean satisfiability. *American Association for Artificial Intelligence*, 2006.

[9] A. Ingólfsdóttir and D. Gudbjartsson. Genetic linkage analysis algorithms and their implementation. *Transactions on Computational System Biology (TCSB)*, 12(III):123–144, 200.

[10] International HapMap Consortium. The international HapMap project. *Nature*, 426(6968):789–96, 2003.

[11] K. Kalpakis and P. Namjoshi. Haplotype phasing using semidefinite programming. In *Proceedings of BIBE*, pages 145–152, 2005.

[12] G. Lancia, M. C. Pinotti, and R. Rizzi. Haplotyping populations by pure parsimony. Complexity, exact and approximation algorithms. *INFORMS Journal on Computing*, 2004.

[13] S. Kelk L.van Iersel, J. Keijsper and L. Stougie. Shorelines of islands of tractability: Algorithms for parsimony and minimum perfect phylogeny haplotyping problems. 2006.

[14] N. Smith M. Stephens and P. Donnelly. A new statistical method for haplotype reconstruction from population data. *Am. J Human Genetics*, 68:978–989, 2001.

[15] M.Davis and H.Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[16] B.V. Halldórsson R. Sharan and S. Istrail. Islands of Tractability for Parsimony Haplotyping. *IEEE/ACM Transactions on Computational Biology and Bioinformatics 3(3)*, pages 303–311, 2006.

[17] K. H. Rosen. *Discrete Mathematics and Its Application*. MC Graw Hill, 2003.

REYKJAVÍK UNIVERSITY

HÁSKÓLINN Í REYKJAVÍK

Department of Computer Science
Reykjavík University
Ofanleiti 2, IS-103 Reykjavík, Iceland
Tel: +354 599 6200
Fax: +354 599 6201
http://www.ru.is