LEBANESE AMERICAN UNIVERSITY

HAPLOTYPING INFERANCE BY PURE-PARSIMONY USING REVAMPED DELAYED HAPLOTYPING SELECTION

By

Raymond Abdallah

A thesis
Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

School of Arts and Sciences
May 2011
THESIS APPROVAL FORM

Student Name: Raymond Abdallah
Thesis Title: HAPLOTYPE INFERENCE BY PURE-PARSIMONY USING REVAMPED DELAYED HAPLOTYPE SELECTION
Program: MS in Computer Science
Department: Computer Science and Mathematics
School: Arts and Sciences, Byblos

We the undersigned hereby certify that we have read and recommend to the School of Arts and Sciences, Byblos for acceptance a thesis entitled “HAPLOTYPE INFERENCE BY PURE-PARSIMONY USING REVAMPED DELAYED HAPLOTYPE SELECTION” by Raymond Abdallah in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Supervisor:  Dr. Daffielle Azar
Readers: Dr. Haidar Harmanani

Dated: May 30, 2011
THESIS /PROJECT COPYRIGHT RELEASE FORM

LEBANESE AMERICAN UNIVERSITY

By signing and submitting this license, I (the author(s) or copyright owner) grant the Lebanese American University (LAU) the non-exclusive right to reproduce, translate (as defined below), and/or distribute my submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video. I agree that LAU may, without changing the content, translate the submission to any medium or format for the purpose of preservation. I also agree that LAU may keep more than one copy of this submission for purposes of security, backup and preservation. I represent that the submission is my original work, and that I have the right to grant the rights contained in this license. I also represent that my submission does not, to the best of my knowledge, infringe upon anyone’s copyright. If the submission contains material for which I do not hold copyright, I represent that I have obtained the unrestricted permission of the copyright owner to grant LAU the rights required by this license, and that such third-party owned material is clearly identified and acknowledged within the text or content of the submission. IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN LAU, I REPRESENT THAT I HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT. LAU will clearly identify my name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to my submission.

Name: Raymond Abdallah

Signature: [Signature]

Date: May 30, 2011
PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that I have read and understood LAU’s Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me. This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Raymond Abdallah

Signature

Date: May 30, 2011
ACKNOWLEDGMENTS

First and foremost, I would like to thank Dr. Danielle Azar who has been a patient and understanding mentor. She has been the primary and sole supporter of my work. She has helped me refine my work to become as perfect as possible. I admire her trust in me and hope my work becomes another badge of victory amongst all her other accomplishments in life.

I would also like to thank the LAU Computer Science Department and Faculty for the opportunities they have given me, and the experience I gained through the Master’s program.

Last but not least, I would like to thank my family and friends who have been a major source of support for me during the writing of this thesis.
Haplotype Inference by Pure-Parsimony using Revamped Delayed Haplotype Selection

Raymond Abdallah

Abstract

In recent years, there has been a worldwide initiative to gather as much information as possible about the human genome, resulting in the Human Genome Project (HGP). The HGP project was founded on the basis of gathering genetic information to be used in various bioinformatics areas. The Human Genome Project’s main purpose is to find the common ancestry among various peoples around the globe in order to identify origins and gene-related diseases. Haplotype Inference (HI) is one of the problems tackled in the HGP, whereby from a given population of genotypes the goal is to find the minimum number of haplotypes from which the genotypes could have derived. Clark’s Algorithm is the first known algorithm to deal with this problem from a Computer Scientist’s perspective. It has been the basis for many other algorithms afterwards. One such algorithm is the Delayed Haplotype Selection (DS). Our work is an improvement of the DS. We call the resulting algorithm Revamped Haplotype Selection (RDS) algorithm. We test our algorithm on real and simulated data, and compare it to the DS algorithm and a Branch-and-Bound approach (known as HAPAR). Results prove that our algorithm significantly outperforms both in the quality of the solution as well as in running time.

Keywords: Haplotype, Branch and Bound, Delayed Haplotype Selection, Pure Parsimony, Haplotype Inference.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction and Biological Background</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Problem Statement</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>Haplotype Inference</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Pure-Parsimony Haplotyping</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Related Work</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>The Delayed Haplotype Selection Algorithm</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>The Branch-And-Bound Algorithm</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>Revamped Delayed Haplotype Selection</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>Experimental Results</td>
<td>25</td>
</tr>
<tr>
<td>7.1</td>
<td>Experiment on human β2-adrenergic receptor gene</td>
<td>25</td>
</tr>
<tr>
<td>7.2</td>
<td>Experiments on simulated data sets</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion and Future Work</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>34</td>
</tr>
</tbody>
</table>
Gregor Mendel discovered what we call today the “gene” [1]. Before Mendel’s experiments everyone noticed that the offspring always inherited features (known as traits) from their parents, but never understood how. When experimenting with pea plants, Mendel noted that traits from parents were passed on to generations according to a certain ratio. He also noticed that each plant trait was formed by “two factors”, one from each of the parents. These factors are found on the pairs of chromosomes of each living being. The terminologies used by Mendel have changed since his time. “Genes” is the new name for his “factors” and each of its possible forms is called an “allele”. Organisms which have two copies of the same allele are said to be homozygous. For example, if both parents pass onto a child an allele encoding brown hair then the child would have a homozygous site for hair color. Organisms that contain copies of two different alleles are said to be heterozygous. For example, if a parent passes an allele for brown eyes and the other passes on an allele for blue eyes, the child would have a heterozygous site for eye color. When an allele is in its most frequent or normal state, it is called wild type. Otherwise, it is called mutant. The former allele is the original unhindered form of a gene that has not been affected by evolution. Mutant alleles, such as blue eyes, are always mutations of a natural wild type allele, such as brown eyes. Figure 1.1 illustrates the above mentioned terms.
In recent years, there has been a worldwide initiative to gather as much information as possible about the human genome, resulting in the Human Genome Project (HGP). The HGP project was founded on the basis of gathering genetic information to be used in various bioinformatics areas. The Human Genome Project’s main purpose is to find the common ancestry among various peoples around the globe in order to identify origins and gene-related diseases. A genetic marker is a specific gene that produces a recognizable trait and can be used in family or population studies. The gathered markers are studied according to a collection of alleles found on a single strand of chromosomes out of the two distinct copies of chromosomes found in diploid organisms (such as humans) [2]. A diploid is a cell that contains a pair of chromosomes unique to its species [1]. This collection of alleles, found in such organisms, is called a haplotype (Haploid Genotype) [3]. Haplotypes can be studied at the level of a whole DNA sequence of a chromosome, or at the level of Single Nucleotide Polymorphisms.
(SNP’s). SNP is the more popular approach to studying Haplotype Inference (HI) problems [2].

The rest of this thesis is organized as follows. In Chapter 2, we state the problem. In Chapter 3, we explain different methodologies and algorithms that were used to solve the stated problem. In Chapter 4, we explain the Delayed Haplotype Selection algorithm which we have based our algorithm on. In Chapter 5, we present the Branch-and-Bound algorithm which was implemented and used to compare our new algorithm with. In Chapter 6, we show the Revamped Delayed Haplotype Selection algorithm that is our mutated version of the Delayed Haplotype Selection algorithm. In chapter 7 we present the results of experiments conducted using the mentioned algorithms, and show that our new algorithm outbeats the other two.
Chapter 2

PROBLEM STATEMENT

2.1 Haplotype Inference

The Haplotype Inference Problem can be seen as a computational problem consisting of \( n \) genotype vectors \((g_1, g_2, \ldots, g_n)\), all of equal length. A genotype vector \( g = \langle a_1, a_2, \ldots, a_k \rangle \) encodes the genetic marker (DNA sequence) of a living being. Each location \( a_k \) in the vector can take one of the three values 0, 1 or 2. A value of 0 indicates a homozygous wild type, such as a pair of “brown eyes” alleles. A value of 1 indicates a homozygous mutant, such as a pair of “blue eyes” alleles. A value of 2 indicates a heterozygous site, such as a “brown eyes” allele paired with a “blue eyes” allele. Given \( n \) genotype vectors, we can infer their haplotype origins by mapping each genotype vector \( g \) to a pair of distinct vectors \( v_1 \) and \( v_2 \) which only contain one of two states, 0 or 1. A haplotype vector explains a genotype vector when it is a possible parent. A haplotype refers to “half of a genotype” and is the set of alleles on a single chromosome.

<table>
<thead>
<tr>
<th>( v_1 )</th>
<th>( v_2 )</th>
<th>( g )</th>
<th>Genotype State Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Homozygous Wild Type</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Homozygous Mutant</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>Heterozygous</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>Heterozygous</td>
</tr>
</tbody>
</table>

Table 2.1: \( v_1 \) and \( v_2 \) are haplotype parents. \( g \) is the resulting genotype.

4
For every 0 (or 1) in $g$, the vectors $v_1$ and $v_2$ will both explicitly have a 0 (or 1) [2]. When the genotype vector has a state 2, this is reflected by one of the vectors having a value of 0, while the other one has a value 1. Table 2.1 illustrates this.

Thus, we are interested in the heterozygous sites where if we have $h$ heterozygous sites, we would have $2^{h-1}$ possibilities for haplotype pairs explaining the $n$ genotype vectors [2]. An example is shown in Table 2.2 where 2 heterozygous sites in the genotype vector $g$ result in 4 possible haplotype pairs indicated by $v_1$ and $v_2$. Note that the last 2 pairs encode vectors that have already been generated. Hence only 2 ($2^{2-1}$) unique haplotype pairs are enough to explain the genotype vector $g$.

<table>
<thead>
<tr>
<th>Genotype Vector</th>
<th>$g$</th>
<th>01212</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Haplotype Pair</td>
<td>$v_1$</td>
<td>01111</td>
</tr>
<tr>
<td></td>
<td>$v_2$</td>
<td>01010</td>
</tr>
<tr>
<td>2nd Haplotype Pair</td>
<td>$v_1$</td>
<td>01110</td>
</tr>
<tr>
<td></td>
<td>$v_2$</td>
<td>01011</td>
</tr>
<tr>
<td>3rd Haplotype Pair</td>
<td>$v_1$</td>
<td>01011</td>
</tr>
<tr>
<td></td>
<td>$v_2$</td>
<td>01110</td>
</tr>
<tr>
<td>4th Haplotype Pair</td>
<td>$v_1$</td>
<td>01010</td>
</tr>
<tr>
<td></td>
<td>$v_2$</td>
<td>01111</td>
</tr>
</tbody>
</table>

Table 2.2: Example of Haplotype Inference. Each block shows the pair of vectors $v_1$ and $v_2$ from which $g$ can be inferred.

2.2 Pure-Parsimony Haplotyping

Pure-Parsimony is one of the well known methods for solving the HI problem. Its objective is to find the smallest number of “distinct” haplotypes to represent all the given genotypes [2]. This is one of the simplest and oldest reliable and efficient methods found for the HI problem. The method uses “known” haplotypes to find new haplotypes for unexplained genotypes. Using the known haplotype parents, the method tries to explain the genotypes by finding a compliment haplotype to form the genotype. For example, given the genotypes 1101011, 1102012, and 1220010 we can infer one haplotype 1101011, since it is the only possible parent for the first
genotype given. In this example we use 1101011 as a parent for the second genotype 1102012, which results in a new known haplotype 1100010. The last genotype 1220010 cannot be explained by 1101011, thus we try to explain it using the second known haplotype 1100010 and results in the new known haplotype 1010010. This concludes with only 3 distinct haplotypes (1101011, 1100010 and 1010010) instead of 5 independently generated haplotypes (1101011, 1100011, 1101010, 1000010 and 1110010). The relationship between the parents and children is shown in Figure 2.1.

Figure 2.1: Example of Pure-Parsimony Haplotyping. In this case 3 haplotypes are enough to explain 3 genotypes.

The Pure Parsimony Haplotyping (PPH) problem has been found to be solvable in polynomial time in the case where the genotype has at most 2 heterozygous sites. However, it has been found to be $NP$-hard when the genotype has 3 or more heterozygous sites [4].
Chapter 3

RELATED WORK

There are many algorithms used for solving the HI problem, some being specifically designed for the problem such as Clark’s Algorithm [5] and the Expectation-Maximization (EM) algorithm [6], while others are based on general algorithms such as exact methods (Branch-and-Bound [7]) and meta-heuristics [8]. Clark’s Algorithm was the first to be presented as a means to finding a solution for the HI problem. This algorithm is based on assigning the smallest number of haplotypes to the genotypes under study by constant updates based on unambiguous haplotypes from given data. We will give the details of this algorithm in Chapter 4 since our work is based on it. Silva et al. [9] implemented their Delayed Haplotype Selection algorithm based on Clark’s Algorithm. The results were compared with Clark’s Algorithm and were found to perform better in most cases. The EM Algorithm starts with an initial estimate of weights for haplotypes, depending on the number of genotypes each can explain. The weights are then adjusted depending on the genotype data given, giving more importance to certain haplotypes as the algorithm runs on the given data sets. Zhao et al. [6] implemented their version of the EM Algorithm with results being quite satisfactory on multiple data sets using different parameters. More generic algorithms have been used to tackle the problem, such as exact methods, particularly Branch-and-Bound [7]. The latter searches through the whole solution space to find
the best results. We explain this algorithm in detail in Chapter 5. Wang and Xu implemented the Branch-and-Bound algorithm, called HAPAR, and found it to generate decent results in comparison with other algorithms [7]. A meta-heuristic approach was implemented for the HI problem using genetic algorithms (GA) [10]. Wang et al. [10] implemented GAHAP using the GA approach and compared their results with HAPAR showing the best solution (or close to best) in places where HAPAR failed to return a solution. Benedettini et al. [8] present an Ant Colony Optimization algorithm to solve the HI problem and the results were quite satisfactory. In our work we present an improved version of the Delayed Haplotype Selection (DS) algorithm [9] that is based on Clark’s Algorithm. We explain the algorithm in detail in Chapter 6. Results show that the modifications we made to the original DS algorithm were good. As a matter of fact our algorithm outbeats DS in all the cases and HAPAR in most cases.
Chapter 4

THE DELAYED HAPLOTYPING SELECTION

ALGORITHM

Clark’s algorithm is one of the most popular algorithms designed to find solutions to the delayed haplotype inference problem. However, the algorithm relies heavily on the greedy technique. It searches for a solution for a genotype by choosing the most recently chosen haplotype only. In other words, a haplotype set $h_1$ is given as a probable solution for genotype $g$, being generated based on genotypes with at most 1 heterozygous site. This creates a new set $h_2$. $h_2$ will not necessarily contain an optimal solution for the next genotype in the series. However, only $h_2$ will be used to try to explain some unexplained genotypes. It relies only on the haplotypes generated from homozygous genotypes or genotypes with one heterozygous sites, and would fail in reaching a result when neither kind of genotype is available. Figure 4.1 illustrates this and Algorithm 1 shows the pseudocode [5].

Delayed Haplotype Selection (DS) works on the basis that the greedy part of Clark’s algorithm be avoided, and thus the selection process becomes delayed by using candidate haplotypes [9]. Candidate haplotypes explain one or more genotypes which have not yet been explained by selected haplotypes. The primary purpose of DS is to obtain an upper-bound on the size of the Pure-Parsimony Haplotyping (PPH) solution. This upper-bound, when small, helps other algorithms, such as
Algorithm 1 Delayed Haplotype Selection [9]

1: ClarkAlgo($G$)
2: $H$ is the set of haplotypes
3: $H \leftarrow$ CalcInitialHaplotypes($G$)
4: $G \leftarrow$ RemoveExplainedGenotypes($G, H$)
5: while $G \neq \emptyset$ do
6: $h \leftarrow h \in H$ that explains a genotype in $G$
7: $G \leftarrow$ RemoveExplainedGenotypes($G, H$)
8: for $g \in G$ do
9: if CanExplain($h, g$) then
10: $H \leftarrow$ CalcExplainPair($h, g$)
11: Associate $h$ with $g$
12: end if
13: end for
14: end while

Figure 4.1: An Illustration of how Clark’s Algorithm works.

Branch-and-Bound, in finding the optimal solution in a smaller amount of time.

The DS algorithm works with two sets of haplotypes, the first is known as selected haplotypes and the second as candidate haplotypes. The former set is used to hold all haplotypes which were chosen by the algorithm as parent haplotypes to the genotypes given. Whereas the latter set holds haplotypes which are possible parents to the genotypes that do not have parent haplotypes in the selected haplotypes set. These genotypes are known as unexplained genotypes [9]. The initial set of selected haplotypes is formed of genotypes with at most one heterozygous site. These mentioned genotypes have a maximum of two parent haplotypes. Since these haplotypes
Algorithm 2 Delayed Haplotype Selection [9]

1: DelayedHaplotypeSelection$(G)$
2: $H_S$ is the set of selected haplotypes; $H_C$ is the set of candidate haplotypes
3: $H_S \leftarrow$ CalcInitialHaplotypes$(G)$
4: $G \leftarrow$ RemoveExplainedGenotypes$(G, H_S)$
5: for $h \in H_S$ do
6:  for $g \in G$ do
7:   if CanExplain$(h, g)$ then
8:    $h_c \leftarrow$ CalcExplainPair$(h, g)$
9:    $H_C \leftarrow H_C \cup h_c$
10:   Associate $h_c$ with $g$
11: end if
12: end for
13: end for
14: while $G \neq \emptyset$ do
15:  if $H_C = \emptyset$ then
16:   $h_c \leftarrow$ PickCandHaplotype$(G)$
17:   $H_C \leftarrow h_c$
18: end if
19: $h \leftarrow h_c \in H_C$ associated with largest number of genotypes
20: $H_C \leftarrow H_C - h$
21: $H_S \leftarrow H_S \cup h$
22: $G \leftarrow$ RemoveExplainedGenotypes$(G, H_S)$
23: for $g \in G$ do
24:  if CanExplain$(h, g)$ then
25:   $h_c \leftarrow$ CalcExplainPair$(h, g)$
26:   $H_C \leftarrow H_C \cup h_c$
27:   Associate $h_c$ with $g$
28: end if
29: end for
30: end while
31: $H_S \leftarrow$ RemoveNonUsedHaplotypes$(H_S)$
32: return $H_S$

are unique parents of these genotypes, they are evidently going to be a part of the final solution. This initial set is then used to generate a candidate set by finding pairs to the haplotypes of the initial set for the unexplained genotypes.

If $h_s$ is a possible parent for an unexplained genotype $g$, a candidate haplotypes $h_c$ is generated as the compliment parent to $h_s$ and is added to the candidate haplotypes set [9]. The algorithm then iterates through the candidate list of haplotypes and chooses the one that explains the largest number of unexplained genotypes. The chosen haplotype $h_c$ is then added to the set of selected haplotypes and the genotypes which are explained by $h_c$ are removed from the list of unexplained genotypes. Then, the newly
selected haplotype is used to generate new candidate haplotypes in the same manner as mentioned in the previous paragraph [9]. This algorithm is illustrated in Figure 4.2.

The algorithm terminates when there are no more entities in the unexplained genotypes set. The set of selected genotypes is then cleaned from all haplotypes which do not explain any genotypes. Given $n$ genotypes each of length $m$, the running time for this algorithm has been found to be $O(n^2m)$ [9]. The pseudocode is shown in Algorithm 2.

To illustrate this, consider the following four genotypes: 101100, 201102, 112201 and 122202. From these four genotypes we form an initial set of selected haplotypes formed of 101100. This haplotype when paired with itself forms the genotype 101100, and sends it to the set of explained genotypes. Then the haplotype 101100 tries to find pairs to explain the unexplained genotypes (201102, 112201, 122202), yielding the two haplotypes 001101 (from 201102) and 110001 (from 122202). The two generated haplotypes are now considered candidates. 001101 can explain one unexplained genotype (201102). Whereas 110001 can explain 2 unexplained genotypes (112201 and 122202).
and 122202). Thus, the latter haplotype (110001) is added to the set of selected haplotypes. The current set of selected haplotypes (101100 and 110001) can explain the unexplained genotype 122202. This genotype (122202) is thus added to the set of explained genotypes. The last selected haplotype (110001) is used to try and explain the remaining unexplained genotypes (201102 and 112201), yielding the haplotype 111101 (from 112201) which is added to the set of candidate haplotypes. Each of the candidate haplotypes (111101 and 001101) can explain only one genotype. In this case, any of the two haplotypes is chosen and added to the set of selected haplotypes. The selected haplotypes 001101 and 101100 form a pair to explain the unexplained genotype 201102, which is thus added to the list of explained genotypes. The last
generated *selected* haplotype (001101) is used to try and explain the *unexplained* genotypes, and results in no new *candidate* haplotypes. The remaining *candidate* haplotype 111101 is then added to the set of *selected* haplotypes. The pair of *selected* haplotypes 111101 and 110001 explain the *unexplained* genotype 112201, which is thus added to the set of *explained* genotypes. This last step renders the set of *unexplained* genotypes empty, thus ending the algorithm. Figure 4.3 illustrates this example.

Figure 4.4 illustrates an example with only 3 given genotypes (201102, 112201 and 122202). The difference between the two examples is that there is no initial set of haplotypes, which forces the algorithm to generate all possible haplotypes and choosing the one that explains the most genotypes. In this case 101100 is chosen as a *candidate* haplotype since it explains the most genotypes. This candidate haplotype then becomes a *selected* haplotype. The algorithm then runs just like the previous example.
Figure 4.4: An Illustration of how DS works on an harder example where there are no non-ambiguous genotypes.
Chapter 5

THE BRANCH-AND-BOUND ALGORITHM

Wang and Xu propose an exact algorithm for the Haplotype Inference Problem by Pure Parsimony [7]. The algorithm consists of two parts and is based on Branch-and-Bound. Since any Branch-and-Bound algorithm is in need of a small initial solution, the authors propose a greedy algorithm for the generation of such a solution.

The proposed greedy algorithm searches for two criteria: The “coverage of a haplotype” which is the number of genotypes a haplotype can explain and the “coverage of a resolution” which is the sum of the coverages of the two haplotypes which explain a certain genotype. The greedy algorithm chooses from each genotype the haplotype parents with the greatest coverage [7].

The branch-and-bound algorithm uses a Matrix $M$ where each row represents a genotype as shown below:

$$M = \begin{pmatrix}
a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\
a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m,1} & a_{m,2} & \cdots & a_{m,n}
\end{pmatrix}$$

The pseudocode is shown in Algorithm 3. This algorithm searches through all viable solutions and selects the one with the least number of haplotypes [7]. When a partial
solution has a larger number of haplotypes than the current bound (the initial solution formed by the greedy algorithm), it is dropped. This is repeated until the best solution (smallest number of haplotypes) is found in the search space, which is all possible solutions since this is an exact algorithm. In theory, the running time of the algorithm is exponential with respect to the input size. If the input has \( k \) SNP sites, then the search space consists of \( 2^k \) possible haplotypes. Each SNP site is equivalent to an allele of value 0, 1 or 2. Wang et al. suggest to reduce the size of resolution lists thus reducing the running time [7]. When two resolutions are equally good, one is chosen randomly and kept, while the other is thrown out since there can be only one optimal solution only. There are two cases to consider that would generate a conflict with regards to choosing haplotype resolutions:

Case 1: a genotype \( m_i \) has two resolutions, each of coverage size 2. In this case, the 2 pairs of haplotypes are unique to the genotype given, and thus only one pair is kept [7].

Case 2: Consider two genotypes \( m_i \) and \( m_j \). Suppose \( m_i \) has two resolutions \((h_1, h_2)\) and \((h_4, h_5)\) and \( m_j \) has two resolutions \((h_2, h_3)\) and \((h_5, h_6)\). If \( h_1, h_3, h_4 \) and \( h_6 \) have coverage 1, and \( h_2 \) and \( h_5 \) have coverage 2, then only the combination \((h_1, h_2)\) and \((h_2, h_3)\) is kept and the combination \((h_4, h_5)\) and \((h_5, h_6)\) is ignored. This is done by randomly choosing a haplotype among the ones with the highest coverage and keeping the resolutions which contain this haplotype only.

To illustrate, consider the following 6 haplotypes: \( h_1 = 0101, h_2 = 1001, h_3 = 1111, h_4 = 0001, h_5 = 1101, h_6 = 1011, m_1 = 2201 \) and \( m_2 = 1221 \). There are two other given genotypes \( m_3 = 2220 \) and \( m_4 = 1020 \) that have nothing to do with the six haplotypes. In this case, only \((h_1, h_2)\) is placed into Array(1) and \((h_2, h_3)\) into Array(2). \((h_4, h_5)\) is ignored in Array(1) and \((h_5, h_6)\) in Array(2). This is illustrated in Figure 5.1.

This improvement proved to significantly lower the number of haplotypes appearing in the resolution arrays [7]. For example, given the Angiotensin Converting Enzyme (ACE) data containing 52 SNP sites, the software uses 483 candidate haplotypes instead of the \( 2^{52} \) possible haplotypes.
The running time for the algorithm\textsuperscript{1} is fairly efficient since it computes the ACE data in 2.25 minutes while PHASE reaches a solution in 12 minutes.

\textsuperscript{1}The implementation was (called HAPAR) written in C++ and is available upon request [7].
Algorithm 3 The Branch-and-Bound Algorithm [7]

**Require:** Input: a genotype matrix $M$ containing $n$ rows
1. List all possible resolutions for each genotype. Let Array(1), Array(2), ..., Array(n) be the arrays of resolutions for the $n$ rows in $M$. Denote $s_i$ to be the length of Array($i$). Let $S$ be a set of resolutions. ($S$ is the number of distinct haplotypes in $S$).
2. Use the greedy algorithm to get a solution and set $f^*(S)$ to be the size of the solution.
3. Search for the optimal solution as follows:
   4. for $j_1 = 1$ to $s_1$ do
   5.     $S = \text{Array}(1)[j_1]$
   6.     if ($f(S) > f^*(S)$) then
   7.         try next $j_1$
   8.     end if
   9. for $j_2 = 1$ to $s_2$ do
   10:    $S = \text{Array}(1)[j_1], \text{Array}(2)[j_2]$
   11:    if ($f(S) > f^*(S)$) then
   12:        try next $j_2$
   13:    end if
   14:    ... 
   15: for $j_n = 1$ to $s_n$ do
   16:     $S = \text{Array}(1)[j_1], \text{Array}(2)[j_2], ..., \text{Array}(n)[j_n]$
   17:     if ($f(S) > f^*(S)$) then
   18:         $f^*(S) = f(S)$
   19:     end if
   20: end for
   21: end for
Chapter 6

REVAMPED DELAYED HAPLOTYPET

SELECTION

In our work, we propose two main changes to the basic DS algorithm. We call our algorithm the Revamped Delayed Haplotype Selection (RDS) and the pseudocode is shown in Algorithm 4, with all changes to the original algorithm marked with a “*”. The DS algorithm does not consider that duplicate haplotypes might be part of the final solution. It also does not take into consideration the complexity of finding the right haplotype parents.

The first change is made to the generation of new candidate haplotypes when the set is empty. The DS algorithm deals with this by generating all possible haplotypes and selecting the haplotype that explains the largest number of unexplained genotypes. In RDS, we handle this step by selecting the genotype with the largest amount of heterozygous sites. This genotype is chosen because it has the largest number of haplotypes that can explain it (and probably other genotypes). There are $2^k$ haplotypes that can explain a genotype with $k$ heterozygous sites. All of them are added to the list of candidate haplotypes. This is shown in Table 6.1, where a genotype with 3 heterozygous sites can be explained by 8 ($2^3$) haplotypes. The candidate haplotype that can explain the most genotypes is then chosen and moved to the set of selected haplotypes. The candidate set is then emptied. New candidate haplotypes are formed.
Algorithm 4 Revamped Delayed Haplotype Selection

1: RevampedDelayedHaplotypeSelection($G$)
2: $H_S$ is the set of selected haplotypes; $H_C$ is the set of candidate haplotypes
3: $H_S \leftarrow$ CalcInitialHaplotypes($G$)
4: $G \leftarrow$ RemoveExplainedGenotypes($G, H_S$)
5: for $h \in H_S$ do
6:   for $g \in G$ do
7:     if CanExplain($h, g$) then
8:       $h_c \leftarrow$ CalcExplainPair($h, g$)
9:       $H_C \leftarrow H_C \cup h_c$
10:       Associate $h_c$ with $g$
11:   end if
12: end for
13: end for
14: while $G \neq \emptyset$ do
15:   if $H_C = \emptyset$ then
16:     $H_C \leftarrow$ GenerateHaplotypesFromGenotype($G$) *
17:   end if
18:   $h \leftarrow h_c \in H_C$ associated with largest number of genotypes
19:   $H_C \leftarrow H_C - h$
20:   $H_S \leftarrow H_S \cup h$
21: if $H_C \leftarrow$ GenerateHaplotypesFromGenotype($G$) then
22:   $H_C \leftarrow \emptyset$ *
23: end if
24: $G \leftarrow$ RemoveExplainedGenotypes($G, H_S$)
25: for $h \in H_S$ * do
26:   for $g \in G$ do
27:     if CanExplain($h, g$) then
28:       $h_c \leftarrow$ CalcExplainPair($h, g$)
29:       $H_C \leftarrow H_C \cup h_c$
30:       Associate $h_c$ with $g$
31:     end if
32: end for *
33: end for
34: end while
35: $H_S \leftarrow$ RemoveDuplicateHaplotypes($H_S$) *
36: return $H_S$

using the newly selected haplotype. The complexity of this algorithm is $O(n^2m)$.

The second change is made to the generation of candidate haplotypes from selected haplotypes. In DS only, the last selected haplotype is used in generating candidate haplotypes (as opposed to RDS where the whole set of selected haplotypes is used to generate candidate haplotypes). This gives a better chance of finding possible combinations, instead of being limited by the last selected haplotype.
Table 6.1: Example of possible haplotype parents generated for a genotype with 3 heterozygous sites.

<table>
<thead>
<tr>
<th>10000</th>
<th>10001</th>
<th>10100</th>
<th>10101</th>
<th>11000</th>
<th>11001</th>
<th>11100</th>
<th>11101</th>
</tr>
</thead>
</table>

Figure 6.1: An Illustration of how RDS works on an easy example where there is a non-ambiguous genotype.
To illustrate the changes, consider the following four genotypes (same as in the DS example): 101100, 201102, 112201 and 122202. From these four genotypes we form an initial set of selected haplotypes formed of 101100. This haplotype when paired with itself forms the genotype 101100, and sends it to the set of explained genotypes. Then the haplotype 101100 tries to find pairs to explain the unexplained genotypes (201102, 112201, 122202), yielding the two haplotypes 001101 (from 201102) and 110001 (from 122202). The two generated haplotypes are now considered candidates. 001101 can explain one unexplained genotype (201102). Whereas 110001 can explain 2 unexplained genotypes (112201 and 122202). Thus, the latter haplotype (110001) is added to the set of selected haplotypes. The current set of selected haplotypes (101100 and 110001) can explain the unexplained genotype 122202. This genotype (122202) is thus added to the set of explained genotypes. The selected haplotype set is used to try and explain the remaining unexplained genotypes (201102 and 112201), yielding the haplotype 111101 (from 112201) which is added to the set of candidate haplotypes. Each of the candidate haplotypes (111101 and 001101) can explain only one genotype. In this case, any of the two haplotypes is chosen and added to the set of selected haplotypes. The selected haplotypes 001101 and 101100 form a pair to explain the unexplained genotype 201102, which is thus added to the list of explained genotypes. The selected haplotype set is then used to try and explain the unexplained genotypes, and results in no new candidate haplotypes. The remaining candidate haplotype 111101 is then added to the set of selected haplotypes. The pair of selected haplotypes 111101 and 110001 explain the unexplained genotype 112201, which is thus added to the set of explained genotypes. This last step renders the set of unexplained genotypes empty, thus ending the algorithm. Figure 6.1 illustrates this example.

Figure 6.2 illustrates an example with only 3 given genotypes (201102, 112201 and 122202). The difference between the two examples is that there is no initial set of haplotypes, which forces the algorithm to generate all possible haplotypes for the genotype with most heterozygous sites (122202), and choosing the one that explains the most genotypes. This is different than DS, in that it generates one set instead
of all sets for all unexplained genotypes. In this case 101100 is chosen as a candidate haplotype since it explains the most genotypes. This candidate haplotype then becomes a selected haplotype. The algorithm then runs just like the previous example.

Figure 6.2: An Illustration of how RDS works on an easy example where there are no non-ambiguous genotypes.
Chapter 7

EXPERIMENTAL RESULTS

We have tested the algorithm against DS and HAPAR on multiple datasets, which include real and simulated data sets. We implemented RDS in Java and ran it on a 2.66G Hz Intel i7 processor with 4 GB DDR3 RAM, using Netbeans 6.9.1 on Mac OS X 10.6.6.

In order to account for the element of randomness in the algorithms, we ran DS and RDS 30 times. We report the average and the standard deviation. We evaluate all three algorithms by three criteria: The \textit{hapNum} which is the number of haplotypes that an algorithm returns, the \textit{errorRate} which is the difference between \textit{hapNum} and \textit{hapMin} (best known possible solution) with respect to \textit{hapMin}, and the \textit{runTime} which is the time (in milliseconds) it took for the algorithm to generate a solution.

7.1 Experiment on human $\beta_2$-adrenergic receptor gene

$\beta_2$-Adrenergic receptors ($\beta_2$ARs) are protein receptors that moderate the actions of certain hormones, known as catecholamines [7]. From a population size of 121 persons, 18 unique genotypes were identified. Ten haplotypes were found to be enough to explain all the genotypes [7]. Hence, the number of haplotypes is considerably less than the number of genotypes. When running the algorithms on this data set\footnote{This data set can be found on http://www.cs.cityu.edu.hk/~lwang/hapar/}, the
best known possible solution was found by both HAPAR and RDS. Table 7.1, figures 7.1 and 7.2 show the results.

<table>
<thead>
<tr>
<th>Algo</th>
<th>hapNum</th>
<th>hapMin</th>
<th>errorRate</th>
<th>runTime (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDS</td>
<td>10(0)</td>
<td>10</td>
<td>0.0</td>
<td>106.2(70.83)</td>
</tr>
<tr>
<td>DS</td>
<td>15.43(0.5)</td>
<td>10</td>
<td>0.5</td>
<td>1830.2(611.77)</td>
</tr>
<tr>
<td>HAPAR</td>
<td>10(0)</td>
<td>10</td>
<td>0.0</td>
<td>466.0(0)</td>
</tr>
</tbody>
</table>

Table 7.1: Comparison results of DS, RDS and HAPAR on β2AR. *hapNum* is the number of haplotypes that an algorithm returns. *hapMin* indicates the best known solution. *errorRate* is the difference between *hapNum* and *hapMin* with respect to *hapMin*. The *runTime* which is the time (in milliseconds) it took for the algorithm to generate a solution.

![Figure 7.1](image.png)

**Figure 7.1:** A comparison of the running time of each algorithm in milliseconds.

Table 7.1 shows that RDS outbeats DS by 50% in the smallest number of possible haplotypes. As a matter of fact, RDS finds 10 haplotypes whereas DS finds 15.45. When comparing RDS with HAPAR we find that they both give the same optimal solution. However, RDS running time complexity is $O(n^2m)$ whereas HAPAR’s running time is $O(2^n)$. This is due to the greedy part of RDS, which efficiently finds the best possible solution at each iteration, instead of searching the whole possible search space as HAPAR does. In order to test the statistical significance of the results, we
performed the Mann-Whitney test to compare DS and RDS. This test was chosen in accordance with the nature of our problem. As a matter of fact, the test is used to compare two unpaired groups, whose results are not dependent on each other. It is also used for a non-gaussian or non-linear data. Our null hypothesis $H_0$ is that RDS does not significantly outbeat DS. The number of outputs for each algorithm is 30 ($n_{DS} = n_{RDS} = 30$), resulting in the standardized value $z$ ($z = 6.65$). This $z$ value is equivalent to $P < 0.0001$, which is less than 2.5%. This is considered a good result given the small size of our data point set. Thus our null hypothesis is invalid, and the results of the two algorithms differ significantly.

### 7.2 Experiments on simulated data sets

In order to test our algorithm on more than one data set we generated our own synthetic data sets. We randomly generate $n$ genotypes with $k$ SNPs. This is done by randomly combining two haplotypes among a set of haplotypes. Nine data sets are thus generated under the parameter settings $k = 20, n = 10, 11, 12, ..., 18$. This was inspired by the simulated data generated for testing GAHAP [10]. The results of these algorithms are listed in Table 7.2 where F denotes that an algorithm fails to
return a solution within two hours\(^2\).

<table>
<thead>
<tr>
<th>n</th>
<th>hapNum</th>
<th>hapMin</th>
<th>errorRate%</th>
<th>runTime(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RDS</td>
<td>DS</td>
<td>H</td>
<td>RDS</td>
</tr>
<tr>
<td>10</td>
<td>6(0)</td>
<td>6(0)</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>8(0)</td>
<td>10(0)</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>9(0)</td>
<td>10(0)</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>9(0)</td>
<td>10(0)</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>14</td>
<td>8(0)</td>
<td>12(0)</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>15</td>
<td>12(0)</td>
<td>15(0)</td>
<td>F</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>17.3(0.9)</td>
<td>21(0)</td>
<td>F</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>13(0)</td>
<td>16(0)</td>
<td>F</td>
<td>13</td>
</tr>
<tr>
<td>18</td>
<td>13(0)</td>
<td>16(0)</td>
<td>F</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 7.2: Results of RDS, DS and HAPAR(H) shown respectively on simulated data. F indicates that the algorithm failed to reach a solution within 2 hours running time.

Table 7.2 shows that RDS could find the minimum number of haplotypes in all but one case with an 8 percent error rate. RDS outbeats DS on almost every dataset. RDS also finds the same number of haplotypes as HAPAR in most cases, where HAPAR generates a solution within 2 hours running time. While inspecting running time of RDS in Table 7.2, we find that it is not proportional to \( n \). As a matter of fact, when \( n \) is 17, the running time is 71.83 ms whereas when \( n \) is 18, the running time is 41.23. This is due to the number of iterations needed to find haplotypes that explain the larger number of genotypes.

<table>
<thead>
<tr>
<th></th>
<th>hapNum</th>
<th>hapMin</th>
<th>errorRate%</th>
<th>runTime(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDS</td>
<td>avg</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>stdv</td>
<td>1.22</td>
<td>1.22</td>
<td>0</td>
</tr>
<tr>
<td>DS</td>
<td>avg</td>
<td>9.6</td>
<td>8</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>stdv</td>
<td>2.19</td>
<td>1.22</td>
<td>19</td>
</tr>
<tr>
<td>HAPAR</td>
<td>avg</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>stdv</td>
<td>1.22</td>
<td>1.22</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7.3: Average and standard deviation for each algorithm on the simulated data sets for \( n \leq 14 \).

Tables 7.3, 7.4 and Fig 7.3, 7.4 show the average and standard deviation of the results of each algorithm for \( n \leq 14 \) and \( n > 14 \) respectively. Table 7.3 demonstrates

\(^2\)Running on a 2.66G Hz Intel i7 processor with 4 GB DDR3 RAM, using Netbeans 6.9.1 on Mac OS X 10.6.6.
that RDS outbeat DS by approximately 1.6%. Since both algorithms have running time complexity $O(n^2m)$, we compare the time (in ms) and we can see that RDS has a much lower and stable running time. Table 7.3 also shows the significant running time difference between RDS and HAPAR, but this is expected due to the exponential time complexity of HAPAR. Figures 7.3b and 7.4b illustrate the stability of RDS running time with respect to DS. This is seen by the high standard deviation in the DS bars, whereby it exceeds 50% of the average running time in both graphs.

In order to test the statistical significance of the results, we performed the Mann-Whitney test on each of the data sets. Since we are testing on many different data sets, we have used the Bonferroni adjustment. Our null hypothesis $H_0$ is that RDS does not significantly outbeat DS. The number of outputs for each algorithm is 30 ($n_DS = n_RDS = 30$), resulting in the standardized value $z$ ($z = 6.65$). This $z$ value is equivalent to $P < 0.0001$, which is less than 0.25%. Thus our null hypothesis is invalid, and the results of the two algorithms differ significantly.

Table 7.4: Average and standard deviation for each algorithm on the simulated data sets for $n > 14$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>hapNum</th>
<th>hapMin</th>
<th>errorRate%</th>
<th>runTime(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDS</td>
<td>avg</td>
<td>13.83</td>
<td>13.5</td>
<td>47.15</td>
</tr>
<tr>
<td></td>
<td>stdv</td>
<td>2.38</td>
<td>1.73</td>
<td>16.65</td>
</tr>
<tr>
<td>DS</td>
<td>avg</td>
<td>17</td>
<td>13.5</td>
<td>137.05</td>
</tr>
<tr>
<td></td>
<td>stdv</td>
<td>2.71</td>
<td>1.73</td>
<td>72.74</td>
</tr>
<tr>
<td>HAPAR</td>
<td>avg</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>stdv</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Table 7.4: Average and standard deviation for each algorithm on the simulated data sets for $n > 14$.
Figure 7.3: (a) A comparison of average number of haplotypes generated between DS and RDS for $n \leq 14$. (b) A comparison of the running time between DS and RDS for $n \leq 14$. 
Figure 7.4: (a) A comparison of average number of haplotypes generated between DS and RDS for n > 14. (b) A comparison of the running time between DS and RDS for n > 14.
Chapter 8

CONCLUSION AND FUTURE WORK

The Haplotype Inference problem has been the focus of international projects, mainly the Human Genome Project. The motivation behind HGP and HI is to find the common ancestry of a given population. This problem started to shine with H.C. Clark in 1990 [5], when he suggested the first known algorithm to solve it, which was called after him, known as Clark’s Algorithm. With the problem gaining ground in the field of bioinformatics, many other algorithms emerged in the effort of finding the best possible solutions in the best possible time frame. Our efforts lay in upgrading a greedy algorithm based on Clark’s method and is known as Delayed Haplotype Selection (DS). These upgrades to the algorithm have yielded amazing results in comparison with the original algorithm and the exact method known as Branch-and-Bound. The DS algorithm is suggested to be an possible upper bound generator to an exact algorithm, such as Branch-and-Bound.

RDS is a greedy algorithm which has given great results on the data sets it was tested on. Although it generated the best results for most of the data sets it ran on, it does have its weakness. In the case where the initial set of candidate haplotypes is empty, RDS heavily relies on the genotype with the largest number of heterozygous site. This reliance, at the initial stage of the process, is a great weakness in generating candidate haplotypes which will be the base for the generation of other haplotypes. This initial generation of a haplotype set should be studied further, since the haplotype found
to be the best at that initial point may not be part of the optimal solution found by other algorithms. This slight error would lead to non-optimal solution generation, and with the greedy nature of the algorithm, this solution might be the only one it would find.

We have proven our algorithm to be extremely time efficient in generating optimal solutions (most of the time). RDS may be integrated into other algorithms which depend on some initial solution to compare its results with. This helps reduce running time for other types of algorithms, such as exact methods and meta-heuristics, which would have an effective comparison measure. Thus these algorithms can be pushed to generate the best possible solution by comparing it with a close-to-optimal solution generated in a minimum amount of time.
BIBLIOGRAPHY


