Bioinformatics Advance Access published July 7, 2005

BIOINFORMATICS

A Parsimonious Tree-Grow Method for Haplotype Inference

Zhenping Li [a,b], Wenfeng Zhou [a], Xiangsun Zhang [b]*, Luonan Chen [c]†

[a]Beijing Materials Institute, Beijing 101149, China; [b]Chinese Academy of Sciences, Beijing 100080, China; [c]Osaka Sangyo University, Osaka 574-8530, Japan

ABSTRACT

Motivation: Haplotype information has become increasingly important in analyzing fine-scale molecular genetics data, such as disease genes mapping and drug design. Parsimony Haplotyping is one of haplotyping problems belonging to NP-hard class.

Results: In this paper, we aim to develop a novel algorithm for the haplotype inference problem with the parsimony criterion, based on a parsimonious tree-grow method (PTG). PTG is a heuristic algorithm that can find the minimum number of distinct haplotypes based on the criterion of keeping all genotypes resolved during tree-grow process. In addition, a block-partitioning method is also proposed to improve the computational efficiency. We show that the proposed approach is not only effective with a high accuracy but also very efficient with the computational complexity in the order of $O(m^2n)$ time for n SNP sites in m individual genotypes.

Availability: The software is available upon request from the authors, or from http://zhangroup.aporc.org/bioinfo/ptg/

Contact: chen@elec.osaka-sandai.ac.jp

1 INTRODUCTION

Single nucleotide polymorphisms (SNPs) characterize most of genomic variation in human populations. A *haplotype* is a SNP sequence from each of the two copies of a given chromosome in a diploid genome. In contrast, a *genotype* is a description of the mixture information of the two haplotypes in a given chromosome. Recently, haplotype information has become increasingly important in analyzing fine-scale molecular-genetics data for a variety of purposes, such as disease genes mapping and drug design. However, current sequencing technology typically determines genotypes rather than haplotypes due to the requirement of tedious and costly experiments. Such restriction makes *in silico* haplotyping attractive.

So far, many inference and statistical methods have been proposed for haplotyping, such as Clark method (Clark,1990), parsimony approaches (Gusfield,2001; Lancia *et al.*,2001; Wang, *et al.*,2005), maximum-likelihood methods (Excoffier and Slatkin, 1995; Hawley and Kidd, 1995), phylogeny-based approaches (Gusfield,2002; Chung and Gusfield,2003; Halperin and Eskin, 2004), and Bayesian methods (Stephens *et al.*, 2001; Niu *et al.*, 2002). In particular, the parsimony criterion that seeks the minimum number of haplotypes to explain a given set of genotypes, has been widely investigated due to its intuitive simplicity and biological implication. Recently both Wang *et al.* (Wang and Xu, 2003) and Brown *et al.*(Brown and Harrower, 2004) developed an exact algorithm to solve the haplotype inference problem based on the parsimony condition, by the branch-and-bound method and by integer programming method respectively. However, the pure parsimony haplotype inference problem is NP-hard (Gusfield,2001). Any exact algorithm generally suffers from the curse of dimensionality, which impedes the application for analyzing large-scale genomic data.

In this paper, we aim to develop a novel algorithm for the haplotype inference problem with the parsimony criterion, based on a parsimonious tree-grow method (PTG). We show that the proposed approach is not only effective with a high accuracy but also very efficient with the computational complexity in the order of $O(m^2n)$ time for n SNP sites in m individual genotypes. The rest of this paper is organized as follows. Section 2 gives a formal definition of the haplotype inference problem. In Section 3, we explore PTG to develop a new algorithm for the haplotype inference problem and further analyze its computational complexity and optimality. Several numerical experiments are provided in Section 4 to demonstrate the proposed algorithm. In Section 5, we provide an algorithm to reduce the genotype matrix to a smaller block matrix so as to improve the efficiency of PTG algorithm. Finally, we give several general remarks to conclude the paper in Section 6.

2 NOTATION

In this paper, we restrict ourselves to biallelic SNPs. Without loss of generality, assume that the values of the two involved alleles of each SNP are always 0 and 1, which represent the common allele and the rare alleles respectively. Since the SNPs are located sequentially on a chromosome, a *haplotype* with length n is a vector over $\{0,1\}^n$, where each position i is also called a *site* or a *locus*. On the other hand, a *genotype vector*, or simply a *genotype*, represents two haplotypes as a sequence of unordered pairs over the set $\{0,1\}$. Each pair represents the nucleotides in a given site. Since the pairs are unordered, we are not able to determine the two haplotypes from the genotype alone. For example two haplotypes with length 3 are $(0,1,1)$ and $(1,0,1)$ that are combined into the genotype $((0,1),(0,1),(1,1))$.

Whenever a pair is made of two identical values, the SNP site is *homozygous*, otherwise it is *heterozygous*. Clearly, by the assumption on the values of the alleles, the pair for a homozygous site is $(0,0)$ or $(1,1)$, while the pair for an heterozygous site is $(0,1)$. In contrast to the unordered pairs, the genotype can also be represented by a compact form, i.e., a compact representation of the genotype consists of a vector over the alphabet $\{0,1,2\}$, where

*This author's work is partly supported by Informatics Research Center for Development of Knowledge Society Infrastructure, Graduate School of Informatics, Kyoto University, Japan

†to whom correspondence should be addressed

one of the first two symbols is used if the site is homozygous, and a 2 encodes a heterozygous site. For example, the compact representation of the genotype $((0,1),(0,1),(1,1))$ is therefore $(2,2,1)$. Next, we only use the compact form of genotype in this paper. If a genotype has no heterozygous site, then we call it *homozygote*; otherwise we call it *heterozygote*.

Given a genotype $\mathbf{g} = (g_1, \cdots, g_n) \in \{0,1,2\}^n$, then a *resolution* of $\mathbf{g}$ is a pair $\langle \mathbf{h}, \mathbf{k} \rangle$ of haplotypes, where $\mathbf{h} = (h_1, \cdots, h_n)$ and $\mathbf{k} = (k_1, \cdots, k_n)$ are defined in such a way that $h_i = k_i = g_i$ if $g_i \neq 2$; and $h_i, k_i \in \{0,1\}$ with $h_i \neq k_i$ if $g_i = 2$. When the above conditions hold, we also say that $\langle \mathbf{h}, \mathbf{k} \rangle$ *resolves* $\mathbf{g}$, which is denoted by $\mathbf{h} \oplus \mathbf{k} = \mathbf{g}$. Next, we give several definitions as well as a basic result which is used in the proposed algorithm.

DEFINITION 1. *Let $\mathcal{G} = \{\mathbf{g}_1, \cdots, \mathbf{g}_m\}$ be a set of m genotypes where $\mathbf{g}_i = g_{i1} \cdots g_{in}$ is the expression of the i-th genotype, and $g_{ij} \in \{0,1,2\}$ is the j-th SNP value in the i-th genotype. Then*

$$\mathbf{G} = \begin{pmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ g_{m1} & g_{m2} & \cdots & g_{mn} \end{pmatrix} \quad (1)$$

is called a genotype matrix with m genotypes in n SNP sites.

Let

$$\mathbf{G}[i,j] = \begin{pmatrix} g_{1i} & g_{1i+1} & \cdots & g_{1j} \\ g_{2i} & g_{2i+1} & \cdots & g_{2j} \\ \vdots & \vdots & \cdots & \vdots \\ g_{mi} & g_{mi+1} & \cdots & g_{mj} \end{pmatrix} \quad (2)$$

denote a submatrix comprising the columns from the i-th to the j-th in $\mathbf{G}$. Then this submatrix is a genotype submatrix of m genotype fragments with consecutive $j - i + 1$ SNP sites, i.e., from the i-th SNP site to the j-th SNP site. Denote the k-th row of $\mathbf{G}[i,j]$ by $\mathbf{g}_k[i,j]$, that is, $\mathbf{g}_k[i,j] = g_{ki}g_{ki+1}\cdots g_{kj}$, which is called a genotype fragment.

Pure parsimony haplotype inference problem for a given input set of m genotype vectors, is to find a set of m pairs of haplotypes, one for each genotype vector, such that the number of distinct haplotypes is minimum. For convenience, let $\mathcal{H}(\mathbf{G})$ be a solution set of a haplotype inference problem for a given genotype matrix $\mathbf{G}$, and $\mathcal{H}^*(\mathbf{G})$ be the optimal haplotype solution set with the parsimony criterion. Then, $\mathcal{H}^*(\mathbf{G})$ is the solution with the smallest number of distinct haplotypes that can resolve the genotypes $\mathbf{G}$. Clearly $|\mathcal{H}^*(\mathbf{G})| \leq |\mathcal{H}(\mathbf{G})|$ where $|\cdot|$ means the number of elements in the set.

PROPOSITION 1. *For any $1 \leq j \leq n-1$, we have*

$$|\mathcal{H}^*(\mathbf{G}[1,j])| \leq |\mathcal{H}^*(\mathbf{G}[1,j+1])|.$$

3 PARSIMONIOUS TREE-GROW METHOD (PTG)

In this section, we propose a novel algorithm, called Parsimonious Tree-Grow method (PTG), to solve the pure parsimony haplotype inference problem for a given genotype matrix $\mathbf{G}$. It is a heuristic algorithm to find the minimal number of distinct haplotypes based on the criterion of keeping all genotypes resolved during a tree-grow process.

3.1 Main Idea of PTG

If a genotype matrix $\mathbf{G}$ has only one column, we can easily resolve all genotypes in $\mathbf{G}$ by no more than two distinct haplotypes of length 1. If a genotype matrix (or submatrix) has k columns and we have resolved the genotype submatrix $\mathbf{G}[1,k-1]$ by a haplotype fragment set $\mathcal{H}(\mathbf{G}[1,k-1])$ of length $k-1$, then we can resolve the genotype matrix (or submatrix) $\mathbf{G}[1,k]$ by a haplotype set $\mathcal{H}(\mathbf{G}[1,k])$ of length k with every haplotype obtained by adding one SNP value 0 or 1 to one of the haplotype fragments in $\mathcal{H}(\mathbf{G}[1,k-1])$. In other words, we can resolve the genotype matrix columns one by one. The resolving process is executed by a growing-tree with minimal branching principle, which we called as PTG. In the growing-tree, successive layers of the tree correspond to the successive columns, from the left to the right of $\mathbf{G}$. We denote a submatrix of $\mathbf{G}$ as $\mathbf{G}[1,j]$, and call the rows $\mathbf{g}_k[1,j]$, or simply $\mathbf{g}_k(j)$ of $\mathbf{G}[1,j]$ as genotype fragments or row fragments.

Each column of $\mathbf{G}$ is resolved one by one in a consecutive way. Suppose that $\mathbf{G}[1,j]$ has been resolved and the tree has grown to the j-th layer. In the process of resolving $\mathbf{G}[1,j+1]$, the tree grows or extends to a new layer, i.e., the $(j+1)$-th layer, where every node in the $(j+1)$-th layer corresponds to a distinct haplotype fragment of length $j+1$ that can be used to resolve some row fragments in $\mathbf{G}[1,j+1]$. All nodes in the $(j+1)$-th layer correspond to all distinct haplotype fragments that resolve all row fragments in $\mathbf{G}[1,j+1]$. When all the columns of $\mathbf{G}$ are resolved, each node in the final layer corresponds to a unique haplotype, and thereby we can obtain the parsimony haplotype solution set corresponding to the genotype matrix $\mathbf{G}$.

Before we describe the algorithm of PTG in detail, we introduce several definitions. Let the genotype matrix $\mathbf{G}$ in (1) have m rows and n columns, and $v_{01} = \{1, \cdots, m\}$ be the index set of rows, which is also the index set of genotypes.

DEFINITION 2. *For a given genotype matrix $\mathbf{G}$, if there is an l for $1 \leq l \leq j$ such that $g_{il} = 2$, then i is called a divided index in the j-th layer. Otherwise, i is called an undivided index in the j-th layer. To distinguish them, we use a boolean variable $f(i)$ to denote whether the index i is divided at each iteration of PTG program.*

3.2 Algorithm of PTG

We first give the detail procedure of PTG, and then use an illustrative example to demonstrate the performance of the algorithm.

ALGORITHM 1. *PTG.* **Initialization***: Input an $m \times n$ genotype matrix $\mathbf{G}$. Set a root node denoted by v_{01}, which represents the index set $\{1, \cdots, m\}$, i.e., $v_{01} = \{1, \cdots, m\}$. Set $f(i) = false$, for every $i = 1, \cdots, m$. Let $j = 0$, and go to step 1.*

Step 1 *Resolve submatrix $\mathbf{G}[1,j+1]$.*

Suppose that there are p nodes $v_{j1}, \cdots, v_{jk}, \cdots, v_{jp}$ in the j-th layer of the growing-tree, which corresponds to p distinct haplotype fragments resolving $\mathbf{G}[1,j]$. The nodes $v_{j1}, \cdots, v_{jp}$ also represent their corresponding index sets. Do Substeps 1.1 and 1.2 depicted below.

Substep 1.1 For each $1 \leq k \leq p$, and each i, ($1 \leq i \leq m$), if $i \in v_{jk}$, resolve the i-th genotype fragment in $\mathbf{G}[1,j]$ when i satisfies either of the following two conditions:

Condition 1*: $g_{i,j+1} \neq 2$;*

Condition 2*: $g_{i,j+1} = 2$, and $f(i) = false$.*

Otherwise, *record the i in a set $I(j)$; and record v_{jk} in a node set T_{ij}, where T_{ij} is a set of the j-th layer nodes that include node i.*

- *if $g_{i,j+1} = 0$, then add a branch of type 0 to the node v_{jk} when there is no branch of type 0 growing from node v_{jk}; add i to the index set of the node $v_{(j+1)\cdot}$, which is connected to the node v_{jk} by the existing or just added branch of type 0.*
- *if $g_{i,j+1} = 1$, then add a branch of type 1 to the node v_{jk} when there is no branch of type 1 growing from node v_{jk}; add i to the index set of the node $v_{(j+1)\cdot}$, which is connected to the node v_{jk} by the existing or just added branch of type 1.*
- *if $g_{i,j+1} = 2$ and $f(i) = false$, then add a branch of type 0 (a branch of type 1) or both branches (one of type 0 and the other of type 1) or nothing to the node v_{jk} according to the following cases: only one type of branch exists, no branch exists, or two types of branches exist. Add i into both index sets of the $(j+1)$-th layer nodes, which are connected to node v_{jk}, set $f(i) = true$.*

Substep 1.2 For $i \in I(j)$, suppose $T_{ij} = \{v_{jk_1}, v_{jk_2}\}$, that is, i belongs to v_{jk_1} and v_{jk_2}. Check whether there are two different types of branches growing separately from node v_{jk_1} and v_{jk_2}.

1. *If there are no such two different types of branches, then add a proper type of branch to node v_{jk_1} or v_{jk_2}, or add two different types of branches, one to node v_{jk_1} while the other to node v_{jk_2}.*
2. *Choose (or randomly choose) a pair of different type branches, one growing from node v_{jk_1}, the other growing from node v_{jk_2}. Add i into both node index sets of the $(j+1)$-th layer which are connected to v_{jk_1} or v_{jk_2} by one of the chosen branches.*

Step 2 *If $j+1 < n$, set $j := j+1$, and return to Step 1. Otherwise assemble haplotypes as follows.*

Trace each path from node v_{01} to every node in the n-th layer of the growing-tree. The sequence of branch type indices (0 or 1) of the path gives a haplotype, which can be used to resolve the genotypes whose indices belong to the corresponding node in the n-th layer. All the haplotypes corresponding to the n-th layer nodes consist of $\mathcal{H}(\mathbf{G})$.

3.3 An Illustrative Example

To demonstrate the algorithm, we resolve a genotype matrix $\mathbf{G}$ as follows.

$$\mathbf{G} = \begin{pmatrix} 2 & 2 & 0 \\ 2 & 0 & 2 \\ 0 & 2 & 2 \end{pmatrix}$$

Let a root node of the tree be v_{01} with index set $v_{01} = \{1, 2, 3\}$.

3.3.1 Resolve Submatrix $\mathbf{G}[1, 1]$ (or the First Column of $\mathbf{G}$) We first resolve genotype fragment $\mathbf{g}_1[1, 1]$. Since $g_{11} = 2$, we must use two distinct haplotype fragments (0 and 1) to resolve $\mathbf{g}_1[1, 1]$, which results in a branch of type 0 and a branch of type 1 growing from node v_{01}. Denote the two new nodes in the first layer by v_{11} and v_{12} respectively, and set $v_{11} = \{1\}$ and $v_{12} = \{1\}$. Because index 1 is now a divided index, set $f(1) = true$. Then we resolve $\mathbf{g}_1[1, 1]$.

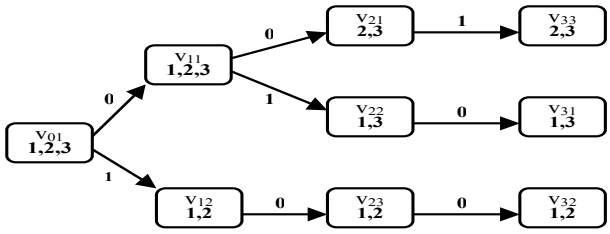

Fig. 1. Growing-tree for resolving all three columns of $\mathbf{G}$

Next, resolve the second genotype fragment $\mathbf{g}_2[1, 1]$. Despite $g_{21} = 2$, since both the branch of type 0 and the branch of type 1 have grown from node v_{01}, we add index 2 into both v_{11} and v_{12} respectively, i.e., $v_{11} = \{1, 2\}, v_{12} = \{1, 2\}$. Set $f(2) = true$. Then, we resolve $\mathbf{g}_2[1, 1]$.

Finally resolve the third genotype fragment $\mathbf{g}_3[1, 1]$. Since $g_{31} = 0$, and there is already a branch of type 0 growing from node v_{01}, we add index 3 into v_{11}. Therefore, $v_{11} = \{1, 2, 3\}$ and $v_{12} = \{1, 2\}$, which resolve $\mathbf{g}_3[1, 1]$. The result is shown in the first layer of tree in figure 1.

3.3.2 Resolve Submatrix $\mathbf{G}[1, 2]$ (or the Second Column of $\mathbf{G}$) First check all indices in v_{11}. Due to $1 \in v_{11}$ with $f(1) = true$ and $g_{12} = 2$, we record the node 1 in a list $I(1)$ to be treated later, and record v_{11} in T_{11}, i.e. $T_{11} = \{v_{11}\}$. Because $2 \in v_{11}$ with $g_{22} = 0$ and there is no branch of type 0 growing from node v_{11} yet, we add a branch of type 0 to node v_{11} and denote the new node by v_{21} and let $v_{21} = \{2\}$. Because $3 \in v_{11}$ with $f(3) = false$ and $g_{32} = 2$ and there is no branch of type 1 growing from node v_{11}, we add a branch of type 1 to node v_{11}, denote the corresponding node attached to it by v_{22}, and further add index 3 to v_{21} and v_{22} respectively, i.e. $v_{21} = \{2, 3\}, v_{22} = \{3\}$. Set $f(3) = true$. Now $\mathbf{g}_3[1, 2]$ is resolved.

In the same manner, we can check all indices in v_{12}. Since $1 \in v_{12}$ with $f(1) = true$ and $g_{12} = 2$, we record v_{12} in T_{11}, i.e. $T_{11} = \{v_{11}, v_{12}\}$. Since $2 \in v_{12}$ with $g_{22} = 0$ and there is no branch of type 0 growing from node v_{12} yet, we add a branch of type 0 to node v_{12} and denote the corresponding node by v_{23} and let $v_{23} = \{2\}$. Now $\mathbf{g}_2[1, 2]$ is resolved.

According to Substep 1.2 of the algorithm, now we consider the indices in $I(1)$. Since $1 \in I(1)$ and $T_{11} = \{v_{11}, v_{12}\}$, there are three branches growing from nodes v_{11} and v_{12}. Hence, we choose the branch of type 1 growing from v_{11} and the branch of type 0 growing from v_{12} to resolve $g_{12} = 2$, add index 1 in v_{22} and v_{23} respectively. Now $\mathbf{g}_1[1, 2]$ is resolved, and

$$v_{21} = \{2, 3\}, \;\; v_{22} = \{1, 3\}, \;\; v_{23} = \{1, 2\}.$$

3.3.3 Resolve Submatrix $\mathbf{G}[1, 3]$ (or the Third Column of $\mathbf{G}$) In the same manner as the above two iterations, we obtain the index sets for the third layer nodes:

$$v_{31} = \{1, 3\}, \;\; v_{32} = \{1, 2\}, \;\; v_{33} = \{2, 3\}$$

which finally solve the haplotyping problem to the given $\mathbf{G}$.

The final tree is depicted in figure 1, which has three nodes in the last layer corresponding to three distinct haplotypes. By tracing all

paths, we obtain three haplotypes for resolving all genotypes in $\mathbf{G}$,

$$\mathcal{H}^*(\mathbf{G}) = \{001, 010, 100\}$$

which is actually the optimal solution of the haplotype inference problem. Because of $1 \in v_{31}$ and $1 \in v_{32}$, the haplotypes 010 (corresponding to v_{31}) and 100 (corresponding to v_{32}) resolve the first genotype $\mathbf{g}_1$ (220). Clearly, according to the index sets of (v_{31}, v_{32}, v_{33}), each haplotype can be used to resolve two genotypes of $\mathbf{G}$, for example, the haplotype corresponding to v_{31} (i.e., 010) can be used to resolve genotypes $\mathbf{g}_1$ and $\mathbf{g}_3$ (since $1 \in v_{31}$ and $3 \in v_{31}$).

3.4 Computational Complexity

There is a bound for the number of haplotypes by PTG. As proven in Proposition 4 in Appendix 1 of Supporting Material, if the genotype matrix $\mathbf{G}$ has m rows and n columns, then the resolution set of haplotype inference problem obtained by Algorithm 1 must satisfy the following inequality

$$|\mathcal{H}(\mathbf{G})| \leq \min\{2m, 2^n\}.$$

From Theorem 1 in Appendix 1 of Supporting Material, we can prove that PTG is a polynomial time algorithm. Specifically, the computational complexity of PTG is $O(m^2n)$, where m denotes the number of genotypes and n is the number of SNP sites in the genotypes or haplotypes. Such a result implies that PTG may be very efficient in terms of CPU cost even for a large amount of genomic data.

4 EXPERIMENTAL RESULTS

In this section, we use both real data and simulation data to demonstrate the performance of PTG. To improve the computational efficiency, input data are preprocessed according to Algorithm 2, which is described in details in Section 5. CPU times in this section are total amount for Algorithms 1 and 2. The program is implemented on a 1.8G Hz 512M RAM Pentium 4 Processor PC using Borland Delphi 5.0 by Pascal, and is available upon request or from website. Throughout our experiment, to measure the performance of PTG, we use *error rate*, a commonly used criterion in haplotype inference problem (Stephen et.al., 2001; Niu et.al., 2003; Wang and Xu, 2003). The error rate is the proportion of genotypes whose original haplotype pairs are incorrectly inferred by the program.

4.1 Experiment on β_2AR Gene Data

β_2-Adrenergic Receptors (β_2AR) are G protein-coupled receptors that mediate the actions of catecholamines in multiple issues. There are 13 variable sites within a span of 1.6kb in the human β_2AR gene. Among 121 individuals, there are 18 distinct genotypes, but only 10 haplotypes resolve all the genotypes. Those 10 haplotypes and 18 genotypes are illustrated in Table 1 (Wang and Xu, 2003).

We run PTG on β_2AR gene data for 100 times. Among 80 times of them, we find 10 distinct haplotypes to resolve all 18 genotypes, where 9 haplotypes of the 10 haplotypes correctly resolve 17 genotypes. The average error rate in 100 runs is 0.056. In particular, in 10 of 100 runs, we found all ten correct haplotypes to resolve all 18 genotypes. The average running time is about 0.016 second, which is considered to be very efficient in contrast to HAPER (over one minute) and PHASE (over ten minutes). Detail computation process for PTG is described in Appendix 2 of Supporting Material.

Table 1. *Ten haplotypes and eighteen genotypes of β_2AR genes.*

	Haplotype	Genotype
$\mathbf{h}_1$	100000010000	$\mathbf{g}_8$,$\mathbf{g}_9$
$\mathbf{h}_2$	100111101000	$\mathbf{g}_1$,$\mathbf{g}_2$,$\mathbf{g}_3$,$\mathbf{g}_6$, $\mathbf{g}_{10}$,$\mathbf{g}_{11}$,$\mathbf{g}_{12}$,$\mathbf{g}_{13}$
$\mathbf{h}_3$	011000010000	$\mathbf{g}_{11}$,$\mathbf{g}_{14}$
$\mathbf{h}_4$	001000010000	$\mathbf{g}_1$,$\mathbf{g}_4$,$\mathbf{g}_5$,$\mathbf{g}_7$, $\mathbf{g}_8$,$\mathbf{g}_{14}$,$\mathbf{g}_{15}$,$\mathbf{g}_{16}$,$\mathbf{g}_{17}$
$\mathbf{h}_5$	001000000000	$\mathbf{g}_6$,$\mathbf{g}_{15}$
$\mathbf{h}_6$	000000000101	$\mathbf{g}_3$,$\mathbf{g}_5$,$\mathbf{g}_9$,$\mathbf{g}_{18}$
$\mathbf{h}_7$	000000000111	$\mathbf{g}_{12}$,$\mathbf{g}_{16}$,$\mathbf{g}_{18}$
$\mathbf{h}_8$	001000010101	$\mathbf{g}_{13}$,$\mathbf{g}_{17}$
$\mathbf{h}_9$	000000000100	$\mathbf{g}_7$
$\mathbf{h}_{10}$	000000000000	$\mathbf{g}_{10}$

Table 2. *Haplotypes for the maize data (Ching et al., 2002).*

	Haplotype	Frequency
$\mathbf{h}_1$	011001001100100101	0.03
$\mathbf{h}_2$	000000000000000000	0.47
$\mathbf{h}_3$	000010001000000000	0.23
$\mathbf{h}_4$	101101110111011010	0.26

4.2 Angiotensin Converting Enzyme Gene Data

Angiotensin Converting Enzyme (ACE) is encoded by the gene DCP1. Complete data for the genomic sequencing of DCP1 from 11 individuals in 22 chromosomes are available (Rieder, *et al.*, 1999). There are 52 SNP sites and 11 genotypes, which are resolved by 13 distinct haplotypes (Rieder *et al.*, 1999; Wang and Xu, 2003). We obtained 13 haplotypes with 9 correct haplotypes that resolve 9 out of the 11 genotypes correctly with a error rate of 2/11=0.182. Such a performance is better than or at least equal to widely used existing programs, i.e., HAPAR with error rate of 0.273, Haplotyper with error rate of 0.182, HAPINFERX with error rate of 0.273 and PHASE with error rate of 0.273 (Wang and Xu, 2003). The relatively low accuracy is mainly due to the small sample size. In these experiments, the average CPU time is 0.320 second.

4.3 Maize Data Set

The maize data set is used as one of a benchmark to evaluate accuracy of haplotype programs (Wang and Xu, 2003). Acetyl-CoA C-acyltransferase which is an enzyme and catalyses the final step of fatty acid oxidation, has 18 SNP sites and 4 haplotypes with frequencies of 0.03, 0.47, 0.23 and 0.26 in the maize data set, as shown in Table 2. We follow the same procedure as Wang and Xu (2003) to generate a sample of n genotypes by randomly picking 2 haplotypes according to their frequencies and conflating them. Table 3 is the simulation results (Wang and Xu, 2003) for five programs. The error rates are average values for 100 random samples. Clearly, PTG correctly resolves all genotypes for sample sizes from 4 to 10, and behaves best among five programs in terms of accuracy. We also conducted simulations for Adh1 in the maize data set for different sample sizes, which has 6 haplotypes and 14 SNP sites with frequencies of 0.031, 0.031, 0.125, 0.25, 0.25 and 0.312. The simulation results are almost the same as those of Table 3, and PTG correctly resolves all genotypes.

Table 3. *Comparison of error rates for five programs on Maize data set.*

Sample size	PTG	HAPAR	HAPLOTYPER	HAPINFERX	PHASE
3	0.02	0.51	0.47	0.86	0.53
4	0	0.10	0.14	0.64	0.15
7	0	0.05	0.05	0.43	0.07
10	0	0	0	0.28	0

Table 4. *The simulation results of PTG with $n = 10$.*

m	n	CPU(sec.)	error rate
10	10	0.010	0.120
15	10	0.010	0.100
20	10	0.010	0.050
25	10	0.021	0.030
30	10	0.027	0.025
35	10	0.032	0.018
40	10	0.036	0.005

Table 5. *The simulation results of PTG with $n = 50$.*

m	n	CPU(sec.)	error rate
50	50	0.211	0.190
100	50	0.283	0.280
150	50	0.292	0.075
200	50	0.316	0.062
250	50	0.371	0.036
300	50	0.431	0.037

Table 6. *The simulation results of PTG with $n = 200$.*

m	n	CPU(sec.)	error rate
100	200	1.260	0.38
200	200	7.120	0.20
400	200	45.024	0.12
600	200	229.941	0.08

4.4 Experiments on Simulation Data

The haplotype generator, ms, in Hudson (2002) is a well-known standard program based on the coalescent model of SNP sequence evolution. In this subsection, we use the software (ms) to generate $2m$ haplotypes, each with n SNP sites, and then randomly pair them to obtain m genotypes, which are used as input for the PTG program.

4.4.1 Coalescence-Based Simulations Without Recombination In this section, the number of SNPs is fixed as 10, 50, 200 respectively, and 100 replications were made for each sample size. When generating haplotypes, we specify recombination parameter to be 0. The CPU times and error rates of PTG are illustrated in Tables 4- 7 respectively, where m denotes the number of genotype matrix rows, and n is the number of genotype matrix columns.

Comparing with Figure 4 of (Wang and Xu, 2003), the computation in Tables 4 - 7 is fairly efficient in terms of both CPU time and accuracy (Halperin and Eskin, 2004), even for large size of genomic data, in contrast to PHASE (Stephens, *et al.*, 2001) and HAPLOTYPER (Niu *et al.*, 2002) and other methods. The results indicate the superiority of our algorithm over the conventional approaches. Both the numbers of genotypes and SNP sites affect the computational cost, which is also proved in Section 3.4 or Appendix 1 of Supporting Material.

Table 7. *The simulation results of PTG on large size of data.*

m	n	CPU(sec.)	error rate
1000	100	303.687	0.01
1000	150	412.052	0.06
1000	200	491.641	0.05

Table 8. *The simulation results of PTG on data with recombination.*

m	n	CPU(sec.)	error rate
10	10	0.015	0.25
15	10	0.025	0.45
20	10	0.030	0.37
25	10	0.031	0.35
30	10	0.040	0.34
35	10	0.045	0.30
40	10	0.046	0.31

4.4.2 Coalescence-Based Simulations With Recombination In this section, we introduce recombination into the model when generating simulated haplotypes. We set recombination parameter ρ to be 100.0 when generating haplotypes by the software ms. The simulation results are illustrated in Table 8.

Comparing with Figure 5 of (Wang and Xu, 2003), the error rate results in Table 8 are similar to those obtained by the existing haplotype softwares. However, in contrast to the cases without recombination shown in Table 4, the error rates are high. The reason resulting in relatively high error rate is that in the simulation data with recombination, the number of correct distinct haplotypes resolving all genotypes is often not the minimum one. For example, in our simulation of 30 genotypes with 10 SNP sites, the number of the correct distinct haplotypes resolving all 30 genotypes is 24. However, by PTG, we can find a solution of 19 distinct haplotypes resolving all 30 genotypes. Since PTG can almost always find the minimum number of haplotypes to resolve all genotypes, the error rate may not be low when recombination rate is high. This is also the reason why the error rate of other programs is also high too. Such a fact implies that parsimony approach may not be suitable for the data with a high recombination rate, or needs to be modified to handle such problems by further considering the characteristics of recombination.

To study the bottleneck effect, we do simulation on large scale of data without recombination, as shown in Table 7. For a sample of 1000 individuals, PTG currently can handle 200 SNPs in no more than ten minutes, which is better than HAPLOTYPER (handling 50 SNPs of 1000 individuals); For a sample of 300 individuals, PTG can handle 400 SNPs, which is also efficient in contrast to HAPLOTYPER (handling 256 SNPs of 100 individuals). PTG can

even resolve problems with much large-size of data if there is sufficient capacity of computer RAM (>512MB).

5 IMPROVING EFFICIENCY OF PTG

Usually in genotype matrix derived from human haplotypes, many columns corresponding to SNP sites are identical. Indeed, as noted in (Patil, N., *et al*., 2001), the number of identical columns in real data is considerably large. It is common to keep only one column out of several identical columns since they are assumed not to carry any additional information (Patil, N., *et al*., 2001). Thus we can improve the performance (in both CPU times and memory requirements) by reducing the number of columns of genotype matrix. This can be executed by dividing the genotype matrix into blocks, as a precomputation process of Algorithm 1.

DEFINITION 3. *Given a genotype $\mathbf{g}_k = g_{k1}\cdots g_{kn}$ and a fragment $\mathbf{g}_k[i,j] = g_{ki}\cdots g_{kj}$, if $g_{kt} = 0$ (or 1) for each $i \le t \le j$, then the fragment $\mathbf{g}_k[i,j]$ is called an identical homozygous genotype fragment of type 0 (or 1) . If $g_{kt} = 2$ for each $i \le t \le j$, then $\mathbf{g}_k[i,j]$ is called an identical heterozygous genotype fragment.*

DEFINITION 4. *Given a genotype submatrix $G[i,j]$, if every row of $G[i,j]$ is either an identical homozygous genotype fragment or an identical heterozygous genotype fragment, i.e., every row of $G[i,j]$ is one of the following three types:*

Type 0: $0\cdots 0$*;*
Type 1: $1\cdots 1$*;*
Type 2: $2\cdots 2$*,*

then $\mathbf{G}[i,j]$ is called a block. *A block is called a* homozygous block *of type 0 (or type 1) if every row is an identical homozygous genotype fragment of type 0 (or type 1). Otherwise, it is called a* heterozygous block.

Clearly, a block is a submatrix of the genotype matrix with all the columns identical.

DEFINITION 5. *Given a haplotype $\mathbf{h}_k = h_{k1}\cdots h_{kn}$ where $h_{kl} \in \{0,1\}$ for $1 \le l \le n$, a haplotype fragment $\mathbf{h}_k[i,j] = h_{ki}\cdots h_{kj}$ is called an identical homozygous haplotype fragment of type 0 (or 1) if $h_{kt} = 0$ (or $h_{kt} = 1$) holds for any $i \le t \le j$.*

With the preparation above, we have a basic proposition below to simplify the computation in PTG algorithm.

PROPOSITION 2. *All the genotype fragments in a homozygous block of type 0 (type 1) can be resolved by one (or two identical) identical homozygous haplotype fragment of type 0 (type 1). All the genotype fragments in a heterozygous block can be resolved by two different types of identical homozygous haplotype fragments in the spirit of parsimony.*

For example, all genotype fragments in the homozygous block

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

can be resolved by one identical homozygous haplotype 000, and all genotype fragments in the heterozygous block

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix}$$

can be resolved by two identical homozygous haplotype fragments, i.e., 000 and 111. Given a genotype matrix $\mathbf{G}$, we can use the following algorithm to divide $\mathbf{G}$ into blocks, which are further combined into a block matrix $\mathbf{B}$.

ALGORITHM 2. *Dividing genotype matrix $\mathbf{G}$ into blocks.*

- *Initialization: input a genotype matrix $\mathbf{G}$ with m individual genotypes and n SNP sites, and let $k := 1, j := 1, i_k := j$.*
- *Step 1. if*

$$\begin{pmatrix} g_{1i_k} \\ g_{2i_k} \\ \vdots \\ g_{mi_k} \end{pmatrix} - \begin{pmatrix} g_{1j+1} \\ g_{2j+1} \\ \vdots \\ g_{mj+1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

  then $j := j+1$, go to Step 2; otherwise go to Step 3.
- *Step 2. if $j = n$, go to Step 3; otherwise go to Step 1.*
- *Step 3. $\mathbf{G}_k = \mathbf{G}[i_k, j]$, which is defined in (2). If $j = n$, stop and output all blocks $\mathbf{G}_k$ of $\mathbf{G}$; otherwise, let $k := k+1,\ j := j+1,\ i_k := j$, and go to Step 1.*
- *Step 4. Combine all blocks into a block matrix $\mathbf{B}$ where each column represents a block.*

It can be easily shown that Algorithm 2 can divide the genotype matrix $\mathbf{G}$ into blocks in no more than $O(mn)$ arithmetic operations. Since all columns in a block are identical, we can use one of them to represent the block. After doing this to all blocks of $\mathbf{G}$, we obtain a matrix $\mathbf{B}$, which is called a *block matrix* of $\mathbf{G}$. Obviously, each block is comprised of consecutive identical columns of a genotype matrix. Clearly, the algorithm can be easily extended to find an *extended block* matrix with the minimum number of "blocks", where each extended block is composed by all the identical columns in $\mathbf{G}$.

PROPOSITION 3. *Given a genotype matrix $\mathbf{G}$ and its corresponding block matrix $\mathbf{B}$, let $\mathcal{H}^*(\mathbf{G})$ and $\mathcal{H}^*(\mathbf{B})$ respectively be the optimal haplotype solution sets of the haplotype inference problem with $\mathbf{G}$ and $\mathbf{B}$. Then $|\mathcal{H}^*(\mathbf{G})| = |\mathcal{H}^*(\mathbf{B})|$, and the haplotypes in $\mathcal{H}^*(\mathbf{G})$ can be obtained from those in $\mathcal{H}^*(\mathbf{B})$ by adding the corresponding SNPs. On the other hand, the haplotypes in $\mathcal{H}^*(\mathbf{B})$ can be obtained from those in $\mathcal{H}^*(\mathbf{G})$ by deleting the SNPs according to the rule of dividing $\mathbf{G}$ into blocks.*

EXAMPLE 1. *Given a genotype matrix*

$$\mathbf{G} = \begin{pmatrix} 2 & 2 & 2 & 0 & 0 \\ 2 & 2 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 & 2 \end{pmatrix}$$

the corresponding block matrix is

$$\mathbf{B} = \begin{pmatrix} 2 & 2 & 0 \\ 2 & 0 & 2 \\ 0 & 2 & 2 \end{pmatrix}$$

according to Algorithm 2.

An optimal solution for the block matrix is $\mathcal{H}^*(\mathbf{B}) = \{001, 010, 100\}$ *by PTG algorithm, which means* $\mathcal{H}^*(\mathbf{G}) = \{00011, 00100, 11000\}$ *by Proposition 3. Clearly, every haplotype in* $\mathcal{H}^*(\mathbf{B})$ *is a "compression" of the haplotype in* $\mathcal{H}^*(\mathbf{G})$*, and every haplotype in* $\mathcal{H}^*(\mathbf{G})$ *is an "extension" of the haplotype in* $\mathcal{H}^*(\mathbf{B})$.

Therefore, instead of the original genotype matrix $\mathbf{G}$, we can use the block matrix $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_t)$ that has less columns, to improve the computational efficiency in Initialization of Algorithm 1, in particular, for large scale data. Generally, such a compression not only reduces the combination of possible haplotypes, but also loses no information of genotype data. Hence, given a genotype matrix $\mathbf{G}$, we first use Algorithm 2 to reduce the number of columns in $\mathbf{G}$ and obtain the block matrix $\mathbf{B}$, then apply Algorithm 1 (PTG) to the block matrix $\mathbf{B}$. After resolving the block matrix $\mathbf{B}$, we recover haplotypes of full length to resolve the genotype matrix $\mathbf{G}$. The program in this paper is coded by both Algorithms 2 and 1.

Using this modified PTG algorithm, we obtained an optimal solution of $\beta_2 AR$ gene data, the corresponding growing tree and the index sets are depicted in Appendix 2 of Supporting Material.

6 FURTHER DISCUSSION OF PTG ALGORITHM

In this paper, we proposed a novel graphic algorithm-PTG, which not only is very efficient with polynomial arithmetic operations but also has high accuracy for the haplotype inference problem. In particular, the computational cost is very low even for large scale genomic data as indicated in Table 7 and proven in Theorem 1. In contrast to Clark's method, there is no restriction for the given genotypes, i.e., PTG can resolve the case that every genotype has more than one heterozygous site such as the illustrative example and in the simulation tests in Section 4.4.

Although PTG is very efficient, it is based on the parsimony criterion, which generally does not directly take the count information of genotypes into consideration, as indicated in Section 4. To alleviate such a disadvantage, instead of pure parsimony, a modified parsimony criterion may be required, such as by adding weighting parameters to approximately incorporate frequency information of genotypes in the model. In addition, PTG algorithm currently has no function to handle gaps in the genotype matrix. As a future topic, we will improve the PTG algorithm to incorporate missing data in optimization.

We are grateful to Wang and Xu for kindly giving us haplotype data sets and related information. This work is partially supported by the National Natural Science Foundation of China under grant No.10471141, and the Stress Foundation of Beijing Materials Institute, Beijing, China.

REFERENCES

[1]Bondy, J.A. and Murty,U.S.R. (1976) Graph Theory With Applications. Macmillan, London.

[2]Brown,D.G. and Harrower,I.M. (2004) A new integer programming formulation for the pure parsimony problem in haplotype analysis. In I. Jonassen, J. Kim, ed., *Algorithms in Bioinformatics*, 4th International Workshop (WABI), 254-265, Springer.

[3]Ching, A., Caldwell, K.S., Jung, M., Dolan, M., Smith, O.S., Tingey, S., Morgante, M. and Rafalski, A.J. (2002) SNP frequency, haplotype structure and linkage disequilibrium in elite maize inbred lines. *BMC Genet.*, **3**, 19.

[4]Chung, R.H. and Gusfield,D. (2003) Perfect phylogeny haplotyper: Haplotype inferral using a tree model. *Bioinformatics*, **19**,780-781.

[5]Clark, A.G. (1990) Inference of haplotypes from PCR-amplified samples of diploid populations. *Molecular Biology and Evolution*, **7**,111-122.

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

[7]Greenberg,H., Hart,W.E. and Lancia,G. (2002) Opportunities for combinatorial optimization in computational biology. Technical report, University of Colorado at Denver, Mathematics Department, Denver,CO.

[8]Gusfield, D. (2001) Inference of haplotypes from samples of diploid populations: Complexity and algorithms. *Journal of Computational Biology*, **8**,305-324.

[9]Gusfield, D. (2002) Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. *Proceedings of RECOMB 2002: The sixth Annual International Conference on Computational Biology*, 166-175.

[10]Halperin,E. and Eskin, E. (2004) Haplotype reconstruction from genotype data using imperfect phylogeny. *Bioinformatics*, **20**, 1842-1849.

[11]Hawley, M. and Kidd, K. (1995) Haplo: a program using the EM algorithm to estimate the frequencies of multi-site haplotypes. *J. Heredity*, **86**, 409-411.

[12]Hudson, R. (2002) Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*, **18**, 337-338.

[13]Lancia,G., Bafna,V., Istrail,S., Lippert,R. and R. Schwartz. (2001) SNPs problems, complexity and algorithms. In *Proceedings of Annual European Symposium on Algorithms (ESA)*, **2161**, Lecture Notes in Computer Science, 182-193, Springer.

[14]Lin,S., Cutler,D.J., Zwick,M.E. and Chakravarti,A. (2002) Haplotype inference in random population samples. *American Journal of Human Genetics*, **71**,1129-1137.

[15]Lincia,G. and Perlin,M. (1998) Genotyping of pooled microsatellite markers by combinatorial optimization techniques. *Discrete Applied Mathematics*, **88**, 291-314.

[16]Niu,T., Quin,Z.S, Xu,X. and Liu, J.S. (2002) Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms. *American Journal of Human Genetics*, **70**,157-169.

[17]Patil,N., *et al.* (2001) Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. *Science*, **294**, 171923.

[18]Qian,D. and Beckmann,L. (2002) Minimum-recombinant haplotyping in pedigrees. *American Journal of Human Genetics*, **70**,1434-1445.

[19]Rieder,M., Taylor,S., Clark,A. and Nickerson,D. (1999) Sequence variation in the human angiotensin converting enzyme. *Nat. Genet.*, **22**, 59-62.

[20]Stephens,M., Smith,N.J. and Donnelly,P. (2001) A new statistical method for haplotype reconstruction from population data. *American Journal of Human Genetics*, **68**, 978-989.

[21]Wang, L.S. and Xu, Y. (2003) Haplotype inference by maximum parsimony. *Bioinformatics*,**19**,1773-1780.

[22]Wang,R., Wu,L., Li,Z., Zhang,X. (2005) Haplotype Reconstruction from SNP Fragments by Minimum Error Correction. *Bioinformatics*, **21**, 2456 - 2462.

[23]Xu,C.F., *et al.*(2002) Effectiveness of computational method in haplotype prediction. *Human Genetics*, **110**,148-156.

Supporting Material

1 COMPUTATIONAL COMPLEXITY OF PTG

We first show that there is a bound for the number of haplotypes by PTG, and then analyze its computational complexity.

PROPOSITION 4. *If the genotype matrix $\mathbf{G}$ has m rows and n columns, then the resolution set of haplotype inference problem obtained by Algorithm 1 must satisfy the following inequality*

$$|\mathcal{H}(\mathbf{G})| \leq \min\{2m, 2^n\}.$$

PROOF. According to Algorithm 1, when a node corresponds to only one divided index in its index set, it must grow only one branch in the next layer, and the new node corresponds to only one divided index in its index set. In the last layer of the tree, every index can be marked in no more than two index sets of nodes. Hence, the total nodes of the last layer are no more than $2m$. Since one node of the last layer corresponds to a unique haplotype, the total haplotypes can not be more than $2m$, that is $|\mathcal{H}(\mathbf{G})| \leq 2m$.

On the other hand, every node can grow at most 2 branches, and the tree has only one root node. Therefore, there are at most 2^n nodes in the n-th layer, which implies that there are at most 2^n haplotypes, that is, $|\mathcal{H}(\mathbf{G})| \leq 2^n$.

THEOREM 1. *The computational complexity of PTG is $O(m^2n)$, where m denotes the number of genotypes and n is the number of SNP sites in the genotypes or haplotypes.*

PROOF. For m genotypes and n SNP sites, the corresponding genotype matrix is an $m \times n$ matrix.

In the growing-tree, every layer has no more than $2m$ nodes. Executing Substep 1.1 to resolve the i-th column needs $O(m^2)$ arithmetic operations. Executing Substep 1.2 needs $O(m)$ arithmetic operations. Hence, resolving every column of $\mathbf{G}$ needs $O(m^2)$ arithmetic operations. Because the genotype matrix $\mathbf{G}$ has n columns, resolving all columns of $\mathbf{G}$ needs $O(m^2n)$ arithmetic operations, which completes the proof for the computational complexity $O(m^2n)$ for PTG.

2 RESULTS OF PTG ON $\beta_2 AR$ GENE DATA

We first divide the genotype matrix $\mathbf{G}$ into blocks. In this example, columns 4 to 7 belong to one block, and any other column consists of a block. Hence, there are 9 blocks in the genotype matrix $\mathbf{G}$, and the block matrix is as follows.

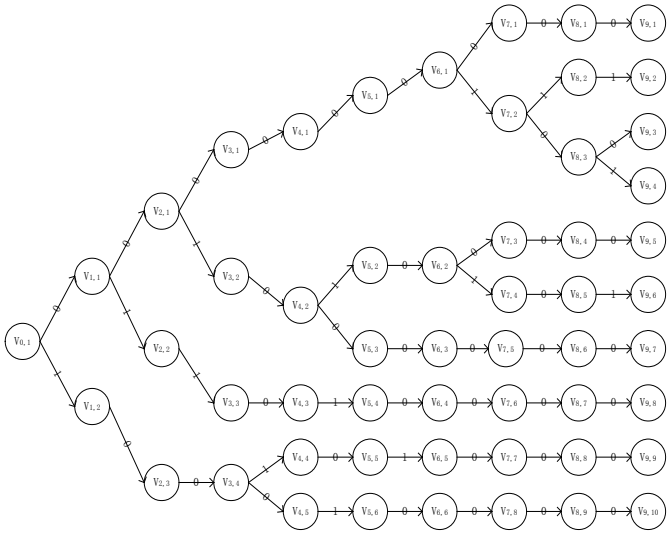

Fig. 2. A growing-tree for $\beta_2 AR$ gene data by PTG

$$\mathbf{B} = \begin{pmatrix}
2 & 0 & 2 & 2 & 2 & 2 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
2 & 0 & 0 & 2 & 0 & 2 & 2 & 0 & 2 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 \\
2 & 0 & 2 & 2 & 0 & 2 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 0 \\
2 & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 2 & 0 & 2 & 0 & 2 \\
2 & 0 & 0 & 2 & 0 & 2 & 0 & 0 & 0 \\
2 & 2 & 2 & 2 & 2 & 2 & 0 & 0 & 0 \\
2 & 0 & 0 & 2 & 0 & 2 & 2 & 2 & 2 \\
2 & 0 & 2 & 2 & 2 & 2 & 2 & 0 & 2 \\
0 & 2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 2 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 2 & 0 & 2 & 2 & 2 \\
0 & 0 & 1 & 0 & 1 & 0 & 2 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1
\end{pmatrix}$$

Then we use PTG algorithm to resolve block matrix $\mathbf{B}$. The growing-tree for this problem is depicted in Figure 2, where v_{jk} denotes the k-th node of the j-th layer and also represents the corresponding index set. Each v_{jk} is listed as follows.

The tree has 9 layers of nodes, and there are 10 nodes in the last layer, which represent 10 haplotypes respectively. For example, by tracing the branches, the haplotype corresponding to node $v_{9,6}$ is 001010101, and the haplotype corresponding to node $v_{9,5}$ is 001010000. These two haplotypes resolve the 17-th row of block matrix $\mathbf{B}$. It is easy to verify that the resolution of every genotype obtained by our algorithm can be easily recover to the corrected haplotypes resolving all 18 genotypes, that is, the error rate is 0.

$v_{0,1} = \{1, \cdots, 18\}$
$v_{1,1} = \{1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18\}$
$v_{1,2} = \{1, 2, 3, 6, 8, 9, 10, 11, 12, 13\}$
$v_{2,1} = \{4, 5, 7, 15, 16, 17, 18, 1, 3, 6, 8, 9, 10, 12, 13, 14\}$
$v_{2,2} = \{14, 11\}$
$v_{2,3} = \{1, 2, 3, 6, 8, 9, 10, 11, 12, 13\}$
$v_{3,1} = \{18, 5, 7, 9, 10, 12, 16, 3\}$
$v_{3,2} = \{4, 15, 17, 5, 7, 14, 16, 13, 8, 6, 1\}$
$v_{3,3} = \{14, 11\}$
$v_{3,4} = \{1, 2, 3, 6, 8, 9, 10, 11, 12, 13\}$
$v_{4,1} = \{18, 5, 7, 9, 10, 12, 16, 3\}$
$v_{4,2} = \{4, 15, 17, 5, 7, 14, 16, 13, 8, 6, 1\}$
$v_{4,3} = \{14, 11\}$
$v_{4,4} = \{2, 13, 6, 10, 11, 12, 1, 3\}$
$v_{4,5} = \{8, 9\}$
$v_{5,1} = \{18, 3, 5, 7, 9, 10, 12, 16\}$
$v_{5,2} = \{4, 17, 15, 1, 5, 7, 8, 14, 16, 13\}$
$v_{5,3} = \{15, 6\}$
$v_{5,4} = \{14, 11\}$
$v_{5,5} = \{2, 1, 3, 6, 10, 12, 13, 11\}$
$v_{5,6} = \{8, 9\}$
$v_{6,1} = \{18, 5, 7, 9, 16, 3, 10, 12\}$
$v_{6,2} = \{4, 17, 5, 7, 8, 15, 16, 1, 13\}$
$v_{6,3} = \{15, 6\}$
$v_{6,4} = \{14, 11\}$
$v_{6,5} = \{2, 1, 3, 6, 10, 11, 12, 13\}$
$v_{6,6} = \{8, 9\}$
$v_{7,1} = \{10\}$
$v_{7,2} = \{18, 12, 9, 7, 5, 3, 16\}$
$v_{7,3} = \{4, 1, 8, 14, 15, 17, 5, 7, 16\}$
$v_{7,4} = \{17, 13\}$
$v_{7,5} = \{15, 6\}$
$v_{7,6} = \{14, 11\}$
$v_{7,7} = \{2, 1, 6, 10, 11, 13, 12, 3\}$
$v_{7,8} = \{8, 9\}$
$v_{8,1} = \{10\}$
$v_{8,2} = \{18, 16, 12\}$
$v_{8,3} = \{18, 3, 5, 7, 9\}$
$v_{8,4} = \{4, 1, 8, 14, 7, 15, 17, 5, 16\}$
$v_{8,5} = \{17, 13\}$
$v_{8,6} = \{15, 6\}$
$v_{8,7} = \{14, 11\}$
$v_{8,8} = \{2, 1, 3, 6, 10, 11, 13, 12\}$
$v_{8,9} = \{8, 9\}$
$v_{9,1} = \{10\}$
$v_{9,2} = \{18, 16, 12\}$
$v_{9,3} = \{7\}$
$v_{9,4} = \{18, 3, 5, 9\}$
$v_{9,5} = \{4, 1, 8, 14, 15, 5, 7, 16, 17\}$
$v_{9,6} = \{13, 17\}$
$v_{9,7} = \{6, 15\}$
$v_{9,8} = \{11, 14\}$
$v_{9,9} = \{2, 1, 6, 10, 11, 12, 3, 13\}$
$v_{9,10} = \{8, 9\}$