Journal of Biomedical Informatics 41 (2008) 953–961

Contents lists available at ScienceDirect

Journal of Biomedical Informatics

journal homepage: www.elsevier.com/locate/yjbin

A new framework for the selection of tag SNPs by multimarker haplotypes

Yao-Ting Huang [a], Kun-Mao Chao [b,c,d,*]

[a] Department of Computer Science and Information Engineering, National Chung-Cheng University, Chia-Yi, Taiwan
[b] Department of Computer Science and Information Engineering, National Taiwan University, #1 Roosevelt Road, Section 4, Taipei, Taiwan
[c] Graduate Institute of Biomedical Electronics and Bioinformatics, National Taiwan University, Taipei, Taiwan
[d] Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei, Taiwan

ARTICLE INFO

Article history:
Received 2 October 2007
Available online 12 April 2008

Keywords:
Algorithm
Haplotype
Linkage disequilibrium
NP-hardness
SNP

ABSTRACT

This paper proposes a new framework for the selection of tag SNPs based on haplotypes instead of on a single SNP. The tag SNPs found by this framework form a set of haplotypes completely predictive of the alleles of all untyped SNPs. We refer to this problem as MTMH, which is defined as follows: given a set of SNPs, find a minimum subset of SNPs (called tag SNPs) which defines a set of haplotypes completely predictive of the alleles of all untyped SNPs. The MTMH problem is solved by dividing into three subproblems, two of which are shown to be NP-hard. Several exact and approximation algorithms are proposed to solve these subproblems. We describe a framework which integrates these algorithms and develop a program called HapTagger for finding tag SNPs. HapTagger is compared with existing methods as well as the official tagging tool (called Haploview) of the International HapMap project using a variety of real data sets. Our theoretical analysis and experimental results indicate that HapTagger consistently identifies a smaller set of tag SNPs and runs much faster than existing methods. HapTagger avoids the need of incorporating a linkage disequilibrium statistic and thus significantly improves the computational efficiency. We also present an algorithm (specific to HapTagger) for reconstructing alleles of untyped SNPs. It is worth mentioning that these predictive haplotypes selected by HapTagger can be used as signatures of recent positive selection or co-evolution.
HapTagger is available at http://www.csie.ntu.edu.tw/~kmchao/tools/HapTagger/.

1. Introduction

Single nucleotide polymorphisms (SNPs) are the most abundant form of genetic variations observed in the human population. Through recent linkage disequilibrium (LD) analysis across the entire human genome, the SNPs in proximity are shown to usually have strong correlation with each other [1,9,14,17]. The correlation structure of entire genome indicates that the human chromosome can be partitioned into high LD regions interspersed by low LD regions. Within each high LD region, only a small subset of SNPs (called tag SNPs) is sufficient to be typed, whereas the alleles of untyped SNPs can be indirectly predicted by typed tag SNPs, due to the strong correlation among them [3,6,21,26,33,34]. In 2002, the International HapMap project is launched to characterize the LD patterns in the human genome such that this information can be used to guide the selection of tag SNPs [1,16]. Recently, with the advent of high-throughput genotyping array (e.g., Affymetrix 500K GeneChip array), the cost of assaying tens of thousands of SNPs has been greatly reduced [22,24]. As a consequence, genome-wide association studies using tag SNPs together with genotyping array are going to be used for studying complex genetic diseases presumably induced by multiple unknown genes throughout the genome [8,19,23]. In contrast to traditional association studies or linkage analysis, the genome-wide association studies using tag SNPs make no assumption on the location of disease genes and is a promising approach for discovering disease susceptibility genes of complex diseases.

However, due to the limited size of the genotyping array, it is difficult to type all tag SNPs to capture the allele of each common SNP on the human genome. Therefore, investigators are usually forced to select a subset of tag SNPs, to prioritize them, or to relax the threshold of LD [1,10]. But these approaches often sacrifice the statistical power in subsequent association studies or analysis. In addition, due to the incompleteness of the HapMap data, less common SNPs (e.g., minor allele frequencies less than 5%) which may induce the disease are usually ignored and not captured by existing algorithms. To capture these less common SNPs, it is expected that more tag SNPs have to be used. Moreover, a portion of tag SNPs may not be always successfully typed and these missing data may greatly decrease the power of using tag SNPs [34]. To avoid the influence from missing data, it has been shown that additional

* Corresponding author. Address: Department of Computer Science and Information Engineering, National Taiwan University, #1 Roosevelt Road, Section 4, Taipei, Taiwan. Fax: +886 2 23628167.
E-mail address: kmchao@csie.ntu.edu.tw (K.-M. Chao).

1532-0464/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jbi.2008.04.003

tag SNPs have to be included into the solution [6,20]. As a consequence, sophisticated methods for reducing the number of tag SNPs are still highly demanded.

A number of methods have been proposed to identify the minimum tag SNPs using different criteria. Most methods are mainly based on the pairwise LD between two diallelic SNPs [2,5,17,28]. However, these pairwise-based methods tend to produce numerous tag SNPs having little or no correlation with untyped SNPs. The singleton tag SNPs (i.e., SNPs having no correlation with others) can even account for more than 50% in their solutions [17]. A few initial studies overcome the limitation of pairwise LD between two diallelic SNPs by further considering the multiallelic LD between a diallelic SNP and a multimarker (multiallelic) haplotype [3,23,30,32]. These approaches can reduce the number of tag SNPs or increase the statistical power but come at the cost of heavy computational overhead, due to the exponential number of possible haplotypes to be tested. Recently, de Bakker et al. use a peelback approach and a multiallelic LD statistic for selecting tag SNPs. The developed program is incorporated into Haploview, which is the official tagging tool used in the International HapMap project [4]. However, Haploview is still quite inefficient, because the number of possible haplotypes to be tested by the LD statistic still grows exponentially with respect to the number of SNPs. As a consequence, Haploview has to make several restrictions to gain efficiency (e.g., test at most 10,000 haplotypes).

In this paper, we design and implement algorithms for the selection of tag SNPs by multimarker haplotypes. In contrast to previous studies using a single tag SNP to predict the alleles of an untyped SNP, our algorithms search for tag SNPs which define a set of haplotypes completely predictive of the alleles of all untyped SNPs. Moreover, our methods do not rely on any statistic to measure the LD between tag SNPs and untyped SNPs. We start by studying a problem called multimarker haplotype tagging by perfect LD (MHTP), which is defined as follows: given a set of SNPs and a target SNP to be replaced, find a minimum length haplotype completely predictive of alleles at the target SNP. We prove that the MHTP problem is NP-hard and give an approximation algorithm. This algorithm is used as a subroutine for solving the main problem studied in this paper (referred to as MTMH), which is defined as follows: given a set of SNPs, find a minimum set of tag SNPs which defines a set of haplotypes completely predictive of the alleles of all untyped SNPs. The MTMH problem is solved by dividing into three subproblems, two of which are shown to be NP-hard. Several exact and approximation algorithms are developed to solve these subproblems and their extension for tolerating missing data is also presented. We integrate these algorithms and develop a program called HapTagger for finding tag SNPs by multimarker haplotypes. The HapTagger is compared with the pairwise LD-based approach and the official tagging tool Haploview. Our theoretical and experimental results indicate that HapTagger consistently finds a smaller set of tag SNPs on a variety of real data sets and runs much faster than existing methods. The efficiency of various LD statistic and the comparison of distinct methods for reconstructing untyped SNP alleles are also discussed in this paper. It is worth mentioning that these predictive haplotypes selected by HapTagger can be used as the signature of recent positive selection or co-evolution.

2. Algorithms for the selection of predictive haplotypes

In this section, we study the MHTP problem, which aims to find a minimum length haplotype completely predictive of the alleles at a target SNP to be replaced. The algorithm introduced in this section is used as a subroutine for solving the MTMH problem studied in the next section. Informally speaking, we seek for a haplotype which is always observed together with some alleles at a target SNP. We first formulate the MHTP problem, show the hardness of this problem, and finally give an approximation algorithm.

2.1. Formulation and hardness of the MHTP problem

Given a $k*(n+1)$ haplotype matrix, where k is the number of haplotypes and $(n+1)$ is the number of SNPs. Denote $C=\{S_1, S_2, \ldots, S_n\}$ as the set of n SNPs and $S_T \notin C$ as the target SNP to be replaced. Let $H=\{h_1, h_2, \ldots, h_k\}$ denote these k haplotypes, where $h_i=\{0,1,x\}^{n+1}$, and 0, 1, or x denote that the allele at this SNP locus is the major type, minor type, or missing data, respectively. Note that for unphased genotypes, the phased haplotypes can be inferred by a number of existing methods [12,21,27,29]. The MHTP problem aims to find a minimum length haplotype which is defined by a subset of SNPs in C and is predictive of the alleles of SNP S_T. Fig. 1 illustrates an example for the MHTP problem. There are five haplotypes (i.e., $h_1, \ldots, h_5$) composed by six SNPs and the target SNP to be replaced is S_6. In this example, the haplotype (0,0) defined by SNPs S_1 and S_3 is predictive of SNP S_6, because haplotype (0,0) perfectly co-occurs with all minor alleles at SNP S_6 (but never with the major allele). As a consequence, only SNPs S_1 and S_3 only need to be typed for predicting the alleles at SNP S_6. That is, if haplotype (0,0) is observed at these two SNPs from a testing sample, we can predict that this sample contain minor allele at SNP S_6. Otherwise, the allele is predicted as the major type. We would like to note that the input may not always contain a feasible solution (i.e., no haplotypes can replace the target SNP). Then the target SNP will be selected as the tag SNP in our final solution (see Section 3).

In this paper, we say that this sort of haplotypes has "perfect LD" with the target SNP, which is close to the definition of perfect LD between two diallelic SNPs (i.e., $r^2=1$). A formal definition of the MHTP problem is given below.

> [*Input:*] a $k*(n+1)$ haplotypes matrix, where $C=\{S_1, S_2, \ldots, S_n\}$ represents a set of n SNPs and $S_T \notin C$ is the target SNP to be replaced.
> [*Output:*] a minimum subset of SNPs $C' \subset C$ which defines a haplotype having perfect LD with SNP S_T.

For the example in Fig. 1, the set of SNPs $C'=\{S_1, S_2, S_3\}$ is one feasible solution for MHTP but the minimum solution is $C'=\{S_1, S_3\}$, because the haplotype (0,0) composed of SNPs S_1 and S_3 perfectly co-occurs with all minor alleles at SNP S_6. In this paper, we also say that a set of SNPs C' can *replace* a SNP S_T if there exists a haplotype defined by SNPs in C' having perfect LD with SNP S_T.

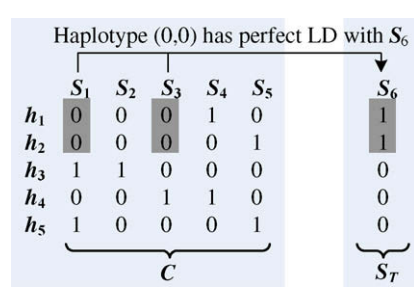

Fig. 1. An input example for the MHTP problem. $H=\{h_1, \ldots, h_5\}$, $C=\{S_1, \ldots, S_5\}$ and $S_T=S_6$. $\{S_1, S_2, S_3\}$ is a set of feasible solution but the minimum solution is $C'=\{S_1, S_3\}$, since the haplotype (0,0) defined by these two SNPs has perfect LD with SNP S_6.

Theorem 1. *The MHTP problem is NP-hard.*

Proof. We make a reduction from an NP-hard problem called *Set Cover* [15] to the MHTP problem with a reduction technique similar to Bafna et al. [3]. The set covering (SC) problem is defined as given a collection C of subsets of k elements, find a minimum subcollection $C' \subset C$ (called *set cover*) such that each element appears in at least one subset of C'. Given an instance of the SC problem, a collection $C = \{C_1, C_2, \ldots, C_n\}$ of elements $\{E_1, E_2, \ldots, E_k\}$, we construct an instance of the MHTP problem by first creating k haplotypes (h_1 to h_k) with n SNPs (S_1 to S_n). Each element $E_i \in C_j$ produces a major allele on haplotype h_i at SNP S_j and the remaining positions are all minor alleles. Then we construct an additional haplotype h_{k+1} with all minor alleles from SNPs S_1 to S_n. Finally, we construct the target SNP to be replaced S_{n+1} with major alleles only occurred in haplotypes h_1 to h_k and one minor allele occurred in haplotype h_{k+1}.

Example. Given an instance of the SC problem $C = \{C_1,\ C_2,\ C_3\}$ over elements $\{1,2,3,4,5\}$, where $C_1 = \{1,2,5\}$, $C_2 = \{3,4\}$, $C_3 = \{1,4\}$, we construct 6 haplotypes composed of 4 SNPs, where $h_1 = (0,1,0,0)$, $h_2 = (0,1,1,0)$, $h_3 = (1,0,1,0)$, $h_4 = (1,0,0,0)$, $h_5 = (0,1,1,0)$, and $h_6 = (1,1,1,1)$,

Note that the additional haplotype (i.e., h_{k+1}) contains all minor alleles at all SNP loci and the SNP to be replaced (i.e., S_{n+1}) has only one minor allele occurred in h_{k+1}. Thus, the haplotype h_{k+1} defined by SNPs S_1 to S_n has perfect LD with the minor allele at S_{n+1}. For the above example, the haplotype (1,1,1) defined by the first three SNPs indicates the occurrence of minor allele at SNP S_4 and thus SNP S_4 can be replaced by haplotype (1,1,1). We then show that the minimum set cover for SC implies a minimum subset of SNPs which can replace SNP S_{n+1} for MHTP, and vice versa.

Example. For the sample example, the minimum set cover for SC is $C' = \{C_1, C_2\}$, which covers all elements $E = \{1,2,3,4,5\}$. On the other hand, the minimum subset of SNPs which can replace SNP S_4 in the MHTP problem is $C^* = \{S_1, S_2\}$, since haplotype (1, 1) defined by SNPs S_1 and S_2 can indicate the occurrence of minor allele at SNP S_4 (i.e., haplotype (1, 1) has perfect LD with SNPs S_4).

Consequently, there exists a set cover of size k for the SC problem if and only if there exists a subset of SNPs of size k which can replace SNP S_{n+1} for the MHTP problem. Therefore, MHTP is NP-hard. □

2.2. An approximation algorithm for the MHTP problem

In this subsection, we describe an approximation algorithm which solves MHTP by first removing SNPs impossible to form a solution, reducing to an existing NP-hard problem, introducing a greedy algorithm, and finally presenting its extension for handling missing data. To simplify the presentation, the described algorithm focuses on identifying a haplotype which perfectly co-occurs with all minor alleles at SNP S_T. The algorithm for capturing major alleles is similar. This algorithm starts by removing SNPs impossible to form the solution using the following two steps.

Step 1. Identify the subset of haplotypes $H_T \subset H$ in which the minor alleles are observed at the target SNP S_T. The set of haplotypes containing the major alleles for SNP S_T is denoted as $\overline{H}_T$. For the example shown in Fig. 1, $H_T = \{h_1,\ h_2\}$ and $\overline{H}_T = \{h_3,\ h_4, h_5\}$.

Step 2. Identify the set of SNPs $C' \subseteq C$ which are *consistent* with SNP S_T. A SNP is said to be consistent to SNP S_T if either only major alleles or minor alleles (but not both) are observed in haplotypes in H_T. On the other hand, a SNP is said to be *inconsistent* with SNP S_T if it contains mixed major and minor alleles observed in haplotypes of H_T. For the example shown in Fig. 1, SNP S_4 is inconsistent with SNP S_6 because one major and one minor alleles are both observed in haplotypes h_1 and h_2, respectively.

The consistent SNPs in C' have either all major or all minor alleles observed in haplotypes in H_T. Consequently, the haplotypes in H_T defined by these consistent SNPs have only one pattern (e.g., {0,0,0} in Fig. 1). Then, for each consistent SNP $S_i \in C'$, we identify the set of haplotypes $H_i \subset \overline{H}_T$ in which the observed allele is complementary to those observed in H_T. For the example shown in Fig. 1, $H_1 = \{h_3, h_5\}$, $H_2 = \{h_3\}$, and $H_3 = \{h_4\}$. The algorithm proceeds by adopting a greedy method which iteratively selects a SNP $S_i \in C'$ that can incur complementary alleles for most haplotypes in $\overline{H}_T$ (i.e., $\max\{|H_i \cap \overline{H}_T|\}$), until all haplotypes in $\overline{H}_T$ contain at least one allele complementary to those in H_T. After running this algorithm, the haplotypes in H_T defined by SNPs in C' perfectly co-occur with all minor alleles at SNP S_T, and all haplotypes in $\overline{H}_T$ are distinguished from those in H_T. For example, in Fig. 1, the SNPs S_1 and S_3 are selected in order by this algorithm. If no feasible solutions exist, this algorithm will return a null symbol which implies no SNPs can replace the target SNP. The running time of this algorithm is bounded by $O(nk^2)$, where n is the number of SNPs and k is the number of haplotypes. A pseudo code of this algorithm (referred to as MHTagger) is given below.

Algorithm: MHTAGGER(C, S_T)

1 Construct H_T and $\overline{H}_T$ containing minor and major alleles for SNP S_T, respectively.
2 Identify the set of SNPs $C' \subseteq C$ which are consistent with SNP S_T.
3 For each SNP $S_i \in C'$, construct $H_i \subset \overline{H}_T$ containing alleles complement to those in H_T.
4 $R \leftarrow \phi$
5 **while** $\overline{H}_T \neq \phi$ **and** $C' \neq \phi$ **do**
6 Let S_j be the SNP $S_i \in C'$ that maximizes $|H_i \cap \overline{H}_T|$.
7 $\overline{H}_T \leftarrow \overline{H}_T - H_j$
8 $C' \leftarrow C' - S_j$
9 $R \leftarrow R \bigcup S_j$
10 **end of while**
11 **if** $\overline{H}_T \neq \phi$
12 **return** ϕ
13 **else**
14 **return** R.

Theorem 2. *The MHTagger algorithm gives a solution within a factor of $O(\log k)$ of the optimal solution.*

Proof. Note that Lines 1–3 reduce MHTP to an instance of the set-covering (SC) problem [15] and Lines 4–11 solve the instance of SC by a greedy algorithm. The greedy algorithm for solving the SC problem has been shown to have $O(\log n)$ approximation [7], where n is the number of elements to be covered. The number of elements (to be covered in the SC problem) corresponds to the number of haplotypes in $\overline{H}_T$ in the MHTP problem, where $|\overline{H}_T| < |H| = O(k)$. Therefore, the MHTagger algorithm also gives a solution of $O(\log k)$ approximation for the MHTP problem. □

2.3. Extension for handling missing data

In reality, a portion of SNPs may not be always typed successfully and these missing SNPs can greatly reduce the power of using

tag SNPs for association studies. For the example shown in Fig. 1, although the minimum solution is $C' = \{S_1, S_3\}$, we would fail to predict the allele at SNP S_6 if any of the two SNPs is missing. As pointed out previously [20], the negative effects from missing data can be avoided by selecting a slightly larger set of SNPs for genotyping. Consequently, we extend the MHTagger algorithm for tolerating a fixed amount of missing SNPs, because the missing rates of the genotyping array is usually limited (<∼10%).

However, there is a tradeoff between the number of tag SNPs and ability of tolerating missing data. In the following, we use the strict requirement which guarantees that if up to m tag SNPs are missing, it has no effects on predicting the alleles at the target SNP. With loose requirement, the number of tag SNPs can be reduced, but then we will not be able to make the correct prediction in all circumstances. The detailed discussion of tolerating missing data can be found in [20]. An extended definition of the MHTP problem for tolerating missing data is given below.

Input: a $k * (n+1)$ haplotypes matrix, where $C = \{S_1, S_2, \ldots, S_n\}$ represents a set of n SNPs and $S_T \notin C$ is the target SNP to be replaced; denote m as the number of missing SNPs to be tolerated.

Output: a minimum subset of SNPs $C' \subset C$ which defines a haplotype having perfect LD with SNP S_T, even when up to m SNPs in C' are missing.

The algorithm is briefly described below. Recall that H_T and $\overline{H}_T$ are two sets of haplotypes containing the minor or major alleles of SNP S_T, respectively. After Line 3 in the MHTagger algorithm, all haplotypes in H_T have the same pattern. The remaining steps (i.e., Lines 5–10) are revised for finding a minimum set of SNPs which defines a haplotype pattern having Hamming distance at least $(m+1)$ with each haplotype in $\overline{H}_T$, whereas the original algorithm only requires the Hamming distance to be at least one. Note that when m SNPs are missing, the Hamming distance between haplotypes in H_T and $\overline{H}_T$ decreases at most m and thus is at least equal or greater than one. Therefore, the haplotypes in H_T can still be distinguished from all haplotypes in $\overline{H}_T$, which still satisfies the requirement of perfect co-occurrence with all minor alleles at SNP S_T.

3. Algorithms for the selection of tag SNPs by predictive haplotypes

In this section, we study the problem of MTMH defined as follows: given a set of SNPs, find a minimum set of tag SNPs which defines a set of haplotypes completely predictive of the alleles of all untyped SNPs. The MTMH problem is divided into three subproblems which are separately solved in the following three stages: (1) find a minimum set of tag SNPs based on pairwise perfect LD between diallelic SNPs; (2) for each of the found tag SNP, identify a minimum length haplotype having perfect LD with the tag SNP by solving the MHTP problem; (3) select a minimum subset of tag SNPs which defines a set of haplotypes completely predictive of alleles of all removed tag SNPs. In the first stage, we describe a linear-time algorithm for finding a minimum set of tag SNPs based on pairwise perfect LD. The second stage iteratively solves the MHTP problem by setting each tag SNP as the target SNP to be replaced and running the MHTagger algorithm to find a haplotype predictive of the alleles at the target SNP (see Section 2). The last stage is shown to be another NP-hard problem and two algorithms are presented.

3.1. Stage 1: finding a minimum set of tag SNPs by pairwise perfect LD

The first stage of our algorithm solves the problem of finding a minimum set of tag SNPs based on pairwise perfect LD between diallelic SNPs, which is defined as follows: given a set of SNPs find a minimum subset of SNPs (called tag SNPs) such that each untyped SNP has perfect LD with some tag SNP. A generalization of this problem with arbitrary LD setting (non-perfect LD) has been shown to be NP-hard and numerous methods have been proposed [2,5,28]. Existing methods usually take $O(n^2k)$ time, where n is the number of SNPs and k is the number of haplotypes, due to the need of computing LD (r^2) between all pairs of SNPs. We observe that SNPs in perfect LD usually have identical 0/1 (major/minor alleles) encoding in all haplotype samples. Instead of explicitly computing r^2 for all pairs of SNPs, we consider SNPs with identical encoding to be perfect LD. Although this looks like a more stringent requirement, our experimental results indicate that the solution found by this method is the same as those found by other programs based on explicitly evaluating $r^2 = 1$ (see Table 1, Section 4). This is mainly due to the sufficiently large sample size in real data sets, which lead to different frequencies of major and minor alleles at each SNP. Thus, any two SNPs with $r^2 = 1$ have identical allele pattern when encoded into 0/1 representation. For example (see Fig. 2), SNPs S_1 and S_2 are in perfect LD and they contain the same allele pattern (0,0,1,0) observed at all haplotypes. Note that this stringent requirement satisfies various definitions of perfect LD (e.g., $r^2 = 1$, $D\prime = 1$, or no-four-gamete property).

The algorithm (FastPerfectLD) intrinsically uses a technique similar to the bucket sorting [7] to divide SNPs into bins of perfect LD and then select a tag SNP from each bin, which is briefly described below. The FastPerfectLD algorithm starts by scanning the first haplotype (e.g., h_1) and divide these SNPs into two groups

Table 1
Numbers of tag SNPs on ENCODE data sets numbers of tag SNPs found by HapTagger, FastPerfectLD, and Haploview on ten HapMap ENCODE data sets

ENCODE Region	ENr112	ENr131	ENr13	ENm010	ENm013
No. of SNPs	1157	1221	1393	618	1042
HapTagger	150 (7 s)	179 (6 s)	128 (5 s)	104 (4 s)	104 (4 s)
FastPerfectLD	424 (2 s)	307 (1 s)	4 (2 s)	307 (1 s)	264 (2 s)
Haploview (pairwise)	424 (5 min)	307 (4 min)	4 (5 min)	307 (4 min)	264 (5 min)
Haploview (aggressive)	*f*	*f*	*f*	265 (4 days)	*f*
	ENm014	ENr321	ENr232	ENr23	ENr213
No. of SNPs	1124	768	614	1052	796
HapTagger	127 (5 s)	136 (5 s)	136 (5 s)	110 (4 s)	132 (4 s)
FastPerfectLD	319 (2 s)	305 (1 s)	293 (1 s)	358 (2 s)	282 (1 sec)
Haploview (pairwise)	319 (6 min)	305 (4 min)	293 (4 min)	358 (5 min)	282 (5 min)
Haploview (aggressive)	*f*	*f*	257 (4 days)	*f*	*f*

The approximately elapsed time of each program for processing each data set is shown in parentheses.
f: fail to output a solution in ten days.

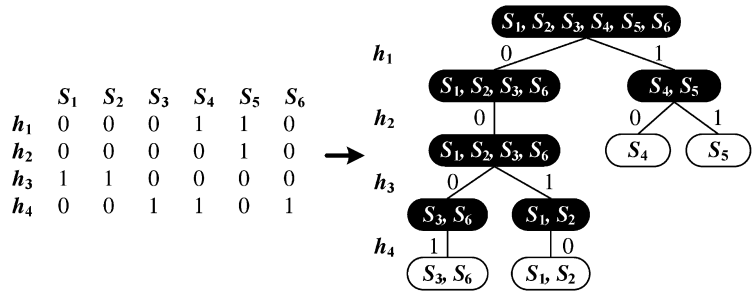

Fig. 2. An example for dividing SNPs into bins of perfect LD. The left-hand side is the input example which contains four haplotypes composed of six SNPs. The right-hand side illustrates each execution stage of this algorithm. The black nodes represent the intermediate groups of SNPs during the algorithm. The white nodes at the leaves of the tree represent the bins of SNPs having perfect LD with each other.

according to the major or minor alleles observed at this haplotype. The algorithm recursively divides SNPs in each group into subgroups according to the major and minor alleles observed in the next haplotype, until all haplotypes are tested. The black nodes in Fig. 2 illustrates the intermediate groups of SNPs during the execution of this algorithm. After finishing testing all haplotypes, each resulting group (i.e., the white node in Fig. 2) stands for a bin of SNPs having perfect LD with each other, whereas the SNPs in distinct bins do not have perfect LD. As a result, we can select any SNP from each bin as the tag SNP and construct a set of tag SNPs as the solution. Note that if we wish to tolerate m missing tag SNPs, we can select arbitrary $m+1$ SNPs from each bin as the solution. This algorithm only needs to test at most n SNPs for each of the k haplotypes. The running time of FastPerfectLD is thus bounded by $O(nk)$.

3.2. Stage 2: identifying a minimum length haplotype for replacing each tag SNP

The input of the second stage is the tag SNPs found in the first stage. The subproblem solved in the second stage is defined as follows: given a set of tag SNPs $C = \{S_1, \ldots, S_n\}$, for each tag SNP S_i $(1 \leqslant i \leqslant n)$, find a minimum set of SNPs from $C - S_i$ which defines a haplotype having perfect LD with the tag SNP S_i. In other words, this stage iteratively solves the MHTP problem (see Section 2) by setting each of the n SNPs as the target SNP to be replaced. We apply the MHTagger algorithm for solving MHTP to identify a subset of tag SNPs for replacing the target tag SNP S_i. If the MHTagger subroutine returns no feasible solution for a target SNP S_i, this irreplaceable tag SNP will be included in the final solution. Let R_i be the subset of tag SNPs found by the MHTagger algorithm which can replace tag SNP S_i. We formulate the dependency of replacement among all SNPs as a directed graph called "replacement graph" (Fig. 3). The vertices in the replacement graph represent each SNP and vertex S_i is connected to vertex S_j with a directed edge if $S_i \in R_j$. That is, SNP S_i is in the set of tag SNPs which can replace SNP S_j.

The replacement graph is the output of the second stage. Note that the replacement graph may contain cycles (e.g., S_4, S_5, and S_6 in Fig. 3). The following lemma describes an additional property of the replacement graph.

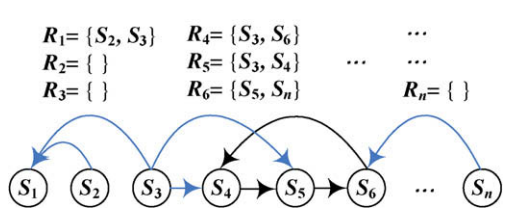

Fig. 3. The replacement graph defined by tag SNPs S_1 to S_n. Note that there exists a cycle defined by SNPs S_4, S_5, and S_6.

Lemma 1. *Each vertex in the replacement graph has at most $k-1$ incoming edges.*

Proof. Recall that the incoming edges in the replacement graph are resulted from the output of running the MHTagger algorithm.

The worst case of the MHTagger algorithm takes place when the target SNP to be replaced S_T has only one haplotype carrying the minor allele at this locus, and all other haplotypes carrying major alleles. Thus, we have to select a set of SNPs which produces complementary alleles for remaining $(k-1)$ haplotypes. Note that the greedy manner of MHTagger guarantees that each selected SNP incurs at least one complementary allele for those $(k-1)$ haplotypes. Therefore, MHTagger outputs at most $k-1$ SNPs as the solution. As a consequence, there are at most $k-1$ incoming edges produced for each vertex in the replacement graph. □

3.3. Stage 3: reserving a minimum subset of tag SNPs based on the replacement graph

The input of the last stage is the replacement graph produced in the second stage. Denote the set of all tag SNPs in the replacement graph as C. The replacement graph gives us the information as to which tag SNPs in C can be replaced. Hence, we can select a minimum subset of tag SNPs $C' \subseteq C$ such that the alleles of each removed tag SNP (i.e., $C - C'$) can be predicted by a haplotype defined by tag SNPs in C'. However, not all SNPs can be safely removed because tag SNPs of these removed SNPs may be also removed (e.g., SNPs S_4, S_5, and S_6 in Fig. 3). That is, the alleles of these dependent SNPs can not be completely reconstructed if all of them are removed from the final solution. The Haploview resolves this problem by sequentially removing a tag SNP on the basis of the remaining SNPs in a peel-back manner [10]. However, the tag SNPs removed in the early stage could be used to replace more tag SNPs, and this global dependent relation is not considered. In the last stage, we introduce an improved algorithm which considers the overall dependency among all tag SNPs and selects a smaller set of tag SNPs based on the replacement graph as the final solution. In the following, we describe two lemmas regarding the set of tag SNPs which can be safely removed.

Lemma 2. *A tag SNP with incoming edges can be safely removed if it is not contained within a cycle in the replacement graph.*

Proof. A tag SNP with incoming edges in the replacement graph implies that there exist some other tag SNPs which can replace it. For example, SNP S_1 in Fig. 3 can be directly removed from the final solution since it can be replaced by SNPs S_2 and S_3. On the other hand, if the tag SNP is contained in a directed cycle, it can not be

safely removed, because each SNP is dependent on others in the cycle for predicting its alleles. For example, SNPs S_4, S_5, S_6 form a directed cycle. If all of them are removed, we can not reconstruct the alleles of these SNPs even though we type all other tag SNPs. □

Lemma 3. *For each cycle, only one tag SNP needs to be kept while the other tag SNPs in this cycle can be safely removed, if they are not contained in other cycles.*

Proof. If a tag SNP within a cycle is kept, we can remove its incoming edges from the graph since the allele of this SNP will be directly typed and known. Therefore, the cycle can be broken and becomes a simple path, if the remaining tag SNPs are not contained in other cycles. By Lemma 2, the remaining tag SNPs in this simple path can now be safely removed since all of them have incoming edges and are not contained in any cycle. □

By Lemmas 2 and 3, we have to reserve at least one tag SNP in each cycle and to remove its incoming edges from the replacement graph. Note that the outgoing edges cannot be removed. Otherwise, we will fail to reconstruct the alleles of untyped tag SNPs. Recall that MTMH asks for a minimum set of tag SNPs as the final solution. Therefore, the last stage is solving a problem (referred to as MTSR) defined as follows: given a replacement graph, find a minimum set of vertices such that the removal of their incoming edges breaks all cycles in the replacement graph.

Theorem 3. *The MTSR problem is NP-hard.*

Proof. Without loss of generality, we assume that the number of incoming edges of each vertex in the replacement graph is bounded by an integer k (see Lemma 1). We make a reduction from a variant of the vertex cover problem referred to as $k-$VC [15,25]. The $k-$VC problem is known to be NP-hard and is defined as follows: given a graph G=(V,E) with degrees bounded by an integer $k \geqslant 3$, find a minimum subset of vertices $V' \subseteq V$ (called *vertex cover*) such that each edge $(u,v) \in E$ has at least one of u and v belonging to V'.

Given an instance of the $k-$VC problem, we construct a new graph $\widetilde{G} = (U_v \cup U_e, \tilde{E})$, where vertices of $U_v = V$ correspond to original vertices of G and vertices of U_e correspond to each edge of G. For each edge $e = (v_1, v_2)$ in G, we construct three edges in $\widetilde{G}$ which form a directed cycle: an edge from $(v_1$ to $v_2)$, an edge from v_2 to e, and an edge from e to v_1. Note that since the degree of each vertex in G is bounded by k, there are at most k directed cycles created for each vertex in $\widetilde{G}$, which produces at most k incoming edges. It is easy to observe that a vertex cover in G implies a set of vertices in $\widetilde{G}$ which can break all cycles by removing their incoming edges, because each edge in G corresponds to one cycle in $\widetilde{G}$. Therefore, G has a minimum vertex cover of size c if and only if $\widetilde{G}$ has a minimum set of vertices of size c such that the removal of their incoming edges breaks all cycles. In summary, MTSR is NP-hard. □

The MTSR problem can be solved by reducing to an NP-hard problem called "minimum feedback vertex set" (MFVS) [15]. The MFVS problem is defined as given a directed graph $G = (V, E)$, find a minimum subset of vertices $V' \subseteq V$ such that V' contains at least one vertex for every directed cycle in G. Let V' be a minimum solution of the MFVS problem. Note that each vertex in a directed cycle of G has one incoming and one outgoing edges both contained in this cycle. The removal of incoming edges of all vertices in V' can also break all cycles in G, which implies that V' is also a minimum solution of MTSR. Therefore, MTSR can be solved by applying existing algorithms for MFVS.

The best approximation algorithm for the MFVS problem gives a solution within a factor of $O(\log v * \log\log v)$ of the optimal solution, where v is the number of vertices [13]. However, it requires solving a linear-programming instance with exponential number of constraints, which is impractical when applying on genome-wide data sets with millions of SNPs. To efficiently break all cycles in the replacement graph, we turn to solve a relaxed problem which asks for a minimum subset of vertices such that the removal of their incoming edges eliminates all *back edges* in the replacement graph. An edge (u,v) connecting from vertex u to vertex v is said to be a back edge if vertex v is the ancestor of vertex u in the depth-first-search (DFS) traversal of the graph [7]. Note that the DFS traversal in a graph produces back edges if and only if the graph has cycles, which implies that all cycles can be indirectly broken by removing all back edges. Consequently, the solution of this relaxed problem is a feasible solution to MTSR.

This relaxed problem can be solved in polynomial time since each back edge (u,v) uniquely corresponds to one incoming edge of the vertex v and we require that only incoming edges of a vertex can be removed. In the following, we describe an algorithm which removes all back edges in the replacement graph by revising the DFS algorithm. During the DFS traversal, all incoming edges of a vertex v are removed once a back edge (u,v) connecting from a descendant vertex u to an ancestor vertex v is found. Therefore, the removal of incoming edges of vertex v eliminates the back edge (u,v) and indirectly breaks cycles associated with this back edge. We repeat this process until all back edges in this replacement graph are found and removed.

We integrate this algorithm with the FastPerfectLD and MHTagger algorithms introduced in previous subsections and develop a program called HapTagger to solve all subproblems of MTMH in three stages. After the last stage, the set of vertices in the replacement graph without incoming edges indicate those tag SNPs of the output. A pseudo code of HapTagger is given below.

Algorithm: HAPTAGGER (C)

```
1   Run FastPerfectLD to find a minimum set of tag SNPs C' ⊆ C
2   Construct a replacement graph G with vertices corresponding to SNPs in C'
3   for each SNP S_i ∈ C'
4     Run MHTagger(C' − S_i, S_i) to obtain a set of tag SNPs R_i which can replace S_i
5     Add directed edges connecting from vertices in R_i to S_i in the replacement graph G
6   end of for
7   for each vertex S_i in G
8     Conduct a DFS traversal starting from vertex S_i
9       if a back edge (u, v) is found during the traversal
10        remove all incoming edges of vertex v from G
11  end of for
12  Identify the set of vertices T in G without incoming edges
13  Return T.
```

The time complexity of HapTagger is analyzed as follows. Line 1 is bounded by the FastPerfectLD algorithm which takes $O(nk)$. Lines 3–6 take $O(n^2k^2)$ time since for each of the n SNPs, we have to run the MHTagger algorithm which takes $O(nk^2)$. Lines 7–11 is bounded by the time of running DFS to traverse the entire graph, which takes $O((V+E)) = O(n^2)$ time, where V and E are the numbers of vertices and edges, respectively. Therefore, the entire HapTagger algorithm runs in $O(n^2k^2)$ time.

4. Experimental results

We implement the HapTagger algorithm in JAVA for finding tag SNPs by multimarker haplotypes. HapTagger is freely available on

http://www.csie.ntu.edu.tw/~kmchao/HapTagger/. Due to the inefficiency of pairwise-LD based methods, the FastPerfectLD algorithm in Section 3 is separately implemented and used as a reference for the solutions of pairwise-LD based approaches. These programs along with the official tagging tool Haploview [10] used by the International HapMap project are tested on a variety of real data sets. The Haploview can identify tag SNPs in two modes: pairwise or aggressive modes. The Haploview in pairwise mode finds tag SNPs only based on pairwise LD between two diallelic SNPs. The Haploview in aggressive mode identifies tag SNPs by first finding tag SNPs using pairwise LD and then reduce the tag SNPs using a peel-back approach with a multiallelic LD statistic. In the following experiments, the minimum LD threshold required for Haploview in both modes is set to 1.0. We download the phased haplotype data from HapMap and choose the population of Utah residents with ancestry from northern and western Europe (CEU) as our experimental target [1]. The following experiments are conducted under the hardware environment with Pentium 3.2 GHz CPU and with 8 GB RAM.

4.1. Experiments on HapMap ENCODE data sets

We first test these programs on ten ENCODE data sets (corresponding to ten 500-kilobase regions) resequenced and genotyped in the HapMap project. Each data set contains 180 haplotype samples originated from 30 CEU trios. Table 1 lists the number of tag SNPs found by HapTagger, FastPerfectLD, and Haploview (in pairwise or aggressive modes) on each ENCODE data set. The Haploview in aggressive mode fails to output a solution in most data sets within a reasonable period of time (e.g., longer than ten days). The results indicate that HapTagger consistently finds a smaller set of tag SNPs with size less than half of other programs. The FastPerfectLD and Haploview in pairwise mode identify the same number of tag SNPs in all data sets. Recall that FastPerfectLD requires that SNPs in perfect LD have identical allele pattern, which is more stringent than the requirement of $r^2 = 1$ used by Haploview in pairwise mode. However, the results indicate that this stringent requirement produces the same number of tag SNPs as Haploview in pairwise mode. This phenomenon is mainly due to the sufficiently large samples in HapMap data and SNPs with $r^2 = 1$ all have identical allele pattern when encoded into 0/1 representation. The Haploview in aggressive mode outperforms FastPerfectLD and pairwise mode as expected, because it further refines the solution (i.e., the solutions of pairwise mode) by multimarker haplotypes. However, the improvement is not significant in comparison with HapTagger. The reason is that Haploview has several default constraints (e.g., allowing up to three SNPs in a haplotype and testing at most 10,000 haplotypes) which prevent it from finding a better solution. It should be noted that the Haploview in aggressive mode fails to output a solution in all data sets within a reasonable period of time when we relax all of its default constraints.

In terms of efficiency, FastPerfectLD is the fastest program because of the internal linear-time algorithm. The HapTagger is able to output a solution in several seconds, whereas Haploview in pairwise mode requires a bit longer time (from four to six minutes). Although the theoretical time complexity of these two programs are the same (i.e., $O(n^2)$), the HapTagger internally employs the FastPerfectLD algorithm to group SNPs in perfect LD and avoid the heavy computation of r^2 values for each pair of SNPs. Thus, it is slightly faster than Haploview in pairwise mode in practice. The Haploview in aggressive mode is the slowest program which takes at least four days for outputting a solution.

4.2. Experiments on HapMap chromosome data sets

We then test these programs on a number of large genome-wide data sets. We download 22 phased haplotype data sets corresponding to human Chromosome 1 to Chromosome 22 from the Phase I release of HapMap data. The Haploview in aggressive mode fails to output a solution for all data sets within a reasonable period of time. The Haploview in pairwise mode also fails to output a solution due to out-of-memory error when we relax all the default constraints but is able to output a solution when all default constraints are retained. Thus we only report the results of Haploview in pairwise mode with its default constraints retained. The Haploview in pairwise mode takes one to four hours to finish processing each data set. The HapTagger returns a solution from one to two hours for each data set. The FastPerfectLD is the fastest program which returns a solution only in several minutes on all data sets.

Table 2 lists the number of tag SNPs found by each program. The HapTagger consistently outperforms Haploview and FastPerfectLD on all data sets as expected, since it further considers the LD between a diallelic SNP and a multimarker haplotype instead of only pairwise LD between diallelic SNPs. The FastPerfectLD also consistently outperforms Haploview in pairwise mode since its solution is not restricted by any constraint. The number of tag SNPs found by HapTagger is less than half of the numbers outputted by Haploview or FastPerfectLD. In summary, the HapTagger only requires 23% of original SNPs to be typed as tag SNPs, whereas the tag SNPs identified by Haploview and FastPerfectLD account for roughly 60% of original SNPs. Most tagging programs reduce the number of tag SNPs by relaxing the threshold of LD (e.g., $r^2 \geqslant 0.8$). It is worth mentioning that HapTagger not only runs faster but also reduces the number of tag SNPs under the requirement of perfect LD.

Table 2
Numbers of tag SNPs found by Hap Tagger, FastPerfectLD, and Haploview on HapMap chromosome data sets

Chromosome No.	Chr1	Chr2	Chr3	Chr4	Chr5	Chr6	Chr7	Chr8
No. of SNPs	61814	69753	56737	48952	48831	53458	41046	60234
HapTagger	14660	14819	12325	11055	11075	11894	9919	10641
FastPerfectLD	37526	40262	33234	28761	28810	30884	25469	30161
Haploview (pairwise)	38731	40993	33860	29332	293	31663	26077	30589
	Chr9	Chr10	Chr11	Chr12	Chr13	Chr1 4	Chr15	Chr16
No. of SNPs	47682	38940	36287	39189	28816	24128	21138	19922
HapTagger	10066	10207	8325	9462	6920	5957	5512	5733
FastPerfectLD	27448	24189	21650	24149	17412	14788	13721	13914
Haploview (pairwise)	27903	24790	22055	24782	17799	15016	13959	14169
	Chr17	Chr18	Chr19	Chr20	Chr21	Chr22	Total	
No. of SNPs	19767	32177	14175	17096	16199	15548	811889	(100%)
HapTagger	5255	6235	4254	4885	3689	3893	186781	(23%)
FastPerfectLD	12647	16820	9694	11493	9190	9417	481639	(59%)
Haploview (pairwise)	12870	17066	9849	11697	9247	9513	491283(61%)	

Haploview in aggressive (multimarker) mode failed to output a solution within a reasonable period of time.

5. Discussion

5.1. Reconstruction of alleles of untyped SNPs with HapTagger

In previous methods, the alleles of an untyped SNP can be directly reconstructed by a typed tag SNP. But in HapTagger, the alleles of an untyped SNP have to be reconstructed in a more complex manner. It is because each untyped SNP is now predicted by a haplotype instead of by a single tag SNP, and the predictive haplotype itself may also contain partial untyped SNPs. In other words, we have to resolve the dependency among all SNPs in order to reconstruct the alleles of all untyped SNPs. Nevertheless, we can simply apply the algorithm of topological sorting [7] to obtain the dependency ordering among all SNPs based on the replacement graph introduced in Section 3. Given an acyclic directed graph, the topological sorting algorithm sorts a vertex S_i precedent to a vertex S_j if there is a directed edge from S_i to S_j and finally gives a linear ordering of these vertices. Note that the replacement graph is also acyclic because we have broken all cycles after the last stage. Consequently, we can reconstruct alleles of all untyped SNPs one by following the linear ordering of these SNPs, which takes $O(n^2)$ time.

5.2. An improved algorithm for breaking cycles in the replacement graph

The previous algorithm for breaking cycles by removing all back edges may fail to obtain the optimal solution. Fig. 4(A) illustrates an example in which the previous algorithm may not perform well. If the DFS traversal starts from vertex S_1 (instead of vertex S_5), we would obtain four back edges (i.e., b_1, b_2, b_3, and b_4) and use four SNPs (i.e., S_1, S_2, S_3, and S_4) to remove all back edges. However, the optimal solution is the SNP S_5 since the removal of its incoming edge also breaks all cycles in this graph, even though this edge is not a back edge.

In order to overcome the limitation of only removing back edges, we consider the removal of other edges which can also break cycles. Although we do not explicitly enumerate all cycles in the replacement graph, each back edge is implicitly associated with some cycles and the removal of this back edge can break these associated cycles. The following lemma indicates that the cycles associated with one back edge can be broken by removing incoming edges of either vertex on this back edge.

Lemma 4. For a directed back edge (u, v), the removal of incoming edges of either vertex u or of vertex v breaks the same set of cycles associated with this back edge.

Proof. Denote the set of cycles broken by removing the back edge (u, v) as $C_{u,v}$. The removal of incoming edges of v removes the back edge (u, v) and thus breaks all cycles in $C_{u,v}$. Note that each cycle in $C_{u,v}$ must contain an edge which is also the incoming edge of vertex u, since it has to pass through vertex u to v. As a consequence, the removal of incoming edges of vertex u also breaks all cycles in $C_{u,v}$. □

By Lemma 4, we can select any of the two vertices on each back edge and remove its incoming edges to break cycles associated with this back edge. Denote the set of vertices at two ends of all back edges as $C = \{S_1, \ldots, S_n\}$ (Fig. 4(B)). The problem is redefined as follows: given a set of back edges $B = \{b_1, \ldots, b_m\}$ discovered during DFS traversal of the replacement graph, find a minimum set of vertices $C' \subseteq C$ such that C' contains at least one vertex from either end of a back edge. The removal of all incoming edges of vertices in C' can thus break all cycles in the replacement graph. However, this problem becomes NP-hard, which can be easily shown by a reduction from the $k-$VC problem similar to the proof in Theorem 3. On the positive side, this problem is just an instance of the set covering problem which asks for a minimum subcollection $C' \subseteq C$ such that each element in B is covered by at least one set in C'. Therefore, we can employ a typical greedy algorithm which iteratively selects a vertex shared by most back edges, until all back edges have at least one vertex (from either end) selected. For example, in Fig. 4, only SNP S_5 is selected by this greedy approach as the solution. Furthermore, it is easy to observe that each $b_i \in B$ appears in exactly two sets in C corresponding to its two end vertices. Therefore, this is a restricted version of the set covering problem with each element in B appears in two sets in C, which is shown to be APX-hard [25] and can be approximated within a factor of 2 of the optimal solution [18].

5.3. Efficiency of various LD statistic

A number of measures for computing the LD between two diallelic SNPs have been widely used for the selection of tag SNPs (e.g., r^2, D', or four-gamete property) [5]. On the other hand, only a few studies consider the LD between a diallelic SNP and a multiallelic haplotype for selecting tag SNPs (e.g., multiallelic D' or the relative information [11]). One major difference between these two directions is the number of tests required for obtaining a predictive SNP or a predictive haplotype. For example, on the basis of LD between diallelic SNPs, we can obtain a SNP which is predictive of another SNP S_T by computing the correlation coefficient (r^2) between SNP S_T and all other SNPs, which takes $O(n)$ time, where n is the number of SNPs. On the other hand, to obtain a haplotype predictive of SNP S_T, one has to compute the multiallelic LD statistic between a SNP and all possible haplotypes. However, the number of all possible haplotypes grows exponentially with respect to the number of SNPs, because a haplotype can be composed by arbitrary combination of SNPs. On the contrary, HapTagger implicitly estimates the multiallelic LD between a SNP and a haplotype using a combinatorics approach but does not rely on any explicit LD sta-

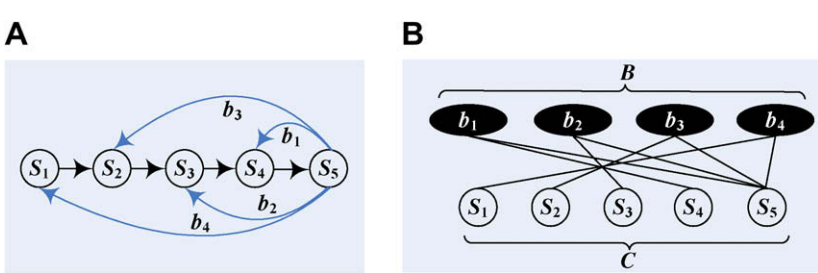

Fig. 4. An example of a replacement graph composed by five tag SNPs (i.e., $S_1, \ldots, S_5$). The DFS traversal starts from SNP S_1 and results in four back edges (i.e., $b_1, \ldots, b_4$). (B) The back edges and vertices are reformulated to elements and sets in a set covering problem, respectively. The element b_i is covered by the set S_j if S_j is one vertex on the back edge b_i.

tistic. The major advantage of our approach is that approximation algorithms are allowed for efficiently finding the predictive haplotypes. As indicated by our theoretical and experiment results, HapTagger runs much faster than other methods since it avoids the test of exponential number of haplotypes.

5.4. Signatures of positive selection or co-evolution

Recent large-scale analysis of recent strong selection using the HapMap data indicates that humans are still under fast evolution [31]. The classical signature of recent positive selection is the elevating-haplotype homozygosity surrounding the favored allele at one SNP (i.e., genetic hitchhiking). The haplotypes flanking the favored allele at one SNP locus usually show very low sequence diversity. Therefore, it is natural that HapTagger will identify haplotypes predictive of alleles at one SNP under recent positive selection. As to alleles not at close loci, they might still co-evolve through the heredity due to their functional dependency in the biological pathway. Thus HapTagger is also able to identify haplotypes of coevolving alleles at several SNP loci. The tag SNPs selected by previous LD-based methods only reflect the extent of past chromosome recombination. It is worth mentioning that the predictive haplotypes selected by HapTagger is not only used for capturing untyped SNP alleles, but these haplotypes may also indicate the signature of recent positive selection or co-evolution.

However, the length of haplotypes under positive selection or co-evolution will be reduced because chromosome recombination will break the linkage of SNPs in these haplotypes. Thus, HapTagger seeks for the minimum-length haplotype for replacing a target SNP in the algorithm. In addition, the requirement of minimum-length haplotype is helpful in the stage 3 of our algorithm, because the dependency (i.e., edges in the replacement graph) among these SNPs can be reduced and less cycles would be generated. Consequently, the number of tag SNPs is expected to be reduced by this requirement.

6. Conclusion

In this paper, we designed and implemented algorithms for the selection of tag SNPs using multimarker haplotypes without relying on any explicit multiallelic LD statistic. The tag SNPs found by our algorithms define a set of haplotypes completely predictive of the alleles of all untyped SNPs. Several exact and approximation algorithms are proposed to efficiently find these tag SNPs. We integrated these algorithms and implemented a program called HapTagger. Our theoretical analysis and experimental results indicated that HapTagger consistently identifies a smaller set of tag SNPs and runs much faster than existing methods on a variety of real data sets. We also discussed the efficiency of various LD statistic and compared distinct approaches for reconstructing untyped SNP alleles. It is worth mentioning that these predictive haplotypes selected by HapTagger may be the signature of positive selection or co-evolution.

Acknowledgments

Yao-Ting Huang and Kun-Mao Chao were supported in part by NSC Grants 93-2213-E-002-029, 94-2213-E-002-091, and 96-2218-E-194-006 from the National Science Council, Taiwan.

References

[1] Altshuler D, Brooks LD, Chakravarti A, Collins FS, Daly MJ, Donnelly P. A haplotype map of the human genome. Nature 2005;437:1299–320.
[2] Ao SI, Yip K, Ng M, Cheung D, Fong PY, Melhado I, et al. CLUSTAG: hierarchical clustering and graph methods for selecting tag SNPs. Bioinformatics 2004;21(8):1735–6.
[3] Halldórsson BV, Schwartz R, Clark AG, Istrail S. Haplotypes and informative SNP selection algorithms: don't block out information. In: Proceedings of the RECOMB'03; 2003. p. 19–27.
[4] Barrett JC, Fry B, Maller J, Daly MJ. Haploview: analysis and visualization of LD and haplotype maps. Bioinformatics 2005;21(2):263–5.
[5] Carlson CS, Eberle MA, Rieder MJ, Yi Q, Kruglyak L, Nickerson DA. Selecting a maximally informative set of single-nucleotide polymorphisms for association analyses using linkage disequilibrium. Am J Hum Genet 2004;74:106–20.
[6] Chang C-J, Huang Y-T, Chao K-M. A greedier appraoch for finding tag SNPs. Bioinformatics 2006;22:685–91.
[7] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. The MIT Press; 2001.
[8] Crawfod DC, Nickerson DA. Definition and clinical importance of haplotypes. Annu Rev Med 2005;56:303–20.
[9] Daly MJ, Rioux JD, Schaffner SF, Hudson TJ, Lander ES. High-resolution haplotype structure in the human genome. Nat Genet 2001;29(2):229–32.
[10] de Bakker PI, Yelensky R, Pe'er I, Gabriel SB, Daly MJ, Altshuler D. Efficiency and power in genetic association studies. Nat Genet 2005:1217–23.
[11] de Bakker PI, McVean G, Sabeti PC, Miretti MM, Green T, et al. A high-resolution HLA and SNP haplotype map for disease association studies in the extended human MHC. Nat Genet 2006:1166–72.
[12] Douglas JA, Boehnke M, Gillanders E, Trent JM, Gruber SB. Experimentally-derived haplotypes substantially increase the efficiency of linkage disequilibrium studies. Nat Genet 2001;28(4):361–4.
[13] Even G, Naor J, Schieber B, Sudan M. Approximating minimum feedback sets and multicuts in directed graphs. Algorithmica 1998;20:151–74.
[14] Gabriel SB, Schaffner SF, Nguyen H, Moore JM, Roy J, Blumenstiel B. The structure of haplotype blocks in the human genome. Science 2002;296(5576):2225–9.
[15] Garey MR, Johnson DS. Computers and intractability. New York: Freeman; 1979.
[16] Helmuth L. Genome research: map of the human genome 3.0. Science 2001;293(5530):583–5.
[17] Hinds DA, Stuve LL, Nilsen GB, Halperin E, Eskin E, Ballinger DG, et al. Whole-genome patterns of common DNA variation in three human populations. Science 2005;307:1072–9.
[18] Houchbaum DS. Approximation algorithms for the set covering and vertex cover problems. SIAM J Comp 1982;11:555–6.
[19] Hu N, Wang C, Hu Y, Yang HH, Giffen C, Tang Z-Z, et al. Genome-wide asspciation study in esophageal cancer using genechip mapping 10 K array. Cancer Res 2005;65(7):2542–6.
[20] Huang Y-T, Zhang K, Chen T, Chao K-M. Selecting additional tag SNPs for tolerating missing data in genotyping. BMC Bioinform 2005;6:263.
[21] Huang Y-T, Chao K-M, Chen T. An approximation algorithm for haplotype inference by pure parsimony. J Comput Biol 2005;12:1261–74.
[22] Kennedy GC, Matsuzaki H, Dong S, Liu WM, Huang J, Liu G, et al. Large-scale genotyping of complex DNA. Nat Biotechnol 2003;21:1233–7.
[23] Lin S, Chakravarti A, Cutler DJ. Exhaustive allelic transmission disequilibirium tests as a new approach to genome-wide association studies. Nat Genet 2004;36:1181–8.
[24] Liu WM, Di X, Yang G, Matsuzaki H, Huang J, Mei R, et al. Algorithms for large scale genotyping microarrays. Bioinformatics 2003;19:2397–403.
[25] Papadimitriou CH, Yannakakis M. Optimization, approximation, and complexity classes. J Comput System Sci 1991;43:425–40.
[26] Patil N, Berno AJ, Hinds DA, Barrett WA, Doshi JM, Hacker CR, et al. Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. Science 2001;294:1719–23.
[27] Qin Z, Niu T, Liu J. Partitioning-ligation-expectation-maximization algorithm for haplotype inference with single-nucleotide polymorphisms. Am J Hum Genet 2002;71:1242–7.
[28] Qin ZS, Gopalakrishnan S, Abecasis GR. An efficient comprehensive search algorithm for TagSNP selection using linkage disequilibirium criteria. Bioinformatics 2006;99(11):7335–9.
[29] Stephens M, Donnelly P. A comparison of Bayesian methods for haplotype reconstruction from population genotype data. Am J Hum Genet 2003;73:1162–9.
[30] Stram DO, Haiman CA, Hirschhorn JN, Altshuler D, Kolonel LN, Henderson BE, et al. Choosing haplotype-tagging SNPs based on unphased genotype data using a preliminary sample of unrelated subjects with an example from the multiethnic cohort study. Hum Hered 2003;55:27–36.
[31] Voight BF, Kudaravalli S, Wen X, Pritchard JK. A map of recent positive selection in the human genome. PLoS Biol 2006:446–58.
[32] Weal ME, Depondt C, Macdonald SJ, Smith A, Lai PS, Shorvon SD, et al. Selection and evaluation of tagging SNPs in the neuronal-sodium-channel gene SCN1A: implications for linkage diequilibrium gene mapping. Am J Hum Genet 2003;73:551–65.
[33] Zhang K, Sun F, Waterman MS, Chen T. Haplotype block partition with limited resources and applications to human chromosome 21 haplotype data. Am J Hum Genet 2003;73:63–73.
[34] Zhang K, Qin ZS, Liu JS, Chen T, Waterman MS, Sun F. Haplotype block partitioning and tag SNP selection using genotype data and their applications to association studies. Genome Res 2004;14(5):908–16.