最大节约原则下单倍型推导问题的实用算法*

张强锋[1+], 车皓阳[2], 陈国良[1], 孙广中[1]

[1](中国科学技术大学 计算机科学技术系,安徽 合肥 230027)

[2](中国科学院 软件研究所,北京 100080)

A Practical Algorithm for Haplotyping by Maximum Parsimony

ZHANG Qiang-Feng[1+], CHE Hao-Yang[2], CHEN Guo-Liang[1], SUN Guang-Zhong[1]

[1](Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

[2](Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: +86-551-3603747, E-mail: qfzhang@mail.ustc.edu.cn

Received 2004-05-31; Accepted 2004-11-15

Zhang QF, Che HY, Chen GL, Sun GZ. A practical algorithm for haplotyping by maximum parsimony *Journal of Software*, 2005,16(10):1699−1707. DOI: 10.1360/jos161699

Abstract: Haplotypes, rather than genotypes are required in some disease susceptibilities and drug response tests. However, it is both time-consuming and expensive to obtain haplotypes experimentally. Therefore usually genotype data are collected in the laboratory at first, then, haplotype data are inferred from them resorting to some computational approaches. Different from Clark's well-known haplotype inference method, Gusfield and Wang *et al*. proposed a new model according to the maximum parsimony principle. It tries to find a minimum set of haplotypes that can explain the genotype samples. This parsimony model overcomes some weaknesses of Clark's method. For the parsimony this paper presents model a polynomial time greedy algorithm and a compound algorithm that combines the greedy policy with the branch-and-bound strategy in a uniform framework. Compared with the original complete algorithm proposed by Wang *et al*., the greedy approximation algorithm runs much faster, and in the meanwhile, produces relatively higher accurate results. The compound algorithm is also a complete algorithm. Simulation results show that it is much more efficient and can be applied to instances of much larger scales than the original complete algorithm.

Key words: genotype; haplotype; SNP; haplotyping; maximum parsimony; greedy algorithm

* Supported by the National Grand Fundamental Research 973 Program of China under Grant No.G1998030401 (国家重点基础研究发展规划(973))

ZHANG Qiang-Feng was born in 1979. He received his BS degree in computer science (CS) from University of Science and Technology of China (USTC) in 2000. Now he is a Ph.D. candidate in CS at USTC. His research areas are bioinformatics, combinatorial optimization, P2P networks. **CHE Hao-Yang** was born in 1977. He is a Ph.D. candidate at the Institute of Software, the Chinese Academy of Sciences. His research areas are optimization algorithms, recommendation system, trust management, etc. **CHEN Guo-Liang** was born in 1938. He is a fellow of Chinese Academy of Sciences, a professor in the Department of CS at USTC, and a CCF senior member. His research areas are parallel computing, computer architecture and combinatorial optimization. **SUN Guang-Zhong** was born in 1978. He received the B.S. degree in CS from USTC in 2000. Now he is a Ph.D. candidate in CS at USTC. His current research areas are scheduling problems and combinatorial optimization.

摘 要: 在疾病的易感基因研究和药物反应实验中,常常需要知道单倍型,而不仅仅是基因型数据.但是直接通过生物学实验手段来测定单倍型在时间和成本上消耗过大,所以在实验室里往往仅测得基因型,而通过一些计算手段来推导出单倍型.不同于 Clark 著名的单倍型推导模型,Gusfield 和 Wang 等人提出了一种通过基因型样本推导单倍型的新模型.这种模型试图按照最大节约原则去寻找可以解释基因型样本的最小单倍型集合.这种基于节约原则的模型克服了 Clark 模型的一些缺陷.提出了节约原则模型的一个多项式时间的贪心算法以及一种把贪心策略和分支限界策略集合在统一框架下的复合算法.相对于 Wang 原来提出的分支限界完全算法,贪心的近似算法运行快得多,而且同时保持了比较准确的推导结果.新的复合算法也是一种完全算法.实验结果表明,与原来的分支限界算法相比,复合算法可以极大地提高运行效率以及可应用的实例规模.

关键词: 基因型;单倍型;SNP;单倍型推导;最大节约原则;贪心算法
中图法分类号: TP301 文献标识码: A

The modeling of human genetic variation is critical to the understanding of the genetic basis for complex diseases. Single nucleotide polymorphisms (SNPs) are the most frequent form of this variation. Supported by the Human Genome Project (HGP), dense human SNP maps are currently under construction. Then polymorphism screens in a population can be carried out to find genes related to disease susceptibilities and drug responses according to the dense SNP maps.

Humans are diploid. In diploid organisms there are two copies of each chromosome. The data from a single copy of the interested region are depicted as a *haplotype*, while the conflated data on the two copies of the interested regions are depicted as a *genotype*. Sometimes haplotype data rather than genotype data are required to give more detailed information. However, it is both time-consuming and expensive to examine the two copies separately using biological methods. Therefore usually blended genotype data are obtained in the laboratory, from which haplotype information is inferred resorting to some other kinds of tools. The way to infer haplotype data using computer programs is called *in-silico haplotyping*. There are many kinds of such methods nowadays, including Clark[1], Gusfield[2], Excoffier[3] and Stephens[4], etc.

The first *in-silico* haplotyping algorithm was proposed by Clark[1] and examined more thoroughly by Gusfield[5]. Clark's method has been used extensively and demonstrated effective in practice[6−8]. However, to apply Clark's method, it needs the assumption that there must be some homozygotes or single-site heterozygotes in genotype samples. A new model was proposed to avoid such limitation[9,10]. Given a sample population of genotypes, its objective is to find out a set of minimum number of haplotypes that explains these genotypes, which fits in with the maximum parsimony principle. Compared with Clark's method, the parsimony model could be applied in wider fields. Wang proposed a complete algorithm using branch-and-bound methods. However, it is an exponential time algorithm and cannot be applied to large-scale instances. In this paper, we present an efficient polynomial time approximation algorithm, which is desired in Ref.[10].

Each genotype will be split into a pair of haplotypes in the process of haplotyping. In order to explain the genotype set with least haplotypes, the greedy algorithm repeats the following steps until all the genotypes are split into a pair of haplotypes: i) seek for a maximum set that contains some unresolved genotypes with a common pattern, ii) split these genotypes according to the common pattern.

Our greedy algorithm runs very fast and keeps a relatively high accurateness to reconstruct the original haplotypes. However, it is an approximation algorithm. We can combine the greedy policy with the branch-and-bound strategy in a uniform framework, using the result of the greedy algorithm as the initial bound of the branch-and-bound algorithm. The compound algorithm is also a complete algorithm. Our experimental results show that it is much more efficient and can be applied to instances of much larger scales than the original one.

The rest of the paper is organized as follows: in Section 1, we formalize the problem of haplotyping and the

model by maximum parsimony. Section 2 is devoted to the design of the greedy algorithm and the compound algorithm. The experimental results based on the simulation data are given in Section 3. At last, Section 4 concludes the whole paper and brings forward some promising directions for future work.

1 Preliminary Definitions

First of all, we formalize the problem of *in-silico* haplotyping and the parsimony model.

Sequences of genotype and haplotype data are presented as string vectors. Each site corresponds to a position of interest on the chromosome. Mostly, there are two possible nucleotides that appear at any site of haplotypes, so we just focus on these cases and denote the nucleotides by "0" and "1". Then a haplotype is presented as a vector where each component has a value of "0" or "1".

A genotype g is formed by combining two haplotypes h_1 and h_2. On each SNP site, the value of the associated position in g is determined by the state of corresponding sites in h_1 and h_2. For example, if h_1 and h_2 have the same state "0"(respectively "1") on one SNP site, then the corresponding site of g should also be "0"(respectively "1"), which denotes a *homozygous* (also referred as *resolved*) site. If h_1 has state "0" and h_2 has state "1" on one SNP site then the corresponding site of g should be "2", which denotes a *heterozygous* (also referred as *ambiguous*) site. We put it down as $g=h_1\oplus h_2$. Haplotype pair (h_1,h_2) is reversely called an *explanation* of genotype g, and haplotypes h_1 and h_2 are *compatible* with genotype g. The genotype g can be *split* to generate a pair of haplotypes h_1 and h_2.

A vector is *ambiguous* if it contains heterozygous sites; otherwise it is *resolved*. Haplotype vectors are always resolved vectors; and genotype vectors are usually (but not always) ambiguous vectors.

Under this definition, a sample population of genotypes can be viewed as a set of vectors $G=\{g_1,g_2,\ldots,g_n\}$ where each component has a value of "0", "1" or "2". Each vector in the set is a sequence associated with m sites of interest on the two copies of a chromosome. The purpose of haplotyping is to find out an explanation for each vector g_i in G.

Definition 1. **Haplotyping** Given a set $G=\{g_1,g_2,\ldots,g_n\}$of genotypes of length m, where each component has a value of "0", "1" or "2", find out a set $H=\{h_1,h_2,\ldots,h_t\}$ of haplotypes (of length m), so that for any genotype $g\in G$, there is a pair of haplotypes $h, k\in H$ that $g=h\oplus k$.

We then call that H *explains* G. Obviously, with the number of heterozygous site (k) increases, the number of possible haplotypes (2^k) increases exponentially. Without additional biological background, one cannot figure out which way gives the solution nearest to the reality. How to infer the actual haplotypes from the daunting number of all possible ones thus becomes the problem of *in-silico* methods.

Simulation and real data experiments showed that solutions near to reality tended to be produced when the amount of the result from the distinct resolved vectors was small. The observation and such idea is formalized as the model of *haplotyping by maximum parsimony*[9,10].

Definition 2. **Haplotyping by maximum parsimony** Given a set $G=\{g_1,g_2,\ldots,g_n\}$ of genotypes of length m, where each component has a value "0", "1" or "2", find out a set $H=\{h_1,h_2,\ldots,h_t\}$ of haplotypes (of length m) that explains G such that $|H|$ is minimized.

This parsimony model surmounts Clark's method by overcoming its drawbacks: it needs no starting set and leaves no orphans. It tries to use least haplotypes to explain most genotypes. In this meaning, it accords with the principle of maximum parsimony, which is a ubiquitous principle in nature.

2 Algorithms

Wang[2] proposed a complete branch-and-bound algorithm for it. Although it works well for some small-scale instances, it is an exponential time algorithm. So it cannot be applied to large-scale instances.

In this section, we will present a greedy algorithm and a compound algorithm that combines the greedy policy with the branch-and-bound strategy in a uniform framework.

2.1 Branch-and-Bound algorithm

Given a set of genotypes G, the branch-and-bound algorithm searches all possible solutions and finds the best one. The whole search space is explored via depth-first search (DFS) to try each possible resolution for each genotype. When the size of a partial solution is greater than or equal to the current bound, it skips the bad subspace and moves to the next possible good choice. Please refer to the subsection 2.3 for a more formal illustration.

Theoretically, the running time of the algorithm is exponential in terms of the input size.

2.2 Greedy algorithm

In the worst cases, the branch-and-bound algorithm must try out all of the leaves in the search space to find the optimal solution. This would be unbearable in time. Usually, heuristic policies are then employed to make a trade-off between performance and cost.

Our idea comes from a straightforward observation. If two different vectors g_1 and g_2 in the original set G are split into two haplotype pairs that contain a common vector h, then the result H will have one less distinct resolved vectors. The more common vectors we get in the process, the less distinct vectors we get as the result. If we could design an algorithm that produces the maximum common vectors during splitting all genotypes in G, we have got another optimal algorithm. Nevertheless, trying to produce the maximum common vectors requires a global view of the given genotype sample, which is difficult to obtain.

Greedy policy thus becomes a good alternative. We design an algorithm based on this idea, seeking for a maximum set S in each local iteration step. S contains some vectors, which can be split to generate a common vector s, called the *common pattern* of S. Then we can split the vectors in S according to s: for each vector $c \in S$, c is split to s and a complementary vector ($c=s \oplus c'$).

The whole framework of Algorithm *Greedy_haplotyping* is shown in Fig.1.

```
ALGORITHM Greedy_haplotyping (INPUT: genotype set G OUTPUT: haplotype set H)
BEGIN
    S0←{all the vectors in G}
    Seek_for_S(S0,S)
    While (|S|≥2)
        S0←S0−S
        Split S according to s and put the result vectors into S′
        S0←S0+S′                                   } Update to S
        Seek_for_S(S0,S)
    Post process: split all unsettled vectors in S0 arbitrarily and output it as haplotype set H
END
```

Fig.1 The framework of algorithm *Greedy_haplotyping*

In Fig. 1, we firstly initialize a set S_0 to contain all of the vectors in the original set G. Vectors in S_0 will be split step by step in a manner of "*update* to S_0":

Update: Find a maximal subset S of S_0, which has a common pattern s. Delete the vectors in S from S_0, and split them according to the common pattern s. The newly obtained resolved vectors will be put back into S_0. Remove the duplicate ones. It is called an update on S_0.

"*Update* to S_0" won't stop until S_0 contains no ambiguous vectors or we can't find an $S(S \geq 2)$. At last, examine S_0 once again and split all the unresolved vectors in it.

The internal procedure *Seek_for_S* should be refined. Its objective is to find out the maximal S with a common pattern for S_0. Let $S_0=\{g_1,\ldots,g_t\}$, we perform the seeking process by eliminating the incompatible vectors step by step till vectors in $S_k^{(m)}$ have a common pattern. Please refer to Fig.2 for detail.

```
PROCEDURE Seek_for_S(S0,S)
BEGIN
    S1(1) = S1(1) = … = St(1) = S0
    for i = 1 to m do
        for j = 1 to t do
            if gj,i = 0 (or 1, respectively), then for k = 1 to t do
                if gk,i = 1 (or 0, respectively),   then Sj(i) = Sj(i) - gk
        for j = 1 to t do Sj(i+1) = Sj(i)
    S = {Sk0(m) : | Sk0(m) | = maxk | Sk(m) |}
END
```

Fig.2 The procedure *Seek_for_S*(S_0, S)

2.3 Compound algorithm

Although the greedy algorithm performs well, it is an approximation algorithm. In the case that high accurateness is required, we may still want a complete algorithm. A good idea is to combine the greedy policy with the branch-and-bound strategy in a uniform framework. Firstly, we run the greedy algorithm and obtain one approximate solution. The solution can be used as an initial bound of the branch-and-bound algorithm. Therefore we can prune away many unnecessary searching paths and greatly reduce the search cost.

Suppose that there are k_i (>1) ambiguous sites in genotype g_i, so there are 2^{k_i-1} ways to split g_i, each way generates a pair of resolution haplotypes. Figure 3 illustrates the framework of the algorithm.

```
ALGORITHM Compound_haplotyping (INPUT: genotype set G OUTPUT: haplotype set H)
BEGIN
    S = Φ; H* = the solution returned by the greedy haplotyping
    For i1 = 1 to 2^(k1-1) do,
        S = S + the i1 pair of resolution haplotypes of g1; if |S| > | H* |, exit;
        For i2 = 1 to 2^(k2-1) do,
            S = S + the i2 pair of resolution haplotypes of g2; if |S| > | H* |, exit;
            ……
                For in = 1 to 2^(kn-1) do,
                    S = S + the in pair of resolution haplotypes of gn; If |S| < | H* |, then H* = S
                    S = S - the in pair of resolution haplotypes of gn
                ……
            S = S - the i2 pair of resolution haplotypes of g2
        S = S - the i1 pair of resolution haplotypes of g1
END
```

Fig.3 The framework of algorithm *Compound_haplotyping*

2.4 Discussion on the complexity of the algorithms

The time complexity of the branch-and-bound algorithm is $O(2^{k_1+k_2+\cdots+k_n})$. If there is no limitation to k_i, it is $O(2^{mn})$. It may be unbearable if m,n are very large. The greedy algorithm runs fast and it can obtain the optimal solution with high probability. Now we analyze its time and space complexity in more detail.

Theorem 1. The time complexity of our greedy algorithm is $O(n^3 \cdot m)$, and its space complexity is $O(n \cdot m)$.

Proof. The outer iteration may be executed as much as $n-1$ times. Each time it splits one ambiguous vector. The time complexity of procedure *Seek_for_S* is $O(m \cdot t^2) = O(m \cdot n^2)$ (for that $t = O(n)$), so, the time complexity of the greedy algorithm is $O(n^3 \cdot m)$.

The space to keep all subsets $S_k^{(m)}$ is clearly $O(n \cdot m)$, and it's the largest space the algorithm will consume. □

Theoretically, although the greedy algorithm can speedup the branch-and-bound algorithm greatly, the compound algorithm is still an exponential time algorithm. However, our experimental results demonstrates its efficiency in practice (please refer to the next section).

Possibly, we may come out with a set S with an ambiguous common pattern s. If s is resolved, there would be a unique way to split the vectors in S according to s. However, if s is ambiguous, there would be several ways to split the vectors in S: each ambiguous site in s could be either "0" or "1". Without any further information, we could not tell which way is better, so we simply enumerate all cases. Our experiments demonstrate that enumerating all cases doesn't largely lower the algorithm's performance.

3 Experiments

All the algorithms are implemented in C++. The programs are now available upon request. Experiments based on simulation data have been done to examine their performance, which is discussed below.

3.1 Random data and simulation data on the coalescent model

As pointed out by Niu[11], to adopt which kind of data in the experiment will dramatically affect the performance of *in silico* haplotyping programs. We employ two kinds of data for the sake of justness: random data and simulation data.

Simulation data sets generated based on some model usually would favor or impair the programs. In order to evaluate the systems on any sample that has no clue about evolutionary history, the first kind of data we adopted in our experiments is random. Firstly, we generate a number of sequences of haplotypes randomly, simply setting every bit evenly and independently. Then two of them are randomly chosen and paired to generate a genotype. According to the number of SNPs, different number of genotypes is generated to form a sample population.

However, real data from living cells are believed to fit into some specific genetic models. To test the programs under more realistic data samples, we introduce a powerful model called the coalescent theory. Developed by R. Hudson[12], the program ms is the most widely used program, which uses coalescent theory to generate simulating population samples of haplotypes. Many independent replication samples under a variety of assumptions about migration, recombination rate, number of gametes and polymorphic sites are generated in our experiments. Provided with the simulation haplotypes, genotype samples could also be generated as before.

3.2 Comparisons with the branch-and-bound algorithm

To evaluate the performance of our programs, we introduce two parameters: running time t and accurate rate ρ. Let n_c = *number of correctly resolved genotypes*, n_t = *total number of individual genotypes*, ρ is defined as:

$$\rho = \frac{n_c}{n_t}$$

All experiments were conducted on a single-CPU P-III PC with an i686 CPU at 550MHz and 512M RAM. We tested our algorithms in three groups of scales. For each scale 100 data sets are generated and the performance is evaluated by computing the average numbers in these 100 runs.

We use simulation data in this subsection. Our genotype sample is generated as follows: firstly, a set H_0

consisting of n_{hap} n_{hap} distinct haplotypes is generated by *ms*. Each vector contains m sites. Then the haplotypes are randomly paired to create n_{temp} genotypes. A post-examination is executed to delete the duplicate ones from these n_{temp} genotypes. The residual n_{geno} strings will be exported into a file as the input of the haplotyping algorithms.

3.2.1 Small-Scale instances

For the three algorithms, these instances all end within 1 second. The output set H of the complete algorithm (the output set H of the compound algorithm is the same as that of the branch-and-bound algorithm) always contains 6 haplotypes. In most cases, but not always, it could restore the original haplotype set H_0. Theoretically, we could find an H with less than 6 haplotypes, but we do not observe such cases in our experiments.

The output set H of the greedy algorithm contained 6 to 12 haplotypes. In about 1/2 cases H contains 6 haplotypes (good result); most were original haplotypes in H_0. In about 1/3 cases H contains 7 to 10 haplotypes (acceptable result). It also restores the 6 haplotypes in H_0 with introducing a few new ones. But we also observed for several times, H of greedy algorithm contains more than 12 haplotypes (bad result). It restores few original haplotypes while introducing many new ones. Table 1 shows the statistical data in detail.

Table 1 Experimental results of different algorithms(small-scale)

Algorithms	Performance of small-scale ($m=10, n_{hap}=6, n_{temp}=10, n_{geno}=6\sim9$) test cases		
	Runtime (s)	$\lvert H\rvert$	ρ
Branch-and-Bound algorithm	0.207	6	0.967
Greedy algorithm	0.011	7.3	0.771
Compound algorithm	0.092	6	0.967

3.2.2 Medium-Scale instances

These instances are closest to the cases in practice. The running time of branch-and-bound algorithm ranges from several seconds to more than 1 hour. Refer to Table 2 for the detail.

Table 2 Experimental results of different algorithms (medium scale)

Algorithms	Performance of medium scale ($m=15, n_{hap}=10, n_{temp}=20, n_{geno}=14\sim18$) test cases		
	Runtime (s)	$\lvert H\rvert$	ρ
Branch-and-Bound algorithm	848.2	10	0.960
Greedy algorithm	0.122	11.9	0.716
Compound algorithm	22.9	10	0.960

The proportion of good results, acceptable results and bad results in the output of the greedy algorithm is somewhat similar to that of the small-scale instances, so the discussion is omitted here.

We notice that if we get a tight initial bound from the greedy algorithm, the running time of the compound algorithm is greatly reduced. Otherwise, if the greedy algorithm gives a bad initial bound, it could hardly reduce the running time of the compound algorithm.

3.2.3 Large-Scale instances

Due to the exponential increasing in time, the original branch-and-bound algorithm is not available for the large-scale test cases. Please refer to Table 3 for the result of the greedy algorithm and the compound algorithm.

Table 3 Experimental results of different algorithms (large-scale)

Algorithms	Performance of large scale ($m=20, n_{hap}=15, n_{temp}=30, n_{geno}=26\sim30$) test cases		
	Runtime (s)	$\lvert H\rvert$	ρ
Branch-and-Bound algorithm	*N/A*	*N/A*	*N/A*
Greedy algorithm	0.578	17.5	0.701
Compound algorithm	1062.0	15	0.954

3.3 Comparisons with the other prevailed programs

To demonstrate the efficiency and performance of our algorithms, we'll have them compared with some other prevailed programs. HAPINFERX is based on Clark's algorithm and offered by Dr. Clark. It's a quite simple yet widely applied program. PHASE is another prevailed program based on a statistical model. It can be downloaded freely from Dr. Stephens' homepage. In order to evaluate the systems on any sample that has no clue about evolutionary history, we adopt random data in this subsection.

Their runtimes are recorded in Table 4. From it we can figure out that HAPINFERX and the greedy algorithm run much more quickly, and their execution times increase smoothly as the numbers of samples increase. PHASE runs far slower, although its execution times also increase smoothly. The execution times of Branch-and-Bound and the compound algorithm increase rapidly, in the manner of exponential increase. However, the latter is still much more efficient than the former and can be applied to larger instances.

Table 4 Comparison of the running times of different programs (m=10, n_{hap}=20)

Algorithms	Runtime (s) of instances of			
	n_{geno}=10	n_{geno}=20	n_{geno}=30	n_{geno}=40
HAPINFERX	0.012	0.015	0.018	0.020
PHASE	710.041	1529.273	2222.304	2746.186
Branch-and-Bound	0.134	22.186	1782.153	N/A
Greedy	0.011	0.083	0.265	0.628
Compound	0.072	1.782	19.622	220.717

We plot the accurate rates of the four algorithms on instances of m =10, n_{hap} =20 and different sample sizes (n_{geno}=10, 20, 30, 40. Different from the former subsection, there are few duplicate genotypes.) in Fig.4. Both the branch-and-bound algorithm and the compound algorithm are complete algorithms for the parsimony model, so their performances are the same, which is denoted by a single line (*the line of Complete*) in the figure.

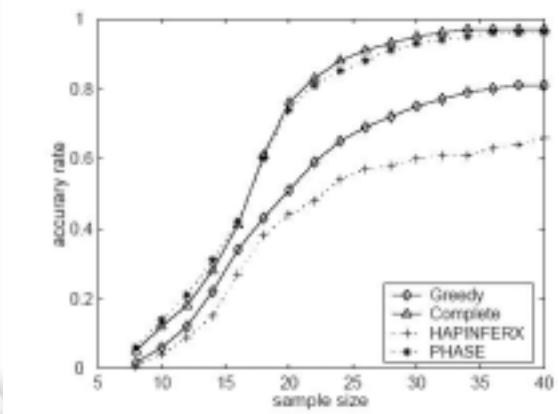

Fig.4 Comparison of the accurate rates of different programs (m=10, n_{hap} =10)

As it shows, the accurate rates increase as the number of samples increases. But in all cases, our complete algorithms and PHASE are better than the greedy algorithm and HAPINFERX. All the algorithms perform badly when n_{geno} is as small as m/2. When n_{geno} is large enough, our complete algorithms and PHASE can resolve the sample genotypes correctly with high probabilities (the accurate rates are greater than 0.9).

3.4 Discussion

We observe from our experiments that if we find the optimal or near optimal solutions for the instance, we can recover the original haplotypes with a high probability and high precision (both ρ_{rec} and ρ_{acc} are close to 1). Which demonstrate the validity of the parsimony model.

Due to the exponential increasing characteristics of haplotyping by maximum parsimony, the original branch-and-bound algorithm can not deal with large-scale instances. Our greedy algorithm always ran fast, yielding good or acceptable result for more than 2/3 cases. The compounded algorithm has both strength of the greedy algorithm and the branch-and-bound algorithm: it runs fast and keeps a high accurate rate to reconstruct the original haplotypes.

4 Conclusions

Compared with the existing models on the haplotyping problem, the new one that integrates the maximum parsimony principle could be applied in wider fields. In this paper we bring forward a greedy algorithm and a compound algorithm that combines the greedy policy with the branch-and-bound strategy in a uniform framework. The approximation greedy algorithm runs much faster than the complete branch-and-bound algorithm, and outputs pretty well results. The newly compounded algorithm is also a complete algorithm and is much more efficient than the original branch-and-bound algorithm. It can be applied to instances of much larger scales. Our experiments on some simulation data demonstrate their practicability.

However the running time of the compound algorithm increases rapidly with the scale of the data sample. It is efficient in most cases, but in the cases of much larger scales, more delicate algorithms are desirable.

Acknowledgements We would like to thank Prof. Lusheng Wang in City University of HK for introducing us to the work of Clark[1] and Gusfield[5]. Thanks Dr. Chao Yan, and Dr. Yinyu Wan for their helpful comments.

References:

[1] Clark AG. Inference of haplotypes from PCR-amplified samples of diploid populations. Molecular Biology and Evolution, 1990,7(2):111−122.

[2] Gusfield D. Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. In: Proc. of the RECOMB02. 2002. 166−175.

[3] Excoffier L, Slatkin M. Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population. Molecular Biology and Evolution, 1995,12(5):921−927.

[4] Stephens M, Smith NJ, Donnelly P. A new statistical method for haplotype reconstruction for population data. American Journal of Human Genetics, 2001,68:978−989.

[5] Gusfield D. Inference of haplotypes from samples of diploid populations: complexity and algorithms. Journal of Computational Biology, 2001,8:305−323.

[6] Clark AG, Weiss KM, Nickerson DA, Taylor SL, Buchanan A, Stengard J, Salomaa V, Vartiainen E, Perola M, Boerwinkle E, Sing CF. Haplotype structure and population genetic inferences from nucleotide-sequence variation in human lipoprotein lipase. American Journal of Human Genetics, 1998,63:595−612.

[7] Drysdale C, McGraw D, Stack CB, Stephens C, Judson RS, Nandabalan K, Arnold K, Ruano G, Liggett SB. Complex promoter and coding region -adrenergic receptor haplotypes alter receptor expression and predict in vivo responsiveness. In: Proc. of the National Academy of Sciences, 2000,97(19):10483−10488.

[8] Rieder M, Taylor SL, Clark AG, Nickerson DA. Sequence variation in the human angiotensin converting enzyme. Nature Genetics, 1999,22:59−62.

[9] Gusfield D. Haplotyping by pure parsimony. Technical Report, CSE-2003-2, University of California at Davis, 2003.

[10] Wang LS, Xu Y. Haplotype inference by maximum parsimony. Bioinformatics, 2003,19(14):1773−1780.

[11] Niu T, Qin ZS, Xu X, Liu JS. Bayesian haplotype inference for multiple linked single nucleotide polymorphisms. American Journal of Human Genetics, 2002,70:157−169.

[12] Hudson RR. Generating samples under a wright-fisher neutral model of genetic variation. Bioinformatics, 2002,18(2):337−338.