Building minimum recombination ancestral recombination graphs for whole genomes

Nguyen Thi Phuong Thao
Institute of Information Technology
Vietnam Academy of Science and Technology
18 Hoang Quoc Viet, Cau Giay, Hanoi, Vietnam
thaontp@ioit.ac.vn

Le Sy Vinh
University of Technology and Engineering
Vietnam National University
144 Xuan Thuy, Cau Giay, Hanoi, Vietnam
vinhls@vnu.edu.vn

Abstract – **Building ancestral recombination graphs (ARGs) with the minimum number of recombination events is an essential problem in genomic population studies. Exhaustive search methods have been proposed to solve the problem with few sequences and markers. Recently, we introduced ARG4WG algorithm [1] to build plausible ARGs for whole genomes of thousand samples. However, the ARG4WG algorithm was not designed to optimize the number of recombination events. In this work, we propose a new design of the ARG4WG algorithm to optimize the number of recombination events. The key idea is to analyze and combine different criteria (i.e., the longest shared ends, the maximum similarity between sequences, and the length of sequences) to select the best candidates for recombinations. The experiments on different data sets showed that our new algorithms produced ARGs with smaller number of recombination events in comparison to other methods. The methods allow us to build minimum recombination ARGs from whole genome data sets.**

Keywords – ancestral recombination graph, minimum recombination, whole genome, population genetics

I. Introduction

Recombinations play an important role in the human evolutionary processes. Since recombination events in the human evolutionary processes are typically undetectable, the true number of happened recombinations is unobtainable. Finding the minimum number of recombination events must have occurred in the evolutionary history of samples is then the aim of research. A number of different methods were introduced to estimate the lower bound of the number of recombination events in the history of a samples [2,3,4,5]. However, the lower bounds estimated by these methods are not always close to the minimum number recombinations.

An evolutionary history can be represented in the form of an ARG as illustrated in Figure 1. Exhaustive search methods have been proposed [6,7] to construct the ARG with the minimum number of recombination events for a data set. These methods can yield the minimum number, however, are only applicable to small data sets with few samples and markers. Thus, they are impossible for large-scale data sets due to computational expense.

Heuristic methods such as Margarita [8] and ARG4WG [1] have been proposed to tacke the problem. These methods are designed to infer plausible ARGs, and do not focus on finding the minimum recombination ARGs. Margarita is able to handle a thousand sequences with hundreds of markers. It first performs all possible coalescence and mutation events. It then searches for a pair of sequences that are identical over the longest segment to perform recombination events. If the shared tract is found inside the sequence, Margarita must put two recombination events and generates three subsequences from one sequence to get a subsequence freely to coalesce (Figure 2a). This strategy leads to the increase of number of recombination events.

We have proposed ARG4WG algorithm to work with whole genome data sets. The key idea is to select the pair of sequences with the longest shared ends for recombinations. This strategy overcomes the computational burden when working with whole genome data sets. Although ARG4WG results in AGRs with fewer recombination events in comparision to the Margarita (Figure 2b), it was not designed to optimize the number of recombination events.

In this work, we proposed a new design of ARG4WG, called REARG, to optimize the number of recombination events for ARGs built from large whole genome data sets. The key idea comes from our experimental observation that the selection of sequence pair with the longest shared ends for recombinations in ARG4WG is not unique. In other words, ARG4WG randomly selects one sequence pair for recombinations from equal longest shared end sequence pairs. For example, as in the figure 2b, there are 2 candidates, (S_1, S_2) and (S_2, S_4), having the same longest shared end, then ARG4WG will randomly choose either (S_1, S_2) or (S_2, S_4) to perform recombinations.

Our experimental analyses revealed that beside the longest shared end criterion, other factors such as the similarity of the selected sequence pair or the length of the selected sequence also considerably affect the number of recombination events. Therefore, combining all these criteria in selecting the best pair for recombinations could optimize more the number of recombination events. Experiments on real data sets with different sizes (100 and 200 sequences with 2000, 5000, and 1000 markers) showed that our proposed algorithms outperformed both Margarita and ARG4WG algorithms.

The paper is organized as following: We introduce ARG4WG and our proposed algorithms in section II. Then,

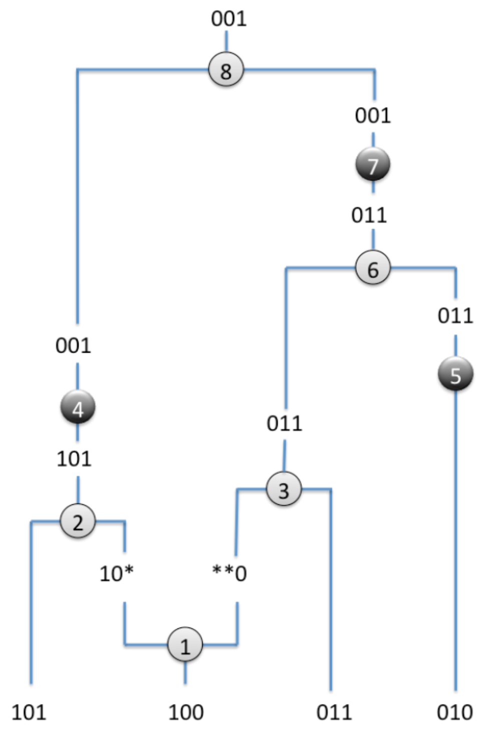

Fig. 1. An ARG with 8 events (numbered from 1 to 8 in cicles) is built backward in time, starting from the input data of 4 sequences with 3 markers {101, 100, 011, 010}, until a single common ancestor (001) is reached. 3 different evolutionary events are performed in the building process. A recombination event at state 1 breaks sequence 100 into two sequences 10* and **0. A coalescence event as in the state 2,3,6,8 combines two sequences into one. A mutation event as in the state 4,5,7 changes a mutated site from 0 to 1 and vice versa.

we describe experiments and results in section III. Finally, we give our conclusions and plans for the future work in the last section.

II. METHODS

We first introduce some notions to describe different algorithms.

Let D = $\{S_1, \ldots, S_N\}$ be a set of input sequences (haplotypes), S_x has m markers, $1 \le x \le N$; and let $S_x[i]$ denote site i of S_x that has the value of either 0 (one of the SNP alleles), or 1 (another allele), or * (missing value or non-ancestral material), $1 \le i \le m$.

As in [1], we define

- $S_1[i]$ matches $S_2[i]$ if $S_1[i] = S_2[i]$.
- S_1 matches S_2 with a maximal length l from the left, $(S_1,S_2)\{d=left,l\}$, if $S_1[i]$ matches $S_2[i]$ for all $1 \le i \le l$ and either $l = m$ or $S_1[l+1]$ does not match $S_2[l+1]$
- S_1 matches S_2 with a maximal length l from the right, $(S_1,S_2)\{d=right,l\}$, if $S_1[i]$ matches $S_2[i]$ for all $m-l < i \le m$ and either $l = m$ or $S_1[m-l]$ does not match $S_2[m-l]$.
- $(S_1,S_2)\{d,l\}$ exits if and only if there are at least one marker i in matching region that $S_1[i] = S_2[i] \ne *$.

We also define:

The similarity of two sequences S_1 and S:

$$Sim(S_1,S_2) = \sum_1^m Sim(S_1[i],S_2[i])$$

where

$$Sim(S_1[i],S_2[i]) = \begin{cases} 1 & if\ \ S_1[i] = S_2[i] \ne * \\ 0 & otherwise \end{cases}$$

The length of a sequence S:

$$Len(S) = \sum_1^m Len(S[i])$$

where

$$Len(S[i]) = \begin{cases} 1 & if\ \ S[i] \ne * \\ 0 & if\ \ S[i] = * \end{cases}$$

ARG4WG algorithm: We now briefly describe the ARG4WG algorithm. The ARG4WG algorithm starts from time $t = 1$. The set of sequences at time t is denoted as D_t (D_1=D). For each D_t, three candidate lists for coalescence, mutation and recombination events is constructed as the following:

- Coalescence list **C**: For a shared end pair $(S_1,S_2)\{d,l\}$ of sequences S_1 and S_2, if $l = m$, we put this pair to the coalescence list.
- Mutation list **M**: For a marker i ($1 \le i \le m$), if there exist exactly only one sequence S_1, where, $S_1[i] = 1$ and $\forall S_2 \in D_t \setminus \{S_1\}: S_2[i] \ne 1$ or $S_1[i] = 0$ and $\forall S_2 \in D_t \setminus \{S_1\}: S_2[i] \ne 0$, then $S_1[i]$ is added into mutation list.
- Recombination list **R**: For a shared end pair $(S_1,S_2)\{d,l\}$ of sequences S_1 and S_2, if $0 < l < m$, $(S_1,S_2)\{d,l\}$ is added into the recombination list.

When one of three events occurs, the next sequence set D_{t+1} is created from the current sequence set D_t as described below and three candidate lists are updated.

- If a coalescent event is encountered between two sequences S_1 and S_2, two sequences S_1 and S_2 are merged into a common ancestor S'. Remove S_1 and S_2 from the sequence set D_t; and sequence S' is added into D_{t+1}.
- If a mutation event occurs on a sequence S, a new sequence S' is created from sequence S with the mutation. Replace sequence S by new sequence S'.
- If a recombination occurs on a shared end pair (S_1, S_2), pick the shorter sequence to do recombination. Assuming $Len(S_1) < Len(S_2)$, sequence S_1 will be broken into two new subsequences. Replace S_1 by two new subsequences. The subsequence containing shared region will be coalesced with S_2.

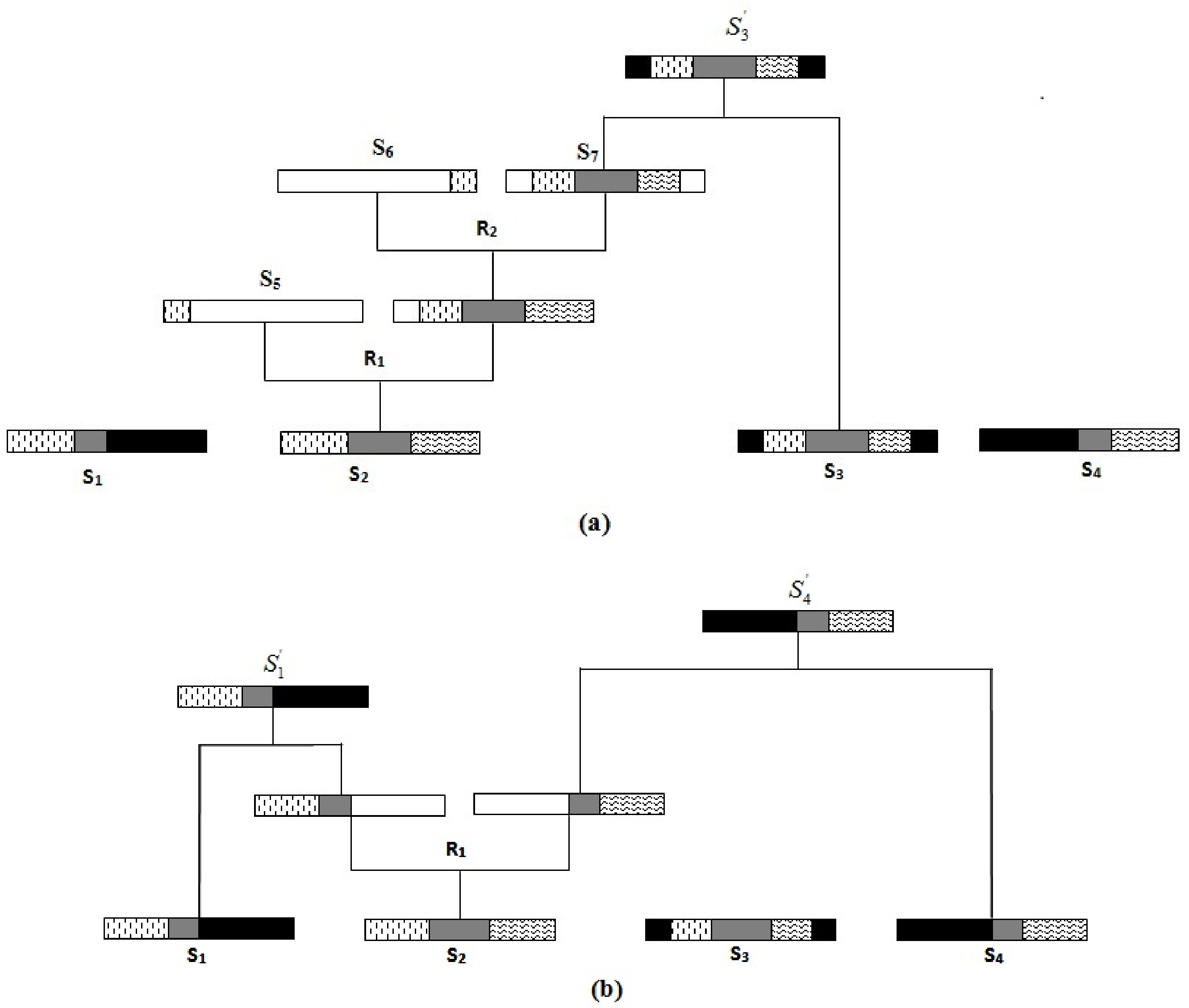

Fig. 2. Recombination event presented in Margarita algorithm (a) and ARG4WG algorithm (b) for a set of 4 sequences S_1, S_2, S_3 and S_4. (a) The longest shared segment is detected between S_2 and S_3, 2 recombination events R_1 and R_2 are putted on S_2 to produce 3 subsequences S_5, S_6 and S_7. S_7 then coalesce to the sequence S_3. So from 4 input sequences, Margarita has to take 2 recombination events and generates 5 sequences for the next generation. (b) A longest shared end detected between S_1 and S_2, a recombination event R_1 is putted on S_2 to produce 2 subsequences. 2 subsequences then coalesce to the sequence S_1 and S_4 respectively. So from 4 input sequences, ARG4WG just performs one recombination event and generates 3 sequences for the next generation.

The ARG4WG algorithm

Input: A set of *N* sequences with *m* snp markers

Output: The relations (coalescences, mutations or recombinations) among sequences.

```
• Step 1: If Coalescence list is not empty,
      do all possible coalescence.
• Step 2: If Mutation list is not empty,
      do a mutation then go to 1.
      If no mutation possible, go to Step 3.
• Step 3: If Recombination list is not empty,
      do a recombination followed by a coalescence.
      Go to Step 1.
• Step 4: Repeat Step 1, Step 2 and Step 3 until a single
      common ancestor is reached.
```

The candidate from the coalescence list or the mutation list to perform coalescence or mutation is taken randomly. In recombination step, the pair of sequences with the longest shared end will be chosen for recombination. If there is more than one candidate having the same longest shared end, one is picked randomly. The random choices in ARG4WG algorithm result in different ARGs for different runs.

REARG algorithm: We design the REARG algorithm to optimize the number of recombination events. To this end, we also use other criteria to determine the best candidate for recombinations. The procedures for coalescence and mutation step are remained as the same as in the ARG4WG algorithm. In the following, we describe three versions of REARG algorithm, called REARG_SIM, REARG_LEN, and REARG_COM.

A. Recombination step of *REARG_SIM algorithm*

- Step 1: Calculate lengths of shared ends for all sequence pairs. Sequence pairs with the longest shared ends are kept as candidate pairs for recombinations.
- Step 2: Calculate similarity scores of all candidates. Select the candidate pairs with the highest similarity score to perform recombination. If several candidates have the same highest similarity score, we randomly pick one for recombination.

B. Recombination step of *REARG_LEN aglorithm*

- Step 1: Calculate lengths of shared ends for all sequence pairs. Sequence pairs with the longest shared ends are kept as candidates for recombinations.
- Step 2: Calculate the length of the shorter sequence of all candidate pairs. Select the candidate with the longest length to perform recombination. If several candidates have the same longest length, we randomly pick one for recombination.

C. Recombination step of *REARG_COM algorithm*

- Step 1: Calculate lengths of shared ends for all sequence pairs. Sequence pairs with the longest shared ends are kept as candidate pairs for recombinations.
- Step 2: Calculate similarity scores of all candidates and length of the shorter sequence of all candidate pairs.
- Step 3: Randomly select one candidate pair with the highest similarity score or one candidate pair with longest length to perform recombination.

III. Experiments and Results

We used real data to analyze the performances of Margarita, ARG4WG, and proposed algorithms. All experiments were conducted on 18 data sets extracted from 3 different regions of Chromosome 1 from the 1000 Genomes Projects [9] with different number of haplotypes and snps as described in Table I.

TABLE I. Data sets from the 1000 Genome Project

Data sets	#haplotypes	#SNPS
{DS1, DS2, DS3}	100	2000
{DS1, DS2, DS3}	100	5000
{DS1, DS2, DS3}	100	10000
{DS1, DS2, DS3}	200	2000
{DS1, DS2, DS3}	200	5000
{DS1, DS2, DS3}	200	10000

On each data set, 1000 ARGs were built by each algorithm and the ARG with the smallest number of recombination events was recorded. Since Margarita could not finish the experiments for all data sets, the tree with smallest number of recombinations found after 9 days was considered as the best tree for Margarita.

TABLE II. The smallest number of recombination events found by 5 algorithims for 100 haplotypes, of (A) DS1, (B) DS2, and (C) DS3

(A)

SNPS	2000	5000	10000
Margarita	30749	77574	13234
ARG4WG	2596	5837	10490
REARG_SIM	2579	5786	10324
REARG_LEN	**2560**	**5741**	10336
REARG_COM	**2560**	5766	**10307**

(B)

SNPS	2000	5000	10000
Margarita	33820	86041	10874
ARG4WG	**1924**	4335	8741
REARG_SIM	1949	4284	8623
REARG_LEN	**1924**	4288	8627
REARG_COM	1930	**4279**	**8613**

(C)

SNPS	2000	5000	10000
Margarita	32807	81829	11714
ARG4WG	1555	4120	9100
REARG_SIM	1545	4087	8987
REARG_LEN	**1527**	4077	8960
REARG_COM	1545	**4063**	**8934**

TABLE III. The smallest number of recombination events found by 5 algorithims for 200 haplotypes, of (A) DS1, (B) DS2, and (C) DS3

(A)

SNPS	2000	5000	10000
Margarita	55788	139136	22692
ARG4WG	4218	9583	17083
REARG_SIM	**4176**	9480	16860
REARG_LEN	4180	**9437**	16834
REARG_COM	4182	9451	**16829**

(B)

SNPS	2000	5000	10000
Margarita	60337	153145	18467
ARG4WG	3072	6998	14158
REARG_SIM	3027	6909	13900
REARG_LEN	**3020**	6877	**13887**
REARG_COM	3021	**6861**	13925

(C)

SNPS	2000	5000	10000
Margarita	58313	144777	20047
ARG4WG	2620	6652	14813
REARG_SIM	2586	6607	**14620**
REARG_LEN	2595	**6564**	14634
REARG_COM	**2584**	6600	14642

Experiment results (see Table II and Table III) show that our proposed REARG algorithms produced ARGs with smaller number of recombination events in comparison to that of Margarita and ARG4WG. Notably, the number of recombination events of ARGs built from REARG algorithms is significantly smaller than that of Margarita. For long sequences (e.g. 10000 snps), the number of recombination events from Margarita is about 1.3 times higher than that of REARG. However, Margarita requires a huge number of recombination events, about 10 to 20 times that of ARG4WG to build ARGs for shorter sequences (2000 and 5000 snps). The number of recombination events needed to build an ARG from Margarita for 10000 snps are even much smaller than that for all data sets with 2000 and 5000 snps. In other words, Margarita is not stable and does not suite for small data sets with short sequences.

Although all three versions of REAR algorithm outperform ARG4WG, they do not completely outperform each other. The REARG_LEN and REARG_COM are better than REARG_SIM on most of data sets; however, REARG_SIM is still the best algorithm for two data sets (DS1 with 200 haplotypes and 2000 snps and DS3 with 200 haplotypes and 10000 snps). REARG_COM is the best algorithm in almost data sets with 100 sequences as it inherits the ability of the similarity factor combining with the length factor in ARG building process.

TABLE IV. THE AVERAGE OF THE RUNNING TIME (SECOND) OF 5 ALGORITHIMS FOR 100 HAPLOTYPES, OF (A) DS1, (B) DS2, AND (C) DS3

(A)

SNPS	2000	5000	10000
Margarita	442.00	5690.00	201.50
ARG4WG	0.57	1.26	2.08
REARG_SIM	3.27	17.66	60.64
REARG_LEN	0.62	1.35	2.46
REARG_COM	3.39	18.34	62.80

(B)

SNPS	2000	5000	10000
Margarita	754.00	7709.00	140.00
ARG4WG	0.41	0.99	1.90
REARG_SIM	2.41	12.98	53.50
REARG_LEN	0.44	1.06	2.15
REARG_COM	2.56	13.77	55.88

(C)

SNPS	2000	5000	10000
Margarita	743.50	7627.50	166.00
ARG4WG	0.36	0.96	1.96
REARG_SIM	2.04	13.01	53.63
REARG_LEN	0.39	1.02	2.08
REARG_COM	2.22	13.34	55.72

TABLE V. THE AVERAGE OF THE RUNNING TIME (SECOND) OF 5 ALGORITHIMS FOR 200 HAPLOTYPES, OF (A) DS1, (B) DS2, AND (C) DS3

(A)

SNPS	2000	5000	10000
Margarita	2826.5	31459.5	1157
ARG4WG	1.334	3.261	5.501
REARG_SIM	8.844	49.848	168.088
REARG_LEN	1.434	3.505	5.762
REARG_COM	9.71	49.197	169.769

(B)

SNPS	2000	5000	10000
Margarita	3953	42623.5	839.5
ARG4WG	0.998	2.442	4.739
REARG_SIM	6.181	35.391	145.266
REARG_LEN	0.999	2.558	4.879
REARG_COM	7.001	35.246	146.843

(C)

SNPS	2000	5000	10000
Margarita	3710.5	39230.5	1074.5
ARG4WG	0.96	2.42	4.941
REARG_SIM	5.547	34.77	149.669
REARG_LEN	0.954	2.551	5.45
REARG_COM	6.28	34.456	153.764

All the averages of the running time to build an ARG by each algorithm for each test are shown in Table IV and Table V. It is clear that the running time of Margarita is much higher, from hundreds to thousands times, than that of other algorithms. The running time of REARG_LEN is nearly equal to that of ARG4WG. Due to the wasting time of the similarity calculation, REARG_SIM and REARG_COM require more time to build an ARG, from several times to dozens of times, than that of ARG4WG and REARG_LEN. However, these algorithms are still capable to build ARGs for whole genome data.

IV. CONCLUSION

Constructing ARGs from large data sets is an emerging important task in population genomic analyses. Different ARGs inference methods have been proposed, however, they are not practical for real large-scale data sets due to their computational burden. Although ARG4WG algorithm was proposed to work with thousand of genomes, it was not designed to optimize the number of recombination events. In this work, we proposed the REARG algorithm in an attempt to build ARGs with the minimum number of recombination events. All versions of REARG algorithm outperformed Margarita and ARG4WG; however, no version of REARG algorithm is the best for all tested data sets. Therefore, we recommend users to test all versions of REARG algorithm to determine the minimum recombination ARG.

In the future, we will examine other factors and optimization strategies to further reduce the number of recombination events in building ARGs from large-scale whole genome data sets.

ACKNOWLEDGMENT

Nguyen Thi Phuong Thao is supported by Institute of Information Technology, Vietnam Academy of Science and Technology (CS17.01).

REFERENCES

[1] N. T. P. Thao, L. S. Vinh, H. B. Hai, L. S. Quang, "Building Ancestral Recombination Graphs for Whole Genomes", IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 14, no., pp. 478-483, March-April 2017.

[2] R.R. Hudson, N.L. Kaplan, "Statistical properties of the number of recombination events in the history of a sample of DNA sequences", Genetics **111,** 147–164, 1985.

[3] S.R. Myers, R.C. Griffiths, "Bounds on the minimum number of recombination events in a sample history", Genetics **163,** 375–394, 2003.

[4] Y.S. Song, Y. Wu, D. Gusfield. "Efficient computation of close lower and upper bounds on the minimum number of recombinations in biological sequence evolution", Bioinformatics, 21.suppl_1, i413-i422, 2005.

[5] V. Bafna, V. Bansal, "Improved recombination lower bounds for haplotype data", Proceedings of the 9th Annual International Conference on Computational Molecular Biology (RECOMB), p. 569-584, 2005.

[6] Y.S. Song, J. Hein, "Constructing minimal ancestral recombination graphs", Journal of Computational Biology, 12.2: 147-169, 2005.

[7] R. Lyngsø, Y. S. Song, J. Hein, "Minimum recombination histories by branch and bound", In *WABI,* pp. 239–250, 2005.

[8] M. Minichiello, R. Durbin, "Mapping trait loci using inferred ancestral recombination graphs", Am. J. Hum. Genet., vol. 79, pp. 910– 922, 2006.

[9] 1000 Genomes Project Consortium, "A map of human genome variation from population-scale sequencing", Nature, vol. 467, no. 7319, p. 1061-1073, 2010.