Reliable and fast estimation of recombination rates by convergence diagnosis and parallel Markov Chain Monte Carlo

Guo, Jing; Jain, Ritika; Yang, Peng; Fan, Rui; Kwoh, Chee Keong; Zheng, Jie

2013

Guo, J., Jain, R., Yang, P., Fan, R., Kwoh, C. K., & Zheng, J. (2013). Reliable and Fast Estimation of Recombination Rates by Convergence Diagnosis and Parallel Markov Chain Monte Carlo. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 99, 1.

https://hdl.handle.net/10356/80080

https://doi.org/10.1109/TCBB.2013.133

Reliable and Fast Estimation of Recombination Rates by Convergence Diagnosis and Parallel Markov Chain Monte Carlo

Jing Guo[1], Ritika Jain[1], Peng Yang[1,3], Rui Fan[1], Chee Keong Kwoh[1], and Jie Zheng[1,2*]

Abstract—Genetic recombination is an essential event during the process of meiosis resulting in an exchange of segments between paired chromosomes. Measuring of recombination rate is crucial for understanding evolutionary inference. Experimental methods are normally difficult and limited to small scale estimations. Thus statistical methods using population genetic data are important for large-scale analysis. LDhat is one of a few available statistical methods for large-scale datasets. It is an extensively used program using rjMCMC algorithm for prediction of recombination rates. Due to the complexity of rjMCMC scheme, LDhat is a resource-consuming algorithm that may take quite a long time to generate results, especially for large-scale SNP data. In addition, rjMCMC parameters should be manually defined in the original program that directly impact results. To address these issues, we designed an improved algorithm based on LDhat implementing MCMC convergence diagnostic algorithms to automatically predict values of parameters and monitor the mixing process. Then parallel computation methods were employed to further accelerate the new program. The new algorithms have been tested on ten samples from HapMap phase 2 datasets. The results were compared with previous code and showed nearly identical outputs, however our new methods achieved significant acceleration proving that they are more efficient and reliable for the estimation of recombination rates. The stand-alone package is freely available for download at http://www.ntu.edu.sg/home/zhengjie/software/CPLDhat/.

Index Terms—Recombination hotspot; reversible jump MCMC; convergence diagnosis; parallel computation; genome instability

1 Introduction

Meiotic recombination occurs in the pairing of homologous chromosomes in meiosis leading to the generation of novel gene combinations. The transfer of genes from parents into offspring by genetic recombination during meiosis is a major engine of genetic variation [1]. The meiotic recombination events break down the genealogical history within a genome which is critical for analyses of genetic variations [2]. The improper segregation of chromosomes can lead to aneuploidy, a significant risk factor for fetal loss and developmental disability in humans [3]. In addition, deleterious variations can be removed from the gene pool by recombination.

The rate and location of meiotic recombination have implications for understanding of recombination process and its evolution. They vary markedly between species and among individuals. The estimation of the rate at which recombination occurs can theoretically provide guidance for biologists to explore biological problems, e.g. gene targeting, mutation mechanisms [4]. Tracking distance between two genes on a chromosome by recombination rate could detect the presence of certain disease-causing genes [5].

Obtaining accurate prediction of recombination rates could be challenging and prohibitively expensive through direct experimental methods. Sperm typing produces high-resolution estimates; however, this procedure is complex, only applicable for male [6], and limited to small scale prediction. Hence, indirect statistical methods are useful. Patterns of genetic variation among DNA sequences have been used to analyze recombination rate [7]. Hudson [8] proposed a composite-likelihood estimator of the population recombination rate that combines the coalescent likelihoods of all pairwise comparisons for segregating sites. McVean, et al. [9] extended Hudson's method to allow for a finite-sites mutation model, and also introduced a likelihood permutation test. Later the heterogeneity implied by recombination hotspots is incorporated to improve the accuracy [10]. Li [11] developed a method considering all loci simultaneously rather than pairwise comparisons based on an approximation to the conditional likelihood (implemented in PHASE). Instead of approximate likelihood method, Wang [12] proposed a full-likelihood Markov chain Monte Carlo method (implemented in InferRho).

[1] *School of Computer Engineering, Nanyang Technological University, Singapore*
[2] *Genome Institute of Singapore, A*STAR (Agency for Science, Technology, and Research), Biopolis, Singapore 138672*
[3] *Institute for Infocomm Research, Agency for Science, Technology & Research*
[*] *Correspondence: zhengjie@ntu.edu.sg*

The algorithm of [10] has been implemented in the program LDhat package. It has been extensively used for detection and calculation of variable recombination rates in population genetic data via composite likelihood method. A typical change point scheme of reversible jump Markov chain Monte Carlo (rjMCMC) algorithm [13] is employed to predict the interval constant rates. The final recombination rate is composed of the morphology of hotspots and background rate. Due to the complexity of rjMCMC scheme, LDhat is a time-consuming program that would take several hours to generate results. In addition, the accuracy of the outputs and execution time are determined by parameters of rjMCMC which can only be manually specified by users. The rjMCMC parameters consist of a set of numbers, including transition times, initial samples discarded called 'burn-in' and sampling frequency. Insufficient running would cause unstable status of the Markov Chain. Conversely, over calculation would waste extra resources.

To address the above issues, we propose an improved algorithm for the prediction of recombination ratse based on LDhat. Firstly we evaluated the performances of LDhat identifying the bottleneck of running time and testified the impact of rjMCMC parameters on recombination profile and time complexity. Secondly, to avoid manually setting the parameters of rjMCMC, we incorporated algorithms for MCMC convergence assessment to automatically predefine those arguments. In addition, the chain convergent status is monitored during iteration process until it reaches the target distribution. Then we made use of parallel computation methods in order to further speed up the process of calculation.

In order to evaluate our new algorithm, we utilized 10 sets of test samples extracted from HapMap phase 2 data. We compared the recombination profiles, running time and iteration numbers of the original LDhat program and our improved methods. The result showed that our methods achieved significant speedup without affecting the accuracy of outputs. The parallel computation method resulted in even more significant reduction of execution time with identical outputs.

2 METHOD

In this section, we analyzed the LDhat program to identify the most time-consuming part. In addition, we evaluated the influence of parameters, i.e. iteration number and SNP number, on output profiles. In allusion to rjMCMC scheme, we proposed an improved algorithm applying MCMC convergence diagnostic methods and parallel computation.

2.1 LDhat program analysis

LDhat (specifically, the *rhomap* program) employs the rjMCMC algorithm which incorporates genomic polymorphisms to estimate the pairwise constant rates by composite likelihood. Composite likelihood[8] is an approximation of the coalescent likelihood [14] which is more easily implemented and based on independent pairwise single nucleotide polymorphisms (SNPs) to esitmate the recombination rate ρ. According to the composite likelihood estimator, the maximum-likelihood estimate of ρ can be obtained as the maximum product of conditional-likelihood functions of all independent pairs in n samples. These two-locus conditional-likelihoods of a fix n samples can be precalculated and stored for future researches.

However, the *ad hoc* estimator have underestimated the effects of mutations to genetic variation assuming infinite site model whereby mutation rate θ tends to be negligible. To address this problem, McVean et al. extended Hudson's work to provide an improved estimation procedure incorporating mutation models [10]. Suppose that the population mutation rate θ is constant across the sequence, it is estimated by Watterson algorithm [15]. Then pairwise segregating sites with 2 alleles are classified into equivalent sets for further likelihood calculation. The execution burden depends on the number of segregating sites with an order of n_{seq}^3, where n_{seq} is the number of haplotypes. Assuming that pairs of SNPs are independent, given the number of haplotypes, all of the possible combinations of allelic states could be consulted from tabulated files which contain precalculated likelihoods. Then the likelihood of each pair of segregating sites is estimated over a grid extracted from those files.

In addition to the contribution of background rate, the morphology of hotspots that reveals the relationship between recombination and genome features [16, 17] is incorporated into the pseudoposterior distribution of recombination rate. The rjMCMC algorithm is implemented to determine the parameters of the mutation model, i.e. change-points of SNPs, background rate, hotspot locations, hotspot heat and hotspot scale.

The scheme of rjMCMC algorithm is a typical change point problem [13] (Appendix Table A.1). Set L as the position of the last SNP, and let k be the number of change-points drawn from a Poisson distribution. The locations of change points are s_i, where $0 < s_1 < s_2 < ... < s_k < L$. The recombination rate is given by a step function $x(.)$ on $[0, L]$.

In the algorithm of LDhat, the interval background rate h_j on the jth block $[s_j, s_{j+1}]$ is initialized with the prior as exponential distribution, denoted as $P(h_j) \sim Exp(\phi)$. And the prior on the kth hotspot rate is defined as a truncated double-exponential curve, presented as $f_k \propto \lambda Laplace(\mu, b)$, where b is the central position of hotspot. Two parameters λ and μ are defined for evaluating the heat and scale of hotspots. The priors on both of them are in gamma distribution, i.e. $\lambda \sim \Gamma(\alpha_1, \beta_1), \mu \sim \Gamma(\alpha_2, \beta_2)$. The hyperparameters $\alpha_1, \beta_1, \alpha_2, \beta_2$ were obtained by Maximum Likelihood estimation to fit a gamma distribution to empirical hotspot datasets [10]. The contribution of a recombination hotspot to the final recombination rate depends on its relative location to blocks.

The mutation model is designed as four independent random transitions for background model: (a) 'death' of a randomly chosen block, (b) 'birth' of a new block at a randomly chosen location in $[0, L]$, (c) a change of the height of a randomly chosen block, and (d) a change of the position of a randomly chosen block. In addition, there are five transitions for hotspot model: (a) 'delete' of a randomly chosen hotspot, (b) 'insert' of a new hotspot at a randomly chosen location in $[0, L]$, (c) a change of the heat of a randomly chosen hotspot, (d) a change of the scale of a randomly chosen hotspot, and (e) a change of the position of a randomly chosen hotspot.

According to the rjMCMC algorithm, mutations occur during each transition. To compute the Metropolis-Hastings acceptance ratio, the recombination map composed of constant rates and a pseudo-likelihood of the data in each transition have to be calculated which take most of the execution time. For N iterations, the complexity of LDhat program scales with an order of $N \times l_{seq}$, where l_{seq} is the number of SNPs. For large scale matrix, the running time of LDhat is prohibitively lengthy. Moreover, in order to get accurate recombination rate profiles, an appropriate setting of parameters, specifically the number of iterations, is critical to guarantee that a Markov chain reaches its equilibrium distribution.

Therefore, we evaluated the influences of parameters on LDhat. Two-sample Kolmogorov-Smirnov test [18] is employed to compare two outputs. The control object $\mathbf{x}$ is attained from the results of the same dataset with 11 million iterations, 1,000,000 burn-in and 2000 sample. The accuracy of the predicted recombination profile $\mathbf{x}^{'}$ with n intervals is estimated by Kolmogorov-Smirnov statistic, defined as

$$KSZ = \sqrt{\frac{n}{2}} \, {}^{max}_{\ i} |x_i - x_i^{'}|, \ \ where \ i = 0, 1, 2...n-1 \quad (1)$$

Given rejection level $\alpha = 0.05$, the reference $KSZ_{ref} = 1.36\sqrt{\frac{2}{lseq}}$. Firstly, a test dataset is applied to examine the impact of the number of iterations on execution time and recombination profile. Then we use a group of datasets with different sizes to analyse the correlation between data size and running time with the same number of iterations. The running time shows approximately linear correlations between the number of iterations and data sizes which is consistent with the analysis on LDhat complexity. In addition, the accuracy of outputs is highly correlated with the two parameters.

Thus in the original LDhat program, a major limitation is that the parameters have to be defined by users without references. Estimation of the parameters of MCMC, such as the iteration number and the number of discarded initial samples, is a critical issue for the application of LDhat. To address the issue, several algorithms have been developed to determine how many steps are needed to ensure the convergence of Markov chains. However, due to the complexity and specialization, direct prediction of parameters is only theoretically described and thus impractical [19]. Hence a variety of empirical tools for the diagnosis of MCMC convergence which are well designed and implemented are used, e.g. Gelman and Rubin diagnostic method [20], Brook and Giudici's method [21], Raftery and Lewis diagnostic algorithm [22].

In the next section, we presented our convergence diagnostic method based on the framework of Raftery and Lewis diagnostic algorithm [22], thereby solving the above major issue of LDhat, as well as speeding up large-scale estimation of recombination rates. For standard Markov Chain Monte Carlo algorithms, the dimension of the parameter vector is fixed, whilst in rjMCMC scheme it has varying dimensions. Normal convergence assessment algorithms cannot be applied directly to outputs from an rjMCMC sampler. Castelloe and Zimmerman's method extends the work of [20] by encompassing all of the parameter spaces and monitoring several parameters simultaneously [23] which is especially designed for rjMCMC situation. Thus we employed the method of Castelloe and Zimmerman in this paper to monitor the status of Markov chain.

2.2 Convergence diagnostic methods

Here we propose an improved algorithm for the prediction of meiotic recombination rates which makes use of convergence diagnostic methods. The original LDhat program not only takes large amount of time on calculation, but also requires users to specify the values of parameters which cannot ensure the convergence of Markov chains. Thus our main purpose is to control the process of rjMCMC iteration to monitor the Markov chain convergent status and supervise the adaptation of parameters in order to accelerate the mixing process and ensure the accuracy. To achieve these goals, the key point is to determine the appropriate number of iterations for the convergence of Markov chains.

Raftery and Lewis diagnostic algorithm and Castelloe and Zimmerman convergence assessment method are adopted in our program. The former is widely used to predict the number of iterations, burn-in and sample parameters in MCMC applications, and the latter cannot predict parameters, but it is dedicated for rjMCMC convergence diagnosis. In our program, the numbers of iterations and burn-in are determined by Raftery and Lewis diagnostic algorithm for a given level of precision. Castelloe and Zimmerman's convergence assessment method runs periodically to check if the chain has reached its target distribution.

Firstly, a pilot chain is run with initial iterations. Using the output sample, Raftery and Lewis diagnostic algorithm will generate a new Markov chain to predict how many steps are needed for each parameter to get equilibrium status and how long the burn-in should be. A reliable factor I will be calculated as well. Values of I must be greater than 1, but when $I > 5$ it often indicates problems [22]. Since a bad starting value or high posterior correlations may cause unreliable results, the estimations are just used as references of initiate settings. The set of parameters satisfying the threshold with the maximum number of iterations will be chosen as the input arguments. Sometimes all of the reliable factors are larger than 5. In that case, none of the results are reliable. The minimal iteration number is selected as the initial values of parameters. Castelloe and Zimmerman convergence assessment method will check the status repeatedly to rectify the values.

Castelloe and Zimmerman's algorithm needs multiple testing chains running for certain steps. It is particularly designed for rjMCMC convergence diagnosis. Since the solution dimention of rjMCMC is not fixed, different models with variant sizes of parameter vectors are generated. Markov chains transit between these models. Thus the evaluation of the variations within each sample, testing chains, different models, could reflect the convergent status of whole Markov chains.

Let C be the number of chains required by Castelloe and Zimmerman's method, T be the number of sweeps in each chian and $\boldsymbol{\theta}$ be a vector of parameters. M is the number of distinct models visited by any chain. R_{cm} stands for the number of times model m occurrs in chain c. The total variance $\hat{V}$ is estimated by

$$\hat{V} = \frac{1}{CT-1} \sum_{c=1}^{C} \sum_{m=1}^{M} \sum_{r=1}^{R_{cm}} (\boldsymbol{\theta}_{cm}^{r} - \bar{\boldsymbol{\theta}}_{..}^{\cdot})(\boldsymbol{\theta}_{cm}^{r} - \bar{\boldsymbol{\theta}}_{..}^{\cdot})' \tag{2}$$

where $\bar{\boldsymbol{\theta}}_{..}^{\cdot}$ is the average of all samples. Variation within chains, variation within models and variation within models and chains are defined as

$$W_c = \frac{1}{C(T-1)} \sum_{c=1}^{C} \sum_{m=1}^{M} \sum_{r=1}^{R_{cm}} (\boldsymbol{\theta}_{cm}^{r} - \bar{\boldsymbol{\theta}}_{c.}^{\cdot})(\boldsymbol{\theta}_{cm}^{r} - \bar{\boldsymbol{\theta}}_{c.}^{\cdot})' \tag{3}$$

$$W_m = \frac{1}{CT-M} \sum_{c=1}^{C} \sum_{m=1}^{M} \sum_{r=1}^{R_{cm}} (\boldsymbol{\theta}_{cm}^{r} - \bar{\boldsymbol{\theta}}_{.m}^{\cdot})(\boldsymbol{\theta}_{cm}^{r} - \bar{\boldsymbol{\theta}}_{.m}^{\cdot})' \tag{4}$$

$$W_m W_c = \frac{1}{C(T-M)} \sum_{c=1}^{C} \sum_{m=1}^{M} \sum_{r=1}^{R_{cm}} (\boldsymbol{\theta}_{cm}^{r} - \bar{\boldsymbol{\theta}}_{cm}^{\cdot})(\boldsymbol{\theta}_{cm}^{r} - \bar{\boldsymbol{\theta}}_{cm}^{\cdot})' \tag{5}$$

The above four factors reflect the convergence in different levels. The convergence assessment algorithm is used to check whether they reach the stable states. Four ratios in equations (6), (7), (8) and (9) are created to evaluate the chain mixing status. When $MPSRF_1$ and $MPSRF_2$ are settled close to 1, $\hat{V}$ and W_c, W_m and $W_m W_c$ are all settled approximately to a common value, indicating that it has achieved the desired distribution of convergence.

$$PSRF_1 = \frac{max\ eigen\ \hat{V}}{max\ eigen\ W_c} \tag{6}$$

$$PSRF_2 = \frac{max\ eigen\ W_m}{max\ eigen\ W_m W_c} \tag{7}$$

$$MPSRF_1 = max\ eigen\ [W_c]^{-1}\hat{V} \tag{8}$$

$$MPSRF_2 = max\ eigen\ [W_m W_c]^{-1} W_m \tag{9}$$

The convergence algorithm needs to calculate the inverse matrixes and eigenvalues. The complexities of them are about $O(n^3)$, where n is the dimension of the matrix, equal to the number of parameters. When the diagnosis algorithm is frequently called with a large number n, it may take considerable time. To accelerate the process, a new parameter $addon$ is defined to control this process. It is initially set to 2 and will be added by 5% of the number of SNPs in each round of convergence diagnosis. Then the iteration number will be set to $addon$ times the estimated value. Once the C chains are diagnosed as convergent, the final output is generated by combining the results of all chains.

In our improved program for estimating recombination rates, a convergence diagnostic model is invoked to estimate MCMC parameters and monitor the convergence process. The computational workflow is shown in Fig. 1.

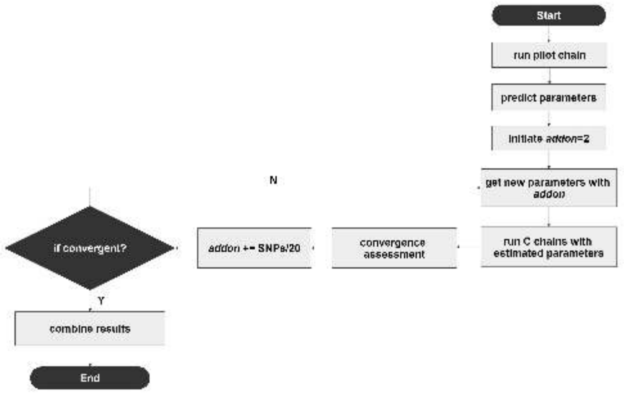


Fig. 1. Workflow of convergence diagnosis model.

Instead of manually setting values, an automatic definition process of parameters is initially run, then a convergence diagnosis procedure repeatedly to check the status. Finally, the results from each chain are combined to a final recombination profile. Since the rjMCMC assessment method requires multiple chains for diagnosis, the sequential scheme would make this new method even more time-consuming than running a single chain when the iteration loop is fixed. However, in most cases the iteration number is unknown to users, thus we cannot directly compare our improved method with the original program with the same number of iterations.

2.3 Parallel method

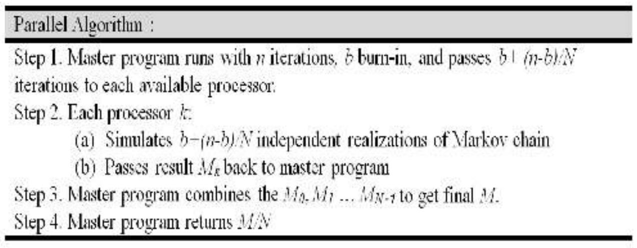
Due to the increasing availability of cheap computing power, parallel computing has received impetus. It has long been employed for scientific computing. Next, we are concerned with parallel implementation of MCMC in the context of accelerating our convergence diagnostic method.

Current algorithms for parallelizing MCMC can be classified into two main categories: one is parallelization of a single chain, and the other is parallel generation of multiple different chains [24]. Conceptually, parallel processing can be applied to almost any problem. However, MCMC is not easy to run in parallel owing to its serial nature. Due to the tight synchronization requirements of MCMC, the single-chain parallelization strategy requires considerable modification of the serial algorithm [25]. While the total iterations could be divided by multiple processors due to the independence nature of samples [25]. We concentrate on introducing a parallel algorithm that signicantly decreases the execution time on multiple short Markov chains.

Assuming that each iteration takes roughly the same time to compute, an iteration may be used as a unit of time. Since the samples collected from MCMC chains are independent, it is possible to allocate the n required samples to N available processors, where the same program is run on each processor.

For a long chain the burn-in only happens once, whereas for several short chains, each must have a respective burn-in, resulting in many wasted samples [25]. With increasing numbers of processors, the performance of parallel computation becomes limited owing to redundant burn-in. Thus the issue of burn-in is of particular concern in a parallel computing environment. Here we make each process with the same burn-in phase identical with the sequential program.

We take advantage of parallel computation integrating convergence diagnosis model. The scheme of integrated method is nearly the same as convergence diagnosis program except that the convergent diagnostic tasks are divided by N processors. Each of the C diagnostic chains is run on N/C processors. The algorithm for parallel simulation of a single Markov chain can be described in Fig. 2.



```
Parallel Algorithm :
Step 1. Master program runs with $n$ iterations, $b$ burn-in, and passes $b+(n-b)/N$
iterations to each available processor.
Step 2. Each processor $k$:
        (a) Simulates $b-(n-b)/N$ independent realizations of Markov chain
        (b) Passes result $M_k$ back to master program
Step 3. Master program combines the $M_0, M_1 \dots M_{N-1}$ to get final $M$.
Step 4. Master program returns $M/N$
```

Fig. 2. Parallel algorithm for a single Markov chain.

The theoretical speed-up is proportioned to the number of processors. However, due to the burn-in overhead, if N processors run one chain with a burn-in of b and n total iterations, then $b + (n - b)/N$ iterations are allocated for each chain. When neglecting the communication time between processors and the handling time on file combination, it gives an optimal speed-up of

$$SpeedUp_1(N) = \frac{n}{b + \frac{n-b}{N}} \tag{10}$$

Let $n = 10b$, then $SpeedUp_1(5) = 3.5714$, $SpeedUp_1(8) = 4.7059$. However, when using 10 processors, there is only 5-fold speed-up indicating that the effect of parallel computation on large clusters becomes limited. Theoretically, due to the burn-in, it could reach a maximum of 10 times speed-up, when $N \to \infty$. Furthermore, effective utilization of multiple processors is also limited due to the aggregation of communication time.

Based on a single chain parallel algorithm, the C diagnostic chains are divided into multiple sub-tasks. In Fig. 3, it shows the strategy of the parallel approach employed in our program. Each processor runs independent copies of the program with n/N iterations, and generates individual output files. The length of burn-in period keeps the same ratio with sequential execution. These numerous files are then compiled to obtain the final outputs of diagnostic chains for convergence evaluation. The implementation of this approach is done using OpenMPI programming language for communicating messages between multi-core processors.

Suppose that C parallel chains run on N processors with consistent burn-in of b and n total iterations, then n/C iterations and N/C processors are assigned to each chain. This gives a speed-up of

$$SpeedUp_2(N) = \frac{n}{b + \frac{n/C-b}{N/C}} \tag{11}$$

Empirically $n = 10bC$, $C = 5$ is a useful rule-of-thumb. We can get $SpeedUp_2(5) = 5$, $SpeedUp_2(10) = 9.09$, $SpeedUp_2(20) = 15.38$ with a maximum speed-up of 50 theoretically.

Another issue we have addressed is the random number generator. The correlation among random number streams on separated processors should be reduced by assigning identical random number seeds

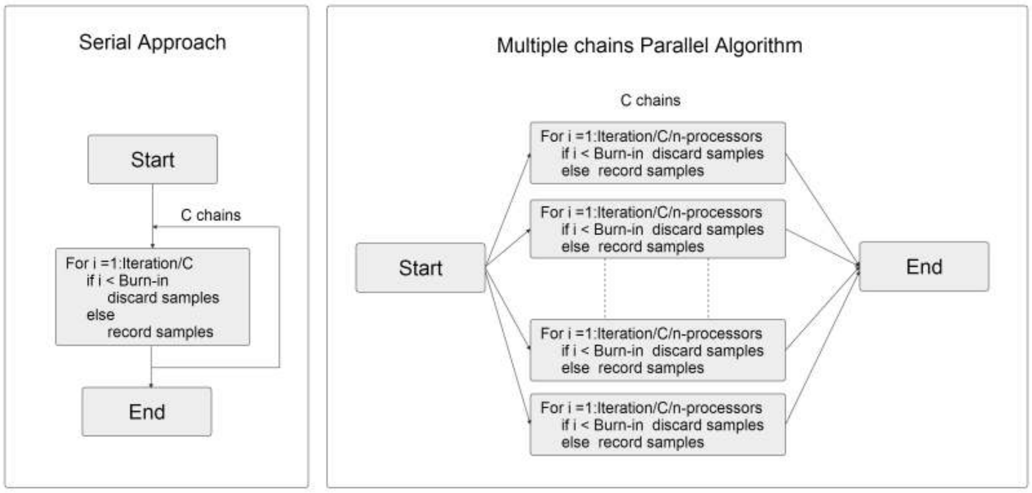


Fig. 3. Parallel algorithm for multiple chains.

to each machine [26]. The original LDhat program uses the default random number generator provided by the C language. It can only be used by one processor at a time. The other processors need to wait for their turn to obtain the random number losing the benefit of parallelization. Since each loop of the program requires random number generation over ten times, the over-all impact of improving the random number generator can be very significant. A sophisticated approach is to change the random number generator to SIMD-Oriented Fast Mersenne Twister (SFMT), which supports multicore parallel random number generation and has been shown suitable for use in Monte Carlo simulations [27]. Therefore we replace the random number generator in order to make it applicable for parallel computation.

3 Results

To investigate the performance of the new method, we conducted two comparison studies. Since the program is used for fundamental genetics studies, it is imperative that the optimization techniques used do not affect the results. The new program should expedite the calculation process meanwhile retaining the accuracy. Not only the recombination rates but also the change positions should be predicted within the acceptable deviation. Hence we analyzed recombination profiles, running time and iteration numbers to evaluate the performance of the new method. We use *LDhat* to refer to the original LDhat implementation, *CLDhat* to refer to the convergence method, and *PLDhat* to refer to the parallel approach.

Ten sets of test data with equal iterations are used to evaluate the performance. They are drawn from human genomes with different numbers of haplotypes, SNPs and sequence lengths (Table 1). In the first study, we compared recombination profiles on outputs of 10 datasets by *LDhat*, *CLDhat* and *PLDhat*. In the second study, the execution time and the number of iterations are compared to show the efficiency of our improved programs.

TABLE 1
Test datasets

Datasets	Haplotypes	SNPs	Length(kb)
Test1	48	61	9.730
Test2	180	100	38.771
Test3	120	110	35.449
Test4	50	251	504.492
Test5	120	401	796.496
Test6	70	520	1102.571
Test7	70	610	983.183
Test8	60	790	1385.201
Test9	60	850	1437.376
Test10	50	1000	2643.03

The experiments were implemented on an IBM cluster of 24 quan-CPU 2.53 GHz Intel Xeon Linux Systems, connected to each other by 100Mbps Ethernet connections.

3.1 LDhat analysis

Before the comparison studies, we analyzed the influence of parameters, i.e. the numbers of SNPs and iterations, on execution time and recombination profiles. Firstly, a test dataset with 61 SNPs is used to examine the effect of iteration number on output profiles and execution time. The iteration numbers are set 3000, 6000, 10000, 15000 and 18000 respectively. Assuming that each mutation transition consumes the same time, the iteration loop in Line 6 of LDhat pseudocode (Appendix Fig. A.1) constructs the main component of rjMCMC leading to a significant linear correlation between iterations and execution time. The comparison results in Fig. 4a shows an approximately linear

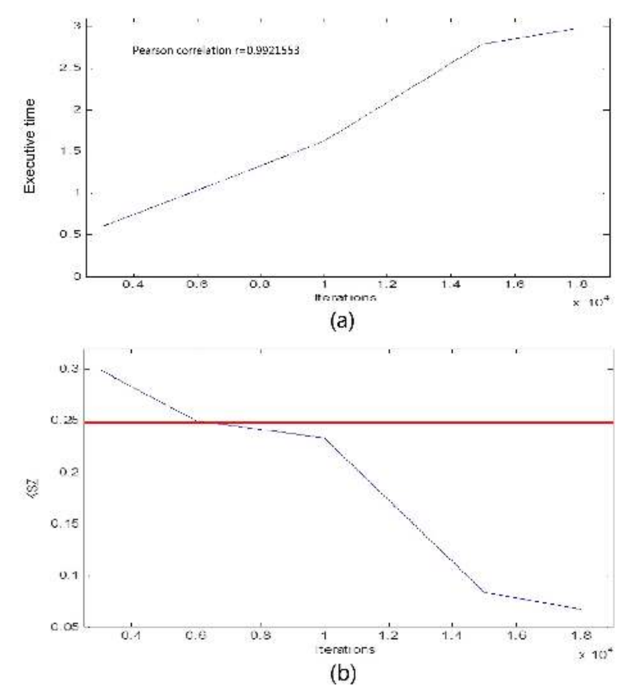


Fig. 4. Execution time(a) and KSZ(b) analysis on 61SNP test dataset with different iterations. The red line in (b) represents the reference KSZ value for the datasets with an acceptance probability of 0.95.

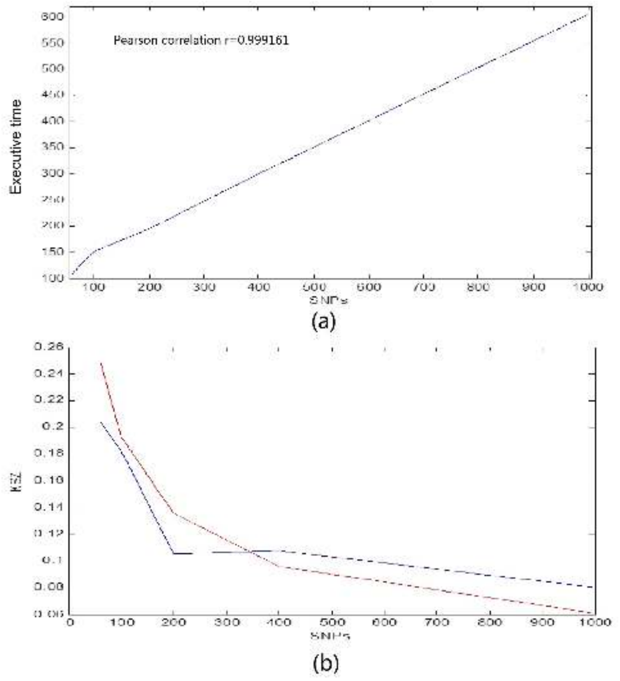


Fig. 5. Execution time(a) and KSZ(b) analysis on different size of test datasets with the same iterations. The red line in (b) represents the reference KSZ value for the individual dataset with an acceptance probability of 0.95.

relationship with $r = 0.992$. Thus controlling the loop length is important for the speed-up of LDhat. With increasing iterations, the KSZ value decreases gradually (Fig. 4b) indicating that the Markov chain is close to the target distribution. However, when the iteration is set too small, e.g. less than 6000, in this case, a deviation occurs in the output profile. On the contrary, over calculation with a large number of iterations would waste computational time without gain of additional information.

We analyzed the correlation between the number of SNPs and execution time, the number of SNPs and recombination profile with a fixed iteration number 10000. As demonstrated above, the time complexity of LDhat has a linear correlation with the number of SNPs (Fig. 5a, r=0.999). The red line in Fig. 5b shows the threshhold of KSZ values for different SNPs with an acceptance value of 0.95. For small scale datasets, the setting of 10000 iterations is enough for the convergence of markov chains. When the number of SNPs exceeds 400, more training is required to make the chains reach the target distribution.

3.2 Comparison of recombination profiles

In the first study, we conduct experiments to compare recombination profiles of *LDhat*, *CLDhat* and *PLDhat* on the 10 datasets in Table 1. For all datasets, *LDhat* is running for 11 million iterations, and the initial 1,000,000 samples are discarded as burn-in. Samples of the chain are taken every 2,000 iterations after the burn-in. Then the output recombination rates are recorded as control groups to evaluate other methods. By contrast, we don‘t have to specify the numbers of iteration, burn-in and samples in the *CLDhat* method. By convention, 5 chains are generated to check the mixing status [28].

For *PLDhat*, the sequential procedure is divided into 5 parallel tasks making use of 15 processors. One processor operates as the master running the Raftery and Lewis diagnostic algorithm to estimate the parameters and control the process of convergence assessment. Every 3 processors are applied to generate a single chain with parameters received from master processor.

Comparing the output graphs, the *CLDhat* and *PLDhat* methods got almost the same figures as the original program (Appendix Fig. A.1). Although the peak values are slightly changed in some points, the outputs showed high correlation coefficients among the three methods. The error is acceptable by KS test (see methods). Fig. 6 shows the KSZ of 10 datasets for our improved methods with reference values in red line. In most cases, the KSZ values of *CLDhat* are smaller than *PLDhat*. This may be due to the loss of accuracy in frequent ‘split-and-combine‘ process during parallel computation. But for small datasets, it converges more quickly with no need for frequently

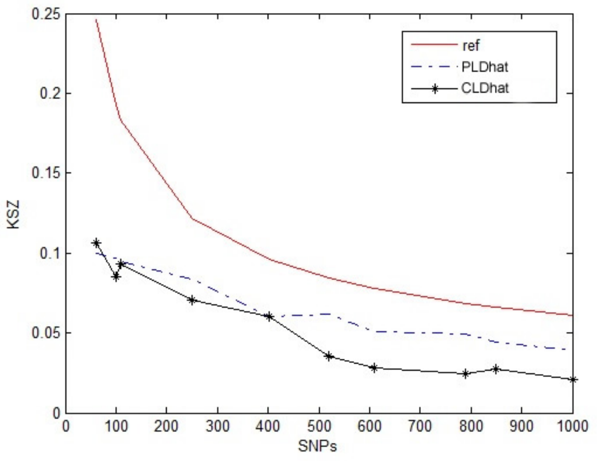


Fig. 6. KSZ values on the 10 datasets by *CLDhat* and *PLDhat*. The red line indicates reference values.

calling diagnosis and combination. Even so, both the outputs of *CLDhat* and *PLDhat* are under the threshold indicating the accuracy of our new methods.

Take Test 2 for instance. Fig. 7a shows the posterior distribution of the number of hotspots. More than 50% of models contain 3 hotspots. Conditional on values of hotspots k=3 and 4, Fig. 7b shows the posterior densities of the step positions. The positions of three hotspots are accurately identified. The density estimates are obtained using a Gaussian kernel with standard deviation.

3.3 Comparison of running time

In the second study, the total time consumed by *LDhat* and the improved methods on the 10 datasets are shown in Fig. 8a (details in Appendix Table A.2). The execution time was tremendously decreased when using our new methods. There are almost 80 times speed-up in *CLDhat*. Using *PLDhat* on 15 processors, we got 622 times acceleration. In Fig. 8b, it shows the separate running time of each test data for *LDhat*, *CLDhat* and *PLDhat*. As the number of SNPs increases, our methods take a linear growth in time which is consistent with previous analysis in section 2.1.

Unlike *LDhat* program, *CLDhat* method is a non-parameter approach under rjMCMC scheme. It is a more reliable and faster method. The mixing process is automatically monitored and checked periodically for convergence. So the MCMC chain could reach the equilibrium distribution rapidly in moderate iteration. In Table 2, the iteration numbers of test datasets by *CLDhat* are significantly decreased compared with *LDhat* leading to an expressively optimization of time efficiency. The *PLDhat* approach has successfully obtained more significant speed-up than *CLDhat*.

We replace the original random number generator with SFMT for parallel computation. Since the program frequently requests for random number generation and SFMT is an efficient and faster random number generator, the replacement of original function reaches approximately 3 times speed-up (data not shown). For large datasets, such as the calculation of Test 7-10, they take more iterations for convergence when the accelerating effect by parallel becomes more apparent.

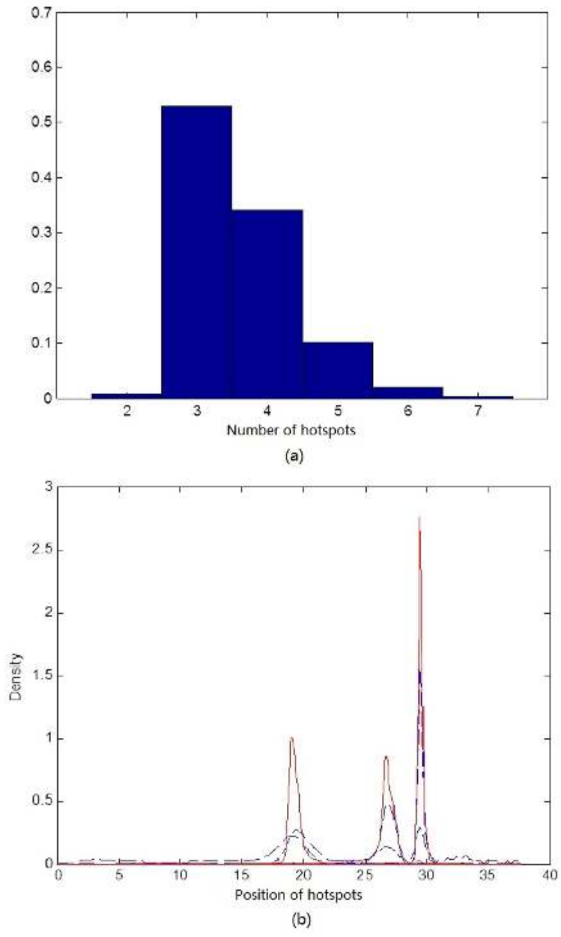


Fig. 7. Posterior distribution of the number of hotspots (a) and posterior density estimates of positions of hotspots (b), conditional on the number of hotspots k=3(solid curve) and k=4(broken curves).

The parameter $addon$ controls the span length of each diagnosis round that correlates with the number of SNPs which makes large datasets mix faster. Conversely, this jumping scheme is suboptimal for small datasets.

4 Discussion and Conclusions

The main purpose of optimization of LDhat is to decrease the time complexity and increase the accuracy and reliability of output recombination profiles. Besides, there are no strategy to set the rjMCMC parameters in the original LDhat program, such as

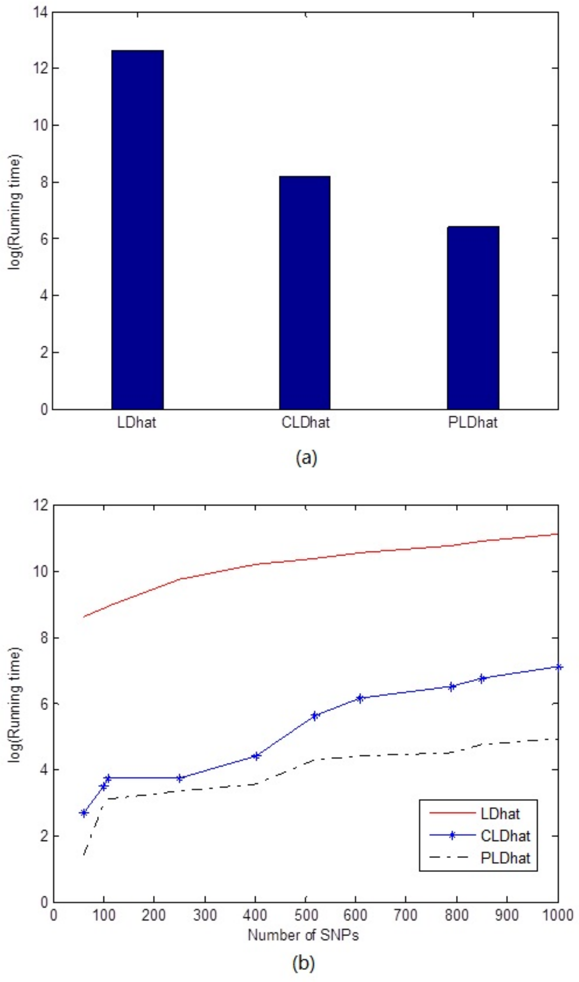


Fig. 8. Total running time(a) and seperate execution time(b) on the 10 datasets by *LDhat*, *CLDhat* and *PLDhat*.

TABLE 2
Iterations by *CLDhat* and *PLDhat*

Datasets	CLDhat	PLDhat
Test1	156109	22561
Test2	185691	23768
Test3	293663	33746
Test4	297257	68746
Test5	390294	108746
Test6	1525275	144821
Test7	2395086	158746
Test8	3215746	297466
Test9	3695314	396121
Test10	4721811	447752

the iteration number, burn-in length and sample frequency. The bottleneck identified as the main loop in the original LDhat program is normally suggested to carried out a million iterations or more which may result in over calculation or insufficient running.

In this paper, we exploited MCMC convergence diagnostic algorithms and proposed two improved methods based on LDhat. A major advantage of the new methods is significant acceleration compared with original program. In addition, the parameters are automatically estimated by our algorithms and only depend on input data. The mixing process is dynamic and monitored until the Markov chain reaches its target distribution. This could avoid unnecessary consumption of resources while also guarantees the accuracy of outputs.

Although the running time of the convergence method is tremendously decreased compared to the original program, it was further improved by implementation of parallel computation method due to the sequential scheme of the generation process of diagnostic chains. Hence we developed a parallel algorithm to allocate separate tasks to individual processors running a single chain in parallel. It achieves significant speed-up.

The outputs of the above two methods were compared with the original LDhat program which showed similar output graphs. Since the results were generated through strict convergence assessment procedure, our methods achieved low values of KSZ (i.e. high accuracy) in much less iterations presenting extraordinarily similar recombination rate profiles.

Therefore our improved programs provide efficient and accurate methods for recombination rate prediction. Especially the parallel program provides a practicable, time saving and effective method. The improved methods, *CLDhat* and *PLDhat*, including the original LDhat (*rhomap*) program are implemented in a stand-alone package written in Java which is freely available for download at web site http://www.ntu.edu.sg/home/zhengjie/software/C-PLDhat/. It could run in both Linux and Windows OS.

Appendix A

Table A.1. Pseudocode of reversible jump MCMC algorithm in LDhat.
Table A.2. Execution time on 10 datasets by *LDhat*, *CLDhat* and *PLDhat*.
Fig. A.1. Comparison of recombination profiles for 10 datasets by *LDhat*, *CLDhat* and *PLDhat*.

Acknowledgments

This project is supported in part by Singapore Ministry of Education (MOE) AcRF Tier 1 Grant RG32/11.

References

[1] E. Shabanova, *Patterns of genetic recombination and variation in the human genome* ,Universitt zu Kln, 2009.
[2] J. C. Avise, Phylogeography: the history and formation of species: Harvard University Press, 2000.
[3] B. L. Dumont, and B. A. Payseur, *Genetic analysis of genome-scale recombination rate evolution in house mice* ,PLoS genetics, vol. 7, no. 6, pp. e1002116, 2011.

[4] U. Mller, *Ten years of gene targeting: targeted mouse mutants, from vector design to phenotype analysis* ,Mechanisms of development, vol. 82, no. 1, pp. 3-21, 1999.

[5] N. J. Risch, *Searching for genetic determinants in the new millennium* ,Nature, vol. 405, no. 6788, pp. 847-856, 2000.

[6] R. Hubert, M. MacDonald, J. Gusella, and N. Arnheim, *High resolution localization of recombination hot spots using sperm typing* ,Nature genetics, vol. 7, no. 3, pp. 420-424, 1994.

[7] R. A. Gibbs, J. W. Belmont, P. Hardenbol, T. D. Willis, F. Yu, H. Yang, L.-Y. Ch'ang, W. Huang, B. Liu, and Y. Shen, *The international HapMap project* ,Nature, vol. 426, no. 6968, pp. 789-796, 2003.

[8] R. R. Hudson, *Two-locus sampling distributions and their application* ,Genetics, vol. 159, no. 4, pp. 1805-1817, 2001.

[9] G. McVean, P. Awadalla, and P. Fearnhead, *A coalescent-based method for detecting and estimating recombination from gene sequences* ,Genetics, vol. 160, no. 3, pp. 1231-1241, 2002.

[10] A. Auton, and G. McVean, *Recombination rate estimation in the presence of hotspots* ,Genome research, vol. 17, no. 8, pp. 1219-1227, 2007.

[11] N. Li, and M. Stephens, *Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data* ,Genetics, vol. 165, no. 4, pp. 2213-2233, 2003.

[12] Y. Wang, and B. Rannala, *Bayesian inference of fine-scale recombination rates using population genomic data* ,Philosophical Transactions of the Royal Society B: Biological Sciences, vol. 363, no. 1512, pp. 3921-3930, 2008.

[13] P. J. Green, *Reversible jump Markov chain Monte Carlo computation and Bayesian model determination* ,Biometrika, vol. 82, no. 4, pp. 711-732, 1995.

[14] P. Beerli, and J. Felsenstein, *Maximum-likelihood estimation of migration rates and effective population numbers in two populations using a coalescent approach* ,Genetics, vol. 152, no. 2, pp. 763-773, 1999.

[15] G. Watterson, W. J. Ewens, T. Hall, and A. Morgan, *The chromosome inversion problem* ,Journal of Theoretical Biology, vol. 99, no. 1, pp. 1-7, 1982.

[16] A. J. Jeffreys, A. Ritchie, and R. Neumann, *High resolution analysis of haplotype diversity and meiotic crossover in the human TAP2 recombination hotspot* ,Human Molecular Genetics, vol. 9, no. 5, pp. 725-733, 2000.

[17] M. I. Jensen-Seaman, T. S. Furey, B. A. Payseur, Y. Lu, K. M. Roskin, C.-F. Chen, M. A. Thomas, D. Haussler, and H. J. Jacob, *Comparative recombination rates in the rat, mouse, and human genomes* ,Genome research, vol. 14, no. 4, pp. 528-538, 2004.

[18] H. W. Lilliefors, *On the Kolmogorov-Smirnov test for normality with mean and variance unknown*, Journal of the American Statistical Association, vol. 62, no. 318, pp. 399-402, 1967.

[19] M. Plummer, N. Best, K. Cowles, and K. Vines, *CODA: Convergence diagnosis and output analysis for MCMC* ,R news, vol. 6, no. 1, pp. 7-11, 2006.

[20] A. Gelman, and D. B. Rubin, *Inference from iterative simulation using multiple sequences* ,Statistical science, pp. 457-472, 1992.

[21] S. Brooks, and P. Giudici, *Markov chain Monte Carlo convergence assessment via two-way analysis of variance* ,Journal of Computational and Graphical Statistics, vol. 9, no. 2, pp. 266-285, 2000.

[22] A. E. Raftery, and S. Lewis, *How many iterations in the Gibbs sampler* ,Bayesian statistics, vol. 4, no. 2, pp. 763-773, 1992.

[23] J. M. Castelloe, and D. L. Zimmerman, *Convergence assessment for reversible jump MCMC samplers* ,Department of Statistics and Actuarial Science, University of Iowa, Technical Report, vol. 313, 2002.

[24] J. Ye, A. M. Wallace, A. Al Zain, and J. Thompson, *Parallel Bayesian inference of range and reflectance from LaDAR profiles* ,Journal of Parallel and Distributed Computing, 2012.

[25] D. J. Wilkinson, *Parallel bayesian computation* ,STATISTICS TEXTBOOKS AND MONOGRAPHS, vol. 184, pp. 477, 2006.

[26] A. Brockwell, *Parallel Markov chain Monte Carlo simulation by pre-fetching* ,Journal of Computational and Graphical Statistics, vol. 15, no. 1, pp. 246-261, 2006.

[27] M. Saito, and M. Matsumoto, *SIMD-oriented fast Mersenne Twister: a 128-bit pseudorandom number generator*, Monte Carlo and Quasi-Monte Carlo Methods 2006, pp. 607-622: Springer, 2008.

[28] M. K. Cowles, and B. P. Carlin, *Markov chain Monte Carlo convergence diagnostics: a comparative review*, Journal of the American Statistical Association, vol. 91, no. 434, pp. 883-904, 1996.

Appendix A

TABLE A.1
Pseudocode of Reversible Jump MCMC Algorithm in LDhat.

Input: ($mydata, mylikelihood, N_{its}, N_{burn}, N_{sample}$), where $mydata \to lseq$ is the number of SNPs, $mylikelihood \to l_{ij}$ contains the composite likelihood of data, $N_{its}, N_{burn}, N_{sample}$ stand for the numbers of iteration, burn-in and sample.

```
1: Set the current model indicator as $M = (M_{block}, M_{hotspot})$
2: Initialize $N_{block} = lseq - 1, h_j \sim Exp(\phi), where\ j \in [0, N_{block} - 1]$
3: Initialize $N_{hotspot} = 0, f_k \sim \lambda Laplas(\mu, b), where\ k \in [0, N_{hotspot} - 1]$
4: Set interval rate $rmap[j] = h_j + f_k$, where $f_k$ is the contribution from hotspot in $[s_j, s_{j+1}]$
5: Estimate the log likelihood ratio and update $l_{ij}$
6: For $i = 0$ to $N_{its}$ then
7:      $\triangle$ block model
8:      Draw a uniform random variable $u \sim U(0, 1)$
9:      if $u < b_k$ then
10:             Do death move
11:             $M'_{block} = M_{block} - 1$
12:             Procedure Metropolis-hasting sampling $(M, M')$
13:                 Update $rmap$ and $l_{ij}$
14:                 Compute the acceptance probability $\alpha(M, M')$
15:                 Draw a uniform random variable $u \sim U(0, 1)$
16:                 if $i < \alpha$ then
17:                     Accept the proposal state and set $(M, M')$
18:                 else
19:                     Remain in current state
20:                 end if
21:             Metropolis-hasting sampling $(M_{block}, M'_{block})$
22:       else if $u < b_k + d_k$ then
23:             Do birth move
24:             $M'_{block} = M_{block} + 1$
25:             Metropolis-hasting sampling $(M_{block}, M'_{block})$
26:       end if
27:       Do height change move
28:       $M'_{block} = M_{block}$
29:       Metropolis-hasting sampling $(M_{block}, M'_{block})$
30:       Do position change move
31:       $M'_{block} = M_{block}$
32:       Metropolis-hasting sampling $(M_{block}, M'_{block})$

33:       $\triangle$ hotspot model
34:       if $i > N_{burn}/4$ then
35:             Draw a uniform random variable $u \sim U(0, 1)$
36:             if $u < h\_b_k$ then
37:                 Do delete move
38:                 $M'_{hotspot} = M_{hotspot} - 1$
39:                 Metropolis-hasting sampling $(M_{hotspot}, M'_{hotspot})$
40:             else if $u < h\_b_k + h\_d_k$ then
41:                 Do insert move
42:                 $M'_{hotspot} = M_{hotspot} + 1$
43:                 Metropolis-hasting sampling $(M_{hotspot}, M'_{hotspot})$
44:             end if
45:             Do heat change move
46:             $M'_{hotspot} = M_{hotspot}$
47:             Metropolis-hasting sampling $(M_{hotspot}, M'_{hotspot})$
48:             Do position change move
49:             $M'_{hotspot} = M_{hotspot}$
50:             Metropolis-hasting sampling $(M_{hotspot}, M'_{hotspot})$
51:             Do scale change move
52:             $M'_{hotspot} = M_{hotspot}$
53:             Metropolis-hasting sampling $(M_{hotspot}, M'_{hotspot})$
54:             if $i > N_{burn}$ and $i\%N_{sample} == 0$ then
55:                 Save the current $rmap$
56:             end if
57:       end if
58:end for
```

TABLE A.2
Execution time(second) on the 10 datasets by *LDhat*, *CLDhat* and *PLDhat*[*].

Datasets	LDhat	CLDhat	PLDhat	ratio(CLDhat/PLDhat)
Test1	5450.75	14.716	4.16	3.5
Test2	7034.23	33.076	19.246	2.2
Test3	7435.58	42.304	22.61	1.5
Test4	16749.4	64.646	28.87	2.2
Test5	27033.9	81.928	34.61	2.4
Test6	31664.5	281.322	74.22	3.8
Test7	38172.7	466.916	82.71	5.6
Test8	46151.5	671.952	91.85	7.3
Test9	54725.6	848.074	117.4	7.2
Test10	67595.6	1221.232	137.18	8.9

[*] Parallel program is running on 15 processors.

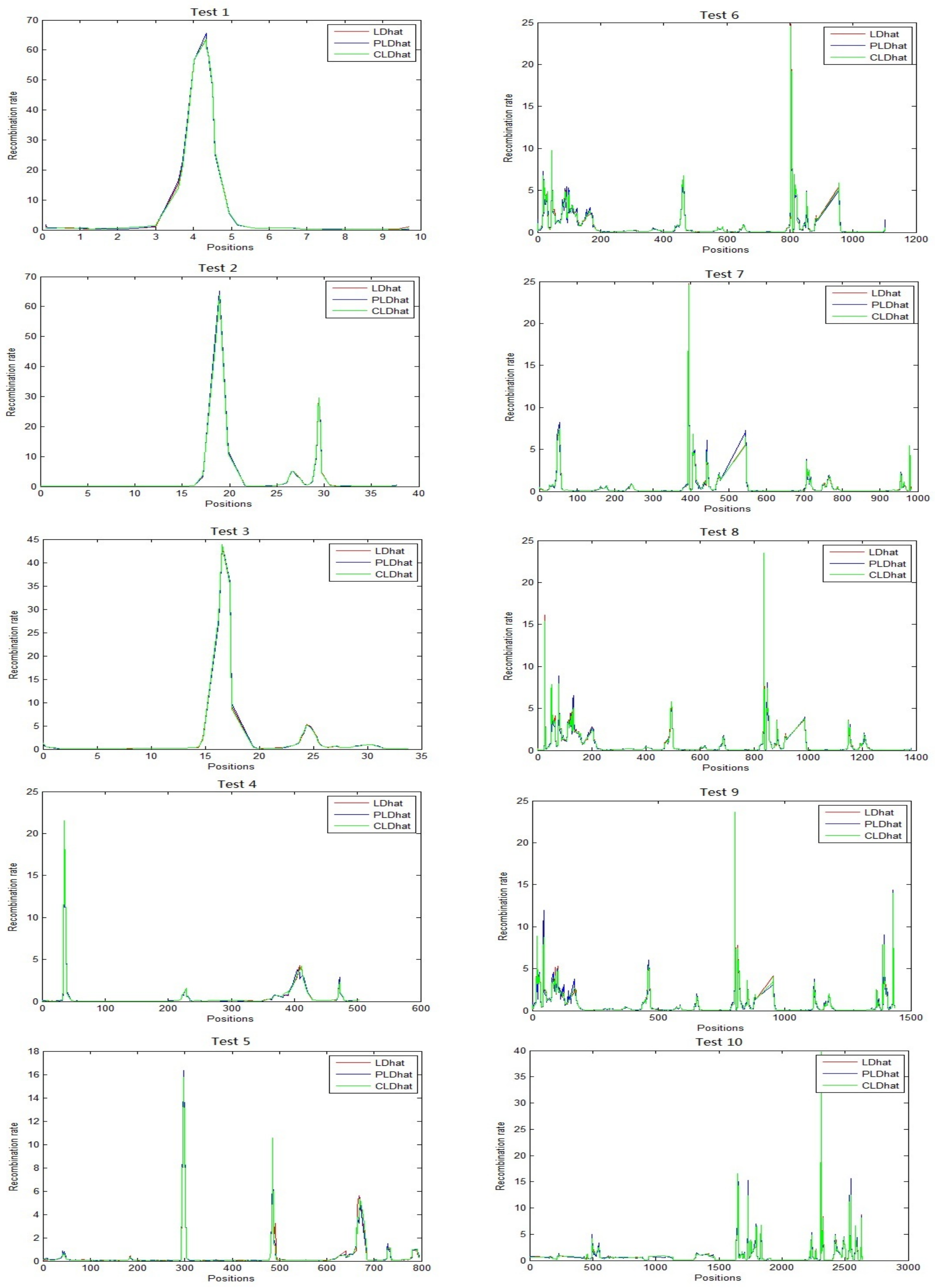


Fig. A.1. Recombination rate profiles comparison for the 10 datasets by *LDhat*, *CLDhat* and *PLDhat*.