GWAsimulator: A rapid whole-genome simulation program

Version 1.1

Chun Li and Mingyao Li

September 21, 2007 (revised October 9, 2007)

1. Introduction

GWAsimulator is a C++ program that simulates genotype data for SNP chips. It implements a rapid moving-window algorithm (Durrant et al. 2004) to simulate whole genome case-control or population samples. For case-control data, the program retrospectively sample cases and controls according to user-specified multi-locus disease models. The program can use HapMap phased genotype data as input and has the flexibility of simulating genotypes for different populations and SNP chips. When the HapMap data are used as the input, the simulated data have similar linkage disequilibrium (LD) patterns as the HapMap data.

The simulation algorithm faithfully follows the local LD structure of the input phased data. Switching to a different population or SNP chip only requires a simple change of input files. The program is efficient in several aspects. Simulated data are internally stored as bit vectors, which minimizes the amount of memory and allows data simulation of a large sample size. Unlike prospective simulation algorithms, GWAsimulator samples cases and controls retrospectively and avoids data throwaway. In addition, there is no need to store a large pool of population chromosomes to sample from. Compared to prospective and coalescence based algorithms, GWAsimulator is faster, making it feasible for evaluating the performance of GWA analysis methods through realistic simulations. The program is also easy to be built upon with user's data analysis functions; this avoids saving whole-genome data to files, which can be time-consuming. The program has been successfully applied in our methodology development (Li et al. 2008).

2. Download and compile the program

GWAsimulator is portable to a variety of operating systems. The program can be downloaded from http://biostat.mc.vanderbilt.edu/GWAsimulator. Once the package is downloaded, unzip the package into a directory using software such as WinZip (in Windows) or command line (in Unix/Linux/Mac OS X)

```
tar -zxf GWAsimulator_v1.1_linux.tar.gz
```

The package should include the program source code (`GWAsimulator.cpp`), a compilation shortcut program (`build`), an example control file (`control.dat`), an example data analysis program (`dataanalysis.cpp`), a manual (`GWAsimulator.pdf`; this file), and a subdirectory of example input phased data files.

Precompiled executable files are available for Windows/x86, Linux/x86, and Mac OS (a universal program for x86/ppc/ppc64). **(1)** The Windows/x86 version (`GWAsimulator.exe`) was compiled by Cygwin g++ v3.4.4 on Window XP with service pack 2. However, the feature of output file compression relies on an external program, gzip, which may not be available in many Windows systems. If you need this feature, you can follow the instructions in the `README.txt` file in the Windows package to install gzip. **(2)** The Linux/x86 version (`GWAsimulator`) was compiled by g++ v4.1.2 on Linux 2.6.20. **(3)** The Mac OS x86/ppc/ppc64 universal program (`GWAsimulator`) was compiled by g++ 4.0.1 on Mac OS X 10.4.

Although the executable file may be ready to use, you can always recompile the program. Recompilation often can take advantage of the latest compiler technology and can optimize the program to your local hardware/software configurations.

The program g++ can be used to compile the program. Please check the version of the g++, because different versions of the compiler require different declarations of `bit_vector`, the data type used by the program to store simulated genotype data. To check the version, use

```
g++ --version
```

If the version is 3.4 or later, `bit_vector` needs to be declared using the following statement near the beginning (this is the default):

```
typedef std::vector<bool, std::allocator<bool> > bit_vector;
```

For earlier versions of g++, this line may need to be removed or commented out. To compile the program, you can use the compilation shortcut program by issuing

```
./build
```

Alternatively, you can issue the full command

```
g++ -O2 -lm -o GWAsimulator GWAsimulator.cpp
```

3. Input phased data

The program requires the user to provide phased data, in 23 files, as its input. The 23 input files should be named consistently with the only difference being the chromosome number, from 1 to 23 (for chromosome X). In the input files, each row is a phased chromosome, with a space between the alleles at two adjacent SNPs. The program only considers biallelic SNPs, and the alleles should be coded as either 0 or 1. The pathname of the input files will be specified by the user in the control file.

If one wants to simulate data for a SNP chip and the input data are obtained from the HapMap database (www.hapmap.org), phased data for only the SNPs on the chip should be extracted and used

as input data. Each phased chromosome should be a row in the input file and will be assumed to be unrelated to the other chromosomes. For the HapMap data, the numbers of unrelated phased autosomes and X chromosomes should be 120 and 90 for both CEU and YRI, 90 and 68 for CHB, and 90 and 67 for JPT. Simulation of a different population or a different SNP chip requires a simple change to a different set of input files.

4. Control file

The program requires a control file, in which the user specifies various parameters for the program to use. An example control file is structured as the following:

```
/home/lic3/GWASIMUL/Illumina300KCEU/chr   _Illu300k.phased
120     90
1       genotype
5
1000    1000    1000    1000
7       0
0.05
19      2885    1       1.5     1000    4000
11      9067    1       1.5     8000    10000
23      7607    0       1.5     7000    9000
20      3357    0       1.1     1000    5000
18      9659    1       1.1     6000    10000
2       10714   0       1.1     10000   12000
6       4322    1       1.1     3000    5600
```

Line 1 contains the prefix and affix of the pathname of the input data files. For each chromosome, the program will insert the chromosome number (single digit for 1-9 and two digits for 10-23) to obtain the full pathname for the input file. For example, for the two strings in the above example control file, the input file for chromosome 8 is accessed through pathname `/home/lic3/GWASIMUL/Illumina300KCEU/chr8_Illu300k.phased`

Line 2 specifies the number of independent phased chromosomes in the input files. The first number is for autosomes and the second is for chromosome X. If the HapMap phased data are used to prepare the input files, the numbers should be 120 and 90 for both CEU and YRI, 90 and 68 for CHB, and 90 and 67 for JPT. As explained in Section 3, each row in the input files should be a phased chromosome.

Line 3 specifies if the simulated data will be output to files (0=no output, 1=output) and the format of the output files. If the simulated data are not output to files, the program should be incorporated with an analysis program so that the simulated data can be analyzed on the fly without being stored to files.

If the data are output to files, three formats are available in this version.

(1) `linkage`: each row is a person, with pedigree ID, individual ID, father ID, mother ID, sex (1=male, 2=female), affection status (0=unknown, 1=unaffected, 2=affected), followed by SNP alleles (two alleles per SNP, "1"=allele 0, "2"=allele 1).

(2) `genotype`: each row is a person and each column is a SNP, with genotype 0, 1, 2, as the number of copies of allele 1. For X chromosome, male genotypes are 0 or 2.

(3) `phased`: a person has two rows, each being a phased chromosome.

The files will be saved in the directory where the program was started and gzipped to save disk space. They are named `chr#.dat.gz`, where # is the chromosome number (single digit for 1-9 and two digits for 10-23). The program will remove all existing files named `chr#.dat` or `chr#.dat.gz` before saving the new files. For regional simulation (see line 6), only simulated chromosomes are saved.

Note: The files can be big and file saving can take significant amount of time.

Line 4 contains the moving window size for data generation. The default is 5, that is, every 4-SNP haplotype is used to simulate the allele of the next SNP. The program will generate a warning message if a user specifies a window size that is too large compared to the number of input phased chromosomes ($2^{(\text{windowsize}+1)} \geq$ number of input chromosomes).

Line 5 specifies the numbers of female cases, male cases, female controls, and male controls. If the number of disease loci on line 6 is zero, population samples are simulated, with the numbers of females and males being those for female controls and male controls.

Line 6 specifies the number of disease loci and indicator for regional simulation. If the number of disease loci is zero, population samples will be simulated, and the regional indicator and the rest of the control file will be ignored.

If the number of disease loci is positive, case-control samples will be simulated, and the program continues to read additional information of the control file to obtain disease model information. For case-control sampling, if the regional indicator is 0, genomic data (genotypes for all SNPs in input data) will be simulated, and the start/end position information (see below) is not required and will be ignored when present. If the regional indicator is 1, start and end positions (see below) for each disease chromosome are required, and the program generates (and outputs) genotype data only for the specified regions.

Note: If you wish to simulate regional data for population sampling, a trick is to use case-control sampling but specify a disease model with disease genotype relative risks as 1. Similarly, to simulate regional data for non-disease chromosomes in case-control sampling, include the chromosomes as disease chromosomes but with genotype relative risks as 1.

Line 7 is the disease prevalence.

Starting from line 8, each line contains information for a disease locus: chromosome number, position, disease variant allele, genotypic relative risk, and optionally, the start and end positions (see line 6 explanations). The disease locus position is the position in the input phased file, not the physical position on a chromosome. For example, the first disease locus in the above example control file is the 2885th SNP in the chromosome 19 input file, with allele 1 as the disease risk allele. The genotypic relative risk is the risk ratio of the genotype with one copy of the risk allele versus that with zero copy of the risk allele. The start and end positions are required only when the number of disease loci is >0 and the regional indicator is 1 (both on line 6), and will be ignored otherwise.

The current version of the program allows one disease variant per chromosome. See Section 9 for more details on disease model determination.

Note: The number of disease loci (line 6) determines how many extra lines the program will read in. Any additional lines will be ignored. Therefore, the extra lines in the control file that

accompanies the software distribution can be kept to remind the user of the parameters and options.

5. Command line

To start simulation, the command line will need to have two arguments: control file and seed number. The seed number should be a positive integer and should be different for each simulation replicate. It is the seed number for the random number generator that is used by the program. This design allows the user to have a full control over the simulation, which may be useful especially for cluster use (see section 7). An example command line is:

```
./GWAsimulator control.dat 7845412
```

6. Output

The program will output its progress to the standard output (often it is the screen). This can be redirected, especially when running on cluster nodes for which standard output is not available. The command for redirection may vary across different shells. The user should consult the manual for the shell being used.

If case-control data are simulated, the program will output disease model information into a file named `diseasemodel.txt`. See Section 9 for more details on disease model determination.

If the simulated data are output to files (i.e., the first value of line 3 in control file is 1), data files will be generated with names `chr#.dat.gz`, where # is the chromosome number (single digit for 1-9 and two digits for 10-23). Note that the files can be big and file saving can take significant amount of time. See Section 4 for explanations of the available output formats.

7. Cluster use

The seed number should be different across simulation replicates. Seed number generation depending on machine time often is not a good idea because it can result in the same seed number across different simulation replicates, especially when the program is executed at about the same time on multiple nodes. It is better to first generate a long list of seed numbers, and then use those numbers sequentially in each simulation replicate.

Different simulation replicates need to be started from different directories to avoid over-writing each other.

8. Incorporation of user's analysis function

The program can be easily built upon with user's functions for further data analysis. This way, there is no need to save whole-genome simulated data to file. The software distribution has an example data analysis program (`dataanalysis.cpp`), which calculates the allelic chi-squared test statistics for all the simulated SNPs and store them in a single array. To try this program, simply

uncomment the statement

```
#include "dataanalysis.cpp"
```

before the `main()` function and the function call

```
dataanalysis(NUMCASEF, NUMCASEM, NUMCONTF, NUMCONTM);
```

in the `main()` function, then recompile the program using the compilation shortcut program `./build`. User's own programs can be similarly incorporated into GWAsimulator. See Section 2 for more details on compilation.

9. Determination of disease model

For simulations of case-control data, a disease model is needed. The program allows the user to specify the disease model parameters, including disease prevalence, the number of disease loci, and for each disease locus, its location, risk allele, and genotypic relative risk. The frequencies of the risk alleles can be calculated based on the input phased data.

Suppose the user wants to generate an m-locus model with prevalence K. Let $g_i = 0, 1, 2$ denote the number of copies of the risk allele at SNP i ($i = 1, \ldots, m$). Let p_i be the risk allele frequency at SNP i ($i = 1, \ldots, m$) and r_i be the risk ratio of genotype 1 versus genotype 0. Assuming Hardy-Weinberg equilibrium, the population genotypic frequencies are $\Pr(0) = (1-p_i)^2$, $\Pr(1) = 2p_i(1-p_i)$, and $\Pr(2) = p_i^2$. Let $f(g_1, \ldots, g_m) = \Pr(\text{affected} \mid g_1, \ldots, g_m)$ denote the penetrance for genotype $\{g_1, \ldots, g_m\}$. The program assumes the penetrance is a function of the genotypes such that

$$\text{logit}[f(g_1, \ldots, g_m)] = \alpha + \beta_1 g_1 + \ldots + \beta_m g_m ,$$

where $\text{logit}(x) = \ln[x/(1-x)]$. The values of α and β_i will be determined by the program so that the model's genotypic relative risks and prevalence agree with those specified by the user. We use the following approximations to determine the values of α and β_i .

Determination of β_i: We assume the marginal model for SNP i is approximately $\text{logit}[f(g_i)] = \alpha_i + \beta_i g_i$. To simplify the notation, let f_0, f_1, f_2 be the marginal penetrances for genotypes $g_i = 0, 1, 2$, respectively. Since $f_0 = 1/[\exp(-\alpha_i) + 1]$ and $f_1 = 1/[\exp(-\alpha_i)\times\exp(-\beta_i) + 1]$, we have

$$r_i = f_1/f_0 = [\exp(-\alpha_i) + 1]/[\exp(-\alpha_i)\times\exp(-\beta_i) + 1],$$

and

$$\beta = -\ln\left\{\left[\frac{\exp(-\alpha_i)+1}{r_i} - 1\right] / \exp(-\alpha_i)\right\}.$$

We need to calculate $\exp(-\alpha_i)$. For an autosome, since $K = (1-p_i)^2 \times f_0 + 2p_i(1-p_i) \times f_1 + p_i^2 \times f_2$, we have

$$\begin{aligned} K / f_0 &= (1-p_i)^2 + 2p_i(1-p_i) \times f_1 / f_0 + p_i^2 \times f_2 / f_0 \\ &\approx (1-p_i)^2 + 2p_i(1-p_i) \times r_i + p_i^2 \times r_i^2. \end{aligned}$$

Then $\exp(-\alpha_i)$ can be calculated as

$$\exp(-\alpha_i) = 1/f_0 - 1 \approx [(1-p_i)^2 + 2p_i(1-p_i) \times r_i + p_i^2 \times r_i^2] / K - 1.$$

For X chromosome, since $K = ½ (1-p_i)^2 \times f_0 + p_i(1-p_i) \times f_1 + ½ p_i^2 \times f_2 + ½ (1-p_i) \times f_0 + ½ p_i \times f_2$, we have

$$K / f_0 = ½ (1-p_i)(2-p_i) + p_i(1-p_i) \times f_1 / f_0 + ½ p_i(1+p_i) \times f_2 / f_0$$

$$\approx \tfrac{1}{2}(1-p_i)(2-p_i) + p_i(1-p_i) \times r_i + \tfrac{1}{2} p_i(1+p_i) \times r_i^2.$$

Then $\exp(-\alpha_i)$ can be similarly calculated.

Determination of α: Once we have all the values of β_i, the penetrance of each multi-locus genotype $\{g_1,\ldots,g_m\}$ is a function of α, that is, $f(g_1,\ldots,g_m) = 1/\{\exp[-(\alpha + \beta_1 g_1 + \ldots + \beta_m g_m)] + 1\}$. Using the prevalence constraint, we can solve the following equation for α:

$$K = \sum_{g_1} \cdots \sum_{g_m} \Pr(g_1,\ldots,g_m) f(g_1,\ldots,g_m).$$

For example, for the disease model parameters in the example control file in Section 4, the final disease model is

```
beta0 =   -2.7433
Locus  chromosome   #SNPs  DLposition  DV  DVFreq   GRR     beta
    1          19    5789        2885   1  0.0500   1.500   +0.4308
    2          11   14520        9067   1  0.3417   1.500   +0.4246
    3          23    9120        7607   0  0.5000   1.500   -0.4218
    4          20    7802        3357   0  0.4333   1.100   -0.1001
    5          18   10441        9659   1  0.1417   1.100   +0.1004
    6           2   25215       10714   0  0.2417   1.100   -0.1003
    7           6   20269        4322   1  0.0667   1.100   +0.1005
```

The disease model information is also saved to a file named `diseasemodel.txt`.

10. Simulation algorithm

For simulations of population samples, no disease model is involved, and the program assumes all chromosomes are non-disease chromosomes (see below). For simulations of case-control data, once the disease model is determined, the program calculates the conditional probabilities $\Pr(G \mid \text{case})$ and $\Pr(G \mid \text{control})$ over all disease locus genotypes given the subject is affected or unaffected. The program then generates disease locus genotypes for cases and controls according to these conditional probabilities. This retrospective approach is different from prospective simulation schemes in which a joint genotype $\{g_1, \ldots, g_m\}$ is simulated and is kept or discarded depending on whether a random number is smaller or larger than the penetrance of the genotype. Compared to prospective simulation schemes, which can discard many genotypes, especially for disease models with a small prevalence, our retrospective sampling approach is more efficient.

After the disease locus genotypes are generated, the program then simulates genotypes for the other SNPs on the disease chromosomes using a moving-window algorithm (Durrant et al. 2004). We assume all SNPs follow Hardy-Weinberg equilibrium in the general population. For each disease chromosome, the two alleles at the disease locus, say d, serve as the starting points for growing the two copies of the chromosome. For each copy, the program randomly selects a five-SNP haplotype at loci $[d-2, d+2]$ from the input phased data that has the same allele as the already simulated allele at d. The program then gradually grows the whole chromosome as follows: for SNPs on the right side of the disease locus, it generates an allele at locus $d+i$ given the haplotype at $[d+i-4, d+i-1]$ for $i \geq 3$; the conditional probabilities for the alleles at locus $d+i$ given the haplotype at $[d+i-4, d+i-1]$ are determined based on the input phased data. Similarly, for SNPs on the left side of the disease locus, it generates an allele at locus $d-i$ given the haplotype at $[d-i+1, d-i+4]$ for $i \geq 3$. Genotypes for non-disease chromosomes are generated similarly except that a randomly selected SNP is designated as the starting SNP. In this algorithm, every four consecutive SNPs are used to determine the allele at

the next SNP, but the window size can be modified by the user through the control file.

The simulated chromosomes generated by this algorithm are not exact copies of those in the original input data. Rather, the input phased chromosomes are used to generate plausible haplotypes in a wider population that have a similar local LD structure as the input phased data.

11. Properties and limitations of the program

How representative are the simulated data of the population under study? This depends on how representative the input data are. With respect to the HapMap phased data, various studies have demonstrated the similarity of LD patterns between HapMap samples and other samples. For example, Willer et al. (2006) observed similar LD patterns between HapMap CEU samples and a Finnish dataset. When the HapMap data are used as the input, we found that the simulated data are in general similar to the HapMap data, especially for LD patterns for markers that are close to each other. However, because the simulation algorithm generates a marker allele based on the haplotype of only a few previous markers, we also see variations in the LD patterns for markers that are more apart from each other, reflecting a potentially larger variety of LD patterns as we will see in real data.

Do biases exist in the simulated data? Because the program relies on a set of input data, limitations of the original data may be passed on to the simulated data. If the HapMap data are used as the input data, the ascertainment bias (Clark et al. 2005) in the HapMap can be reflected to some extent in the simulated data. However, our program can use other sources of input data when available.

What effect does the uncertainty of the input phase information introduce into simulated datasets? The simulation algorithm relies on short-range phase information. Using the default window size 5, phased data on five adjacent markers are used. For a SNP chip with 500,000 SNPs, the average length of a five adjacent marker region is 24 kb. For such a short region, phased data often achieve very high certainty. Thus, the longer-range phase uncertainty in the HapMap data is expected to have a limited effect on the simulated data.

Are there limitations? Limitations of the input data can be carried over to the simulated data. The program requires large-scale genotyping data, which is currently only available for Yorubans, Chinese/Japanese and northern Europeans (CEPH Utah) through the HapMap project. This method may not be useful for populations that have not been extensively sampled, such as South Asian, Middle Eastern, sub-Saharan African or Native American populations. In addition, if the input data are not variable enough due to ascertainment bias or small sample size, the generated data might not show enough variability for the population under study.

Is it correct that we must already know the exact disease locus in HapMap in order to generate simulated datasets? The presumed disease locus needs to be specified for the program. If one doesn't want to choose a SNP from the source database as the disease locus, he/she has to create a SNP in the input file with complete phase information with all the other SNPs. However, we recognize such information may not be available and this could limit the applicability of the software.

Can data other than HapMap be used with this software; such as, for instance, Perlegen data? The software is not restricted to the HapMap data. Any phased data can be used as the input.

What is the effect of changing the size of the sliding window? When the window size is 5, we use every four-marker haplotype to predict the allele at the next marker. At these five markers, the input phased data would be effectively tabulated into a 16×2 table with each row representing one of the 16 possible haplotypes at the previous 4 markers, and each column represent one of the 2 alleles at the fifth marker 5. When there are 120 input phased chromosomes, any larger window size will make the table too sparse and the prediction less variable. The program has a built-in warning message if a user specifies a window size that is too large compared to the number of input phased chromosomes.

Is the expected time linear in the number of samples? How does time increase if the window size is increased beyond 5? The expected time is almost linear in the number of samples. The program does spend some time on reading in data. When the sample size is too large, necessary memory swapping may slightly slow down the program. The window size has little effect on the time because the program effectively moves along one marker at a step.

Do the simulated data reflect more of the structure that happens to be in the input data and less of the structure that is in the population but not reflected in the input data? The simulation algorithm dictates that the simulated data mimic the local LD patterns of input data, which might be a limitation of the algorithm. However, when the HapMap data are used as the input, various studies have demonstrated the similarity of LD patterns between HapMap samples and other samples. For example, Willer et al. (2006) observed similar LD patterns between HapMap CEU samples and a Finnish dataset. On the other hand, the algorithm grows a chromosome in a Markov fashion, that is, the next marker allele depends only on the alleles at the previous 4 markers. This way, the program can generate a much richer variety of mid- and long-range LD patterns than those of the input data, while keeping the short-range LD patterns similarly as the input data. This allows the program to simulate data that mimic what could have happened due to recombination between markers of distance, resulting in a wider tree structure than that of the input data.

Does the user specify the allele frequency for the disease SNPs? The user specifies the disease SNPs, but does not need to specify the allele frequency as it can be implicitly determined from the input phased data.

What effect does the assumption of HWE have on the simulations? The program assumes all SNPs are in HWE in the population, which is a good approximation for majority of the SNPs in the genome. However, we recognize that the HWE assumption may be violated for some SNPs. The effect of this assumption on the subsequent analysis often depends on the nature of the analysis. For each chromosome, the algorithm in our program grows the two copies of the chromosome independently from a starting SNP. Because of this, even if the starting SNPs are simulated to follow a genotype distribution that is not in HWE, markers that are far from the starting SNP will eventually behave like they are in HWE.

12. References

1. Clark AG, Hubisz MJ, Bustamante CD, Williamson SH, Nielsen R (2005) Ascertainment bias in studies of human genome-wide polymorphism. *Genome Res*. 15:1496-1502.
2. Durrant C., Zondervan K.T., Cardon L.R., Hunt S., Deloukas P., Morris A.P. (2004) Linkage disequilibrium mapping via cladistic analysis of single-nucleotide polymorphism haplotypes. *Am. J. Hum. Genet*. 75:35-43.

3. Li C., Li M., Lange E.M., Watanabe R.M. (2008) Prioritized subset analysis: improving power in genome-wide association studies. *Hum. Hered.* 65:129-141.
4. Willer CJ, Scott LJ, Bonnycastle LL, Jackson AU, Chines P, Pruim R, Bark CW, Tsai YY, Pugh EW, Doheny KF, Kinnunen L, Mohlke KL, Valle TT, Bergman RN, Tuomilehto J, Collins FS, Boehnke M. (2006) Tag SNP selection for Finnish individuals based on the CEPH Utah HapMap database. *Genet. Epidemiol.* 30:180-190.