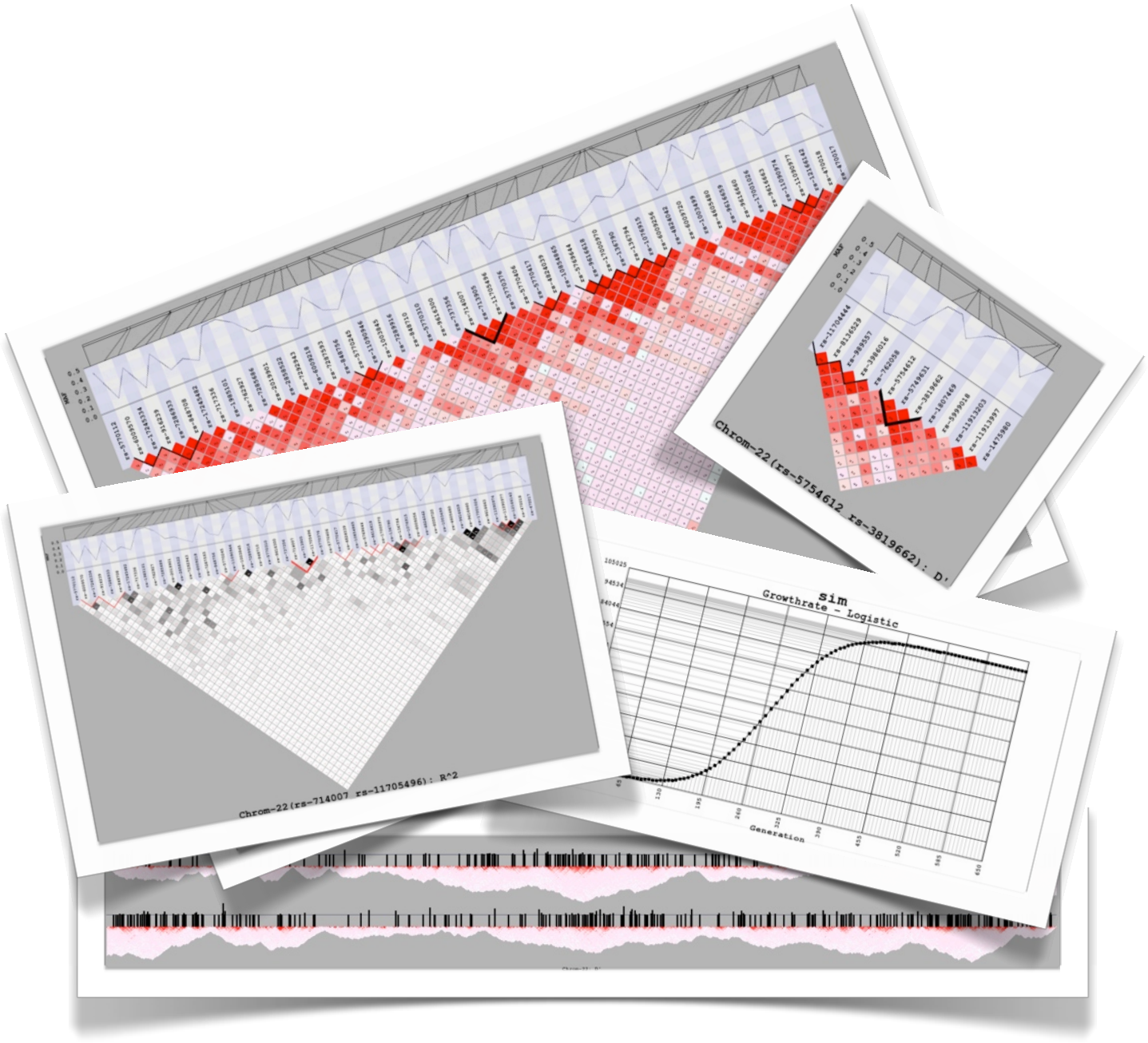
g e n o m e S I M L A

Software Introduction rev. 1.0.0



genomeSIMLA - A forward time simulation for genetic data

http://chgr.mc.vanderbilt.edu/genomeSIMLA

Table of Contents

Introduction

Purpose of this document

This document contains a tutorial to walk users through using the application, genomeSIMLA. Along the way, users will learn about the most common features and build an understanding about how the software is intended to be used. This document is designed to run on hardware commonly found on user's desks and shouldn't take more than 30 minutes to an hour to complete.

At the end of the tutorial, users should be able to generate datasets which contain up to 5000 SNPs with and without LD. Users will be able to configure their own penetrance models for disease status and have the ability to generate both pedigree and case/control datasets.

Conventions Used

There are two conventions used throughout this document. These text conventions are intended to help distinguish examples from configuration parameters.

Example Configuration Settings and Command Line

```
Example Lines are inset and in italics
```

These lines represent what a user might actually use on the command line or within a configuration file.

Example Output

```
Example Output will be in a light gray box and
will generally be captured directly from an
output file or output generated during execu-
tion of a command. Examples will often be trun-
cated to minimize the footprint.
```

Using genomeSIMLA for the First Time

Except in very specific cases, generating data-sets with genomeSIMLA is a multistep process. At the very least, users must run genomeSIMLA forward through time performing at least 1 drop along the way. It is from one of these drops that the user's data-sets will be drawn from. In addition to generational advancement and data-set production, genomeSIMLA can pick up from a specified generation and advance further through time, or perform "complete" LD analysis.

For this document, you should have downloaded the file, GenomeSIMLA-Introduction.tgz. This archive contains a number of files including the files c21-5K.loc and c21-5J.sim. For the example commands to work as they are used inside the document, users should make sure that the executable, genomeSIMLA (or genomeSIMLA64, if you are using the 64bit version) is found in the user's search path. Some executables have been renamed to allow them to exist together on a shared file-system. These include windows and mac versions. If you are using one of these platforms, you will be expected to adjust the command to include the appropriate executable name.

Listing Application Functionality

To get a simple listing of genomeSIMLA's command-line options, simply run the application with no arguments.

```
genomeSIMLA
```

```
----------------------------------genomeSIMLA-------------------------------
genomeSIMLA is a forward-time population simulation application intended to
provide users the ability to produce datasets rich with LD patterns similar to
those seen in real data.

This version of genomeSIMLA is considered under beta. We highly recommend that you
check the genomeSIMLA website to make sure that you are running the latest version:
    http://chgr.mc.vanderbilt.edu/genomeSIMLA

genomeSIMLA - Forward time simulation for SNP data: 1.2.0 build 2 (Tue Sep  1 16:07:14 CDT
2009)
Vanderbilt University
Center for Human Genetics Research


usage: genomeSIMLAs <configuration file> [datasets] [ld] <project name> options
    datasets - Activate dataset generations
    ld       - Activate complete ld plot generation
    -p (--project) name  : Specify part of the filenames generated
    -l (--load-generation) number : Indicates a specific generation to be loaded and run from
    -d (--drop-points) first freq count : Override the drop point configuration
    -s (--seed) number : Override the seed value in the configuration file

When running genomeSIMLA in either dataset or ld mode, no generational advancement will
be performed. Therefore, you should be sure to load the appropriate generation using the
-l flag. The line below will load generation 100 and generate detailed LD plots from it.
    genomeSIMLAs config.sim ld -l 100

If you have questions or issues regarding genomeSIMLA, please direct them to the
genomeSIMLA distribution list, genomeSIMLA@chgr.mc.vanderbilt.edu
```

As stated in the output above, the application can be run in 3 different modes:

1. Generational Advancement (Default)

2. Dataset Generation

3. LD Extraction

By default, the application will attempt to advance the population in time. This assumes one or more drop points have been configured. A drop point simply represents a generation at which the application will store the contents of the pool and perform some LD analysis on the pool. These can be reviewed by the user to assess the value of the drop for their purposes.

Dataset generation is obvious. When users specify the command, datasets, the application will attempt to extract datasets from the current population. By default, the current population is at generation 0. When drawing data from generation 0, the pool is initialized and data-sets are drawn from the randomized chromosomes. Datasets drawn at generation 0 should have no significant amount of LD.

Unless specified in the confguration file, LD plots generated during generational advancement will use sampling. This includes a subset of the entire genome and a small portion of the individuals. This saves a lot of time and allows the user to scan many points on various growth curves looking for ideal pools to draw data from. Once a candidate generation has been selected, the user is expected to perform an LD extraction prior to generating their data. This step generates LD data for every locus in the genome for every individual in the population at the chosen generation. Generations other than zero are chosen on the command-line using the -l command line option.

Basic Configuration

Before we continue, let's take a quick look at some of the important settings in the configuration file:

```
SEED 2225                       # random seed for data simulation
GENERATE_LD_PLOTS       ON      # Required to generate LD plots
BLOCK_REPORT_SIZE       10      # How many blocks you want to generate LD plots/reports for.
WRITE_LD_REPORT         Yes     # Write each LD value into a text report
DRAW_RSQUARED_PLOTS     Yes
DRAW_DPRIME_PLOTS       Yes

SIMULTANEOUS_CHROM      1
THREADS_PER_CHROM       1

LOAD_CHROMOSOME chrom21-5K.loc Chrom-21

GROWTH_RATE RICHARDS 500 0.00 0.01 40000 400 0.1
```

The lines above are a very small selection of what is actually in your simulation configuration, but they are the most relevant items. For more information on any of the items listed in the configuration, please refer to the reference guide. We'll just touch on these in this document.

SEED 2225 Set the random seed to 2225

GENERATE_LD_PLOTS ON This instructs genomeSIMLA to perform LD calculations.

BLOCK_REPORT_SIZE 10 When LD analysis is performed, the entire region for which analysis has been done is rendered to the plot. Users can request 1 or more Block Reports, which are much smaller and thus, much eas-

ier to read. Setting this variable to a number greater than 0 will cause genomeSIMLA to generate up that many reports.

WRITE_LD_REPORT YES Yes indicates that genomeSIMLA will generate text report containing all pair-wise LD for the region that was calculated (remember that sampled LD will analyze only a portion of the SNPs).

DRAW_RSQUARED_PLOTS YES Turns on RSquared Plots

DRAW_DPRIME_PLOTS YES Turns on DPrime Plots

SIMULTANEOUS_CHROM 1 Sets the number of chromosomes to be managed at runtime. Numbers greater than one increases the number of threads that work on advancing chromosomes through time and calculating LD. This can speed up runtimes considerably, but increase the memory requirements accordingly (advancing 3 chromosomes at once will require enough RAM to hold 6 pools in memory at once).

THREADS_PER_CHROM 1 This has less impact on runtimes as parallel chromosomes, but it has less impact to the user's memory (basically, N threads will tackle the same chromosome at the same time and don't add considerably to the memory consumption over a single thread's needs).

LOAD_CHROMOSOME chrom21-5K.loc Chrom21 This line sets up a file based chromosome (based on chrom21-5K.loc) and names the chromosome Chrom21.

GROWTH_RATE RICHARDS 500 0.00 0.01 40000 400 0.1 Sets up the growth curve. In this case, we use a growth based on Richard's Logistic. This has an initial period of no increase, which is responsible for initializing the LD before the population explodes.

Running GenomeSIMLA

GenomeSIMLA will generate a large number of files while it runs. To avoid polluting the folder in which the configuration is found, it is recommended that users write their files to a subdirectory. Let's call ours *data*.

```
mkdir data
genomeSIMLA chrom21-5K.sim -p data/5k -d 200 200 4
```

These two lines create a subdirectory data. Next, we tell genomeSIMLA to use the configuration file, chrom21-5K.sim. The -p option sets the project name to include the directory data. This is used as a prefix for all files that are produced, so it works nicely to organize the output. Finally, the -d option sets our drop points to begin at generation 200. Drops will occur every 200 generations for a total of 4 drops.

This takes about 15 minutes on a dual core Pentium D, 3.2Ghz machine running linux. If you didn't make any changes in the configuration and used the command as written above, there should now be 201 files inside the data directory- all starting with the letters 5k.

Most files are named as follows:

> Project Name "." Generation "." Chromosome "." [other file relevant pieces]

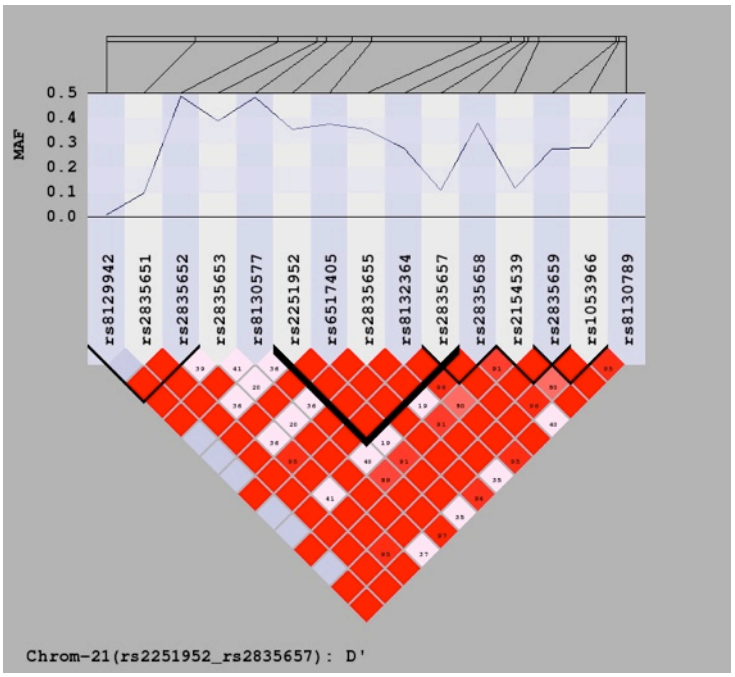
As a result of all of the "."s, windows users might be confused at first. If you see a number of files that appear to be named exactly the same, you should adjust your system to show all extensions.

Below is a listing of useful file endings and their meaning:

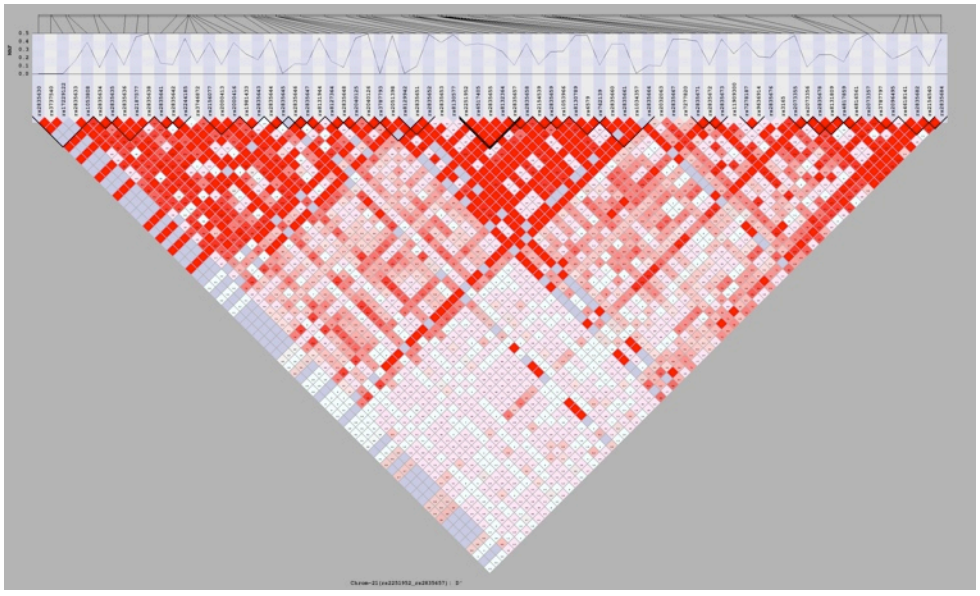
.loc Files ending in .loc are locus files at the given generation/chromosome. These files include allele frequency for each locus on a given chromosome (even if the MAF is 0.0)

.csv This is the pairwise LD at the speficied generation and chromosome.

#-det-dp.png These files contain the detailed DPrime report for a single block. These files will be relatively small, generally only having 10 or 20 additional SNPs in addition to the block for which the report was drawn. An example can be seen below. The block for which the report was made is identified with an extra bold line.



#-ovr-dp.png These files are the same as above, only zoomed out a bit, showing a but more of the surrounding SNPs.

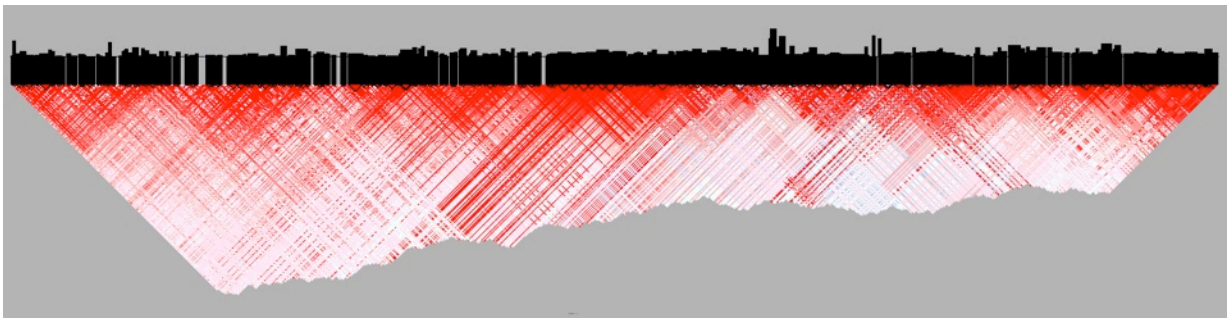


#-det-rs.png This is the same as the D-prime version, but for RSquared reports.

#-ovr-rs.png This is the same as the D-Prime version, but for RSquared reports.

The number associated with each of the graphical reports is just the index associated with the order in which the report was generated. The first report generated is the largest block found.

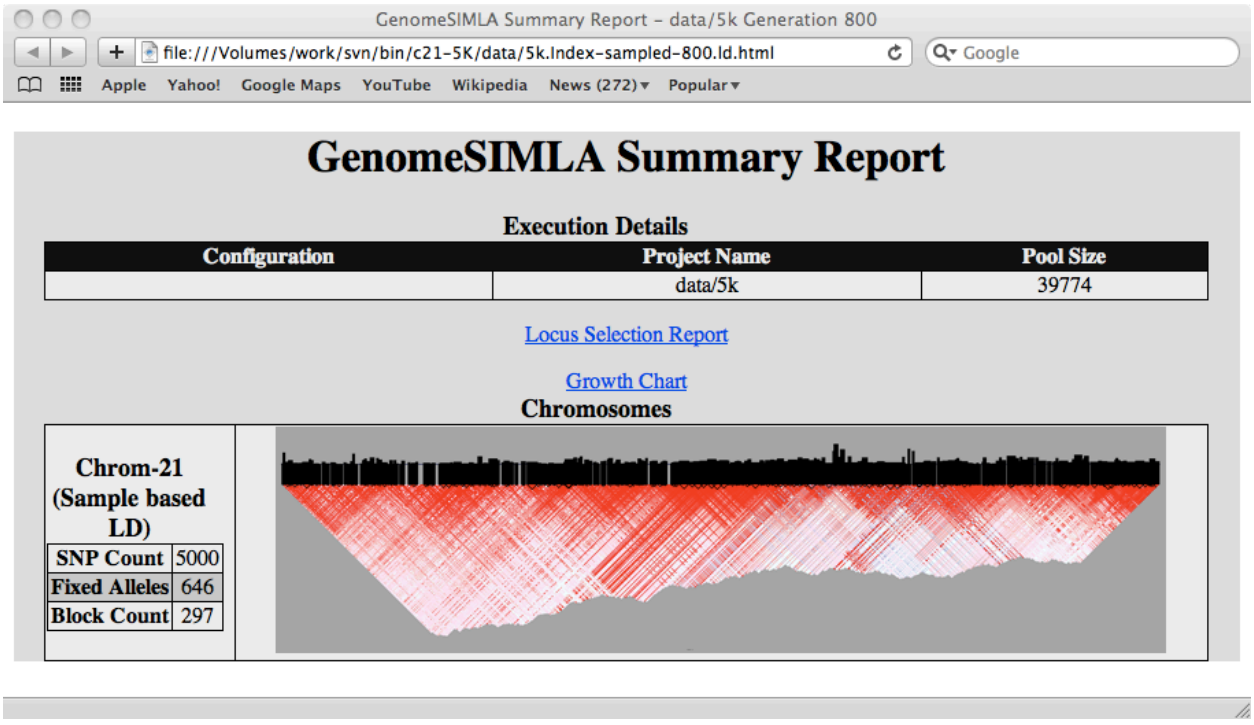
sum-dp.png These files contain the DPrime report for the entire region that was analyzed. When performing LD-Analysis on a specific drop point, this will be the entire chromosome, and can be very large.



sum-rs.png Same as above, except for RSquared values.

.phased These files contain the pool information in a binary format. The can be very large, depending on the size of the chromosome and population.

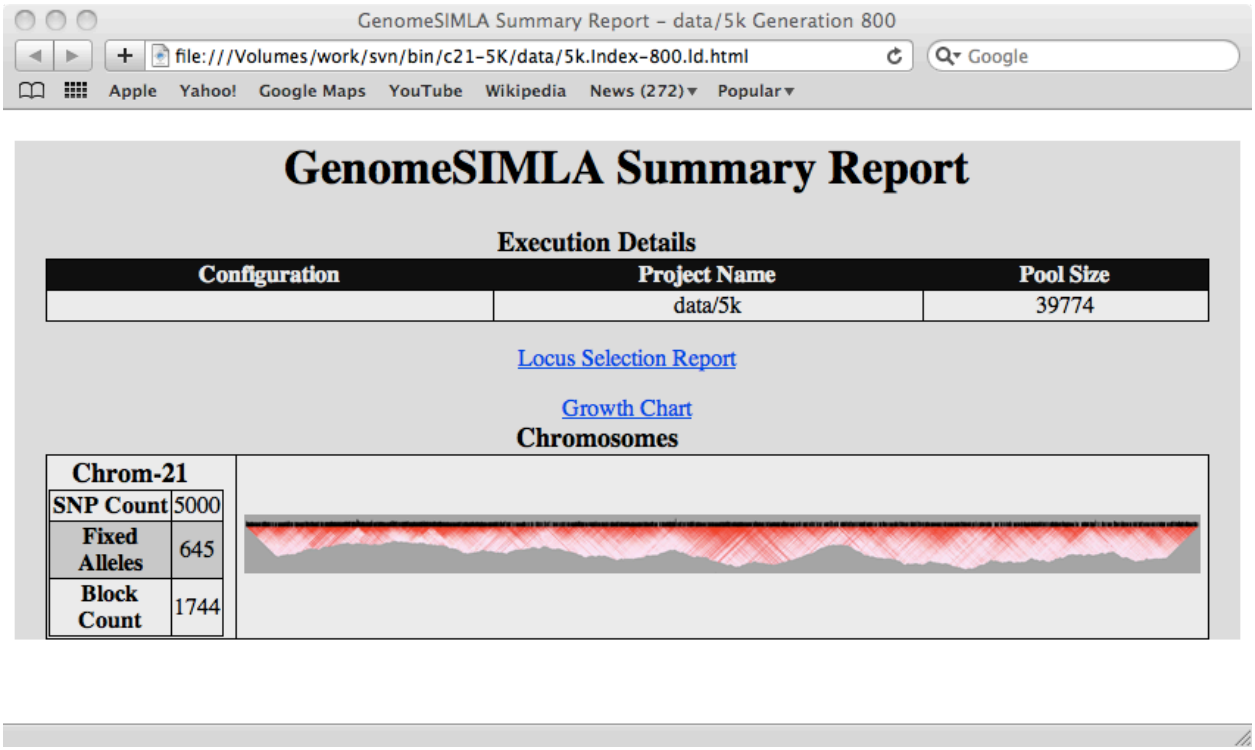
ld.html These files contain basic information about each chromosome in the simulation and some details about the simulation. There are links to *Locus Selection Report*, *Growth Chart* as well as the chromosome reports themselves (click on the pictures).



Notice that our report mentions that the report is sample based. This means that we probably didn't consider everything in our pools, and might not have even looked at all loci. We will fix that in a little bit. Take a look at each of the reports for each of the 4 drops.

Once we have looked over the four LD html reports, we decide that generation 800 looks like a reasonable choice for dropping data. Before we actually do that, though, let's generate a complete LD report. Keep in mind, this will take some time. On my dual core 3.2 ghz machine, it took just over 17 minutes.

If we look through the data directory when its finished, we'll see a file, 5k.Index-800.ld.html. Open that file. It has the summary of what we just did.



The first thing you'll probably notice is that the chromosome is much longer in this report. That's because our report contains LD information on all of the SNPs. If this had been all of chromosome 21, there would be many rows of LD information in our pictures. Also notice that the report doesn't mention sampling anywhere.

Click around and take a look at the various blocks and report elements.

Before we continue, let's take a closer look at some of these reports and make sure we understand what we are looking at.

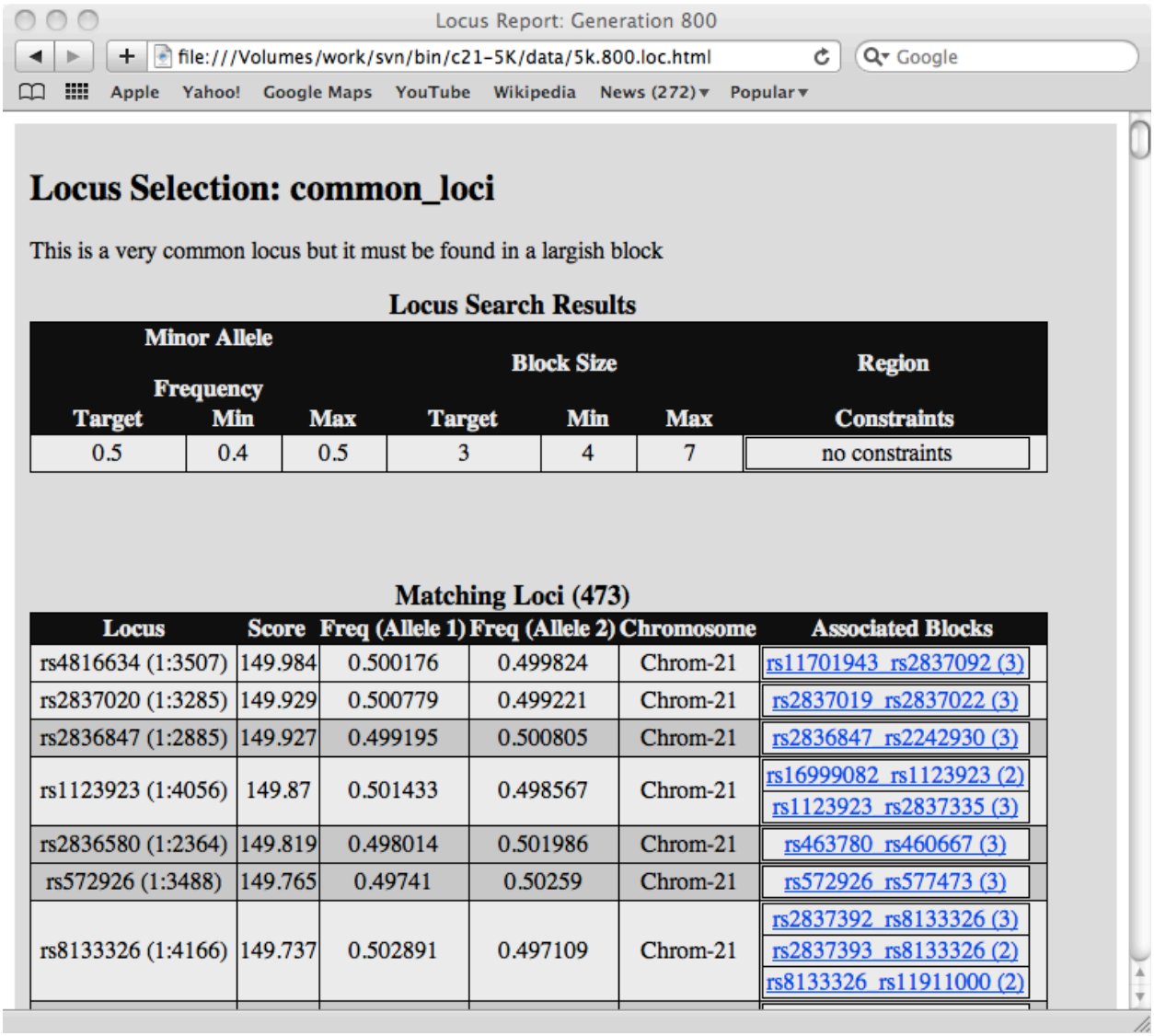
HTML Index Report

The index report is just the entry to a given drop. It can be sampled or complete (as we saw above). This report shows the number of chromosomes in our pool(s) (labeled pool size) as well as the number of SNPs, fixed alleles and blocks. The block count is the number of blocks that were identified using the 4 gammete rule.

Locus Selection Report

This report is available from the index report page, linked under the name, *Locus Selection Report*. It contains the results of our locus searches.

You didn't have anything to do with them, but we added two searches to our configuration, and the results are here in the report. Basically, a locus search specifies a min/max and target value for Block Size and Minor Allele Frequency. Any block that falls within these criterion (it must meet ALL criterion) is scored based on how closely it lies to the target values.



Locus Selection: common_loci

This is a very common locus but it must be found in a largish block

Locus Search Results

Minor Allele Frequency			Block Size			Region
Target	Min	Max	Target	Min	Max	Constraints
0.5	0.4	0.5	3	4	7	no constraints

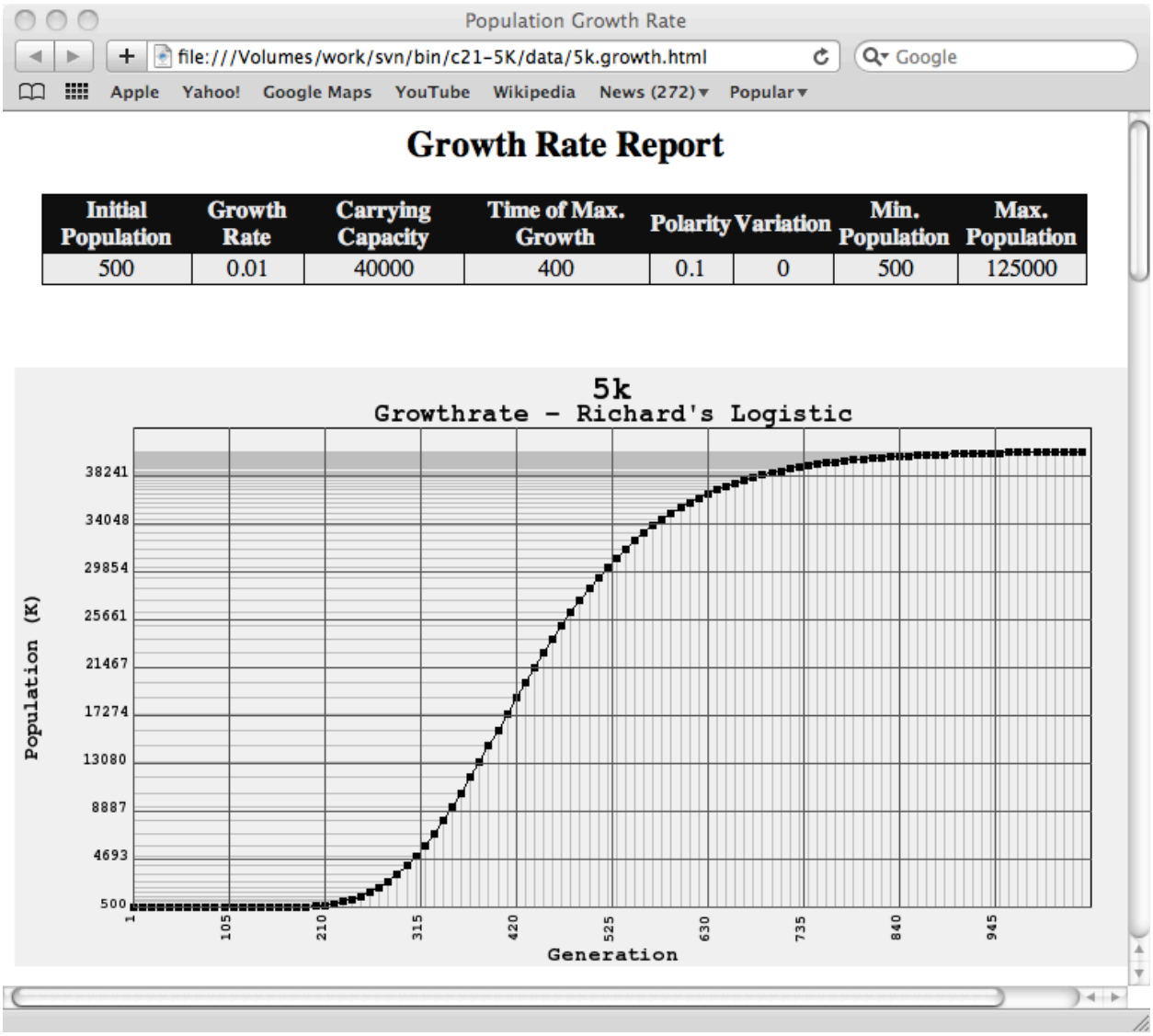
Matching Loci (473)

Locus	Score	Freq (Allele 1)	Freq (Allele 2)	Chromosome	Associated Blocks
rs4816634 (1:3507)	149.984	0.500176	0.499824	Chrom-21	rs11701943_rs2837092 (3)
rs2837020 (1:3285)	149.929	0.500779	0.499221	Chrom-21	rs2837019_rs2837022 (3)
rs2836847 (1:2885)	149.927	0.499195	0.500805	Chrom-21	rs2836847_rs2242930 (3)
rs1123923 (1:4056)	149.87	0.501433	0.498567	Chrom-21	rs16999082_rs1123923 (2) rs1123923_rs2837335 (3)
rs2836580 (1:2364)	149.819	0.498014	0.501986	Chrom-21	rs463780_rs460667 (3)
rs572926 (1:3488)	149.765	0.49741	0.50259	Chrom-21	rs572926_rs577473 (3)
rs8133326 (1:4166)	149.737	0.502891	0.497109	Chrom-21	rs2837392_rs8133326 (3) rs2837393_rs8133326 (2) rs8133326_rs11911000 (2)

The report simply shows the locus (it's label and the chromosome and index:locus position) followed by allele frequencies and a link to associated block reports (if a report was generated). The alternate row shading is actually used to help users differentiate loci which have a MAF at the first or second allele. This subtle detail can be critical when selecting a disease locus for a penetrance table.

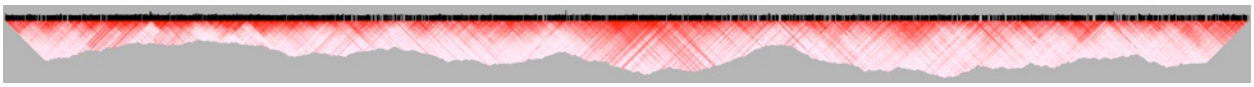
Growth Chart

The growth chart is simply a report on the growth configuration for the current simulation. It can be used to visualize a growth curve before spending large amounts of time dropping and analyzing data, or simply to serve as a record for the growth rate for a given simulation.



Summary LD Plots

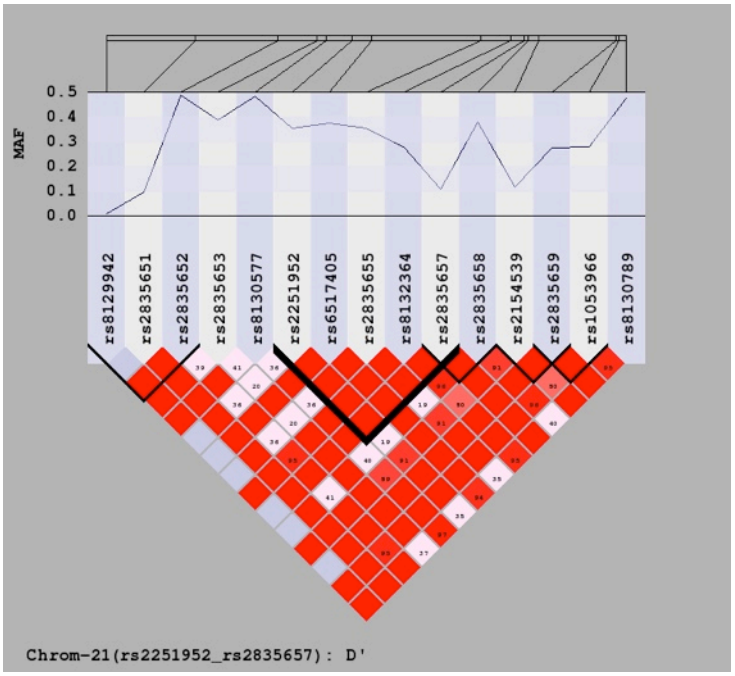
These plots are designed to allow the user to see an overview of a chromosome's complete pairwise LD.



Aside from the familiar shading, there is one additional feature. The block density is displayed as a bar-graph at the very top of the plot. The taller the bar, the more dense a give block is.

LD Block Report

Like the summary plots, most users will be quite familiar with the LD Block Reports. However, there is an additional component, minor allele frequency, which might prove helpful to some users. The block around which the plot was generated is highlighted using an extra thick line.



Generation Advancement Summary

That's about all a user really needs to know. If you have reached this point, and are happy with your pool, you can move on to the next step and get ready for generating datasets.

There are a lot of things we didn't cover.

Additional Drop Points

If we wanted to push further out from generation 800, we could load generation 800 and perform another series of drops:

```
genomeSIMLA chrom21-5K.sim -p data/5k -l 800 -d 1000 100 3
```

This command would instruct genomeSIMLA to load generation 800 and perform three drops starting at generation 1000 at 100 generations apart.

Randomness is an important thing to mention here. If you were to do the following, and decide that you liked generation 1000 best of all and decided later that you wanted to rerun your simulation starting at generation 0, you wouldn't get the exact same pool by simply dropping 1 pool at generation 1000. When you run the command above, loading generation 800, you start with a fresh random number stream as you begin performing advancements. In order to rerun the simulation and get the same pool a second time, you would need to drop the first four, and then load generation 800 and drop the 1000. Intermediate drops should have no effect on the stream of random numbers, but restarting the application will.

Sampling Options

Many of the parameters associated with LD plots are configurable, including some of the thresholds associated with sampled LD plotting. Users interested in changing these settings should consult the reference manual for details on making these changes.

Additional Growth Models

The growth curve is currently the most important component in the production of convincing LD patterns. While it's not perfect, the Richard's Logistic curve produces the most reasonable LD patterns of all of the curves we've implemented. However, users are welcome to use any of the other 3 curves. Please see the reference manual for details on altering the growth curve settings.

General Parameters

The following parameters control the basic behavior of the application.

SEED integer

```
SEED 23125
```

Sets the seed for all random number calls. Seeds can range from 0 - 4.2 billion.

Drop Points

Drop points are points in simulated time (generations) where the entire contents of the pool(s) is written to disk and analyzed. Reports are produced in HTML format to help the user to interpret the current state of the pool. Drop points are designed to allow the user to track the state of LD within the population. If the population is large enough, any drop point can be the source for dataset generation.

The reports initially written for a given generation are done using sampling. Prior to selecting loci for modeling diseases, users are expected to extract detailed reports from the generation of interest.

FIRST_DROP_POINT integer

```
FIRST_DROP_POINT 500
```

Sets the first drop point to be performed at generation 500.

DROP_FREQUENCY integer

```
DROP_FREQUENCY 100
```

Causes genomeSIMLA to drop every N generations once it has reached the initial drop point. The example says to drop every 100 generations.

DROP_COUNT

```
DROP_COUNT 5
```

Indicates the total number of drops to be performed (including the initial drop). If we look at all three of the previous DROP related examples, genomeSIMLA would perform 5 drops at generations: 500, 600, 700, 800 and 900.

It should be noted that the calling parameters can change how drop points are interpreted.

Graphical Plot Settings

The following general parameters control various aspects of the graphical reporting during a given drop.

MAX_SNPS_PER_ROW integer

```
MAX_SNPS_PER_ROW 3000
```

In order to render a general overview of an entire chromosome, it is necessary to set a maximum number of SNPS which can be drawn on a single row. This is not a hard setting- more of a suggestion. genomeSIMLA will distribute the SNPS evenly on all rows in order to avoid having a small chunk at the bottom.

BLOCK_REPORT_SIZE integer

```
BLOCK_REPORT_SIZE 30
```

Determines the number of detailed blocks that are reported. Each report takes disk space as well as time to generate. If your configuration is set to use 22 chromosomes with a report size of 30, there will likely be over 1200 charts drawn and a fair amount of information added to the final report. However, if the settings are too small, it might be more difficult to find the preferred SNP.

Each *Block Report* consists of 2 graphs. The first is expected to be smaller (fewer SNPS on either side of the block of interest). The second is generally zoomed out to show a bit more of the surrounding SNPS.

FONT filename

```
FONT ./FreeMonoBold.ttf
```

genomeSIMLA requires access to a true type font in order to write labels and details onto the graphical portions of the reports. This font should be available to genomeSIMLA during **execution time.** If the file can't be found, there will be a large amount of warnings rendered to STDOUT and none of the graphs will have any textual information on them- but execution will continue.

CSS_FILENAME filename

```
CSS_FILENAME ../genomesimla.css
```

In order to make the reporting flexible, each report refers to a stylesheet which contains the necessary information about shading spacing and other information. An example stylesheet is provided with the application as well as each of the examples. Users are welcome to change this to suit their needs- and should be aware that editing the stylesheet does not require anything be done with genomeSIMLA. However, it is necessary that the stylesheet be found, as stated in the configuration file when the reports are read.

The example above indicates that there will be a file named genomesimla.css that resides in the directory above the one in which the report is read from. In other words:

If there is a report named: */home/torstees/genomesimla/data/test1.index.50.html*

And I used the setting from the example above, the following file **must** exist:
/home/torstees/genomesimla/genomesimla.css.

If I copy the report (s) to a new filesystem, I should be sure to copy the style-sheet to the parent directory of the new location.

If the style-sheet isn't found, the report will just be harder to read.

LD_REPORT_BUFFER_SIZE On/Off

```
LD_BUFFER_SIZE 50
```

This sets the number of SNPS around the block in the *detailed plot*. In the example above, 50 SNPS on either side of a block are drawn.

WRITE_LD_REPORT On/Off

```
WRITE_LD_REPORT On
```

Causes genomeSIMLA to produce a complete complete report of the pairwise LD values. Users should be aware that this file can be very large.

DRAW_RSQUARED_PLOTS On/Off

DRAW_DPRIME_PLOTS On/Off

```
DRAW_RSQUARED_PLOTS Off
```

By default, both RSquared and DPrime plots are drawn. If the user wants to save time and disk space, they can opt to one or both charts off.

MAX_SNP_DISTANCE integer

```
MAX_SNP_DISTANCE 500000
```

This allows the user to determine far apart SNPs can be before genomeSIMLA decides to calculate LD values. Lowering this value from the default (500K) can speed up LD processing.

CLOSE_POOLS_BETWEEN_DROPS On/Off

```
CLOSE_POOLS_BETWEEN_DROPS On
```

In general, it is assumed that genomeSIMLA will be used to produce very large populations (~1 million unique chromosomes) with genomes that approach 500K. In order to manage this on a single computer, genomeSIMLA must close pools down when they aren't currently in use. This frees up valuable memory- allowing us to do this without gigabytes of ram. However, it can slow down processing when only one or two small chromosomes in use.

FAST_LD_POOL_SIZE integer

```
FAST_LD_POOL_SIZE 3000
```

Sets the number of chromosomes that are used in sampled plots. If this is larger than the population, the entire population is used.

FAST_LD_PLOT_SIZE

```
FAST_LD_PLOT_SIZE 10000
```

Sets the max number of SNPs used in sampled LD production. If the size is greater than a given chromosome, the entire chromosome is used.

NO_FASTLD

```
NO_FASTLD
```

Unlike most other options, this takes no parameters. When it is encountered, genomeSIMLA will skip the sampled LD calculations and instead render only complete LD plots. Please be aware that this will increase the length of time during a growth scan where you are unsure where the best generation to draw from lies. It should be used only when you either know ahead of time what your LD will look like, or you want to get statistics on the entire population at every drop.

Locus Generation

genomeSIMLA allows for the creation of *chromosomes* in two different ways. The first involves the description of one or more block types and populating a chromosome randomly with one or more of these blocks. The other approach uses a locus description file containing positional SNP names and positional information.

Block Based Locus Generation

Block based generation is the production of chromosomes using completely random draws. The user specifies one or more block configurations which will be applied randomly to create the loci on a given chromosome. There are 3 elements involved in this process:

Block Definitions: These describe 4 things:

- Min/Max number of snps that can be associated with the "block"
- Min/Max recombination fraction for the first SNP (how far away is that SNP from the previous SNP on the chromosome)
- Min/Max recombination fraction for each of the containing snps.
- Probability this block will be drawn

When a chromosome draws a block definition to be used to construct a set of loci, it will randomly drawn the number of SNPs based on the block Min/Max value. Then, for each SNP, it will determine the distance between each SNP and it's predecessor. All but the first SNP use the second set of Min/Max recombination values. The first SNP is drawn from the first set. This allows the user to space the block further out from the SNPs in front of it (or not).

Default Block: When a chromosome is deciding which block definition to use next, it uses the probabilities associated with the blocks. It is possible for the sum to be less than 1.0. The difference between the sum and 1.0 is the probability that the default block will be used. The default block is common to ALL chromosomes, and should be defined before any other blocks or chromosomes. With the exception of probability, the default block has the same parameters as regular blocks.

Chromosome: To create a block based chromosome, users will use the ADD_CHROMOSOME command, indicating how many blocks to draw and possibly giving it a label. The user then applies blocks to the chromosome by using the ADD_BLOCK command.

When the draws are made, the blocks associated with a given chromosome will be drawn based on their probability (including the possibility of using the default block, if necessary).

DEFAULT_BLOCK min max float float float float

```
DEFAULT_BLOCK 5 10 0.01 0.015 0.00001 0.000025
```

The default block is used when the sum of a given chromosome's blocks probabilities don't sum up to 1.0. Otherwise, it is the same as a regular block.

The first two parameters specify the minimum and maximum number of SNPS will be created. The next two represent the range of distance this block falls from the previous SNP on the chromosome. The last two represent the range of distances of SNPS within the block itself.

DEFAULT_BLOCK should be set prior to the definition of any chromosomes (and thus, any other blocks).

ADD_CHROMOSOME integer label

```
ADD_CHROMOSOME 5 chromosome_1
```

Adds a new chromosome to the genome. The first parameter represents how many blocks to draw and the last (optional) parameter is the label that will be used in naming files and on the reports. The example above will create a chromosome with 5 blocks named "chromosome_1".

ADD_BLOCK chr_idx snp_idx float float float float float

```
ADD_BLOCK 5 10   0.0001 0.0002       0.000001 0.00001          0.5
```

Adds a block definition to the most recently defined chromosome (using ADD_CHROMOSOME). The example above will create a block that ranges from 5 to 10 SNPs. 0.0001 and 0.0002 represent the chance of a cross-over event occuring between the previous SNP (if one exists) and the first in the block. This effectively describes how far away from that last SNP the block is.

The next two describe the chance of a recombination occurring between any two SNPs found inside the block itself.

The last parameter is the probability this block will be drawn.

Other Block Related Settings

DEFAULT_ALLELE_FREQ float float

```
DEFAULT_ALLELE_FREQ 0.1 0.5
```

This allows the user to define min/max allele frequencies to be used during the configuration of a new block based chromosome.

File Based Chromosome Configuration

File based chromosome files have all of the information necessary to simulate a chromosome. There are two reasons one would use such files:

1. To mimic one or more region from a real genome.
2. To precisely control a region's density as part of a research project.

The Ritchie Lab has made a set of these files available which represent a large portion of the Affymetrix 500K coverage. As we produce others, it is expected they will be made available for use as well. These files allow for SNPs to be distributed very similarly to real human assays- though, the actual LD patters will depend largely on the generations the data-sets were extracted from as well as the random seed used.

Locus File Format

Line #1 is just a line used to describe which chromosome the file was derived from. This is ignored when the file is read

Line #2: Indicates the number of loci contained within the file. genomeSIMLA doesn't actually parse that number out- so again, this line is not used for reading.

Line #3: Column Headers. This is for the user's beneift...and is not used during reading

Line #4...N+3: Each line describes a single locus. The following represent the 6 columns that should be present (in the order listed). Each column must have a value for each line, and should be seperated by whitespace (multiple spaces or tabs is fine).

Col #1: Label This is usually the RS Number. However it can be any label one wants to use. All SNPS must have unique labels.

Col #2: Freq Allele 1: Allele 1's allele freqency

Col #3: Freq Allele 2: Allele 2's allele frequency

Col #4: Recombination Fraction: Chance that an odd number of recombinations took place between this SNP and the previous SNP in the genome.

Col #5: Position: This is the physical position on the chromosome (relative to the beginning of that chromosome...NOT the genome) These values should be in base pairs.

Col #6: Description (optional): This is just a note that can be added. Currently, this isn't used anywhere.

The last line of the file should be an empty line (the last entry should contain a return character).

It should be noted that when genomeSIMLA sets up the loci, allele 1 is ALWAYS the minor allele, regardless of the locus' frequency in the file. This is only important if a user were to draw datasets from a pool at generation 0. Their interpretation of 'A' and 'a' could be different from the way genomeSIMLA. When drawing data-sets from generation 0, 'A' is ALWAYS the minor allele.

Also, allele frequencies are not exact, even in large populations. When one is setting up a disease model for generation 0, it is recommended to let genomeSIMLA create the pool, drop generation 0 (it defaults to this) and assign model loci based on allele frequencies found in the locus file generated during the initialization.

Locus Miscellany

ALLELE_FREQUENCY chr_idx snp_idx float float

```
ALLELE_FREQUENCY 1 5 0.25 0.75
```

This sets the frequency of allele 1 of Snp # 5 on chromosome 1 to 25% and the second allele to 75%.

Population Control

Currently, there is a single population in genomeSIMLA, though each individual could have several different chromosomes. This population is grown using one of several growth rates. During a generational advancement, individuals are drawn (with replacement) from the current population, mated using Hardy-Weinburg mating and added to the new pool until it reaches it's target size.

Growth rates share many parameters. Below is a list of parameters that are used in each of the growth curves.

intial_population	integer	The population that is created at generation 0.
variation	float	This value is used to simulate imperfect growth curves. It represents the percentage of fluctuation around the curve's value at a given generation. The amount of fluctuation is actually +/- 1/2 the variation- so it is possible that the population at generation N+1 be smaller than at N.
growth_rate	float	This is the rate of growth. While it is applied differently for each model, the higher the growth-rate, the faster the growth.
carrying_capacity	integer	This is used in logistic style growths and specifies the ceiling of the growth curve. As the population approaches this value, it becomes less and less exponential in nature until it becomes static.
time_of_max_growth	integer	Used only in Richard's Logistic, this parameter effectively moves the exponential part of an S curve about on the X axis (in the direction of the generation specified.
polarity	float	Used only in Richard's Logistic, this parameter affects the "draw" of the curve toward the carrying capacity.

To set up a growth rate, the user should configure one of the following:

GROWTH_RATE LINEAR initial_population variation growth_rate

```
GROWTH_RATE LINEAR 30000 0.05 10.0
```

This is just a straight line that grows by growth_rate each generation

GROWTH_RATE EXPONENTIAL initial_population variation growth_rate

```
GROWTH_RATE EXPONENTIAL 700 0.05 0.3
```

This is just a basic exponential growth based on the growth rate specified.

GROWTH_RATE LOGISTIC initial_population variation growth_rate carrying_capacity

```
GROWTH_RATE LOGISTIC initial_population variation growth_rate carrying_capacity
```

This is considered to be one of the preferred models for describing growth rates. The carrying capacity represents the peak potential (which could be caused by various reasons). For our needs, it is the size of pool required for drawing data-sets.

GROWTH_RATE RICHARDS initial_population variation growth_rate carrying_capacity time_of_max_growth polarity

```
GROWTH_RATE LOGISTIC initial_population variation growth_rate carrying_capacity time_of_max_growth polarity
```

Richard's logistic is just an enhanced logistic curve- with two parameters capable of determining when growth starts and just how steep the growth will be.

By pushing the time_of_max_growth forward, the population hovers at initial_population for some amount of time. This small population will produce rich LD patterns which tend to be carried forward in time once growth begins. However, this small population increases the risk of fixing alleles dramatically.

General Growth rate parameters

MAX_POOL_SIZE integer

```
MAX_POOL_SIZE 90000
```

Sets the hard upper limit for population size. Every generation is compared against this value and can NEVER exceed it, regardless of variation nor growth curve details. It is very important for non-logistic growth rates, especially exponential, where growth could occur very fast and cause memory problems.

MIN_POOL_SIZE integer

```
MIN_POOL_SIZE 1500
```

Sets the semi-hard lower limit for population size. This is evaluated for all generations other than 0. So, it is possible to set the initial population to below MIN_POOL_SIZE and cause a hard spike in population at generation 1.

TARGET_POP_SIZE integer

```
TARGET_POP_SIZE 100000
```

If TARGET_POP_SIZE is greater than 0, genomeSIMLA will use the value as a hard limit for advancement- once it reaches the specified population size all advancement and population growth will cease.

Dataset Generation

The entire purpose of genomeSIMLA is data. genomeSIMLA is capable of generating 2 types of data-sets: case/control and basic pedigrees. genomeSIMLA can produce any number of different data-sets and guarantees that, in an truly diverse population, no single individual will be used in any data-sets generated during a single run.

Both pedigree and case/control data-sets allow for the use of a label. This label is used as part of the filename, and allows the user to quickly recognize different data-sets. These labels can have any character except slashes and spaces.

Both types of data-sets can have the following types of error:

genotype_error	float	Exact Portion of SNPs which are not derived via cross-over. This error is applied evenly across SNPs
phenocopy	float	Percentage of the affected individuals in a given data-set whose affected status was determined not by the chosen model.
missing	float	Percentage of SNPs that will be missing.

Case/Control

DATASET CC label affected unaffected genotype_error phenocopy missing

```
DATASET CC sample-01 500 500 0.05 0.1 0.15
```

This line creates datasets with 500 affected, 500 unaffected each with 5% genotype error and 15% missing data. Of the 500 affected individuals, 50 of them will not have been evaluated with the model.

Pedigree Data

Pedigree data is slightly more complicated because you can specify multiple types of family structures to be added to your dataset. The affected/unaffected numbers simply describe the number of children in those categories.

DATASET PED label genotype_error phenocopy missing

```
DATASET PED family-01 0.05 0.1 0.15
```

This sets up a framework for datasets with 5% genotype error and 15% missing data. 10% of all affected children will not have been evaluated with the disease model.

This just sets up the data-set framework. Until you add family types to it, the data-sets will be empty.

DATASET FAMTYPE affected unaffected extra_sibs number_of_families

```
DATASET FAMTYPE 1 1 1 250
```

This sets up a type of family which will be added to the data-set. A given data-set can have as many different types of families as the user needs. The affected/unaffected counts represent the number of affected/unaffected children a given family MUST have. The number of extra sibs indicates that a random number from 0 to extra_sibs will be added to the family. All children will be evaluated for status, however, by adding extra sibs, you can have larger families which vary by the number of affected siblings.

It is perfect acceptable to have 0 extra_sibs or unaffected sibs. Affected sibs MUST be greater than or equal to 1.

```
DATASET FAMTYPE 1 0 0 150
```

This would add 150 trios to the data-set.

```
DATASET FAMTYPE 2 1 0 75
```

This would add 75 AAU families to the data-set.

```
DATASET FAMTYPE 1 0 3 50
```

This would add 50 families with between 1 and 4 children with at least 1 affected sib in them.

Parent's status is evaluated and written to the dataset, but is not considered for determining whether or not the family will be included into the dataset.

Peformance Note: It is important to note that pedigrees with more than 1 affected individual can be computationally difficult. Children are created by actually crossing over the parents just like is done during generational advancement. If the children don't meet the necessary family shape, all individuals are thrown away. For instance, for a disease with a prevalence of 0.1, it would take the production of almost 1,000,000 families before we found a family with 2 affected sibs and it gets worse as you add in more required affected sibs. Most data-sets can be generated in a few minutes, but be aware of the possibility of long delays for large numbers of affected sibs and rare disease models.

General Data-set Configuration Parameters

DATASET_COUNT integer

```
DATASET_COUNT 500
```

This indicates the number of files that will be created per data-set. In this example, all data-sets created by this configuration would result in 500 unique files.

BINARY_DATASETS Yes/No

```
BINARY_DATASETS Yes
```

This compresses data-sets dramatically, allowing whole genome size data-sets to occupy a minimal amount of disk-space. This format was developed in-house, and won't be supported by any products other than those produced at the lab here (and only now are we beginning to implement it in our own applications).

If you are interested in the format, we will make the format available on the wiki in the near future. In the meantime, feel free to contact us at genomeSIMLA@chgr.mc.vanderbilt.edu.

This is currently not supported for pedigree datasets.

USE_STD_PEDIGREE_HEADER on/off

```
USE_STD_PEDIGREE_HEADER On
```

When on, all pedigree data-sets will have 10 column headers. When off, the header count will be 6 columns.

Locus Searching

The main goal for genomesimla is the production of realistic data-sets. These might be very large, and choosing disease loci can be a daunting task when presented with over 200,000 possible loci. To make the task as easy as possible, genomeSIMLA can limit the loci presented and present them in a sorted fashion where the topmost SNP shown most closely matches the user's specifications.

The following commands are used to set up searches. Users can have as many searches as they like...even if they don't need them all for setting up their models.

A search describes 3 qualities: minor allele frequency ranges, types of blocks the SNP is contained within and location. Each of the ranges contains three pieces. Target Min and Max. Currently, LOCUS_SELECTOR has 2 ranges: minor_allele_frequency and block_size.

LOCUS_SELECTOR label float float float integer integer integer description

```
LOCUS_SELECTOR rare_loci 0.2 0.15 0.23 4 2 10 The following loci are moderately rare and appear in a block
```

This creates a new search called *rare_loci* which will only contain SNPs whose minor allele frequency is between 0.15 and 0.23 and are found in blocks of up to 10 SNPs large.

The SNPs will be ranked so that those that are closest to a minor allele frequency of 0.2 and in blocks with 4 SNPs will be ranked first. Notice that the min/max values are not evenly distributed around the target. The score is ranked on the relative distance from the target for that particular arm. So, a SNP with a minor allele frequency with just a bit larger MAF than 0.2 would score very similarly to one that had a MAF of just under 0.23.

The block size and MAF weights are different. MAF is more currently weighted higher than block size. However, each block a SNP is found in will add more to it's final score- meaning it will rise higher in the report.

The description of a LOCUS_SELECTOR gets used in the locus report. Be descriptive as is necessary. However, there can be no newline characters in it. Spaces are allowed, though.

ADD_REGION label snp_start snp_stop

```
ADD_REGION rare_loci rs-321412 rs-543231
```

This adds a region to the selector, rare_loci. This region is bounded by the two SNPs rs-321412 and rs-543231. Both SNPs must be found and exist on the same chromosome.

By default, all searches are performed over the entire genome. However, if a user wishes to restrict the region to search, they can do so by adding regions (once you add a single region, it will only search the regions that have been added). To add a whole chromosome, simply add the first and last SNP geographically.

MAX_LOCI_PER_CHROM_REPORTED Integer

```
MAX_LOCI_PER_CHROM_REPORTED 50
```

Instructs genomeSIMLA to report at most N loci per chromosome for each sector described. Loci are ranked according to how well they fit the criterion, which could possibly be rather extensive. By setting this value to a reasonable number, the locus report can be kept at a manageable size. Setting the value to -1 will catch all possible matching loci.

Disease Modeling

genomeSIMLA comes with 3 options for modeling affection status: User generated Penetrance tables, simPEN (purely epistatic models) and SIMLA (main effect + interactions). Each method requires it's own configuration details.

Penetrance Table Disease Models

Users can use predefined penetrance tables to assign status to models. The only requirement is that the user specify the allele frequencies associated with each possible allele associated with each model locus. This is to help ensure that the appropriate meaning of a given cell is being applied. genomeSIMLA will not proceed to use a model if the actual allele frequencies differ too much from those specified in the configuration.

Users indicate to genomeSIMLA that a penetrance based model is to be used using a line similar to one of the two lines below.

DEFINE_MODEL PENTABLE INDEX pen-file chrom-id snp-id [chrom-id snp-id, ...]

```
DEFINE_MODEL PENTABLE INDEX disease.pen 1 5
```

DEFINE_MODEL PENTABLE LABEL pen-file snp-label [snp-label, ...]

```
DEFINE_MODEL PENTABLE LABEL disease.pen RL5
```

Both lines do the same thing. The first tells genomeSIMLA to load the penetrance table in disease.pen and use Locus 5 on chromosome 1 as the single disease locus. The contents of the specified penetrance table must match the number of model loci specified on the configuration line. Otherwise, genomeSIMLA will generate an error (or worse, become confused and generate misleading data-sets.)

The second example simply uses labels to specify which loci are to be associated with the disease model. Block based chromosomes (see BLOCK_DEFINITION) are labeled RLN where N is a number between 1 and however many loci there are associated with all chromosomes being simulated. Otherwise, the labels are based on information found inside the locus files that were used to populate the simulation (most likely, this will be an RS number). There can be no duplicately labeled SNPs.

Penetrance File Configuration

A small number of parameters make up the configuration details of a penetrance file. All but the threshold must be fully specified even if the value is 0.0.

FREQ_THRESHOLD Float

Specifies maximum allowed variation from the allele frequencies that will be tolerated before execution is halted.

FREQ [AaBbCcDd....etc] float

```
FREQ A 0.2
FREQ a 0.8
```

Using letter notation for specifying penetrance cells, this command allows the user to tell genomeSIMLA what the intended frequency for a given allele should be. The user MUST specify all alleles that are expected to be involved in the given model.

PENTABLE

This just indicates to genomeSIMLA That the various penetrances are about to follow.

model-identification penetrance

```
AABB 0.171
AABb 0.155
```

Each possible combination must be present, regardless if it's value is anything other than 0.0.

Penetrance tables should be written as a separate file from the main configuration.

Purely Epistatic Models with simPEN

simPEN is a method for using a Genetic Algorithm (GA) to evolve purely epistatic models. With few exceptions, the configuration details are considered to be beyond the scope of this document, however, a few details will be covered, such as those that specify target odds ratios and heritability.

Users indicate to genomeSIMLA that a penetrance based model is to be used using a line similar to one of the two lines below.

DEFINE_MODEL SIMPEN INDEX simpen-cfg chrom-id snp-id [chrom-id snp-id, ...]

```
DEFINE_MODEL SIMPEN INDEX disease.simpen 1 5
```

DEFINE_MODEL PENTABLE LABEL simpen-cfg snp-label [snp-label, ...]

```
DEFINE_MODEL PENTABLE simpen-cfg disease.pen RL5
```

Both lines do the same thing with the exception of how they specify which loci are to be associated with the model.

The configuration file for simpen must be a separate file. When deciding which weights are most appropriate, users should keep in mind that the values themselves can differ drastically, and a weight of 1 differs in effectiveness for a value whose target is 0.1 than that of a value whose target is 0.000001. The values used in the following examples were determined to be reasonable starting points for obtaining good results from the simPEN module.

simPEN File Configuration

A small number of parameters make up the configuration details of a simpen configuration file.

HERIT float

```
HERIT 0.01
```

Specifies that the target heritability will be 0.01.

HERITWEIGHT float

```
HERITWEIGHT 10
```

Adjusting the weight determines how important the attribute is.

ODDSRATIO float

```
ODDSRATIO 1.25
```

ODDSWEIGHT float

```
ODDSWEIGHT 1
```

MARGVAR float

```
MARGVAR 0.0000001
```

Set the target Marginal variance. This determines how pure of an epistatic model you want. The higher the values, the more likely there will be main effects.

MARGWEIGHT float

```
MARGWEIGHT 100
```

PENTARGET float

```
PENTARGET 0.15
```

Specifies the target prevalence of the *disease*.

The following parameters are associated with the GA portion of simPEN. For more information about how to use these parameters, please see the simPEN user's manual (available at the genomeSIMLA website. References to pool sizes, generations, populations and mutation below are completely unrelated to the simulation being performed by the forward time simulation.

GEN 15000

Number of generations to be tried before ending.

POPSIZE 1000

The search population.

DEMES 100

Multiple pools of penetrance values

MUTATE 0.01

Frequency of mutation

CROSS 0.6

Rate of cross over (1 per "genome")

SUBMODELS ON

Turning this on will possibly catch a pattern where a smaller model contained within a larger model does exist with enough strength as to represent a potential problem.

UPDATE 100

Specifies how many generations between progress is reported

The following two parameters are legacy and have no effect on the production of valid models. However, the error checking currently requires them to be present. Just use these values to satisfy the error checking code for now.

LOCI 2

FREQ 0.2 0.8

Main Effects and Interactions with SIMLA

SIMLA is a simulation program that allows the researcher to specify varying levels of both linkage and linkage disequilibrium among and between markers and disease loci. SIMLA was specifically designed for the simultaneous study of linkage and association methods in extended pedigrees, but the penetrance specification algorithm can also be used to simulate samples of unrelated individuals (e.g., cases and controls).

Users indicate to genomeSIMLA that a SIMLA based model is to be used using a line similar to one of the two lines below.

DEFINE_MODEL SIMLA INDEX simla-cfg float int chrom-id snp-id MIN | MAJ float float [...]

```
DEFINE_MODEL SIMLA INDEX interactions.simla 0.05 2 1 5 MIN 0.26 0.0
```

DEFINE_MODEL SIMLA LABEL simla-cfg float int snp-label MIN | MAJ float float [...]

```
DEFINE_MODEL SIMLA LABEL interactions.simla 0.05 2 RL5 MIN 0.26 0.0
```

Both lines do the same thing with the exception of how they specify which loci are to be associated with the model.

The filename specified by simla-cfg represents the list of interactions (see below). If no interactions are required, you may use the keyword, NO_INTERACTIONS- otherwise, the file must exist. The next number (0.05 in the example above) is the target prevalence. The last parameter before the loci is the maximum interaction size. This is just maximum number of loci that will be interacting with one another.

The MIN/MAJ value determines whether the disease is associated with the minor or major allele. The next parameter specifies the beta value associated with that locus. Finally, the last value required for each locus is the type. A 0.0 represents a recessive trait. The locus becomes more dominant as it approaches 1.0. Each locus to be considered must have each of these parameters (locus specification, MIN | MAJ, beta and type).

SIMLA configuration file

The simla-cfg is just a file that specifies the beta values associated with each of the interactions desired. For each interaction, specify them in the following way:

```
1x2x3 0.26
```

This tells genomeSIMLA that the 1rst, 2nd and 3rd locus (in the order they are encountered on the DEFINE_MODEL line) interact with a beta value of 0.26. Users can add as many or as few interactions as they wish.