1.1 Linkage Pipelines and Processing

The strength behind any linkage analysis does not lie solely with the linkage program, but with the quality of the input data that the program processes. An input data set consisting of mostly uninformative markers will not yield any meaningful output data, since the linkage program will be processing ambiguous data and subsequent runs may produce wildly varying results. In order to preserve some consistency across subsequent runs, we must be more selective in our choice of input data such that we only include markers that have a high information content in relation to the sample data set.

1.2 Input Data

1.2.1 **Pedigree** (pedfile.pro)

This file contains the information upon each family member, with every line denoting a single individual in standard LINKAGE[1] format.

An individual's line consists of data that numerically maps their sex ({male→1, female→2, unknown→0}) and their phenotypic status ({unaffected→1, affected→2, unknown→0}). Depending on the program used, an unknown sex may not be permitted. While it is not uncommon to see an unknown affectation, it is generally of no use to include it within a linkage run as it adds no power to the analysis.

Other fields record the relationship of that individual to other family members (mother, father). This mother-father-offspring grouping is referred to as a *trio*.

1 Also referred to as pre-MAKEPED format.

```
FamilyID     PersonID     FatherID     MotherID     Sex     Affectation
311          101           0            0           1       1
311          102           0            0           2       2
311          201           101          102         1       2
311          202           101          102         2       1
```

Text 1: An example pedigree file, showing members of family 311, with mother and father of 102 and 101 respectively, and their offspring 201 and 202

Siblings are not stated within a trio, as this would introduce a great deal of redundancy for individuals that share multiple siblings[2]. Instead, these relationships are inferred under the rubric that trios who share the same set of parents must be siblings. Similarly, (great-)grandparents and (great-)grandchildren are inferred from recursing up through a parent's identifying record, or down through an offspring's.

Individuals without parents are given a null pointer to refer to[3], but they are only of use to an analysis if they have offspring (and are founders).

Extra fields are also permitted, but these are typically used to store genotype data which can be housed in a separate file.

Individual IDs can be alphanumeric, but most linkage programs were written at a time where it was more efficient to use plain integers, and precautious measures have taught us to follow this convention.

Another convention of ours is to follow an individual identifier naming scheme of:

<generation><gender (odd/even)>

2 who would then be stated multiple times at later points in the file
3 usually a '0', where '0' is a protected identifier for a non-existent individual.

Generation begins at 1 for the first generation, and becomes 2 for the second generation and so forth. The gender specifier is the lowest unused integer that is odd for males and even for females.

e.g. 12 – the first described female in the first generation, 45 – the third described male in the fourth generation (where 41 and 43 have already been used).

It is not a necessary rubric, but it facilitates in the creation of large (and often consanguineous) pedigrees.

1.2.2 **Genotypes** (genotypes.dat)

If the pedigree file does not contain genotypic data, then it is stored separately here. This is typically a truncated readout of the genotyping chip output file.

The full genotype output report is paired with map data that contains marker positions as well as the AB genotypic call for each individual that was genotyped under the same chipset. Calls include, but are not limited to, a combination of A and/or B

The first type of calls (AA, AB, BB) relate to the actual binary allele that each individual is supposed to exhibit in an present/not-present manner. These are pure genotypes where phase is not present, and thus a call of BA would make no sense in this context.

The second type of calls (NC, - -, ??) are error messages and attribute a misfire (or 'No Call') of the chip such that the allele could not be determined.
Often the positional data relating to the markers are split into their own map file, leaving just a simple table of marker names and sample data.

```
            12    13    14    15    16    17    18    19    20    21    22
      23    24    25    26    27    28
rs12345     AA    AA    AA    AA    AA    AA    AA    AA    AA    AA    AA
      AA    AA    AA    AA    AA    AA
rs12346     AB    AA    AA    AB    BB    BB    NC    BB    AA    BB    AB
      NC    NC    AB    BB    AB    BB
rs12456     BB    AB    AA    AB    AA    BB    AA    AA    AA    BB    AB
      AB    AA    AA    AA    AB    AB
```

Text 2: Sample genotyping output readout file, with marker identifiers (without positions) as row headers, and sample identifiers as column headers.

In a given sample context, not all markers are as *informative* as each other. Informative markers are those that satisfy the following criteria:

Varied Set – There is enough variation in the genotypes for all samples that the marker covers. This ensures that the linkage algorithm can distinguish the descent of maternal and paternal alleles without too much ambiguity.

High Quality – A good proportion (approximately at least 80%) of the calls are not misfires. One or two No Calls in a small set can be resolved by the linkage analysis to reconstitute the missing genotypes from incomplete data, but only within reasonable limits.

Non-Zero Allele Frequencies (Optional) – Each marker is host to meta-statistics based upon the pool of samples it was used upon, the most relevant statistic being the *minor allele frequency*, which alludes to the representation of the less-likely allele in the overall population. Markers with 0:1 or 1:0 proportions are effectively homozygous[4] across the entire population and can be said to not contribute a varied enough set for linkage.

Known Inheritance Pattern (Optional) - The HapMap project was the first international effort to create a haplotype map of the human genome. Markers within this database are extremely useful for

4 Though the genotype may indeed be heterozygous

linkage since the markers tend to be quite common under the criteria that each allele is present in at least 1% of the population.

1.2.3 **Minor Allele Frequency** (maf.txt)

This file contains a map of markers and their associative minor allele frequencies in context to a super-population (European, African, Asian).

In the earlier days of linkage pipeline filtering, this was typically based upon historical HapMap data, but in more current renditions can be derived from a variety of different sources; the most recent being the calculated allele frequencies from the 1000 Genomes Project (Pilot 1 data) via the **bcftools** commandline utilities.

The markers in this file are used as selection criteria for generating a map, such that any markers with extremely (un)prevalent frequencies can be filtered out due to lack of informativeness.

1.2.4 **Genotype Map and Linkage Map Creation** (map.txt)

The genotype map is the file that comes paired with the genotype call data, such that the markers described in the output point to a physical or genetic map. Where genetic map data is not present, the physical location is used in place under the well-grounded assumption that genetic distance scales with physical distance, and that the marker ordering remains the same.

```
Chr     Name    0.000000        0       Name    x
01      rs3094315       0.752565        752565  rs3094315       x
01      rs2073813       0.753540        753540  rs2073813       x
01      rs2905040       0.770215        770215  rs2905040       x
01      rs12124819      0.776545        776545  rs12124819      x
```

Text 3: Map file describing markers in a genotyping chipset. Headers denote: chromosome, marker identifier, genetic map distance (cM), and physical map distance.

For a given genotype chipset there is an associated map file, and as the progression of chip-sequencing technology has rapidly developed, the map files have gotten significantly larger in accordance. As of 2012, Illumina's HumanOmni1S BeadChip kit boasts just over 1 million markers[5].

As stated before, Lander-Green based linkage analysis scales linearly with the number of markers, making it computationally impractical to use the entire map set. Instead we use a smaller sub-selection of informative SNPs under the same principles described in section 1.2.2.

Genotyping chipsets have fixed buffer sizes, and so if there are not enough members in a pedigree to satisfy the buffer, it is often economical to fill empty slots with individuals from unrelated pedigrees. As a result, this can lead to a very varied set of alleles across a given marker, artificially making it seem more informative (in context of the pedigree in question) than it actually is.

Though there is nothing wrong in including uninformative SNPs in a linkage analysis, it is up to the researcher whether they wish to trim the genotypes file so that it only contains members of the pedigree.

The Python tool **snpbutcher** was developed to facilitate in the creation of these maps under the aforementioned filtering precedents, with the added inclusion of spacing criterion as well as a number of other different optional parameters (see Appendix section <XXX ref> for full program functional disclosure), with all parameters saved to a log file.

Linkage analyses before dense genotyping chipsets allowed for a spacing requirement of 0.2 cM (~10,000 markers) between any two adjacent markers, but subsequent analyses in more modern times have iteratively led to a default spacing requirement of approximately 0.05 cM (~40,000 markers), keeping the marker count low for the linkage algorithm without compromising the power of the analysis.

5 http://support.illumina.com/array/array_kits/humanomni1s-8_beadchip_kit/specifications.html

The "resolution" of a linkage analysis is heavily dependent upon the number of markers used in the analysis, since the spacing between the flanking markers that the disease locus co-segregates with has a centiMorgan precision equal to the minimum spacing parameter used to generate the map.

1.3 Pre-Runtime Configuration

1.3.1 Folder Conventions

All four input files (pedfile.pro, map.txt, maf.txt, genotypes.dat) are placed into their own folder, following a rigid folder naming convention of:

/path/to/working/directory/<projectID>_<projectName>/<runNumber>[_runName]_<date>

e.g. /scratch/Linkage/124_boneitis/02_correctedpedigree_20120423, where:

- */scratch/Linkage/* is the folder of the working directory relative to the root file system
- *124_boneitis* refers to a unique project identifier of 124, and the description of the project's phenotype[6].
- *02_correctedpedigree_20120423* identifies that this is the second linkage run on the same input data set (corrected for minor pedigree changes) on the 23rd April 2012.

This ensures that each linkage project is uniquely identified by a project identifier, such that subsequent analyses can be run in the same master folder to reduce the number of repeating run numbers in the parent folder.

Once an analysis is fully complete (i.e. it is understood that there will be no more subsequent runs on the same input data set for at least a period of a month), then the project should be archived into

6 Boneitis is a dystrophic bone disorder that if left untreated can lead to comically accelerated TV references.

allocated storage.

1.3.2 Effective Filesystem Management

<XXX Maybe this section should go in the discussion: Why Upgrading Disk Technology Wont Necessarily Lend a Performance Increase...>

The working directory should be seen as a temporary file system at best, since many linkage programs generate a substantial amount of temporary files during processing. Though modern file systems are excellent at keeping track of and recovering files (a task known as *journaling*), it is still good practice to “mount a scratch monkey” such that temporary working files are isolated away from archived ones in distinctly separate partitions.

It is preferable to isolate the two file systems on different disks altogether, such that a disk-wide read error on a given hard disk won’t affect the operation of another. Most modern file systems have journaling enabled by default, which writes extra data to disk so that a recovery is possible in the event of a crash, but this becomes more problematic with Solid State Drives (SSDs) which have a limited number of read/write operations, and are prone to seizing should an operating system over-breach those limits.

This is not an improbable occurrence; operating systems routinely write temporary files to disk in order to manage RAM constraints in a process known as *paging*, where not immediately required portions of system memory can be sequestered into slower disk storage to free up space in the RAM for more higher priority tasks.

When a system begins to run low on RAM (as in the frequent case of extensive linkage analysis), a great deal of paging is performed to keep the linkage program in memory, leading to a substantial

amount of disk writes on a journaled file-system. Mechanical (spin) drives can handle these read/write requests robustly, but they are an order of magnitude slower than SSDs and significant bottlenecks can occur where the system has to wait for the disk to be ready in order to perform a block[7] read/write to it.

It is clear that SSDs offer no advantage at all in the use-case of frequent large temporary file activity, since though the operating system will rarely have to wait in order to perform a read/write, the limitations on the number of these operations shortens their lifetime considerably.

The best long-term compromise is to use mechanical disks with a filesystem without journaling; either an older filesystem where journaling was never implemented (though file size constraints may be enacted), or a newer filesystem where journaling is disabled upon initialization. This reduces the total number of read/write operations whilst still ensuring disk longevity.

Recoverability may be jeopardized, but the temporary nature of the files within the system imply that a re-run of the same analysis would not be resumable in any case; for it is the operating system that dispatches the temporary files to the linkage programs, and not the linkage program itself (with the notable exception of **Simwalk**[†]).

RAID configuration

One extra caveat to reduce the number of operations upon a single disk *and* keep some level of redundancy is to distribute the load across several disk drives in a standard redundant array of independent disks (RAID); specifically a RAID-5 level setup. RAID relies on the method of storing contiguous data across several mediums (a concept known as *striping*), as well as ensuring the data

7 If multiple sequential operations are requested on same contiguous portion of a disk, then the operating system groups the individual requests under a large contiguous 'block' request, that performs one long read/write operation instead of several short ones.

is correct and error-checkable without having to scan the entire block of data (a concept known as *parity*). The failure of one disk drive would not be enough to impede the setup and an ongoing analysis could be reconstituted from the other remaining disks, though a minimum of three identically sized disks is required.

These features lend well to a large temporary filesystem, since the storage capacities of multiple disks can now be combined to store a great amount of data, as well as the number of operations on any one disk being reduced by a factor of *n* (for *n* disks in the setup). The setup will still have the same read/write speeds as an ordinary disk, but there will not be any added complexity since the operating system will detect it as a single volume.

1.3.3 User Confirmation and Setup

Once the input files have been set, the **linkage_pipeline.sh** script is called to perform a thorough assessment upon the input files to determine if they are indeed primed correctly for analysis, and then acts as as a pre-analysis screening for the user to set run parameters and perform a double-confirmation that the set parameters are indeed correct. In order, the tasks are performed are:

1. **Encapsulation** – A GNU **screen** session is initiated (if not already present); a program that allows the user to re-access an ongoing process[†]. Multiple linkage analyses can be run at the same time through different screen sessions without interfering with one another, though this is not encouraged for large projects since they would still be competing for the same system resources.

2. **Display Detection** – This portion of the script detects whether there is a user logged on remotely or locally, and forwards graphical applications accordingly so that they appear

† see Program List section on page 34 for more details on this program

either on their machine, or locally on the host machine[†]. This has the added benefit of the user being able to inspect the data remotely during the pre-filter stage described later.

3. **Input Parsing** – The input files are checked for consistency with one another, most of the work being delegated to the Python script **prelim_check.py**, which asserts the following:

   I. Folder naming conventions are adhered to in the current working directory.
   II. Genotype file has correct headers (i.e. are present and numeric), and that marker identifiers are also present.
   III. Pedigree file specifies the correct project ID, contains all minimally required columns (family, id, father, mother, sex, and affectation), and has no duplicate individuals.
   IV. The genotype and pedigree file both contain the same set of IDs. Non-fatal error messages occur when the individuals specified in the pedigree are only a subset of all the individuals genotyped under the same chipset, likely because the user did not trim the genotypes beforehand. In this case, the user is prompted to visually evaluate the IDs that will not be assessed in the linkage and asked for confirmation, where a termination signal is sent upon refusal.

4. **Run Configuration** - The user is prompted for analysis-specific options: dominant or recessive; autosomal[8] or X-linked; single-core or multi-core. Since chromosomes segregate independently, the multi-core option allows for multiple chromosomes to be evaluated simultaneously using the GNU **parallel**[†] framework. Once set, the user is presented with the options they just specified, and if all is correct, a log of what was just entered is stored in a hidden file to be used in readme generation later. At this stage, the appropriate pipeline script is started.

8 It should be noted that even if an autosomal analysis is specified, the genotypes must still contain chromosome X markers for accurate gender detection in the next section.

1.4 Runtime Filtering

Filtering is the first and foremost important step in the entire pipeline. The genotypes, pedigrees, and markers are all checked for relational consistency and preliminary tests are run to better gauge the data. The filtering step is split into three main script components: **auto_stage1.sh**, **auto_stage2.sh**, and **auto_stage3.sh.**

1.4.1 auto_stage1.sh

This script is split into four main components: pedigree checking, gender checking, relationship priming, relationship checking.

Pedigree Check

The first stage involves re-checking the pedigree for inconsistencies as before (duplicates, project IDs), but the Perl program **HaploPainter**[†] goes deeper to also follow the relationship in all trios and check that each member is defined. If no errors are discovered, a PDF of the pedigree is generated and the pipeline resumes.

Gender Check

Here, the Perl program **alohomora**[†] reads in the map file and the maf file, where the minor allele frequencies are mapped to the markers and then compared with the genotypes by counting the number of heterozygous SNPs in chromosome X. Gender detection relies on the ratio of the heterozygous alleles of females against the hemizygous alleles of males towards the q-arm of chromosome X, due to a shortened Y chromosome in the males. Errors at this stage are rare, but are

often caused by sample mix up, and can be rectified by changing pedigree members to their correct genders in the input pedfile.

The program outputs a relationship file containing a mixture of pedigree data and genotypes which is to be used in the next step.

Relationship Priming

This step is a correctional one, only coming into effect if the output relationship file from the previous step is empty. This occurs if the maf file contains allele frequencies for no more than six markers that are present in the map, showing poor overlap.

The frequencies in the maf file apply directly to HapMap markers, but the HapMap project has not kept up with the increasingly dense genotyping chipsets, so modern allele frequencies need to be generated from new data (i.e.. 1000 Genomes allele frequencies from sub-populations).

This can be achieved by regenerating the maf file from new data sources, or creating a “dummy” allele frequencies by assuming that all alleles are equally represented across populations. This is a depreciated step that should never be used for a true analysis, but it exists in the case where the user merely wants to gauge the overall linkage before embarking on a more thorough study.

In this case, a new maf file is generated and the previous step is re-run to generate a valid relationship input file.

Relationship Checking

The next step is the visual inspection of the genotypes through the genetics relationship representation program **GRR**[†]. GRR evaluates the genotype distribution of between related individuals and determines how much allele-sharing is occurring compared to how much is expected, classed by their respective levels of relation. Unrelated individuals will share fewer alleles than more related individuals, and this can be plotted in a graph with axes of standard error vs standard deviation.

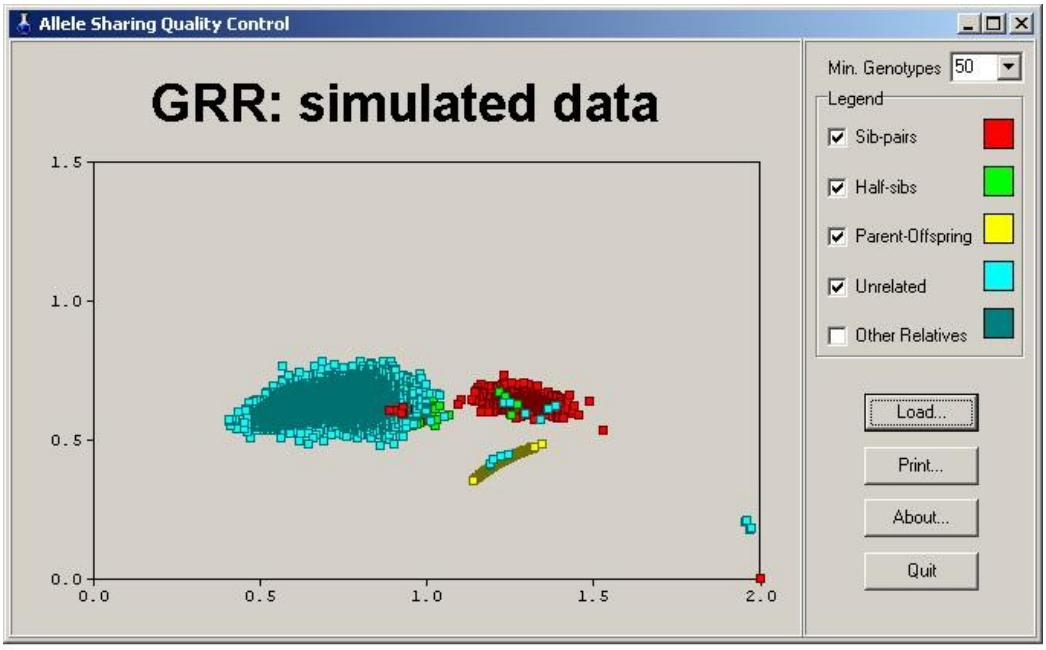

Figure 1: A simulated data set showing relationship clusters coloured by type and plotted by mean against standard deviation.

GRR uses *identity-by-state (IBS)* allele sharing in its metrics, allowing it to identify monozygotic twins (who share identical alleles) without having trace haplotypes.

Each point is formed by querying the relationships between any two individuals, coloured by the type of relationship specified in the pedigree, and plotting the mean against the standard deviation of the allele's being shared.

This forms distinct clusters, where each cluster is typically a representation of a single relationship type (siblings, half-siblings, parent-offspring, unrelated). This display is crucial, for if there is invalid clustering or mixing, then these are alarm signals for the user to terminate the pipeline.

In Figure 1, we can see a single red dot in the bottom-lefthand corner. Red indicates that the pedigree states that they are siblings, but they are somehow skewed far from the main Sib-pair group. GRR allows you to click on the dot and examine the relationship, and in this case it becomes

apparent that this sibling relation has identified a monozygotic twin pair who share identical alleles, giving them an extremely high IBS score.

There are also two unrelated relationships with a near-perfect high mean score, indicating once more a duplicated set of alleles, but in this case likely a sample or pedigree mix up; either the same individual was genotyped twice, or the pedigree did not state they were related, or the pedigree specifies different individuals for the same set of genotypes.

The green Half-sibs seem to be split into different groups of Unrelated and Sib-pairs, as well as a small cluster of Sib-pairs within the Unrelated. Both hint that a restructure of the pedigree may be required due to erroneous information specified in the pedigree, likely due to obfuscated parentage.

Should the relationships cluster improperly, the user must terminate the pipeline and re-evaluate their input before continuing.

1.4.2 auto_stage2.sh

This stage performs fast preliminary linkage that ultimately checks for Mendelian inheritance, and gives an approximate overview of what linkage scores we can expect.

Mendelian Inheritance Check

Mendelian errors are detected very quickly through the examination of trios, where a marker is flagged if a child's genotype contains an allele that could not have possibly been inheriting from their parents. Such errors can be indicative of non-Mendellian inheritance models such as trinucleotide repeat disorders or genomic imprinting (XXXref), but are typically more associated with the low error rate (0.01%) of genotyping chipsets (Saunders et al., 2007)

A list of markers with unlikely genotypes are generated from this checking step, and plotted in a chart of physical position against error frequency. For *n* families, the maximum error count is *n*, and markers with an error frequency greater than 1 are very indicative of poor quality.

In a typical linkage analysis of 40,000 markers an average of 5 erroneous markers is expected, and this scales linearly with the number of markers.

<image of mendel errors>

Higher numbers of erroneous markers are not uncommon (10 – 30) as long as they are sparsely distributed across the genome. Closely packed clusters of markers numbering greater than 50 usually suggest bad parentage in the pedigree, and several clusters totalling more than 150 are a clear sign that the entire pedigree requires re-verification.

This is a user-evaluated step, with a PDF of the plot being displayed on the screen, so it is up to the user to terminate the pipeline if the situation requires it.

Preliminary Linkage Overview

Once the erroneous markers from the list of unlikely genotypes have been removed, the linkage analysis program **Merlin**[†] is started. The error detection suite detects genotyping errors by inferring inconsistent haploblocks in otherwise consistent stretches of alleles between siblings. Such blocks are easily detectable by searching for double recombinations within small marker intervals where the likelihood of such an event occuring is extremely improbable.

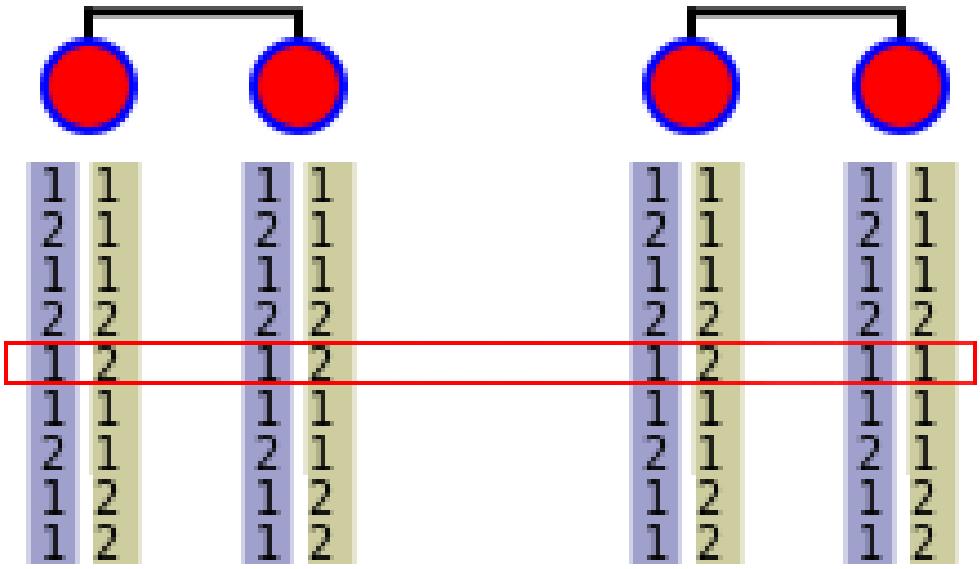

Figure 2: Two siblings have identical genotypes over a given allele (left), compared to the same alleles but with one contradicting genotype (right), indicating an unlikely double recombination event and a likely genotyping error

Another output list of erroneous markers is generated from this analysis and passed into the next section. Merlin performs a rudimentary pass over a small subset of these markers (~5,000 markers) to give a preliminary linkage plot that works to at least exclude regions of linkage.

1.4.3 auto_stage3.sh

This stage does not perform any filtering, but uses Merlin to generate input files for the main three linkage programs in the pipeline: **Allegro**[†], **Simwalk**, and **Genehunter**[†].

Each program is given its own folder, and pre-existing folders (from a previous run where errors were flagged) are renamed with a timestamp suffix (e.g. allegro → allegro-2016-04-02_10-58).

1.5 Runtime Linkage

Only Allegro and GeneHunter are run by default, and they are run sequentially so as not to impede the performances of one another. Though Simwalk input files are generated, they are not used unless further analysis is required.

This is the longest stage of the entire pipeline, with the previous filtering (auto) stages taking no longer than 5 minutes. Here, the rest of the pipeline can take anything from 10 minutes for small pedigrees, and then any interval from 10 hours to over a 100 hours for pedigrees larger than 19 bits.

1.5.1 Preliminary LOD estimation

The first step is to produce a maximum *estimated LOD (ELOD)* score, using a simulated data upon the pedigree. This is produced by the Python script **clodhopper.py**[†] which generates false linkage files for both Allegro and Genehunter using the shortest chromosome input files as templates (chr21).

The maximum ELOD score is parsed straight from the output files, storing largest result in a parent folder, and then all related files are deleted in preparation for the real linkage analysis. Scores between Allegro and GeneHunter tend to differ by only – 0.2. Any larger discrepancies are suspect, and usually prompt a Simwalk run as a precautionary third control.

1.5.2 Simwalk

Though Simwalk is not run by default it is useful to talk about it before Allegro and Genehunter, since it is the more classically robust linkage program with a longer publication history. It is generally one order slower than more modern linkage programs and is not able to process X-linked

inheritance models effectively, which is why it is not included by default. Analysis on moderately sized 19-bit pedigrees can take several days to complete. There are methods we designed to increase its throughput, but these are discussed later (see page <XXXRef>).

1.5.3 GeneHunter

GeneHunter is one of the core linkage programs, and is used as the main secondary control to Allegro. It is by far the most verbose of the linkage programs, and terminal multiplexers such as screen should take care not to enter a buffer mode (such as copy) should the buffer grow too large and unresponsive from the copious verbose output. It is wise to disconnect from a screen session to let this part of the pipeline run unmonitored, only checking in periodically from time to time if warranted.

1.5.4 Allegro

The fastest, memory compact, and consistently accurate linkage program in the entire runtime linkage suite. Allegro's main strength comes from its ability to perform multi-point parametric linkage and haplotype reconstruction within the same pass. For a pedigree under 19-bits, it performs linkage upon the entire genome with an average time of 10 mins. The real bottleneck occurs for any pedigrees outside of this limit, but this is addressed in the next chapter (see the Large Pedigrees section on page 25).

1.6 Post-Linkage Processing

After each linkage step, clean up operations are performed to reclaim the disk space generated by the temporary files that were created at runtime. This largely involves recursively crawling through

the directories within the parent folder, and removing any folders that match numeric-only filenames[9].

Filtering is performed via the script **messner** which parses the output the linkage files in each of the three linkage program folders (allegro, genehunter, simwalk) and extracts the any linkage regions with a score above 2, as well as the name of the flanking markers for each region. This result is stored in a messner_<program>.txt file in the parent folder.

Conversion operations are also performed at this step, with the **chromosomes.sh** taking the output result files of the linkage program folder it is invoked within (allegro, genehunter, simwalk), and generating graphical plots with the data, by using the graphing tool **GNUplot** which outputs the graph in postscript format, later converted to PDF via the **ps2pdf** system utilities. The **mkhaplotypes.sh** script converts the haplotype output files generated by Allegro (ihaplo.out) and feeds them into **HaploPainter**, which then renders them using the **Cairo** library, and outputs them in the PDF format.

This happens automatically, with the final message of the pipeline prompting the user to run **readme_gen.py**, which generates the readme file using the saved logs from snpbutcher and pre-linkage user criterion to fill in the core of the data such as: project ID, genotyping chipset, number of markers used, recessive/dominant, and autosomal/X-linked. It is up to the user to fill in other fields such as: original genotypes filename, name of user/technician, and any extra notes specific to the run.

At the end of the pipeline, all files instrumental to the analysis of the data are extracted into a separate directory (outside of the parent folder) through the **collect.sh** script. The files collected are the 4 input files (map, maf, genotypes, pedigree), the readme file, the mendelian errors plot, the

9 The find command paired with a good regular expression is an excellent one-liner method of discovering these folders: *find ./ -type d -exec [ls {} | grep \b[0-9]+\b] && rm {} \;*

GRR relationship plot, the messner file, the linkage plot PDFs, the haplotype PDFs, and the pedigree PDF.

1.7 Linkage Pipeline Limitations and Solutions

All that was described in the previous chapter was how the linkage program operates under normal input, the main dividing factor between normal and unnormal input being whether the pedigree size is above 19-bits, and/or is X-linked, and/or if the run is requested to be multicore.

1.7.1 Parallelization

The need for parallelization in linkage becomes apparent almost immediately after the first run; chromosomes are evaluated independently, and the program uses only a single core to perform the computation using a small percentage of the total RAM, whilst other cores lay idle.

An ideal run would utilize all the cores/threads available on a system at any given time, by processing several chromosomes at the same time. Such a process must first be *thread safe*, meaning that any two instances of the same program either use independent output resources (temporary files, output files, blocks of memory) or any shared resources are accessed mutually exclusively such that they do not write to the same resource at the same time (see Appendix section <XXXRef> for a quick overview on semaphores).

Three scripts were developed under the parallel framework to ease this process, by queuing up chromosomes (or 'jobs') in a waiting buffer, and dispatching from it whenever a core (or 'resource') became available. The script **gh_parallel.sh** dispatched jobs from Genehunter, and the script **allegro_parallel.sh** did the same from Allegro. The script **simwalk_multicore.sh** worked under the same principle, but operated differently because Simwalk splits its input not just into chromosomes, but smaller fixed-size loci. This has the benefit of being able to partially process an

entire chromosome, since some of the sub-input files might have been processed and others not. This also allows the script to be *resumeable*, in the manner that the script can be terminated prematurely, and then re-run, but it would not start from scratch upon each chromosome, but would resume from the files it had not processed yet. The parallel description in the Program Listing goes into further detail about the nature of these scripts.

1.7.2 Distribution

The initial pipeline was built for a very specific platform in mind: Ubuntu 10.04. Ubuntu is a Linux operating system (OS) that is built pre-packaged with utilities, drivers, and codecs, that make it work for many general purposes without much prior configuration from the user. It is actively developed with two main major release models: Bleeding Edge (all updates, and support for a year until the next Bleeding Edge release), and Long Term Release (major updates only and supported for 3 years until the next LTR). Its base is derived from a more stable Linux OS (Debian) which exists upstream and is the source of all its major core package and kernel updates.

Linux OS's are structured under a tree schema, with all locations being derived from the root /[10] and most Linux subsystems exist off this root[11], locations of note being: /usr/bin and /usr/local/bin, each being included in the systems PATH variable which allows all files within those locations[12] to be seen by the entire system without having to type out the entire filename.

e.g. If the binary *echo* is placed in the location /usr/local/bin/, it now exists as /usr/local/bin/echo. It is cumbersome to have to type out /usr/local/bin/echo every time we wish to use the command, so if the path /usr/local/bin is included within our PATH variable, we can use the command by simply calling 'echo'.

10 Not to be confused with '/root/' which is where the *root* user stores their files, the root user having full access to every part of the system. Other (less-privileged) users keep their personal files in '/home'
11See section Linux XXXref in the Appendix for an overview of the Linux filesystem structure.
12It should be noted that inclusion is not recursive; any items within sub-folders at a given location are not included.

The pipelines interconnecting scripts, program scripts, and binaries, were solely contained within the /usr/bin/ and /usr/local/bin locations, meaning that they were mixed amongst the operating systems binaries making it very hard to distinguish between the two. One of the first operations performed in the second iteration of the pipeline was extracting the binaries specific to the pipeline from these locations and storing them in separate locations.

The method required involved a brute-force approach of cloning the operating system and programatically deleting any scripts or programs not called by the pipeline. The **ldd**† utility was also employed in this regard to discover the locations of libraries, and library dependencies so that these could be extracted too. It was thought at the time that the pipeline was version specific to the utilities it depended upon.

Once extracted, an installer script was developed and through trial-and-error and most of the vast library dependencies were removed as the pipeline became less Ubuntu/Debian based and more Linux-centric. The pipeline was then moved under the **git**† school of version control, so that further development could be managed more effectively with a private backup on the code hosting site bitbucket.org. Changes to the pipeline could be performed upon separate feature branches without affecting the base code, and if the modifications were stable and successful they could be merged into base (or 'main') branch.

Under this new distribution model, the pipeline has since been ported onto more modern versions of Ubuntu (from version 12.04 to 15.04) as well as other Linux OS's such as Arch and Gentoo.

1.7.3 X-linkage

The first iteration of the pipeline was unable to generate the relevant input files required for X-linked analyses. Setting the Allegro and Genehunter input flags to run chromosome X (as per their respective manuals) worked to no avail, instead resulting in numerous errors detailing inconsistent

markers and penetrances; the main problem stemming from the two separate male and female penetrances required in X-linked inheritance models (see page <XXXRef> in the Background for more details).

There were several stages where this problem could originate, the most likely being the Alohomora input file generation stage, but closer inspection revealed that the program merely re-formats the data and makes no assumptions upon the genetic model. The real source of the error messages was from the separate linkage programs themselves.

Upon painstakingly thorough cross-checking between the input formats of the Allegro and Genehunter, penetrances could be set to <RTFM>, and correct run configuration files could be generated for the setup. This process was automated via the script **x_makedat.py** which modifies the chromosome X input files by setting the "use X" flag, and producing an extra line of male penetrances of either {0 1 1} or {0 0 1} (XXXexplain the fieldsXXX) for dominant/recessive models within Genehunter.

Another script (**x_allegroin_haploout.py**) generates the correct the Allegro run parameters in an almost identical manner. The same script is also used after a linkage run to convert haplotype output files into a more consistent formatting scheme, where males which only have 1 allele haplotyped in the analysis, now have 2 (a Y-chromosome consisting of entirely zeroes), aiding in the haplotype visualization process when inspecting the results

1.7.4 Large Pedigrees

Despite evidence pointing to the contrary (XXXref, XXXref), neither Allegro nor Genehunter could handle large bit-size pedigrees very well during our pipeline runs. Even if the number of informative markers were reduced to a 1000 marker range, which was well below the recommended

limit for precision linkage, the programs would either terminate early or throw an unhandled exception during runtime.

Such behaviour are typical of out-of-memory / insufficient-resource errors, but most linkage runs use below 500Mb of memory, and our hardware was more than capable of handling any resource constraints. The problem was most likely occurring once again in the input files.

Genehunter

Genehunter was relatively easy to fix , by simply changing its default MAXBIT size setting 19 to 30. This worked well for pedigrees within a range of 19 to 22 bits, but any larger and Genehunter would split the pedigree or truncate individuals altogether in order to spare the computation. Flags to prevent these processes (XXXName them) were alternated in numerous runs, but Genehunter would override them if it deemed that it could not perform the computation.

Due to the unresolved splitting problem in large pedigrees for Genehunter, Allegro is the only linkage program used for large bit-sized pedigrees.

Allegro

Allegro required much more time and thought, resulting in a costly solution that involved patching a bug within the program binary itself.

The method in which Allegro spared itself extra computation was by making use of MTBDDs (see section XXXRef) which provided a compact structure to access and store ongoing calculations without duplication or redundancy. The MTBDDs were implemented via the CUDD library from the University of <XXXREF>, which at the time of Allegro's conception was on version 2.4.1 and

included within the program source. Careful debugging of the errors thrown by Allegro led to this CUDD source, and it was clear that the fault lay there.

Updating the library to the (then) latest version of 2.6 yielded no success[13], and the program would not compile due to various inconsistent function calls between versions. Many of these had been depreciated in favour of new ones that no doubt simplified and/or merged several tasks under a single call within the library itself, but broke specific functionality in Allegro.

The only other option left was to disable the CUDD library altogether, by forcing Allegro to skip the pre-existing computation checks, and attempt to keep all calculations within memory, caching to disk where possible.

This worked somewhat, and pedigrees smaller than 19-bits had no significant change in their performance. Pedigrees larger than 21-bits was the limit where the system resources started to become more scarce. The analyses would run, but very slowly and at the cost of quickly diminishing RAM and disk space. Haplotype reconstruction required almost no disk space, but often over 10 terabytes of memory to function, the most of which was implemented via VRAM by using disk space as swap space. The default allegro binary could run both multi-point parametric linkage and haplotype reconstruction in the same pass, but the newly patched binary could only now do one at a time.

Large pedigrees can now be run via the **allegro21bitstart.sh** script, which calls the newly compiled allegro binary over the input files, and takes an argument that allows it perform either haplotype reconstruction or multi-point parametric linkage (but not both). The user is greeted with a display detailing how much available swap space, memory, and cores/threads are available. If the scratch partition is not detected, the user is prevented from starting any processes.

13As of 2015 there is now a version 3.0 of the CUDD library, but the changelog reveals that only the overall packaging has changed, and the library code has not been modified since 2.6.

The script also has the capability of parallelizing the process, but given the large memory demands of a single instance, this functionality goes largely unused.

1.7.5 Graphical Tools and Post Analysis Software

Once the data has been collected, it is then available for interpretation to determine whether any linkage peaks were actually discovered, and how informative they are in regard to the regions they are discovered in.

1.7.6 Linkage Visualization

Previously, the output results were simply collated by the GNUplot program and plotted as is, without any modification. This may sound agreeable in theory, but there were several issues that were not being addressed; namely that the plots were using genetic distance, and that all adjacent points were joined together.

Genetic distance, though an extremely useful representation of the expected number of crossovers across a region, does not represent the structure of the chromosome very well. Though genetic distance does scale fairly linearly with physical distance, the position of markers varies largely between different versions of the human genome, leading to some confusion in the actual location of a linkage peak.

The spread of markers are also not taken into account, since most SNPs do not fall within telomeric or centromeric regions, meaning that there should be a noticeable gap within the plot towards the head, tail, and off-middle section.

To add some context to the plots, the overarching script **physical_linkage_plot.sh** was developed with the intention of reading in the output files, and remapping the scores back to physical distance

for a specified version of the human genome[14]. This is performed through the Python script **read_all_linkage.py**, which performs an additional quality check of splitting data over impossible regions, such as any linkage peaks spanning the centromere (possible due to the last informative marker on the p-arm having the same score as the first-informative marker on the q-arm). The script works under the principle of bisecting the data at fixed intervals, such that plots are not variably-spaced and prone to joining over long distances.

The re-plotting involves reconstituting the data within a GNUplot environment, and this is done through the script **plot_linkage_physical.py** which not only takes in the remapped data, but also a map file of chromosomal regions that dictates the true length of each chromosome (rather than the length inferred from the first and last markers in the analysis), and another map file of chromosomal bands which contains the names and regions of each band and sub-band of every chromosomal band. These bands are then overlayed upon the linkage plot to provide instantly visible context to some of the more questionable peaks in an analysis by allowing the user to see the density of the band the linkage peak falls within.

Genomewide and individual plots are generated under this method, with the individual plots being generated simultaneously using the parallel framework. The overarching script can handle output data from allegro, simwalk, and swiftlink; producing the same type of plots for each (though titled accordingly) so as to provide some consistency when comparing plots from two different program.

Previously, visual comparison of different linkage results from different programs was impeded by the varying size of the axes due to spacing inconsistencies introduced by different header labels, and that the analysis was constrained to the size of the analysis and not the size of the chromosome.

The consistent sizing provides a huge advantage when programatically comparing the data, since the points in all the remapped linkage output files all correspond to the same points, and comparing

14 Currently hg19 is used, but any build within the UCSC Genome Browser can be used.

sets is a simple matter of addition/subtraction.

1.1.1 Haplotype Inspection and Rendering

Haplotype inspection is problematic. The allegro output haplotype file (ihaplo.out) follows the pre-MAKEPED structure which represents alleles horizontally. Worse, the marker headers are represented as column headers in a fixed-width text structure that types them out vertically across multiple lines. Dealing with the actual file involves transposing and trimming the headers, and then inspecting the haplotypes with a text editor that uses window-space buffering such that the entire file is not evaluated upon opening (which introduces visible slowdowns in programs such as gedit, and notepad).

Of course, browsing haplotype data alone is not enough for a haplotype analysis, since the real interest in the analysis is the representation of the haplotype blocks which cannot be shown in a plain text format. An application capable of resolving and representing the data as distinct blocks under a genetics model is required for a true analysis.

1.1.1.1 HaploPainter

HaploPainter is the one of the most commonly used open source pedigree and haplotype rendering tools, built upon a framework of powerful and well-maintained Perl libraries such as Cairo and Tk, it can produce a wide variety of output formats from the graphics that it renders.
It is primarily used by researchers for drawing pedigrees, but is also used in rendering haplotypes.

Researchers can examine an allegro output haplotype file by loading it into the application, which then renders the pedigree and then the complete haplotypes trailing underneath each individual.

Though seemingly sensible in relation to the pedigree structure, this ultimately leads to several problems:

- **Pan and Zoom Inspection** – In order to inspect the haplotypes, the user is forced to pan and zoom across the screen in haphazard manner; the sensitivity of movement is set by the program and not appropriately scaled for zoom level, meaning it is a frequent problem for the user to overshoot their region of interest. Distressingly, the arrow keys are bound to other functions, and so the user must navigate sections using the mouse alone.
- **Minuscule Fonts in Large Sets** – Each haplotype representation alludes to a specific chromosome, which depending on which chromosome is being analysed can range greatly between 200 (e.g. chr21) to 2000 (e.g. chr1) for a typical 40k marker SNP analysis. Small marker sets can be easily rendered with normal fonts since the pedigree does not need to be shrunk to fit into working space of the application. Larger sets are more problematic, with minuscule fonts being required to fit everything into the working space. These fonts do not scale very well when being exported into other formats such as JPG or even lossless formats such as PNG or PDF.
- **Out of Range in Large Sets –** The problem of minuscule fonts leads to another issue that becomes apparent as soon as the user attempts to zoom into a region of interest: the minuscule fonts rendered may be just outside of the reach of the maximum zoom level, making the marker names and genotypes unreadable. Exporting to an image format such as JPG or PNG will only make this problem more apparent, since fonts will be rasterized into the image canvas at a fixed resolution, which even if set high will still not scale the fonts well.
- **Couple or Sibling Analysis Only** – Since haplotypes trail vertically with the pedigree, a given region of interest is not contained within a single viewport, but appears once for each generation. For large complex pedigrees, the viewport quickly becomes localised to each set of offspring[15]. This restricts the region of interest to a side-by-side comparison of either siblings or couples, which is somewhat limiting in dominant pedigrees when case and control individuals can appear in any generation.

15 Cousins in the same generation may be vertically aligned, but likely a great deal of horizontal scrolling back and forth would be required to compare genotypes between them.

- **Unsupported X-linked Haplotypes** – At the time of HaploPainter's release, X-linked analyses were not as common as autosomal, mainly due to the distribution of SNPs in earlier genotyping chipsets. As a result, the X chromosome was likely not extensively tested, and bugs were likely to manifest at some point.

In order to address the minuscule fonts as well as the pan and zoom limitations, the extremely feature-rich[16] script **haploregion.py**† was developed to produce a subset haplotypes file that would reduce the number of markers to just the region of interest. The region of interest is usually common regions of heterozygosity/homozygosity across cases which the script also produces.

This, in precession with **HaploPainterRFH**† (a modified HaploPainter clone) produces a much more useable representation of the haplotypes that the user does not need to extensively pan/zoom through.

The modified script also highlights regions specified either manually or through a messner file, overlaying coloured boxes onto the haplotypes. In a given analysis, the two scripts are usually setup in a way that identifies regions of homology (depending upon the genetic model) across affected individuals, which are then used to generate a truncated *ihaplo.out* file that limits the analysis to those regions (usually with a flanking amount of ten markers).

The result of this is a concise representation of the haplotypes: localised to the linkage peak(s) with a flanking marker range of 20; actual linkage peaks highlighted in blue boxes via the messner file; regions of homology highlighted in red boxes. This makes for a quick visual analysis without any room for human error during inspection.

Unfortunately, the scripts do not solve the issue of X-linked haplotype reconstruction and side-by-side case vs control analysis. A bug report paired with a minimal working example of an X-linked

16 It is strongly advised to read the Program Listing section for this script in order to acquire a full appreciation of its toolset.

data set was submitted, and eager email correspondence with the authors was also established, but no update has been issued to resolve this problem.

Months of silence from the developers, and a growing need for a comprehensive haplotype suite not reliant on HaploPainter made the need for another solution very apparent, and it was with that momentum which prompted the development of *HaploHTML5*.

1.7.7 Program List

Allegro – Moe.

Alohomora – A linkage analysis suite written in Perl, primarily used for its quality control modules such as pedigree checking and Mendelian error checking. It is also an excellent format conversion tool, and is primarily responsible for taking the input files set by the user at the beginning of the linkage analysis and converting into input formats for programs such as **Merlin**, **Allegro**, **GeneHunter**, and **Simwalk**.

clodhopper.py – A Python script to estimate the maximum LOD score of an analysis for either Simwalk, GeneHunter, or Allegro. The input files generated are program specific, but all work under the same principle of creating three distinct haplotype blocks: affected individuals (12^g, 11^h, 12^g), unaffected individuals with unaffected parents (12^g, 12^h, 12^g), and unaffected individuals with affected parents (12^g, 22^h, 12^g), where g and h are positive integers and $g \neq h$. The implication being that all affected individuals are subject to the same (in this case, double) recombination event to create a singular well-defined linkage peak spanning h markers. It is then trivial for linkage programs to evaluate this.

GeneHunter – Moe.

Git – Git is a versioning tool.

GRR – The only windows binary in the entire program listing. The program has no commandline interface and can only be run graphically. In order to load genotypes, we must simulate mouseclicks via **Xautomation** and regularly poll the display server via **Xwininfo** to verify that the correct window has appeared in order to resume automation.

haploregion.py – A Python script to address the large marker sets in haplotype output files from Allegro (ihaplo.out). It has many functions:

- **Producing Haplotype Subsets** – It does this by taking pairs of markers as argument as well as an optional flanking parameter. The script transposes the headers in the original file, and attempts to match the marker-pairs specified in the arguments to find consecutive non-overlapping regions within the file. The flanking parameter (by default set to 0) specifies the amount at which the region(s) of interest is expanded outwards, such that a flanking parameter of 10 would expand 10 markers left and 10 markers right of the region specified by the marker pair being evaluated. Due to the script usually preceding the modified **HaploPainterRFH** script, it requires not only the *ihaplo.out* file but also the *map.N* file (where N is a chromosome number) to be in the same directory.

- **Detecting Regions of Homology –** The script can also parse a *ihaplo.out* file to look for regions of heterozygosity and/or homozygosity between cases, depending on the inclusion of the respective settings specified in the arguments. If control exclusion is desired, this can also be optionally specified such that only regions of homology that exist in cases and not in controls are produced. A flag to set the minimum size of a stretch of homology can also be set, with the default value set to 2 such that sparse regions of homology spanning a single marker are not included in the final result (though of course this can be overridden). The general output of this is a list of marker pairs, but a debug flag can be specified to show the step-by-step working of the entire process. The debug output consists of a table with marker identifiers as rows and individuals (with identifying case/control tags) as columns, the data depicting the genotype of a specific individual-marker, as well as a boolean true/false score at the end of each row to show evaluation of cases (vs controls (if specified)) and current size of the stretch of homology.

- **Haplotype Extraction** – Haplotypes in the *ihaplo.out* file are depicted horizontally which is not the ideal format when trying to copy/paste haplotypes spanning a given region, since regions will need to be copied line-by-line manually by the user, which could introduce errors. The haplotype extraction mode works under the default assumption that all genotypes from all individuals are to be extracted, and it does so by transposing the headers

and the data separately and then printing the data out into a table such that marker identifiers are rows, and individual identifiers are columns. It should be noted that this is the same format as the debug mode used in detecting regions of homology (outlined in the previous bullet point), but this mode does not perform any homology checks, and also takes optional parameters to specify a subset of individuals and/or a subset of markers. The resultant output file is then in a much more useable format, and copy/pasting haplotypes in a given region is facilitated via the conventional method of selecting a region of text that spans multiple lines.

HaploPainter – This is a Perl script/program that relies on the TkInter framework for drawing and parsing pedigrees with an interactive graphical user interface (GUI). It has a background batch mode that enables it to run without the need to draw anything, and it can also parse and draw haplotypes. Should an invalid pedigree be given to it, it reports to standard error and terminates.

HaploPainterRFH – A modified clone of HaploPainter, with extra functionality added to be tailored more to the type inspection performed at the RFH. New features include the optional parameters of reading in a messner output file and/or pairs of markers, plotting a transparent box of blue and/or red to highlight multiple regions of interest across multiple generations. A limitation of the script (as with the original HaploPainter) is that it requires both the *ihaplo.out* data file and the *map.N* map file (where N is a chromosome number) with a direct 1:1 mapping of each marker in the map file to each marker in the data file, with no tolerance for any overlaps. As a result, any modification of the *ihaplo.out* file (via **haploregion.py**) must be met with the same modifications within the *map.N* file.

ldd – This utility traces

Merlin – One of the fastest linkage application suites; capable of parametric and non-parametric analysis, as well as various utilities such as gender checking, pedigree checking, and genotyping error detection.

parallel – The GNU parallel framework is an encapsulating program that takes an existing program and a list of argument parameters, and then dispatches each argument to a separate clone of the program, running them simultaneously. It is implemented via a queue-and-dispatch system, where jobs are added sequentially to a queue buffer, and are dispatched in a first-come-first-serve manner to a given work resource until there are no more resources left. As soon as a job finishes, a resource is freed and another job can be dispatched from the queue. The number of active jobs at any one time can be set manually and can also be dynamically changed during runtime.

For efficiency, the number of active jobs should be the number of cores/threads that the CPU can handle, though if the task is memory-intensive then the number of jobs should be limited to the total amount of available RAM divided by the maximum memory usage of that task. Most linkage programs process an entire chromosome at once, and so jobs are queued as chromosomes, with small chromosomes such as chr21 finishing quickly compared to chr1. This is problematic if chromosomes are queued sequentially (either ascending or descending), as the lower-numbered chromosomes can only be effectively processed two at a time whereas the higher-numbered chromosomes can be processed five at a time, requiring a daemon to watch over RAM usage and change the number of active jobs parameter accordingly. This is not always effective, since the daemon must know beforehand how much system resources will be consumed for a given chromosome, and this varies with bit-size and the number of markers (which varies greatly between projects) and so the daemon must make conservative estimates so that the system does not run out of memory and begin killing processes.
A better compromise which requires much less oversight is to alternate chromosomes by counting from either end and meeting in the middle (e.g. the queue order would be: [**1**, 22, **2**, 21, **3**, 20, **4**, 19, **5**, 18, **6**, 17, **7**, 16, **8**, 15, **9**, 14, **10**, 13, **11**, 12]) with at most 3 jobs running at once.

Linkage programs that utilize a sliding-window approach (such as Simwalk) split input data evenly into windows of *n* markers, regardless of chromosome, and so jobs can be more efficiently queued without any resource monitoring at all.

Simwalk – Written in Fortran, it is one of the most trusted comprehensive linkage analysis suites spanning two decades. The current version is 2.91 and it performs parametric linkage, non-parametric linkage, and rudimentary haplotype reconstruction by searching for the optimal chain amongst the various Monte Carlo Markov Chain simulations it performs. On top of the three main input files (pedigree, map, genotypes), Simwalk also requires a penetrance file to complement the map as well as a custom script file to perform user-specified instructions that dictate the type of analysis. Input files are split into chromosomes, and then further split into overlapping marker windows with a 25-marker length. The output files generated come in three main prefix types: SCORE, ERROR, STATS, and VIDEO; each being tailed by an integer that maps it to a specific chromosome, and another integer to indicate which marker window is being examined. LOD scores are parsed from the SCORE files, and associative statistics with significance scores are detailed in the STATS files. ERROR files is where standard error is duplicated, and VIDEO is where standard output is duplicated.

screen – A program that allows for multiple users to access the same terminal session and co-operatively witness and interact with users sharing the same session. It has the added benefit that it detaches the parent process from the application it was spawned from (usually the desktop manager), such that the process is not killed when the application is closed. De-parenting a process is not exclusive to a screen session, but re-accessing the same terminal to resume interacting with an ongoing process is the main benefit of the program (though the **tmux** program is also capable of this).

standard in/error/out - All operating systems are made up several input and output channels known as *file descriptors*, and there is no limit to how many there are (though a static number of

less than 100 is typically enforced by the operating system). The three main file descriptors are: standard input (stdin), standard output (stdout), and standard error (stderr); with the first being an input channel and the remaining two being output channels. These channels are independent of each other, and act as all encompassing virtual buffers for various different processes to use in order to interact with the system, without each process having to create their own separate resources to do so. Programs are dependent upon these channels in order to receive input (via stin) and to dump their data (via stdout and stderr). Channels/file descriptors can be further forwarded into other file descriptors (such as file handlers) via *input/output redirection*. Redirection allows for channels to dump their data into files, or into each other such that stderr and stdout consolidate all output by flowing from the same channel. The separation of these two channels is deliberate however, since it is rarely beneficial to combine error messages and output results under the same context.

X11 – In Unix systems, the X11 display server is the legacy protocol for displaying graphical applications, where in order for the application to draw anything to screen it must first request a portion of the screen to draw to. Though seemingly convoluted for local applications, it has the added benefit of being able to manage multiple displays which may are may not be hosted on the actual machine itself. Through a protocol known as *X11 Forwarding*, a remotely logged in user can see graphical applications on their remote machine (as long as their machine also hosts an X11-capable display). There is some latency over slow connections, but this only affects **Xautomation** scripts, which if setup correctly can factor any delays into their timings.

Xautomation – Refers to either the program **xte** or **xdotool**, both of which have a similar API and function calls though the former is being deprecated and the later is more actively developed. Typical calls include *sleep <interval>*, *mousemove <xposition> <yposition>* and *mouseclick*. Calls can be strung together under the same command, and calls can be made relative to specific windows, but the windows must already exist. Window IDs can be determined via **xwininfo**, and are instrumental to determining accurate sleep intervals between Xautomation calls.

xwininfo – A utility that returns information about application windows displayed under the X11 protocol, such as; ID and title, class instance (the calling program), position, dimensions, and

display port. Specific windows can be selected by mouseclick, or the entire windows system can be scanned and filtered by a keyword. Typically xwininfo is included within a polling *while* loop which periodically scans the display until the desired window is present, and terminates upon the successful detection.