

A SYSTEM FOR AUTOMATED STORAGE AND ANALYSIS
OF SINGLE NUCLEOTIDE POLYMORPHISM DATA

MASTER'S THESIS
DEPARTMENT OF MEDICAL INFORMATICS AND
CLINICAL EPIDEMIOLOGY
OREGON HEALTH AND SCIENCE UNIVERSITY

BY HOLLIS WRIGHT

School of Medicine
Oregon Health & Science University

Certificate of Approval

This is to certify that the Master's Thesis of

Hollis Wright

"A System For Automated Storage And Analysis of Single
Nucleotide Polymorphisms"

Has been approved

[REDACTED]

Thesis Examination Committee Chair

[REDACTED]

Thesis Examination Committee Member

[REDACTED]

Thesis Examination Committee Member

TABLE OF CONTENTS

ABSTRACT.....	3
BACKGROUND.....	4
METHODS.....	9
REQUIREMENTS GATHERING AND DEVELOPMENT METHODOLOGY.....	9
STORAGE OF SNP GENOTYPE DATA... 	10
STORAGE OF HAPLOTYPE DATA.....	12
UPLOAD OF SNP DATA.....	14
INFERENCE OF HAPLOTYPES.....	16
QUERY OF HAPLOTYPES.....	21
RESULTS.....	22
FUTURE DIRECTIONS.....	26
DBSNP COMPATIBILITY.....	26
TAG SNP SELECTION ASSISTANCE.....	28
EXPANSION OF HAPLOTYPE INFERENCE CAPABILITY.....	29
EXPANSION OF INTERFACE CAPABILITIES.....	31
INTEGRATION OF OTHER EXPERIMENTAL DATA.....	32
CONCLUSION.....	34
REFERENCES.....	35
APPENDIX 1.....	37
APPENDIX 2.....	68

ABSTRACT

Single nucleotide polymorphism (SNP) genotyping provides a method of genotyping ideally suited to high-throughput operations. However, individual variation at SNP loci represents little information regarding genotype or phenotype on a locus-by-locus basis. Rather, sets of SNP loci that do not independently assort during meiosis, known as haplotypes, are used as more informative blocks of information for genotyping and linkage analysis, but they cannot be determined from genotype data alone and are difficult and costly to determine through most laboratory methods. Computational inference methods provide a solution for haplotype determination that is low-cost, highly accurate, and suited to high-throughput. The Oregon National Primate Research Center (ONPRC) has begun to utilize SNP genotyping but lacks an integrated solution for upload to the Genome Resource Informatics Program (GRIP) database and for subsequent analysis of that data.

This thesis describes the design and implementation of a data upload and haplotype inference system for ONPRC. Utilizing data from an SNP study in progress at the center, I have developed interfaces and infrastructure that allows researchers utilizing the GRIP website to easily upload their data to the GRIP database and to perform haplotype inference on that data. The system utilizes PedPhase as its haplotype inference program, but is extensible to use other inference programs as well. The system has been tested extensively and is capable of providing rapid uploads and robust haplotype inference and search capabilities. The system is also extensible to help better serve research needs as they evolve and grow.

BACKGROUND

Single Nucleotide Polymorphisms (SNPs) are single base pair variations in the genetic sequence of organisms. These variations are typically biallelic (e.g., for a specific SNP location, only two of the four possible nucleotides will be present as alleles), and, unlike other types of genetic variation used for genotyping purposes, such as microsatellite data, SNPs can be both quickly discovered and individuals subsequently typed by standard gene sequencing techniques. In addition, microarray-based techniques revolving around SNP genotyping have begun to appear (Di et al, 2005), helping to further simplify genotyping of SNP loci. The ease of use and growing variety of SNP genotyping technologies makes SNP-based genotyping an excellent choice for high-throughput genotyping strategies.

As single base pair variations, however, SNPs in most cases impart little information about phenotypic outcome individually as compared to other kinds of genotype information. While it is possible that a particular SNP allele might cause a noticeable effect in an organism's phenotype (as in the case of the point mutations implicated in prion formation (Croes et al, 2004)), most allelic variations at SNP loci have no disease-related or other major observable phenotypic effect. They are also less informative on a locus-by-locus basis in genotyping; for genotyping purposes, a specific SNP locus can only differentiate between two classes of organism (those possessing allele 1 as opposed to those possessing allele 2) whereas a microsatellite-based genotype locus, which uses different lengths of repetitive DNA as allelic identifiers, can differentiate as many classes of organism as there are allelic variations at that locus. However, patterns of SNP loci in the genome that are inherited together can act as more

informative markers; such sets of loci are called haplotypes or haplotype blocks.

Consider a set of polymorphisms located close together on a particular chromosome such that these loci do not independently relocate during crossover events at meiosis. These loci, along with the rest of the intervening DNA that is inherited with them, must be inherited together; hence, they form what is known as a haplotype block. One such block exists on each chromosome of an individual, and since these loci are inherited together their pattern acts as a signature that identifies the block; with a block of 5 SNPs, the pattern of SNP alleles effectively can reflect 2^5 or 32 potential states, making the haplotype much more informative than any specific locus it contains. Figure 1, below, illustrates this concept (It is important to note that the scale of this figure is considerably exaggerated for purposes of clarity; actual haplotype blocks would not be expected to span as large a section of a particular chromosome).

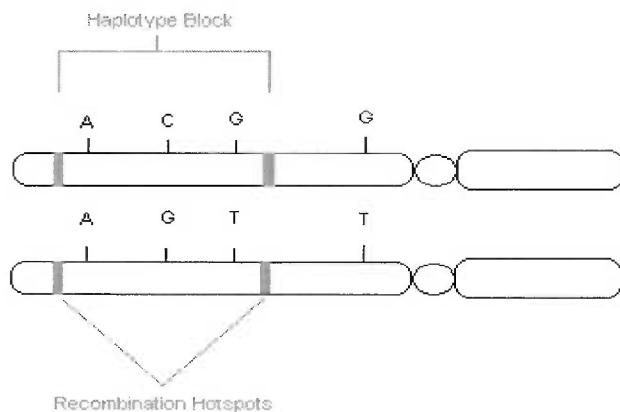


Figure 1. Illustration of Haplotype Block Structure.

Crawford and Nickerson (2005) provide a review of the clinical importance of haplotypes; to summarize, it is currently thought by many in the genetics community that such haplotypes are candidates to be used for markers identifying disease states or other phenotypic variations; further, it is believed that specific "tag" SNP loci can be defined

that will be useful for characterizing larger haplotypes because variations in these tag loci seem to associate with allelic variations in the rest of the haplotype's loci. Identification of tag SNP loci is generally accepted as a desirable end since it reduces the numbers of SNP loci that would need to be typed to genotype an individual, though use of tag SNP loci is less able to identify rare haplotypes (i.e., haplotypes whose allelic variation does not associate as expected with the tag locus's variation) and it is possible that such haplotypes may be very important in some genetic processes (Liu et al, 2005). A wide-scale study of human haplotypes in four diverse populations known as the HapMap project (International HapMap Consortium, 2003) has been launched; one of the primary purposes of this study is to determine if haplotypes will in fact yield useful information and if suitable tag SNPs can be identified in the human genome. Early results have been encouraging, and preliminary haplotype maps for some of the studied populations are publicly available at this time (see www.hapmap.org). An SNP-based haplotype map of the human major histocompatibility complex has recently been published (Miretti et. al, 2005), including "first-generation" tag SNP loci.

The primary difficulty in making use of haplotypes in diploid organisms involves the determination of which chromosomes particular alleles reside on; most genotyping techniques cannot determine which of the two chromosomes in an organism a specific SNP allele occurs on; a signal indicating either homozygosity (the same allele on both chromosomes) or heterozygosity (two differing alleles) is detected, but in the heterozygous case the alleles cannot be traced back to specific chromosomes. Going back to the example in Figure 1, it would be impossible to determine from genotype information alone which of the two chromosomes the G and C alleles at the second locus

resided on. This presents problems for haplotype determination; however, inference techniques present a solution to this difficulty. Consider an organism with two linked SNP loci that when genotyped presents alleles A and G at loci 1 and T and G at loci 2. The organism possesses two haplotypes (one for each chromosome), but these haplotypes will be selected from one of four possible haplotypes (e.g., A/T, A/G, G/T, and G/G) and it is not possible from the genotype information alone to determine which two of those four haplotypes are correct. If we know that one of the parents of this organism possessed a homozygous genotype A/G at these two loci, however, we can infer that the parent must have possessed the A/G haplotype on both chromosomes. Presumably, the child organism we are interested in must have inherited the A/G haplotype, if our assumption of linkage between the SNPs holds, and so we can determine that the child must possess haplotypes A/G and G/T; further, we can then identify the other parent as possessing at least one haplotype of type G/T, since it must have contributed the other haplotype to the offspring. Figure 2 illustrates this example.

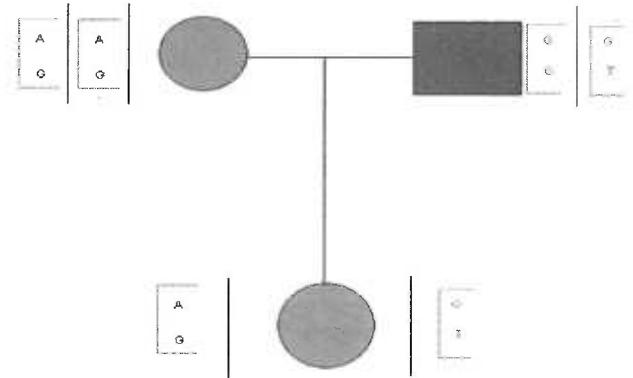


Figure 2. Illustration of haplotype inference

There are generally considered to be two primary types of inference algorithms in

use (most of which, while utilizing pedigree information to make inferences about haplotypes, are considerably more complex in their inference methods than the simple example above); those implementing versions of the expectation-maximization (EM) algorithm and those using Bayesian statistical analysis of population genetics. (Adkins 2004). The EM algorithm, as defined by Excoffier and Slatkin (1995), attempts to determine the most likely haplotypes for individuals from exhaustive analysis of pedigree data, but this is computationally intensive. Bayesian methods attempt to use expectations of population genetics to more efficiently determine the likely haplotypes. Adkins mentions a third class of algorithms termed "parsimony" algorithms, but considers these algorithms to have too many weaknesses for general use. However, programs using both of the primary inference methods have been shown to have relatively high reliability in correctly determining haplotypes (Adkins 2004, Xu et al. 2004) and hence such programs provide an alternative to molecular haplotype identification techniques such as long-range allele-specific PCR (Pont-Kingdon et al, 2004), which tend to be comparatively difficult and expensive and so not suited to high-throughput genotyping.

The Oregon National Primate Center (ONPRC) has begun to investigate the uses of SNP genotyping for its animals; recently, the Ferguson lab at ONPRC has begun a pilot study using SNP genotyping to differentiate Indian-origin rhesus macaques from Chinese-origin animals. As ONPRC moves forward with SNP genotyping projects, the need for automated management and analysis of this data will increase. This thesis describes the development of a prototype system to support the informatics needs of ONPRC scientists interested in SNP genotyping and analysis, using the Ferguson pilot study as a model from which to develop the system. I have constructed a web interface to

ONPRC's Genome Resource and Informatics Project (GRIP) database system that allows researchers to easily upload SNP genotype data to the database. I have also constructed a web interface allowing researchers to both perform semi-automated haplotype inference analysis on SNP genotype data stored in the GRIP database and to query previously inference-generated haplotypes stored in the database, and, working with the Ferguson lab, developed protocols for use of these tools. These interfaces provide both a highly-functional environment for researchers to store and work with SNP data and a starting point for future development of SNP-based bioinformatics tools as both the data sets and analysis techniques grow richer and more robust in the future.

METHODS

REQUIREMENTS GATHERING AND DEVELOPMENT METHODOLOGY

The user base for this system during the time of development consisted of Dr. Ferguson and her lab assistant; due to this small pool of users, requirements gathering was an informal process. From discussions with these two users and with members of the GRIP database development and maintenance team, expected workflows for upload and analysis processes were elucidated:

Upload Process

1. User produces summary file of individuals and discovered/genotyped SNPs.
2. User uploads summary file to processing system via GRIP web page.
3. System uploads all SNP loci and alleles for polymorphisms in summary.

Haplotype Inference Process

1. User selects loci for desired haplotype.
2. System extracts SNP and pedigree information for genotyped individuals at those loci.
3. System converts data into an appropriate format for inference program.
4. System runs inference program.
5. System back-translates results into database.

6. System displays results to user

Haplotype Query Process

1. User selects individuals and loci for query.
2. System queries database.
3. System displays results to user.

Using these expected workflows, the system interfaces and analysis components were developed. Development followed a rapid prototyping model, in which prototype versions of the system with limited functionality were presented to the users and the GRIP team for evaluation and commentary. After these evaluations, the system was further refined. This process continued until such time as this author, the GRIP team, and the users reached a consensus that the system was sufficiently developed for deployment. As of this writing, the system has not been actively deployed on the GRIP's production database due to a pending redeployment of the production database; however, the system has been implemented on a development server on the GRIP network and can be made available to users. It is anticipated that the system will be fully deployed shortly.

STORAGE OF SNP GENOTYPE DATA

The GRIP database schema is fully described in the Master's thesis of Khouangsathiene (2004); refer to Figure 3 for a visual representation of the schema. The database is based on standard relational database concepts and is currently implemented on a Microsoft SQL Server platform; web interfaces to the database are handled by the Apache Tomcat server system and are primarily implemented with Java Server Pages (JSP) technology. Most of the GRIP web interface has been developed and maintained by Samone Khouangsathiene and Carlo Pearson; the interface provides researchers with a

rich array of options for querying the database and generating reports.

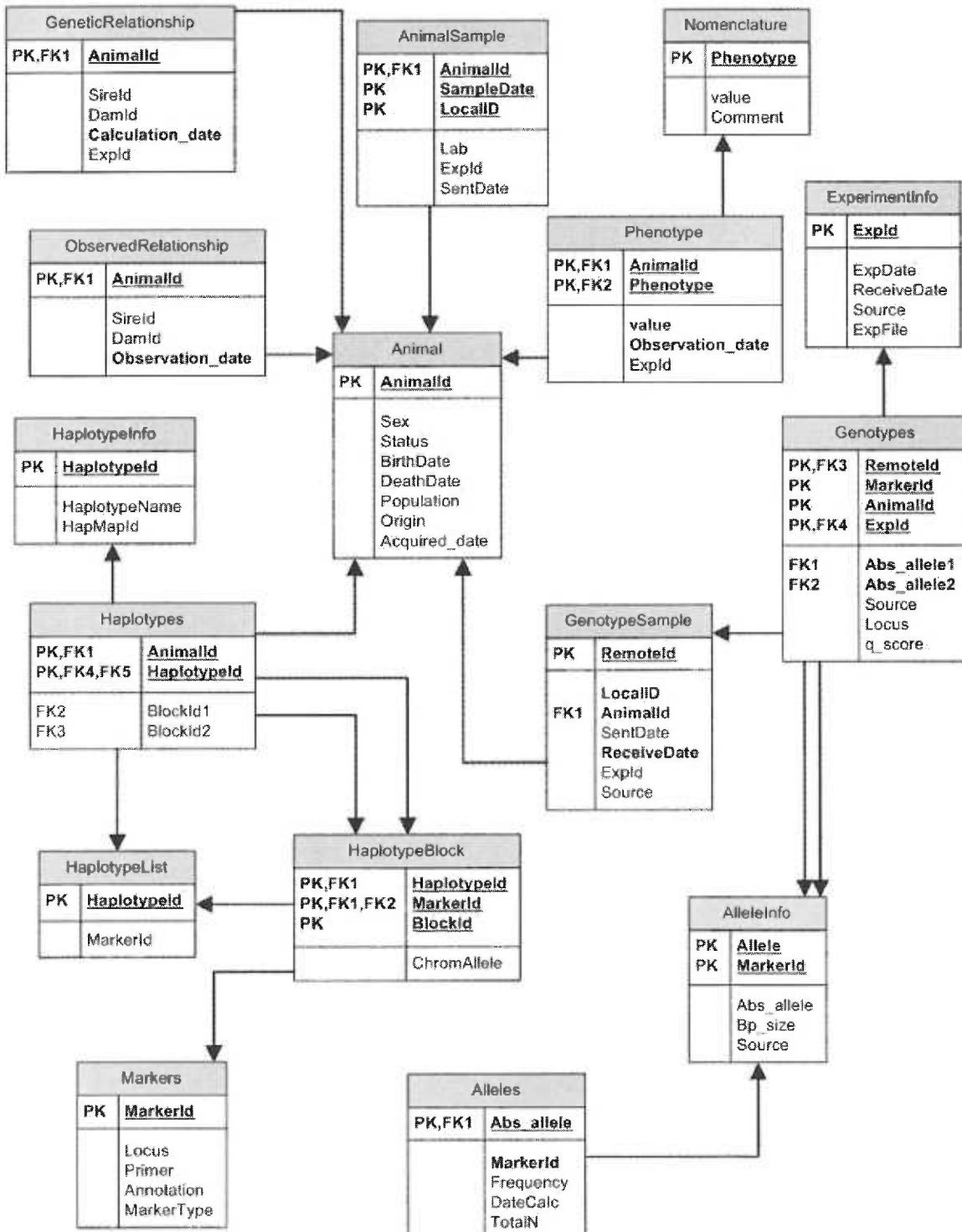


Figure 3. GRIP Database Schema (from Khouangthiene, 2004)

Regarding the schema, the portions of the database most important to the storage

of SNP genotype data are the Genotypes, Markers, AlleleInfo, and Allele tables. SNP genotype data conforms to the storage model for other genotype information. Each individual locus is assigned a Markers entry, containing the name of and information about the marker such as its type (in which field it is indicated that a locus is an SNP) and information about primers for the locus (irrelevant and currently unused for SNP loci). SNP loci are named according to the pattern name: position_flanksequence, where name is an arbitrary name for the segment of genome in which the SNP is located, position is a numeral indicating the distance from the starting base pair of the named genomic segment, and flanksequence represents the 16-base pair sequence surrounding the SNP (8 bases preceding, seven bases following, with the SNP represented by the degenerate base pair symbol representing its two alleles) to allow for unambiguous location of the SNP in the genome. A marker name for an SNP located in a section of genome called “LTA” would appear as, for example, “LTA:168_ACCGTAGAYTAGCCA, where Y is the degenerate codon representing the alleles and position of the SNP. Each allele of a specific locus is assigned associated Allele and AlleleInfo entries. These entries consist of names of specific alleles and information about the allele such as frequency of appearance. SNP alleles are named in a similar fashion to SNP loci, with the pattern name:position_allele, where allele is the specific nucleotide the allele represents: i.e. LTA:168_C. Genotypes of individuals resolved at a specific loci are stored as individual entries in the Genotype table, containing animal IDs, metadata about the typing experiment, and the alleles found. In Genotype entries, the alleles are not resolved to a specific chromosome and are presented in an arbitrary order.

STORAGE OF HAPLOTYPE DATA

In theory, haplotypes consisting of genotype data types other than SNP loci could be constructed; such haplotypes may become desirable to construct at some point in the future and as such the haplotype storage portions of the GRIP schema are not limited to exclusive representation of SNP-based haplotypes. Currently, however, only SNP-based inference-generated haplotypes are contained in the database. The haplotype portion of the schema is analogous in construction to that of the genotype portion described earlier, consisting primarily of the HaplotypeInfo, HaplotypeBlock, and Haplotype tables. The HaplotypeInfo table is similar to the Markers table, containing HaplotypeID, HaplotypeName, and HapMapId fields. For SNP-based haplotypes, the HaplotypeID field consists of the name of the gene segment the haplotype represents, followed by the position portion of the loci involved in the haplotype separated by commas (e.g. LTA:168,227,298). The HaplotypeName is generated with the assumption that haplotypes will contain loci only within the same named region of the genome and consists of the name of the region and an identifying number and the extension “_inf” to indicate that the haplotype is inference-generated; an example HaplotypeName would be LTA_9_inf. Currently, the HapMapID field is unused. HaplotypeBlock entries are essentially similar to Allele entries: the MarkerID field serves to store the specific alleles in the haplotype in a format similar to the HaplotypeID (such as 168_C,227_T,298_A) while the BlockId identifies the block by appending a numeral to the end of the associated HaplotypeID (such as LTA:168,227,298_1). Finally, the Haplotypes table stores the association of a specific haplotype with an individual. The BlockID entries are, as with the genotypes, not schematically associated with a specific chromosome; this is due to cases where inference is performed on individuals whose parentage is unknown,

hence rendering it impossible to definitively determine the parental origin of their haplotypes.

UPLOAD OF SNP DATA

Currently, virtually all SNP discovery and genotyping data produced by the Ferguson lab is visualized using Sequencher (Gene Codes Corporation), a program providing various analytic facilities for gene sequence information. SNP data can be output into a summary file representing typed individuals, the allelic variances of said individuals, and the consensus sequence of the typed region (see Figure 4 for an example of a summary file). These summary files provide the vehicle by which SNP data is uploaded into the database; currently, only upload of these files is supported.

```
Summary view of contig "LTA gene"
>Chn1      #1
>Chn2      #1
>Chn3      #1
>Chn4      #1
>Chn5      #1
>Chn6      #1
>Ind1      #1
>Ind2      #1
>Ind3      #1
>Ind4      #1
>Ind5      #1
>Ind6      #1
#1          TGGAGCCTTC GCTCTGTTAGA ACTTGGAAAA CTCCAGAAAG AAAAAATAAT
+          +
```

Figure 4. Sequencher Summary File

The interface for upload of SNP data consists of an HTML form linked to a JSP designed to process the uploaded files and invoke the upload programs (also written in Java). See figure 3 for an illustration of the form. Fields include the pathname of the summary file, a pathname for a name-masking file (described subsequently), and fields for metadata about the experiment. This form is the entirety of the upload interface.

The screenshot shows a web-based form titled "Upload Files". It contains several input fields and buttons. At the top are two file selection fields: "Sequencher Summary File" and "Name Mask File", each with a "Browse..." button. Below these is a checked checkbox labeled "Use Name Mask File". Further down are three text input fields: "ExperimentID", "RemoteID", and "Source". At the bottom of the form is a large, prominent "Upload Files" button.

Figure 5. SNP Genotype Data Upload Form

Upon submission of the form, the upload-handling JSP (ProcessFileUpload.jsp, see Appendix) utilizes the Apache Commons FileUpload library to transfer the Sequencher summary file and mask file (if one is specified) to the local machine. It then invokes the snploader class's uploadSNP function. (see Appendix for source code). snploader processes the summary file using the parseSNP function, which utilizes a two-pass strategy; first, the consensus sequence is assembled then the SNPs themselves are identified. This function returns a HashMap of HashMaps; the first map uses individuals as keys and keys to the subsequent maps, which key loci to alleles. In addition, a HashMap keyed to "degen" in the initial HashMap keys loci to the degenerate codon symbols for their two alleles. uploadSNP proceeds to create appropriate entries in the database for this data. For each entry, several steps take place. First, if a name-masking has been specified, the name of an individual as uploaded will be replaced by the name in the masking file. These files are simple tab-delimited files of the format:

Original Name	Mask Name
Original Name	Mask Name

and are used in such cases where the Sequencher summary output does not contain the AnimalIDs of the typed individuals; typically, the original name would be masked by the appropriate AnimalID. Allele information is stored as a pair of Strings in an ArrayList object contained in the appropriate individual HashMap, with each String in the format name:position_flanksequence_allele (e.g. LTA:168_ACCGTAGAYTAGCCA_C). This format simplifies extraction of the necessary information to create (as necessary) appropriate Markers, Allele, AlleleInfo, and Genotype entries. Functions to check for the existence of and create these entries if needed are called for the first three of these tables; they are named checkMarker, checkAllele, and CheckInfo. The allele string is passed to the function along with a JDBC database connection and any other required info; it processes it into the appropriate format for the table in question and queries the database using JDBC to see if an entry exists for the data. If not, the function creates an entry. At this point, the uploadSNP function itself parses the allele Strings into the correct SQL to create a Genotype entry for the individual and produces the final Genotype entry. This is repeated for all alleles in the individual's HashMap and all HashMaps representing individuals in the primary HashMap. When this is concluded, ProcessFileUpload.jsp informs the user that the upload has been successful.

INFERENCE OF HAPLOTYPES

There are many haplotype inference programs available to the public; a discussion of the algorithms in general use for haplotype inference can be found in (reference). Most of the approaches used by these programs have been shown to have comparable accuracy for haplotypes consisting of up to 5 linked SNPs (Adkins 2004). With this in mind, my

selection of haplotype inference programs for this project was influenced largely by other factors; in the end, I settled on PedPhase (Li and Jiang 2003) as it seems to be the most stable and best-performing program of those tested in the Windows environment. Other programs, such as HAPLORE (Zhang, Sun, and Zhao, 2005), did not run correctly in the Windows environment. PedPhase's algorithm is closest to the EM-type of inference algorithms discussed previously, though it is highly modified from the original design. One shortcoming of PedPhase as compared to other haplotype inference programs is that it does not output an indication of confidence in its haplotypes. The authors of PedPhase have obtained very accurate results in testing, however, and PedPhase has sufficient advantages in speed, stability, and in the ability to work with data sets missing information, in my experience, that I believe it is suitable for the system.

PedPhase is typical of haplotype inference programs in interface; it is a command-line utility that takes in a tab-delimited file (see figure 6 for an example) containing data regarding the animals for which haplotypes are to be determined, their pedigree information, and their genotypes at each locus in the desired haplotype. The genotype information is abstracted to integer representations of alleles; no specific gene names are used. Readers familiar with linkage analysis programs in general will recognize this as a standard Linkage format for representing pedigree and genotype information; one beneficial side effect of using PedPhase is that any other haplotype inference program utilizing this format could be swapped in without modifying the code to generate the input file. Additionally, such files could be exported in the future if users so desire. The format of an output file from a particular haplotype inference program will probably remain unique to that program, though, and this will require new code to correctly

handle.

1	90218	90215	90216	2	0	0	1	2	1	1
1	90210	0	0	1	0	0	1	1	1	1
1	90215	0	0	1	0	0	1	1	1	1
1	90212	90210	90211	2	0	0	1	1	1	1
1	90213	90210	90211	2	0	0	1	1	1	1
1	90217	90215	90216	1	0	0	1	2	1	1
1	90216	0	0	2	0	0	1	2	1	1
1	90214	90210	90211	1	0	0	1	1	1	1
1	90211	0	0	2	0	0	1	1	2	1

Figure 6. PedPhase Input File

The current implementation of the haplotype inference functions is best referred to as "semi-automated," as the system makes no attempt to automatically infer a haplotype for any gene region. Rather, the system is designed to allow researchers a web-based interface to facilitate inference, storage, and query of haplotypes, using PedPhase as the inference program on the backend.

Researchers interested in generating a haplotype for a specific gene region access a search web page (see figure 7).

Select loci set	<input type="text"/>	<input type="button" value="Submit Query"/>
-----------------	----------------------	---

Figure 7. Gene Region Query Page

This page is simplistic; the researcher need only enter the name of the gene region that he or she is interested in haplotype inference. This form feeds this information to getloci.jsp, a JSP form which queries the GRIP database and generates a form allowing the researcher to select which loci will form the haplotype. These loci are represented in two ways: a dynamically-generated SVG image (see svggen.jsp) which appears at the top of

the page, representing the loci's spatial position along the gene segment of interest, and in an HTML form list. This list is where the selection of loci for haplotype inference is performed. The list is sorted in descending order of the frequency of the less-common (or minor) allele present at each loci; the rationale behind this presentation is that a locus with a higher frequency of its minor allele will be more likely to produce informative haplotypes in combination with other alleles than a locus whose minor allele is fairly rare (though the rarity of that minor allele might be informative in and of itself, as previously discussed). This frequency information is also indicated in a color-coding scheme on the SVG representation of the gene segment. The researcher selects the set of loci in the desired haplotype via checkboxes in the list. Though an unlimited number of boxes may be checked, Adkins has observed that reliability of haplotype inference seems to be best when three to five SNP loci are involved; in accordance with this observation, the inference subsystem only accepts the first five loci checked to as part of a given haplotype. Refer to figure 8, below, for an example of the getloci.jsp page.

Number of rows: 18

Select	Minor Allele Frequency	Locus
<input type="checkbox"/>	1.0	IGF1:542_AAAITAAYTCTGATTGT
<input type="checkbox"/>	1.0	IGF1:365_TATGAGARTTGGGATTA
<input type="checkbox"/>	1.0	IGF1:307_TTACATARAGGCCCAAA
<input type="checkbox"/>	1.0	IGF1:224_ATCTTTIRTCGTGTCT
<input type="checkbox"/>	1.0	IGF1:196_ACTCTGARACCTCAAGC
<input type="checkbox"/>	0.1111111111111111	IGF1:365_TATGAGAATTGGGATTA
<input type="checkbox"/>	0.1111111111111111	IGF1:224_ATCTTTATCTGTGTCT
<input type="checkbox"/>	0.07142857142857142	IGF1:634_AATCATTATCTTACATT
<input type="checkbox"/>	0.07142857142857142	IGF1:286_ATCATGCAGCTTATATT
<input type="checkbox"/>	0.07142857142857142	IGF1:135_CTTAACATCTGATTTGGT
<input type="checkbox"/>	0.06521739130434782	IGF1:762_TTIIGGATAATGTTGG
<input type="checkbox"/>	0.06521739130434782	IGF1:604_CCTAACATGACTTCTAAAA
<input type="checkbox"/>	0.05555555555555555	IGF1:634_AATCATTIRTCCTTACATT
<input type="checkbox"/>	0.05555555555555555	IGF1:542_AAATTAAATTCTGATTGT
<input type="checkbox"/>	0.05555555555555555	IGF1:307_TTACATAAAGGCCCAAA
<input type="checkbox"/>	0.05555555555555555	IGF1:286_ATCATGCRGCCTTATATT
<input type="checkbox"/>	0.05555555555555555	IGF1:196_ACTCTGAAACCTCAAGC
<input type="checkbox"/>	0.05555555555555555	IGF1:135_CTTAACATCYGATTTGGT

Figure 8. Loci Selection Form

When the researcher has selected the desired set of loci for haplotype inference, getloci.jsp is submitted to genuserhaplo.jsp. This JSP file is responsible for initiating calculation of the desired haplotype and displaying the results, using the functions in the haplotyper class (see Appendix for code details). The set of loci is first submitted to the haplotyper.defineUserHaplotype function, which determines if an appropriate HaplotypeInfo entry exists for the specified loci; if one does not exist, the function creates an appropriate entry in the database. The retrieved (or newly generated) HaplotypeId is returned to genuserhaplo.jsp, which then uses it to call the haplotyper.calculateHaplotypes function. This function begins operation by retrieving all

genotype entries for all animals that have been typed at the loci identified in the calling HaplotypeId (recall that the HaplotypeId exists as a list of comma-separated loci positions). These Genotype entries are stored; then, if possible, pedigree data about typed individuals is retrieved and stored from the GeneticRelationship table of the GRIP database, utilizing functions written by this author for a pedigree viewing system on the GRIP website. The combined genotype and pedigree data are then output to a file formatted for PedPhase analysis. As discussed previously, this translation strips the genetic information of locus and allele information; this information is stored by the program to allow reconstruction from the PedPhase output file. The function triggers PedPhase and parses the resultant output file to generate Haplotype entries for the typed individuals in the database. This portion of the function operates in a similar fashion to the database update portion of the snploader.uploadSNPs function; for each individual entry, the function calls the haplotyper.checkHaploBlock function to locate or create an appropriate BlockId for the generated haplotypes, then creates a Haplotype entry for the individual and its HaplotypeBlocks. The JSP terminates in a state similar to that illustrated in Figure 9, retrieving and displaying the results of the calculation.

Number of rows = 90					
AnimalID	HaplotypeID	BlockId1	BlockId2	Block1Alleles	Block2Alleles
906985	Chr10:67224,67934,73893	Chr10:67224,67934,73893_2	Chr10:67224,67934,73893_2	Chr10:67224_T,67934_A,73893_G	Chr10:67224_T,67934_A,73893_G
906991	Chr10:67224,67934,73893	Chr10:67224,67934,73893_2	Chr10:67224,67934,73893_2	Chr10:67224_T,67934_A,73893_G	Chr10:67224_T,67934_A,73893_G
906993	Chr10:67224,67934,73893	Chr10:67224,67934,73893_2	Chr10:67224,67934,73893_1	Chr10:67224_T,67934_A,73893_G	Chr10:67224_T,67934_C,73893_G
906994	Chr10:67224,67934,73893	Chr10:67224,67934,73893_9	Chr10:67224,67934,73893_10	Chr10:67224_C,67934_A,73893_G	Chr10:67224_T,67934_A,73893_T
907000	Chr10:67224,67934,73893	Chr10:67224,67934,73893_9	Chr10:67224,67934,73893_6	Chr10:67224_C,67934_A,73893_G	Chr10:67224_T,67934_C,73893_T
907019	Chr10:67224,67934,73893	Chr10:67224,67934,73893_5	Chr10:67224,67934,73893_2	Chr10:67224_C,67934_A,73893_T	Chr10:67224_T,67934_A,73893_G
907022	Chr10:67224,67934,73893	Chr10:67224,67934,73893_5	Chr10:67224,67934,73893_1	Chr10:67224_C,67934_A,73893_T	Chr10:67224_T,67934_C,73893_G
907029	Chr10:67224,67934,73893	Chr10:67224,67934,73893_10	Chr10:67224,67934,73893_9	Chr10:67224_T,67934_A,73893_T	Chr10:67224_C,67934_A,73893_G
907034	Chr10:67224,67934,73893	Chr10:67224,67934,73893_1	Chr10:67224,67934,73893_1	Chr10:67224_T,67934_C,73893_G	Chr10:67224_T,67934_C,73893_G
907048	Chr10:67224,67934,73893	Chr10:67224,67934,73893_1	Chr10:67224,67934,73893_5	Chr10:67224_T,67934_C,73893_G	Chr10:67224_C,67934_A,73893_T
907055	Chr10:67224,67934,73893	Chr10:67224,67934,73893_5	Chr10:67224,67934,73893_5	Chr10:67224_C,67934_A,73893_T	Chr10:67224_C,67934_A,73893_T
907056	Chr10:67224,67934,73893	Chr10:67224,67934,73893_2	Chr10:67224,67934,73893_2	Chr10:67224_T,67934_A,73893_G	Chr10:67224_T,67934_A,73893_G
907345	Chr10:67224,67934,73893	Chr10:67224,67934,73893_7	Chr10:67224,67934,73893_8	Chr10:67224_N,67934_A,73893_G	Chr10:67224_N,67934_A,73893_T

Figure 9. Haplotype Generation/Query Output

QUERY OF HAPLOTYPES

The GRIP web interface already provides a robust query solution to researchers regarding Genotype data; as SNP entries are stored in the database's standard Genotype format, no particular modifications to the Genotype query design is required to accommodate them. However, no preexisting query system is in place to deal with Haplotype entries; thus, such a system has been developed as part of this project. The search form interface to this system is pictured in Figure 10. Researchers may specify several parameters in their search. A researcher may specify a list of animals to retrieve records for, or leave this field blank to specify all animals; a list of loci that retrieved haplotypes should contain may be specified as well. This latter field may also be left blank to specify all loci. A checkbox associated with this field allows the researcher to specify whether he or she wishes to search for any haplotype containing at least one of the specified loci or only haplotypes containing all of the specified loci. Results are displayed as in Figure 9, ordered by AnimalId.

Enter animal id list (leave blank to select all animals)

Enter SNP list for haplotypes (format is lociposition (i.e., LTA:111) or enter prefix to select all haplotypes with prefix, leave blank to select all haplotypes)

Check to search for haplotypes containing all specified SNPs only (logical AND)

Figure 10. Haplotype Query Interface

RESULTS

While the small number of users and short development cycle prevented any

large-scale usability testing, technical validation of the system was quite feasible. The Ferguson lab has produced a reasonable amount of SNP genotype data for the animals in the study, and in testing with this data the SNP loader has performed well. The system accurately translates and uploads data from Sequencher files rapidly; a typical file from the pilot study encompassing ~900 nucleotides and 9 individuals uploads within 30 seconds. The current state of the Ferguson study precludes a rigorous testing of the haplotype inference system in actual service, however; the haplotype inference portion of the system is difficult to test without data with accurate pedigree information, and most of the animals included in the Ferguson study have partially known ancestry at best. However, validation of the haplotype inference system has been performed using SNP genotype data obtained from the human HapMap data. The North American population used in the HapMap project is part of the CEPH families (www.cephb.fr), families whose members have been extensively genotyped and, for many of whom, extensive pedigrees have been made available. The availability of accurate pedigrees makes this population an excellent test data set for the haplotype inference capabilities of the system.

SNP genotype data for chromosome 10 and pedigree data for the North American population of the HapMap project was obtained from the HapMap website; this information was part of Public Release #16, released 03/01/2005. A subset of the chromosome 10 SNP loci encompassing all loci within the first 100KB of the chromosome was selected; the corresponding genotypes for all North American individuals were uploaded to a testing instance of the GRIP database. The HapMap Project utilizes a program called Haplovview (Barrett et al, 2005) as its primary data visualization solution; it generates haplotypes internally using an expectation-

maximization algorithm from automatically selected SNP loci. This program was given the same set of loci and genotypes from chromosome 10, and its default haplotype inferences from that data were generated as a comparison data set. The web interface was then used to generate haplotypes from the selected loci; results of one of this test and the original data set are reproduced in the Appendix. Also, the Linkage file produced by the web interface was input into Haplovview to verify its accuracy.

In several ways, these tests differed from the expected operation of the system in use, but these differences should not impact the relevance of the tests. First, a custom uploader was written to accommodate the HapMap data format, but since the Sequencher format SNP genotype loader has been tested with live data its operation was not relevant to this test. Second, since some of the alleles selected for upload showed no allelic variation in the North American population, minor allele frequencies of 0.0 appeared in the loci selection table during haplotype inference. This would not be expected to occur during actual use of the system, and none of these loci were involved in the inference tests. Finally, unique to this test is the format of the loci and themselves: only name, position, and degenerate codon at the locus are indicated. The HapMap data does not include the flanking sequence about the alleles, and the flanking sequence is irrelevant to this test; this variation in formatting does not affect the system's operation in any meaningful way.

Comparisons of the test results to the expected haplotypes for parent-child sets revealed some differences in the generated haplotypes and their frequencies. Haplovview identified 4 haplotype blocks from the three loci selected both from the original HapMap data and from the Linkage-format data, while PedPhase identified ten distinct haplotype

blocks. Since Haplovie generated essentially identical results from both the HapMap and Linkage-format files, it is safe to conclude that the different results between the two programs are due to the divergent algorithmic implementations in the inference programs themselves. It should be noted as well that the differences stem partially from the nature of the data set itself; in one family data was unavailable at two loci for two individuals, under which circumstances PedPhase identifies incomplete haplotypes but Haplovie discards the data. In the one completely genotyped family where PedPhase identified haplotypes that Haplovie did not, the potential haplotypes are ambiguous, and either of the programs could be correct. None of the families in the HapMap data set are pedigreed and typed to three generations, which is generally necessary for completely unambiguous phase discrimination, and it is probable that the two programs would have higher agreement on a three-generation deep or better pedigree for the now ambiguous family. Even taking these discrepancies into account, the two systems agreed on 90% of the cases. From the standpoint of interface, Linkage file generation, and database upload functionality, results from the haplotype inference test were indicative of full functionality; the system successfully invoked PedPhase and loaded the generated haplotypes into the database on several iterations of the test. Additionally, the fact that Haplovie was capable of using the Linkage-format files generated by the interface confirms the validity of those files. The system did perform somewhat slowly during identification of candidate loci, taking on the order of 1-2 minutes to generate the webpage for candidate loci; this is due to the fact that allele frequencies are calculated in real time by the test system. The system should perform faster on the GRIP production server, which pre-calculates allele frequencies in the background.

FUTURE DIRECTIONS

Though the upload and analysis system described in this thesis is quite robust and functional, in some ways it is still very much at the proof of concept stage; few users have been exposed to its functionality and there are probably many improvements and additions that could be made to the interface and analysis of SNP and haplotype data. Some of these modifications will be occasioned by the needs of users as the system becomes more widely used; others will come about due to the availability of new tools and new data. Here, I will outline several of the modifications I anticipate the system may incorporate in the future.

dbSNP COMPATIBILITY

NCBI maintains a database of SNP loci known as dbSNP (see www.ncbi.nlm.nih.gov/SNP). In this database, SNP loci are assigned unique numerical IDs and are unambiguously located in the genome of the organism. However, unambiguous location of SNPs in the rhesus genome was not possible at the beginning of the project due to the lack of a published sequence of the genome (though such location would certainly be conceivable for some well-characterized regions of the genome). Recently, however, a draft sequence of the genome has been made available (reference). For several reasons, including the desire for inter-institution data sharing and NIH publishing requirements, it will be necessary to begin assembling information about discovered SNPs into dbSNP-compatible format for upload and possibly to convert the storage format of local SNP data into dbSNP format.

NIH has defined requirements for upload to dbSNP at
http://www.ncbi.nlm.nih.gov/projects/SNP/get_html.cgi?whichHtml=how_to_submit.

Data required about an SNP locus includes such information as its observed alleles, position, allele frequency, the experimental protocol leading to identification, etc.; in general, most of these requirements are met by the current data on SNPs contained in the database, with the exception of some experimental protocol information and the aforementioned unambiguous location of SNPs. It would not be difficult to generate and automated report from the collected SNP data in the format required by NCBI to help streamline upload to dbSNP. Unambiguous location of SNPs would be facilitated by search of the flanking sequence against the draft genome either at a remote site or local to the GRIP database (discussions about installing a local copy of the draft sequence are already, in fact, underway at GRIP), while experimental information would be provided by the entity requesting the report.

Once NIH assigns dbSNP IDs to uploaded SNPs, it may become protocol to assign these IDs in addition to or in lieu of the ad hoc gene segment-based IDs described in the Methods section. While this is not difficult to do from a technical perspective, the replacement may require some redesign of the system, particularly in the haplotype inference portion, to accommodate. In particular, physical proximity of SNPs in currently indicated by the common ad hoc gene name prefix and searching for SNP candidates for haplotype inference is based on this prefix name, which would not be present using the numerical IDs. Hence, the search for candidate SNPs for haplotype inference would have to be predicated on absolute genetic location; storage provisions would need to be made for this information, possibly necessitating modification of the GRIP schema. It may be preferable to continue to store the "original" identifier for an SNP as well as its dbSNP identification and chromosome location; however, the ability to specify arbitrary

stretches of chromosome rather than the specific ad hoc segments for candidate SNP identification considerably increases the flexibility of the haplotype inference system.

TAG SNP SELECTION ASSISTANCE

Though the system does provide a primitive form of tag selection assistance in the form of indicating the minor allele frequency at a particular locus, it is possible to perform a more comprehensive assessment of a particular locus's suitability as a tag SNP. Goldstein et al. (2003) provide a survey of the techniques by which tag SNPs are selected, and the human MHC mapping of Miretti et al. provides an excellent recent example of the techniques. In general, such techniques revolve around estimations of linkage disequilibrium between the putative tag and the other loci it is expected to represent in a haplotype, and there are software packages that can be used to estimate this quantity for purposes of tag SNP identification, such as the GOLD package used by Miretti et al. (Abecasis and Cookson, 2000).

The current version of the system does not emphasize tag selection, partially since the data sets currently available from the Ferguson study are maps of fairly fine resolution from a small pool of animals, and so tag selection is not a priority. However, if tag determination becomes a desired capacity in the future, it would be possible to add it to the system; for example, it might be possible to calculate and then integrate indications of LD statistics for particular loci into the loci selection screen of the haplotype inference portion of the system, or to integrate a search capacity for candidate loci among existing haplotypes.

EXPANSION OF HAPLOTYPe INFERENCE CAPABILITY

The haplotype inference portion of this thesis, as currently implemented, only has the capacity to work with SNP-based haplotypes and with only those SNPs in a certain arbitrarily-defined gene segment as identified in the gene name portion of the SNP identifier. However, haplotype inference of genetic elements other than SNP loci may prove to be useful to users and should be accommodated in future versions of the system. Specifically, two other types of haplotype inference are likely to be of interest in the near future; haplotype inference based on alternate marker varieties and haplotype inference based on smaller haplotypes.

Of the two, alternate-marker haplotype inference is the most straightforward and probably the easiest to accommodate. Since Genotype entries of any experimental derivation are represented with the same database structure and are reduced to integer representation during haplotype inference, there is no technical reason why any set of Genotype entries cannot be specified for haplotype inference. Rather, interface issues are the primary difficulty anticipated in implementing this functionality. Unlike SNPs restricted by gene segment or by range of chromosomal location, genotypes reliant on such techniques as microsatellites are not guaranteed to be in any particular physical proximity and hence linkage of the loci may be suspect, rendering haplotypes potentially useless. Though a location identifier could be added to the database, under the current schema the user would need to be aware of this issue and responsible for considering its impact. This also means that there is no easy way to search for candidate loci, and that the user will have to determine appropriate loci for such haplotypes on their own. However, such haplotypes may be of considerable interest to researchers at ONPRC in the near future; several researchers at the center have research interests involving the

heavily polymorphic major histocompatibility complex of the rhesus. A recent publication by Ottig et al. (2005) has identified a startling complexity of variation in the MHC of the rhesus. A significant amount of the variance appears to emerge from unequal crossover events in the MHC, however, and the presence or absence of entire loci within the complex. Haplotype inference could be used in this case to help identify loci that associate together during crossover events.

Haplotype generation based on smaller, previously-generated haplotypes is marginally more complex to develop, but retains many of the advantages of single-SNP haplotype inference as compared to alternate-marker haplotype inference. The underlying assumption in this method of haplotype inference is that a smaller haplotype block consisting of SNP loci, if these loci are highly linked, can effectively be treated as a single locus and further haplotype inference with other haplotype blocks can be performed; this provides an alternative to tag SNP selection in such cases where tag SNP loci have not been resolved for a region. As with alternate-marker haplotype inference, implementation issues can be expected to revolve more around interface than mechanics; however, unlike alternate-marker loci, haplotype blocks have implicit positional information encoded within them (either within ad hoc gene name in locus names or in positional information about the dbSNP identifier, if it has been assigned). This makes possible searches based upon genetic location and so helps users select haplotype blocks that are more likely to be linked. There are certain technical issues that may need to be addressed with using Haplotype rather than Genotype entries as part of HaplotypeBlocks, but in the overall this should not be a difficult capacity to build into the haplotype inference system.

EXPANSION OF INTERFACE CAPABILITIES

It is a general tenet of interface design that the presentation of information can make a significant difference in the utility of that information to a user; a visualization method suitable for some types of data may make other kinds of information entirely incomprehensible. The current interface to both SNP and haplotype data utilizes a textual, table-based approach with minimal use of graphical components; while this is an entirely usable interface, other types of interface may increase the utility or ease of use of the system.

A specific interface option that is likely to be integrated with the SNP and haplotype inference system in the near future is the GRIP's pedigree viewing system. This system is an SVG-based pedigree-drawing system developed by this author and other GRIP employees in my capacity as a research assistant at ONPRC; as previously discussed, certain of its functions are utilized as part of the pedigree determination step during haplotype inference. Being able to visualize SNP genotype information or haplotype information alongside the pedigree would be of obvious utility. The pedigree viewer already has the capacity to display genotypic information and as such could easily display SNP allele information with no modification. The pedigree viewer, however, does not display haplotypes, nor can haplotype inference for a specific lineage be requested from it. Both of these features would add immensely to the utility of the pedigree viewer.

Display of haplotype information will utilize a glyph-like format; examples of this type of display can be seen in various existing commercial products. Alleles can be displayed on the glyph and color coded to make identification of haplotypes between individuals easy to see. When combined with either known or displayed phenotypic

information about the individuals, inheritance patterns that may correlate with haplotypes become easy to see. Similarly, if a displayed pedigree contains animals with a known phenotypic outcome, the user might wish to request a haplotype inference of SNP or alternate-marker loci that he or she suspects may be associated with the phenotype. Integrating this ability into the pedigree viewer would not be unduly difficult since the SVG specification supports hyperlinking; the viewer could easily call the JSP required for haplotype inference and regenerate the view of the pedigree afterwards.

INTEGRATION OF OTHER EXPERIMENTAL DATA

Ultimately, the goal of any haplotype determination project is to determine association of identified haplotypes with particular genetic or phenotypic states in an organism or population. In a sense, the discussion of integrating haplotype data into the GRIP pedigree viewer above is merely a special case of this. However, there are a number of sources of data that are potentially valuable for integration with haplotype information. Sources that may be of particular interest in the near future include phenotypic data about individuals and data from polymorphism identification methods that do not identify specific alleles (such as conformational analysis). In the longer term, combination of SNP data with promoter region information and wide-scale analysis of multiple haplotype blocks may become useful.

Integration of haplotype data with phenotypic data is arguably the most fundamental data integration step, and is, from a technical standpoint, fairly straightforward. The GRIP database schema does provide a Phenotype table for recording individual phenotypes, and as such no particular database modifications would need to be made; most of the programming work would reside in providing interfaces to the data

(such as described in the discussion of the pedigree viewer) and statistical analysis capabilities. However, the Phenotype table in the database is currently unpopulated, and while the GRIP is working towards getting researchers to deposit phenotypic data in the database, it appears that phenotype data collection is going to be a long-term project. Much as with the haplotype inference facility itself, this is very much a solution waiting for more data to make it useful.

Association of sequenced SNPs with alleles identified by other methods, however, has the potential to be a topic of intense near-term interest at ONPRC. Again considering the Ottig et al. paper, association of MHC sequences with SNP haplotypes could provide confirmation of and further detail about the loci and alleles in the MHC, as well as potentially providing a method of genotyping in the region if the SNP haplotypes of specific insertions can be ascertained.

Using SNP and haplotype information in association with promoter region data and in large-scale genomic analysis are steps towards a systems biology approach to understanding such things as gene control networks. Some types of genetic features in promoter regions (notably, transcription factor binding sites) tend to be highly degenerate and hence highly polymorphic, suggesting that variations in promoter regions should be observable as haplotypes; indeed, Liu et al. (2005) have identified several promoter region haplotypes for a human MHC (or HLA) gene region that vary within different Chinese ethnic subtypes, and recently Mototani et al. (2005) have described SNP alleles in the promoter region for a calmodulin gene that appear to associate with osteoarthritis in Japanese. Discovery and association of haplotypes in noncoding genetic features with variations in coding regions, phenotypic outcomes and/or with known promoter or

transcription factor binding motifs might help elucidate the contribution of promoter region variation to gene control networks and the complex phenotypic outcomes deriving from them. Similarly, association of multiple haplotypes in association with a specific complex phenotypic outcome might suggest the involvement of these haplotypes in the phenotypic outcome. Such analysis would be particularly valuable in the short term in the previously discussed identification of alleles in the MHC complex, where the high variability of the gene implies that a single allele of the entire gene might span several haplotype blocks, and in the longer term in the case of a multigene network of interactions leading to complex outcomes. As a final example utilizing the MHC complex, one can envision that bringing together both promoter region haplotypes and allelic haplotypes in the analysis of an MHC gene region would provide a very rich source of data for correlating such variations with phenotypic outcomes and deconvoluting the influence of promoter region variation vs. allelic variation.

CONCLUSION

This thesis describes a solution for the upload of single nucleotide polymorphism genotype data to ONPRC's GRIP database and subsequent inference of haplotypes from that data. Though this system is in the infancy of its implementation, it is a fully functional solution providing an easy interface for database upload and haplotype inference capabilities and for subsequent query of uploaded or generated information. Integration of upload and haplotype inference capabilities into a single web-based solution appears to be a novel concept in this author's experience; even the HapMap webpage, at the time of this writing, requires the local use of a separate haplotype inference program for detailed analysis of the HapMap data. The system is extensible in

many areas with minimal modification and adaptable to ONPRC's growing needs regarding SNP genotype data and to accommodate future forms of analysis, and will become an important part of the GRIP's informatics support mission.

REFERENCES

- Di X, Matsuzaki H, Webster TA, Hubbell E, Liu G, Dong S, Bartell D, Huang J, Chiles R, Yang G, Shen MM, Kulp D, Kennedy GC, Mei R, Jones KW, Cawley S. (2005). Dynamic model based algorithms for screening and genotyping over 100K SNPs on oligonucleotide microarrays. *Bioinformatics*, 21(9):1958-63.
- Crawford DC, Nickerson DA, (2005) Definition and clinical importance of haplotypes. *Annu Rev Med*, 56, 303-20.
- International HapMap Consortium, (2003). The International HapMap Project. *Nature*, 426(6968), 789-96.
- Liu PY, Zhang YY, Lu Y, Long JR, Shen H, Zhao LJ, Xu FH, Xiao P, Xiong DH, Liu YJ, Recker RR, Deng HW, (2005). A survey of haplotype variants at several disease candidate genes: the importance of rare variants for complex diseases. *J Med Genet*, 42(3), 221-7.
- Miretti MM, Walsh EC, Ke X, Delgado M, Griffiths M, Hunt S, Morrison J, Whittaker P, Lander ES, Cardon LR, Bentley DR, Rioux JD, Beck S, Deloukas Pv, (2005). A high-resolution linkage-disequilibrium map of the human major histocompatibility complex and first generation of tag single-nucleotide polymorphisms. *Am J Hum Genet*, 76(4), 634-46.
- Croes EA, Alizadeh BZ, Bertoli-Avella AM, Rademaker T, Vergeer-Drop J, Dermaut B, Houwing-Duistermaat JJ, Wientjens DP, Hofman A, Van Broeckhoven C, van Duijn CM. Polymorphisms in the prion protein gene and in the doppel gene increase susceptibility for Creutzfeldt-Jakob disease. *Eur J Hum Genet*, 12(5), 389-94.
- Adkins RM (2004). Comparison of the accuracy of methods of computational haplotype inference using a large empirical dataset. *BMC Genetics*, 5(1), 22
- Excoffier L, Slatkin M. Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population. *Mol Biol Evol*, 12(5), 921-7
- Xu H, Wu X, Spitz MR, Shete S, (2004). Comparison of haplotype inference methods using genotypic data from unrelated individuals. *Hum Hered*, 58(2), 63-8.
- Pont-Kingdon G, Jama M, Miller C, Millson A, Lyon E. (2004) Long-range (17.7 kb)

allele-specific polymerase chain reaction method for direct haplotyping of R117H and IVS-8 mutations of the cystic fibrosis transmembrane regulator gene. *Mol Diagn*, 6(3), 264-70.

Khouangtsathiene, C. (2004). Master's Thesis. Oregon Health and Science University.

Li J, Jiang T (2003). Efficient inference of haplotypes from genotypes on a pedigree. *J Bioinform Comput Biol*, 1(1), 41-69.

Zhang K, Sun F, Zhao H. (2005). HAPLORE: a program for haplotype reconstruction in general pedigrees without recombination. *Bioinformatics*, 21(1), 90-103.

Barrett JC, Fry B, Maller J, Daly MJ. (2005) Haploview: analysis and visualization of LD and haplotype maps. *Bioinformatics*, 21(2), 263-5.

Abecasis GR, Cookson WOC (2000). GOLD—graphical overview of linkage disequilibrium. *Bioinformatics*, 16:182–183.

Otting N, Heijmans CM, Noort RC, de Groot NG, Doxiadis GG, van Rood JJ, Watkins DI, Bontrop RE. (2005) Unparalleled complexity of the MHC class I region in rhesus macaques. *Proc Natl Acad Sci U S A*. 102(5), 1626-31.

Liu X, Xu Y, Shen Y, Zhang H, Fu Y, Liu Z, Xu A (2005). HLA-DPA1 promoter haplotypes are differently distributed in southern Chinese ethnic groups. *Tissue Antigens*, 65(2), 172-7.

Mototani H, Mabuchi A, Saito S, Fujioka M, Iida A, Takatori Y, Kotani A, Kubo T, Nakamura K, Sekine A, Murakami Y, Tsunoda T, Notoya K, Nakamura Y, Ikegawa S. (2005) A functional single nucleotide polymorphism in the core promoter region of CALM1 is associated with hip osteoarthritis in Japanese. *Hum Mol Genet*, 14(8), 1009-17

APPENDIX 1: SOURCE CODE

snploader.java

```

//Sequencher SNP loader for ONPRC
//By Hollis Wright
//Release Version 1.0
//April 14, 2005
package snploader;
import java.io.*;
import java.util.*;
import java.sql.*;

public class snploader
{
    public static HashMap parseSNPs(String file, String maskfile, boolean mask) throws Exception
    {
        System.out.println("I am refactored!");
        HashMap individuals = new HashMap();
        Map temp = new HashMap();
        HashMap maskmap = null;
        BufferedReader buf = null;
        String line = null;
        if (mask == true)//using mask file or no?
        {
            maskmap = new HashMap();
            buf = new BufferedReader(new FileReader(maskfile));
            line = buf.readLine();
            while(line != null)
            {
                StringTokenizer str = new StringTokenizer(line);
                maskmap.put(str.nextToken(), str.nextToken());
                line = buf.readLine();
            }
        }
        buf = new BufferedReader(new FileReader(file));
        line = buf.readLine();
        String genename = null;
        int pos = 0;
        String refseq = new String();
        TreeMap polys = new TreeMap();
        int genpos = 0;
        HashMap degen = new HashMap();
        individuals.put("degen", degen);
        ArrayList indivarr;
        while(line != null)
        {
            if (line.indexOf("Summary") != -1) //gets gene name
            {
                genename = line.substring (line.indexOf("\t") + 1, line.lastIndexOf("\t"));
            }
            else if (line.indexOf(">") == -1 && line.indexOf("<") == -1 && line.indexOf("#") != -1)// parses
consensus sequence
            {
                int counter = line.lastIndexOf("#");
                counter = line.indexOf(" ", counter);
                char testchar = line.charAt(counter);
                while (testchar == ' ')
                {
                    ++counter;
                    testchar = line.charAt(counter);
                }
                refseq = refseq + line.substring(counter, line.indexOf(" ", counter + 1));
                counter = line.indexOf(" ", counter + 1);
                while (counter != -1)
                {
                    System.out.println(refseq);
                    if (line.indexOf(" ", counter + 1) != -1)
                    {

```

```

        refseq = refseq + line.substring(counter + 1, line.indexOf(" ",
counter + 1));
    }
    else
    {
        rcfseq = refseq + line.substring(counter + 1);
        counter = -1;
    }
}
line = buf.readLine();
}
System.out.println(refseq);
buf = new BufferedReader(new FileReader(file));
line = buf.readLine();
int abspos = 0;
while(line != null)//parses polymorphisms, gets alleles
{
    if (line.indexOf("Summary") != -1)
    {
        genename = line.substring (line.indexOf("\n") + 1, line.lastIndexOf("\n"));
    }
    else if (line.indexOf(">") != -1 || line.indexOf("<") != -1)
    {
        String individual = null;
        System.out.println(line);
        if(line.indexOf(">") != -1)
        {
            individual = line.substring(line.indexOf(">") + 1, line.indexOf(" "));
        }
        else
        {
            individual = line.substring(line.indexOf("<") + 1, line.indexOf(" "));
        }
        System.out.println(individual);
        if(mask == true)
        {
            individual = (String) maskmap.get(individual);
        }
        System.out.println(individual);
        HashMap arr = (HashMap) ttmp.get(individual);
        int temppos = 0;
        if(arr == null)
        {
            arr = new HashMap();
        }
        pos = linc.indexOf("#");
        try
        {
            temppos = Integer.parseInt(line.substring(pos + 1, line.indexOf(" ", pos)));
        }
        catch (Exception ex)
        {
            try
            {
                temppos = Integer.parseInt(line.substring(pos + 1, line.indexOf(">", pos)));
            }
            catch (Exception ex2)
            {
                try
                {
                    temppos = Integer.parseInt(linc.substring(pos + 1,
line.indexOf("<", pos)));
                }
                catch(Exception ex3)
                {
                    pos = -1;
                }
            }
        }
    }
}

```

```

        }
    }
    pos = line.indexOf(" ", pos);
    char tempchar = ' ';
    if (temppos < abspos)
    {
        System.out.println(pos);
        pos = line.length() - 1;
        tempchar = line.charAt(pos);
        int killflag = 0;
        genpos = abspos + 50;
        while (killflag != 3)
        {
            System.out.println(tempchar);
            if(tempchar == ' ' && killflag == 0)
            {
                --pos;
                --genpos;
            }
            else if(tempchar == ' ' && killflag == 1)
            {
                --pos;
                killflag = 2;
            }
            else if (tempchar == ' ' && killflag == 2)
            {
                killflag = 3;
                pos = pos + 2;
                System.out.println("Adjust");
            }
            else
            {
                --pos;
                --genpos;
                killflag = 1;
            }
            tempchar = line.charAt(pos);
        }
    }
    else
    {
        genpos = temppos;
        abspos = temppos;
    }
    if (pos < line.length() && pos != -1)
    {
        tempchar = line.charAt(pos);
    }
    else
    {
        pos = line.length();
    }
    System.out.println("Genpos = " + genpos + " Abspso = " + abspos);
    while (pos < line.length())
    {
        if(tempchar == ' ')
        {
            ++pos;
        }
        else if(tempchar == '-' || tempchar == '.')
        {
            ++pos;
            ++genpos;
        }
        else
        {
            ++pos;
            polys.put(new Integer (genpos), " ");
            arr.put(new Integer(genpos), new Character(tempchar));
            System.out.println(individual + " " + genpos + " " + tempchar);
        }
    }
}

```

```

        ++genpos;
    }
    if(pos < line.length())
    {
        tempchar = line.charAt(pos);
    }
}
temp.put(individual, arr);
}
else if (line.indexOf(">") == -1 && line.indexOf("<") == -1 && line.indexOf("#") != -1)
{
    System.out.println("terminate");
}
line = buf.readLine();
}
pos = 0;
int flankstart = 0;
int flankend = 0;
String flankseq = null;
System.out.println(polys.keySet());
Iterator iter = polys.keySet().iterator();
while(iter.hasNext())
{
    pos = ((Integer) iter.next()).intValue();
    System.out.println(pos + refseq.length());
    Iterator iter2 = temp.keySet().iterator();
    while (iter2.hasNext())// gets the polymorphism hash for each individual at each locus
    {
        String thisind = (String) iter2.next();
        HashMap temp2 = (HashMap) temp.get(thisind);
        Character thischar = (Character) temp2.get(new Integer(pos));
        //Following segment produces entries for each allele for an individual in an ArrayList
        //Format is genename:position_flanksequenceXflanksequence_allele, like
LTA:108_AGCTGGAAAXTGGXAGA_A
        if(thischar != null)
        {

            System.out.println(thischar.charValue());
            if (thischar.charValue() == 'M')
            {
                flankstart = pos - 8;
                flankend = pos + 9;
                if (flankstart < 0)
                {
                    flankstart = 0;
                }
                if (flankend > refseq.length())
                {
                    flankend = refseq.length();
                }
                flankseq = genename + ":" + pos + " " +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + "_A";
                indivarr = (ArrayList) individuals.get(thisind);
                if (indivarr == null)
                {
                    indivarr = new ArrayList();
                }
                indivarr.add(flankseq);
                flankscq = genename + ":" + pos + " " +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + "_C";
                indivarr.add(flankscq);
                degen = (HashMap) individuals.get("degen");
                degen.put(genename + ":" + pos, new Character('M'));
                individuals.put("degen", degen);
                individuals.put(thisind, indivarr);
            }
            else if (thischar.charValue() == 'R')
            {
                flankstart = pos - 8;
                flankend = pos + 9;
            }
        }
    }
}

```

```

        if (flankstart < 0)
        {
            flankstart = 0;
        }
        if (flankend > refseq.length())
        {
            flankend = refseq.length();
        }
        flankseq = genename + ":" + pos + "_" +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + "_A";
        indivarr = (ArrayList) individuals.get(thisind);
        if (indivarr == null)
        {
            indivarr = new ArrayList();
        }
        indivarr.add(flankseq);
        flankseq = genename + ":" + pos + "_" +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + "_G";
        indivarr.add(flankseq);
        degen = (HashMap) individuals.get("degen");
        degen.put(genename + ":" + pos, new Character('R'));
        individuals.put("degen", degen);
        individuals.put(thisind, indivarr);
    }
    else if (thischar.charValue() == 'W')
    {
        flankstart = pos - 8;
        flankend = pos + 9;
        if (flankstart < 0)
        {
            flankstart = 0;
        }
        if (flankend > refseq.length())
        {
            flankend = refseq.length();
        }
        flankseq = genename + ":" + pos + "_" +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + "_A";
        indivarr = (ArrayList) individuals.get(thisind);
        if (indivarr == null)
        {
            indivarr = new ArrayList();
        }
        indivarr.add(flankseq);
        flankseq = genename + ":" + pos + "_" +
refseq.substring(flankstart, pos - 1) + "X" + rcfseq.substring(pos, flankend) + "_T";
        degen = (HashMap) individuals.get("degen");
        degen.put(genename + ":" + pos, new Character('W'));
        individuals.put("degen", degen);
        indivarr.add(flankseq);
        individuals.put(thisind, indivarr);
    }
    else if (thischar.charValue() == 'S')
    {
        flankstart = pos - 8;
        flankend = pos + 9;
        if (flankstart < 0)
        {
            flankstart = 0;
        }
        if (flankend > rcfseq.length())
        {
            flankend = rcfseq.length();
        }
        flankseq = genename + ":" + pos + "_" +
rcfseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + "_C";
        indivarr = (ArrayList) individuals.get(thisind);
        if (indivarr == null)
        {
            indivarr = new ArrayList();
        }
        indivarr.add(flankseq);
    }
}

```

```

        }
        indivarr.add(flankseq);
        flankseq = genename + ":" + pos + "_" +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + " G";
        degen = (HashMap) individuals.get("degen");
        degen.put(genename + "_" + pos, new Character('S'));
        individuals.put("degen", degen);
        indivarr.add(flankseq);
        individuals.put(thisind, indivarr);
    }
    else if (thischar.charValue() == 'Y')
    {
        flankstart = pos - 8;
        flankend = pos + 9;
        if (flankstart < 0)
        {
            flankstart = 0;
        }
        if (flankend > refseq.length())
        {
            flankend = refseq.length();
        }
        flankseq = genename + ":" + pos + "_" +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + " C";
        indivarr = (ArrayList) individuals.get(thisind);
        if (indivarr == null)
        {
            indivarr = new ArrayList();
        }
        indivarr.add(flankseq);
        flankseq = genename + ":" + pos + "_" +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + " T";
        degen = (HashMap) individuals.get("degen");
        degen.put(genename + "_" + pos, new Character('Y'));
        individuals.put("degen", degen);
        indivarr.add(flankseq);
        individuals.put(thisind, indivarr);
    }
    else if (thischar.charValue() == 'K')
    {
        flankstart = pos - 8;
        flankend = pos + 9;
        if (flankstart < 0)
        {
            flankstart = 0;
        }
        if (flankend > refseq.length())
        {
            flankend = refseq.length();
        }
        flankseq = genename + ":" + pos + "_" +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + " G";
        indivarr = (ArrayList) individuals.get(thisind);
        if (indivarr == null)
        {
            indivarr = new ArrayList();
        }
        indivarr.add(flankseq);
        flankseq = genename + ":" + pos + "_" +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + " T";
        degen = (HashMap) individuals.get("degen");
        degen.put(genename + "_" + pos, new Character('K'));
        individuals.put("degen", degen);
        indivarr.add(flankseq);
        individuals.put(thisind, indivarr);
    }
    else if (thischar.charValue() == 'A')
    {
        flankstart = pos - 8;
        flankend = pos + 9;
    }
}

```

```

        if (flankstart < 0)
        {
            flankstart = 0;
        }
        if (flankend > refseq.length())
        {
            flankend = refseq.length();
        }
        flankseq = genename + ":" + pos + " " +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + "_A";
        indivarr = (ArrayList) individuals.get(thisind);
        if (indivarr == null)
        {
            indivarr = new ArrayList();
        }
        degen = (HashMap) individuals.get("degen");
        degen.put(genename + ":" + pos, new Character(degenLookup('T',
refseq.charAt(pos - 1))));

        individuals.put("degen", degen);
        indivarr.add(flankseq);

        indivarr.add(flnkseq);
        individuals.put(thisind, indivarr);
    }
    else if (thischar.charValue() == 'C')
    {
        flankstart = pos - 8;
        flankend = pos + 9;
        if (flankstart < 0)
        {
            flankstart = 0;
        }
        if (flankend > rcfseq.length())
        {
            flankend = rcfseq.length();
        }
        flankseq = genename + ":" + pos + " " +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + "_C";
        indivarr = (ArrayList) individuals.get(thisind);
        if (indivarr == null)
        {
            indivarr = new ArrayList();
        }
        degen = (HashMap) individuals.get("degen");
        degen.put(genename + ":" + pos, new Character(degenLookup('C',
refseq.charAt(pos - 1))));

        individuals.put("degen", degen);
        indivarr.add(flnkseq);

        indivarr.add(flnkseq);
        individuals.put(thisind, indivarr);
    }
    else if (thischar.charValue() == 'G')
    {
        flankstart = pos - 8;
        flankend = pos + 9;
        if (flankstart < 0)
        {
            flankstart = 0;
        }
        if (flankend > refseq.length())
        {
            flankend = refseq.length();
        }
        flankseq = genename + ":" + pos + " " +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + "_G";
        indivarr = (ArrayList) individuals.get(thisind);
        if (indivarr == null)
        {
            indivarr = new ArrayList();
        }
    }
}

```

```

        }
        degen = (HashMap) individuals.get("degen");
        degen.put(gename + ":" + pos, new Character(degenLookup('G',
refseq.charAt(pos - 1))));

        indivarr.add(flankseq);
        individuals.put("degen", degen);
        indivarr.add(indivarr);

        indivarr.add(flankseq);
        individuals.put(thisind, indivarr);
    }
    else if (thischar.charValue() == 'T')
    {
        flankstart = pos - 8;
        flankend = pos + 9;
        if (flankstart < 0)
        {
            flankstart = 0;
        }
        if (flankend > refseq.length())
        {
            flankend = refseq.length();
        }
        flankseq = gename + ":" + pos + "_" +
refseq.substring(flankstart, pos - 1) + "X" + refseq.substring(pos, flankend) + "_T";
        indivarr = (ArrayList) individuals.get(thisind);
        if (indivarr == null)
        {
            indivarr = new ArrayList();
        }
        degen = (HashMap) individuals.get("degen");
        degen.put(gename + ":" + pos, new Character(degenLookup('T',
refseq.charAt(pos - 1))));

        indivarr.add(flankseq);
        individuals.put("degen", degen);
        indivarr.add(indivarr);

        indivarr.add(flankseq);
        individuals.put(thisind, indivarr);
    }
    else
    {
        flankstart = pos - 8;
        flankend = pos + 9;
        thischar = new Character(refseq.charAt(pos - 1));
        if (flankstart < 0)
        {
            flankstart = 0;
        }
        if (flankend > refseq.length())
        {
            flankend = refseq.length();
        }
        System.out.println(pos + " " + flankstart + " " + flankend + " " +
refseq.length() + thischar);
        flankseq = gename + ":" + pos + "_" + refseq.substring(flankstart, pos - 1) +
+ "X" + refseq.substring(pos, flankend) + "_" + thischar.charValue();
        indivarr = (ArrayList) individuals.get(thisind);
        if (indivarr == null)
        {
            indivarr = new ArrayList();
        }
        degen = (HashMap) individuals.get("degen");
        degen.put(gename + ":" + pos, thischar);
        individuals.put("degen", degen);
        indivarr.add(flankseq);
        indivarr.add(flankseq);
        individuals.put(thisind, indivarr);
    }
}
else
{
}

```

```

        flankstart = pos - 8;
        flankend = pos + 9;
        thischar = new Character(refseq.charAt(pos - 1));
        if (flankstart < 0)
        {
            flankstart = 0;
        }
        if (flankend > refseq.length())
        {
            flankend = refseq.length();
        }
        System.out.println(pos + " " + flankstart + " " + flankend + " " +
refseq.length() + thischar);
        flankseq = genename + ":" + pos + "_" + refseq.substring(flankstart, pos - 1)
+ "X" + refseq.substring(pos, flankend) + "_" + thischar.charValue();
        indivarr = (ArrayList) individuals.get(thisind);
        if (indivarr == null)
        {
            indivarr = new ArrayList();
        }
        //degen = (HashMap) individuals.get("degen");
        //degen.put(gename + ":" + pos, thischar);
        individuals.put("degen", degen);
        indivarr.add(flankseq);
        indivarr.add(flankseq);
        individuals.put(thisind, indivarr);
    }
}

return individuals;//returns hashmap of individuals and polymorphism entries
}

public static Connection establishCon(String server) throws Exception//scif explanatory
{
    Driver d = (Driver) Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
    Connection con = DriverManager.getConnection("jdbc:microsoft:sqlserver:" + server, "sa", "2grip");
    return con;
}

public static void uploadSNPs(String file, String server, String maskfile, String remoteID, String expID, String source,
String mask) throws Exception
{
    boolean maskbool = false;
    if(mask.equals("true"))
    {
        maskbool = true;
    }
    HashMap individuals = parseSNPs(file, maskfile, maskbool);//gets the hashmap from parseSNPs
    System.out.println(individuals);
    Iterator iter = individuals.keySet().iterator();
    Connection con = establishCon(server);
    String thisind = null;
    HashMap degen = null;
    while(iter.hasNext())
    {
        thisind = (String) iter.next();
        System.out.println(thisind);
        if(!thisind.equals("degen"))
        {
            ArrayList arr = (ArrayList) individuals.get(thisind);
            Iterator iter2 = null;
            String allele1 = null;
            String allele2 = null;
            if(arr != null)
            {
                iter2 = arr.iterator();
            }
        }
    }
}

```

```

        allele1 = (String) iter2.next();
    }
    while(iter2 != null && iter2.hasNext())
    {
        degen = (HashMap) individuals.get("degen");
        System.out.println(degen);
        String testallele = allele1.substring(0, allele1.indexOf(" "));
        System.out.println(testallele);
        char dallele = ((Character) degen.get(testallele)).charValue();
        allele1 = allele1.replace('X', dallele); //replaces generic X with proper
degenerate allele

        String markerID = checkMarker(allele1, con);
        checkAllele(allele1, con);
        checkInfo(allele1, con, source);
        if(iter2.hasNext())
        {
            allele2 = (String) iter2.next();
        }
        allele2 = allele2.replace('X', dallele);
        checkAllele(allele2, con);
        checkInfo(allele2, con, source);
        Statement st = con.createStatement();
        System.out.println(allele1);
        allele1 = markerID.substring(0, markerID.indexOf("_")) +
allele1.substring(allele1.lastIndexOf("_"));

        allele2.substring(allele2.lastIndexOf("_"));

        System.out.println(allele1 + " " + allele2);
        String updatecline = "(" + remotelID + "," + markerID + "," + thisind + "," +
allele1 + "," + allele2 + "," + source + "," + allele1.substring(0, allele1.indexOf("_")) + "," + expID + ", 0)";
        System.out.println(updatecline);
        st.executeUpdate("Insert into grip2.dbo.Genotypes values " +
updatecline); //grip2 is test server

        if(iter2.hasNext())
        {
            allele1 = (String) iter2.next();
        }
    }
}

public static String checkMarker(String markerID, Connection con) throws Exception
{
    String newmarker = markerID.substring(0, markerID.lastIndexOf("_"));
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("select markerID from grip2.dbo.Markers where markerID = '" + newmarker +
"');");
    if(rs.next() == false)
    {
        st.executeUpdate("insert into grip2.dbo.Markers values ('" + newmarker + "','" +
newmarker.substring(0, newmarker.indexOf("_")) + "', null, null, 'SNP')");
    }
    return newmarker;
}

public static void checkAllele(String allele, Connection con) throws Exception
{
    Statement st = con.createStatement();
    System.out.println(allele.lastIndexOf("_", allele.lastIndexOf(" ") - 1));
    String testallele = allele.substring(0, allele.indexOf(" ")) + allele.substring(allele.lastIndexOf(" ")));
    System.out.println(testallele);
    ResultSet rs = st.executeQuery("select Abs_allele from grip2.dbo.Alleles where Abs_allele = '" + testallele +
"');");
    if(rs.next() == false )
    {
        String locus = allele.substring(0, allele.lastIndexOf("_"));
        System.out.println(testallele + " locus = " + locus);
        st.executeUpdate("insert into grip2.dbo.Alleles values ('" + locus + "','" +
testallele + "', 0, null, 0)");
    }
}

```

```

        }

    }

    public static void checkInfo(String allele, Connection con, String source) throws Exception
    {
        Statement st = con.createStatement();
        String testallele = allele.substring(0, allele.indexOf("_")) + allele.substring(allele.lastIndexOf("_")));
        ResultSet rs = st.executeQuery("select Abs_allele from grip2 dbo.AlleleInfo where Abs_allele = '" + testallele
+ "'");
        if(rs.next() == false)
        {
            String locus = allele.substring(0, allele.lastIndexOf("_"));
            String snp = allele.substring(allele.lastIndexOf("_") + 1);
            st.executeUpdate("insert into grip2 dbo.AlleleInfo values ('" + snp + "','" + testallele + "','" + locus +
", 16,'" + source + "')");
        }
    }

    public static char degenLookup(char a, char b)
    {
        if(a == 'A')
        {
            if(b == 'C')
            {
                return 'M';
            }
            else if(b == 'G')
            {
                return 'R';
            }
            else if(b == 'T')
            {
                return 'W';
            }
        }
        else if(a == 'C')
        {
            if(b == 'A')
            {
                return 'M';
            }
            else if(b == 'G')
            {
                return 'S';
            }
            else if(b == 'T')
            {
                return 'Y';
            }
        }
        else if(a == 'G')
        {
            if(b == 'A')
            {
                return 'R';
            }
            else if(b == 'C')
            {
                return 'S';
            }
            else if(b == 'T')
            {
                return 'K';
            }
        }
        else if(a == 'T')
        {
            if(b == 'A')
            {
                return 'W';
            }
        }
    }
}

```

```

        }
        else if(b == 'C')
        {
            return 'Y';
        }
        else if(b == 'G')
        {
            return 'K';
        }
    }
    return 'X';
}

haplotyper.java

//Haplotyper for ONPRC
//By Hollis Wright
//Release Version 1.0
//Apri 14, 2005

package haplotyper;
import java.io.*;
import java.util.*;
import java.sql.*;

public class haplotyper
{
    //this function isn't actually used, currently.
    //but if we start doing auto-generated haplotypes, it may bc.
    public static void generateHaplotypes(String server) throws Exception
    {
        Connection con = pedviewer.inbreedcoeff建立Con(server);
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("select MarkerId from grip2.dbo.HaplotypeBlock");
        int killflag = 0;
        int counter = 0;
        String marker = null;
        String haplolist = null;
        ArrayList haploarr = new ArrayList();
        while (killflag == 0)
        {
            if(rs.next() == false)
            {
                killflag = 1;
            }
            else
            {
                haploarr.add(rs.getString("MarkerId"));
            }
        }
        HashMap locusarr = new HashMap();
        rs = st.executeQuery("select MarkerId from grip2.dbo.Genotypes");
        killflag = 0;
        while (killflag == 0)
        {
            if(rs.next() == false)
            {
                killflag = 1;
            }
            else
            {
                marker = rs.getString("MarkerID");
                if(marker.indexOf("SNP_") != -1)
                {
                    locusarr.put(marker, null);
                }
            }
        }
        Iterator iter1 = locusarr.keySet().iterator();
    }
}

```

```

Iterator iter2 = haploarr.iterator();
while (iter1.hasNext())
{
    marker = (String) iter1.next();
    iter2 = haploarr.iterator();
    killflag = 0;
    while (killflag == 0 && iter2.hasNext())
    {
        haplolist = (String) iter2.next();
        if(haplolist.indexOf(marker) != -1)
        {
            killflag = 1;
            locusarr.remove(marker);
        }
    }
    if(killflag == 0)
    {
        ++counter;
    }
}
//System.out.println(locusarr.keySet());
HashMap bins = new HashMap();
iter1 = locusarr.keySet().iterator();
TreeMap tmap = null;
while (iter1.hasNext())
{
    marker = (String) iter1.next();
    int uidx = marker.indexOf("_");
    String genid = marker.substring(0, marker.indexOf("_", uidx + 1));
    tmap = (TreeMap) bins.get(genid);
    int pos = Integer.parseInt(marker.substring(marker.lastIndexOf("_", marker.lastIndexOf("_") - 1) +
1, marker.lastIndexOf("_")));
    if (tmap == null)
    {
        tmap = new TreeMap();
        tmap.put(new Integer(pos), marker);
        bins.put(genid, tmap);
    }
    else
    {
        tmap.put(new Integer(pos), marker);
        bins.put(genid, tmap);
    }
}
System.out.println(bins.keySet());
System.out.println(bins.values());
HashMap htypes = new HashMap();
iter1 = bins.keySet().iterator();
ArrayList arr = null;
while(iter1.hasNext())
{
    String genkey = (String) iter1.next();
    tmap = (TreeMap) bins.get(genkey);
    counter = 0;
    killflag = 0;
    int lastpos = 0;
    int thispos = 0;
    iter2 = tmap.values().iterator();
    arr = new ArrayList();
    while(iter2.hasNext())
    {
        String thislocus = (String) iter2.next();
        System.out.println(thislocus);
        thislocus = thislocus.substring(thislocus.indexOf("_") + 1, thislocus.lastIndexOf("_"));
        System.out.println(thispos + " " + lastpos);
        if (killflag < 5 && iter2.hasNext())
        {
            if(lastpos == 0)
            {
                System.out.println("first one");
            }
        }
    }
}

```

```

        arr.add(thislocus);
        System.out.println(arr);
        thispos =
Integer.parseInt(thislocus.substring(thislocus.lastIndexOf("_") + 1));
        lastpos = thispos;
        ++killflag;
    }
else
{
    thispos =
Integer.parseInt(thislocus.substring(thislocus.lastIndexOf("_") + 1));
    System.out.println("thispos is = " + thispos);
    System.out.println(thispos);
    if(thispos - lastpos <= 100)
    {
        lastpos = thispos;
        arr.add(thislocus);
        System.out.println(arr);
        ++killflag;
    }
else
{
    htypes.put(genkey + "_" + counter, arr);
    ++counter;
    killflag = 0;
    arr = new ArrayList();
    arr.add(thislocus);
    thispos = Integer.parseInt(thislocus.substring(thislocus.lastIndexOf("_") + 1));
    lastpos = thispos;
    killflag = 1;
}
}
else if (killflag >= 5 && iter2.hasNext())
{
    htypes.put(genkey + "_" + counter, arr);
    ++counter;
    killflag = 0;
    arr = new ArrayList();
    arr.add(thislocus);
    thispos = Integer.parseInt(thislocus.substring(thislocus.lastIndexOf("_") + 1));
    lastpos = thispos;
    killflag = 1;
}
else
{
    arr.add(thislocus);
    htypes.put(genkey + "_" + counter, arr);
    System.out.println("done" + arr);
}
}
System.out.println(htypes.keySet() + " " + htypes.values());
iter1 = htypes.keySet().iterator();
while(iter1.hasNext())
{
    String hapid = (String) iter1.next();
    arr = (ArrayList) htypes.get(hapid);
    System.out.println(arr);
    iter2 = arr.iterator();
    String blockid = new String();
    while (iter2.hasNext())
    {
        String nextblock = (String) iter2.next();
        if(iter2.hasNext())
        {
            blockid = blockid + nextblock + ",";
        }
    }
}

```

```

        {
            blockid = blockid + nextblock;
        }
    }
    System.out.println(blockid);
    st.executeUpdate("insert into grip2 dbo.HaplotypeInfo values (" + blockid + "," + hapid + "_inf,
null");
}
}

public static int calculateHaplotypes(String server, String haploid, boolean fullupdate) throws Exception
{
    Connection con = pedviewer.inbreedcoeff.cestablishCon(server);
    Statement st = con.createStatement();
    ResultSet rs = null;
    boolean next = true;
    HashMap family = new HashMap();
    HashMap parentmap = new HashMap();
    TreeMap famsort = new TreeMap();
    ArrayList famstring = null;
    Integer famnum = null;
    int famcounter = 1;
    int filecounter = 0;
    HashMap individuals = null;
    TreeMap absall = null;
    HashMap alleles = null;
    HashMap arr = new HashMap();
    Random ran = new Random();
    if (fullupdate == true)//not generally used; used with above function.
    {
        rs = st.executeQuery("select HaplotypeID, HaplotypeName from grip2 dbo.HaplotypeInfo");
        while(next == true)
        {
            next = rs.next();
            if (next == true)
            {
                String inf = rs.getString("HaplotypeName");
                if(inf.indexOf("inf") != -1)
                {
                    arr.put(rs.getString("HaplotypeID"), null);
                }
            }
        }
    }
    else
    {
        arr.put(haploid, null);
    }
    //System.out.println(arr);
    Iterator iter1 = arr.keySet().iterator();
    String thisindy = null;
    FileWriter fout = null;
    while(iter1.hasNext())
    {
        String fnamec = "" + ran.nextInt();
        fout = new FileWriter("c:\\hapdata\\hapdata" + fnamec + ".txt");
        individuals = new HashMap();
        absall = new TreeMap();
        absmap = null;
        String thislocus = null;
        String thishaploid = (String) iter1.next();
        //System.out.println(thishaploid);
        //String thisgene = thishaploid.substring(0, thishaploid.indexOf(":") + 1);
        //thishaploid = thishaploid.substring((thishaploid.indexOf(":") + 1));
        StringTokenizer str = new StringTokenizer(thishaploid, ",");
        while(str.hasMoreTokens())//gets alleles for all individuals genotyped at selected locus.
        {
            thislocus = str.nextToken();
            thislocus = thislocus.substring(0, thislocus.indexOf("_"));
            //System.out.println(thislocus);
        }
    }
}

```



```

        }
        Iterator iter2 = individuals.keySet().iterator();
        //System.out.println(individuals);
        while(iter2.hasNext())
        {
            thisindv = (String) iter2.next();
            ArrayList parents = pedviewer.inbreedcoeff.getAncestors(thisindv, con);
            String father = null;
            String mother = null;
            if (parents != null)
            {
                father = (String) parents.get(0);
                mother = (String) parents.get(1);
            }
            Object ftest = null;
            Object mtest = null;
            if (!(father == null || mother == null || father.length() < 1 || mother.length() < 1 ||
father.indexOf("O") != -1 || mother.indexOf("O") != -1))
            {
                ftest = individuals.get(father);
                mtest = individuals.get(mother);
                if (ftest != null && mtest != null)
                {
                    parentmap.put(thisindv, parents);
                }
            }
            if (father == null || father.length() < 1 || father.indexOf("O") != -1)
            {
                father = null;
            }
            if (mother == null || mother.length() < 1 || mother.indexOf("O") != -1)
            {
                mother = null;
            }

            famnum = (Integer) family.get(thisindv);
            if (famnum != null)
            {
                System.out.println(thisindv + " already in " + famnum + " " +
famcounter);
                if (ftest == null)
                {
                    family.put(father, famnum);
                }
                if (mtest == null)
                {
                    family.put(mother, famnum);
                }
            }
            else
            {
                if (father != null)
                {
                    famnum = (Integer) family.get(father);
                }
                else
                {
                    famnum = null;
                }
                Integer famnum2 = null;
                if (mother != null)
                {
                    famnum2 = (Integer) family.get(mother);
                }
                System.out.println(father + " " + famnum + " " + mother + " " + famnum2);
                if (famnum == null && famnum2 == null)
                {
                    family.put(thisindv, new Integer(famcounter));
                    if (ftest != null && mtest != null)

```

```

        {
            family.put(father, new Integer(famcounter));
            family.put(mother, new Integer(famcounter));
        }
        System.out.println(thisindv + " not in " + famcounter + " " +
famcounter);
        ++famcounter;
    }
    else if (famnum == null)
    {
        family.put(father, famnum2);
        family.put(thisindv, famnum2);
        System.out.println(thisindv + " mother in " + famnum2 + " " +
famcounter );
    }
    else if (famnum2 == null)
    {
        family.put(mother, famnum);
        family.put(thisindv, famnum);
        System.out.println(thisindv + " father in " + famnum + " " +
famcounter);
    }
    else if (famnum.intValue() == famnum2.intValue())
    {
        family.put(thisindv, famnum);
        System.out.println(thisindv + " mother & father in " +
famnum + " " + famnum2 + " " + famcounter);
    }
    else
    {

        family.put(thisindv, famnum);
        System.out.println(family);
        Iterator famiter = family.keySet().iterator();
        ArrayList addarr = new ArrayList();
        while (famiter.hasNext())
        {
            String changeindv = (String) famiter.next();
            Integer changenum = (Integer) family.get(changeindv);
            if (changenum.intValue() == famnum2.intValue())
            {
                famiter.remove();
                addarr.add(changenum);
            }
        }
        System.out.println(addarr);
        famiter = addarr.iterator();
        while (famiter.hasNext())
        {
            family.put((String) famiter.next(), famnum);
        }
        System.out.println(family);
        System.out.println(thisindv + " execute7 " + famnum + " " +
famnum2 + " " + famcounter);
    }
}
//System.out.println(family);
iter2 = individuals.keySet().iterator();

while (iter2.hasNext())
{
    thisindv = (String) iter2.next();
    ArrayList parents = (ArrayList) parentmap.get(thisindv);
    String father = null;
    String mother = null;
    if (parents != null)
    {
        father = (String) parents.get(0);
        mother = (String) parents.get(1);
    }
}

```

```

        }
    else
    {
        father = "0";
        mother = "0";
    }
    //System.out.println(family);
    //System.out.println(thisindv);
    Integer famid = (Integer) family.get(thisindv);
    System.out.println(thisindv + " " + famid);
    String indivstring = famid.intValue() + "\t" + thisindv + "\t" + father + "\t" + mother +
    "\t" + pedviewer.inbreedcoeff.getScx(thisindv, con) + "\t0\t0";
    Iterator iter3 = absall.keySet().iterator();
    //System.out.println(absall.keySet());
    while (iter3.hasNext())
    {
        thislocus = (String) iter3.next();
        //System.out.println(thisindv + " at " + thislocus);
        alleles = (HashMap) individuals.get(thisindv);
        String absalleles = (String) alleles.get(thislocus);
        absmap = (HashMap) absall.get(thislocus);
        //System.out.println(thislocus + " " + absalleles + " Map = " + absmap);
        Integer abs1 = null;
        Integer abs2 = null;
        if (absalleles == null)
        {
            abs1 = new Integer(0);
            abs2 = new Integer(0);
        }
        else
        {
            abs1 = (Integer) absmap.get(absalleles.substring(0,
            absalleles.indexOf("/")));
            abs2 = (Integer)
            if (abs1 == null)
            {
                abs1 = new Integer(0);
            }
            if (abs2 == null)
            {
                abs2 = new Integer(0);
            }
        }
        //System.out.println(abs1.intValue() + " " + abs2.intValue());
        indivstring = indivstring + "\t" +
        abs1.intValue() + "\t" + abs2.intValue();
    }
    famstring = (ArrayList) famsort.get(famid);
    if (famstring == null)
    {
        famstring = new ArrayList();
    }
    famstring.add(indivstring);
    famsort.put(famid, famstring);
}
iter2 = famsort.keySet().iterator();
while (iter2.hasNext())
{
    famnum = (Integer) iter2.next();
    famstring = (ArrayList) famsort.get(famnum);
    if (famstring != null)
    {
        Iterator iter3 = famstring.iterator();
        while (iter3.hasNext())
        {
            fout.write((String) iter3.next() + "\n");
        }
    }
}
}

```

```

fout.flush();
fout.close();
System.out.println("I should run once");
Runtime.getRuntime().exec("c:\\pedphase\\PcdPhase.exe -l c:\\hapdata\\hapdata" + fname + ".txt");

int killflag = 0;
TreeMap tmap = new TreeMap();
str = new StringTokenizer(thishaploid, ",");
//System.out.println(thishaploid);
while (str.hasMoreTokens() && killflag < 5)
{
    thislocus = str.nextToken();
    //System.out.println(thislocus);
    tmap.put(new Integer(Integer.parseInt(thislocus.substring(thislocus.indexOf(":") + 1,
thislocus.indexOf("_")))), thislocus);
    ++killflag;
}
Iterator iter = tmap.values().iterator();
thishaploid = new String();
while (iter.hasNext())
{
    String nextblock = (String) iter.next();
    System.out.println("this is nextblock =" + nextblock);
    if(nextblock.indexOf(":") != -1)
    {
        String locid = nextblock.substring(0, nextblock.indexOf(":"));
        if(thishaploid.indexOf(locid) != -1)
        {
            nextblock = nextblock.substring(nextblock.indexOf(":") + 1);
        }
        if(nextblock.indexOf("_") != -1)
        {
            nextblock = nextblock.substring(0, nextblock.indexOf("_"));
        }
    }
    if(iter.hasNext())
    {
        thishaploid = thishaploid + nextblock + ",";
    }
    else
    {
        thishaploid = thishaploid + nextblock;
    }
    ++killflag;
}
int failflag = 0;
FileReader fr = null;
String line = null;
BufferedReader buf = null;
while (line == null || line.length() < 1)//this is an ugly bit of code
try
{
//but windows doesn't let
{
//the output file go correctly, hence the loop here
fr = new FileReader("c:\\hapdata\\hapdata" + fname + ".out");
buf = new BufferedReader(fr);
line = buf.readLine();
}
catch (Exception ex)
{
    ex.printStackTrace();
    line = null;
    ++failflag;
    if(failflag > 10000)
    {
        return 0;
    }
}
System.out.println("Open " + fname);
}

```

```

while(line != null)
{
    System.out.println(line);
    str = new StringTokenizer(line);
    System.out.println(str.nextToken());
    thisindv = str.nextToken();
    System.out.println(thisindv + " " + str.nextToken());
    System.out.println(str.nextToken());
    System.out.println(str.nextToken());
    System.out.println(str.nextToken());
    System.out.println(str.nextToken());
    ArrayList hap1 = new ArrayList();
    ArrayList hap2 = new ArrayList();
    iter2 = absall.keySet().iterator();
    while(iter2.hasNext())
    {
        absmap = (HashMap) absall.get(iter2.next());
        try
        {
            thislocus = str.nextToken();
            System.out.println(thislocus);
            Integer thistype = new Integer(Integer.parseInt(thislocus.substring(0,
thislocus.indexOf("/"))));
            //System.out.println(absmap);
            Iterator iter3 = absmap.keySet().iterator();
            killflag = 0;
            while(iter3.hasNext() && killflag == 0)
            {
                String thiskey = (String) iter3.next();
                if (thistype.intValue() == ((Integer)
absmap.get(thiskey)).intValue())
                {
                    killflag = 1;
                    //System.out.println(thiskey);
                    hap1.add(thiskey);
                }
            }
            thistype = new
Integer(Integer.parseInt(thislocus.substring(thislocus.indexOf("/") + 1)));
            iter3 = absmap.keySet().iterator();
            killflag = 0;
            while(iter3.hasNext() && killflag == 0)
            {
                String thiskey = (String) iter3.next();
                //System.out.println(thiskey + thistype.intValue());
                if (thistype.intValue() == ((Integer)
absmap.get(thiskey)).intValue())
                {
                    killflag = 1;
                    //System.out.println(thiskey);
                    hap2.add(thiskey);
                }
            }
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }
    String block1 = checkHaploBlock(thishaploid, hap1, con);
    String block2 = checkHaploBlock(thishaploid, hap2, con);
    rs = st.executeQuery("Select BlockId from grip2.dbo.HaplotypeBlock where
HaplotypeID = '" + thishaploid + "' and MarkerID = '" + block1 + "'");
    if(rs.next() == false)
    {
        System.out.println("upload error has occured for block = " + thishaploid + "
for individual = " + thisindv);
    }
    else
    {
}
}

```

```

        block1 = rs.getString("BlockId");
        rs = st.executeQuery("Select BlockId from grip2.dbo.HaplotypeBlock where
HaplotypeID = '" + thishaploid + "' and MarkerID = '" + block2 + "'");
        if(rs.next() == false)
        {
            System.out.println("upload error has occured for block = " +
thishaploid + " for individual = " + thisindy);
        }
        else
        {
            block2 = rs.getString("BlockId");
            st.executeUpdate("delete from grip2.dbo.Haplotypes where
AnimalId = '" + thisindy + "' and HaplotypeID = '" + thishaploid + "'");
            st.executeUpdate("insert into grip2.dbo.Haplotypes values (" +
thisindy + "," + thishaploid + "," + block1 + "," + block2 + ")");
        }
        //System.out.println(thishaploid);
        line = buf.readLine();
    }
    buf.close();
    fr.close();
    ++filecounter;
    //Runtime.getRuntime().exec("del c:\hapdata\*.out");
}
return 1;
}

public static String checkHaploBlock(String haploid, ArrayList arr, Connection con) throws Exception
{
    //System.out.println(haploid);
    Iterator iter1 = arr.iterator();
    String marker = null;
    TreeMap tmap = null;
    int killflag = 0;
    while (iter1.hasNext())
    {
        marker = (String) iter1.next();
        System.out.println(marker);
        int uididx = marker.indexOf(":");
        String genid = marker.substring(0, marker.indexOf("_"));
        int pos = Integer.parseInt(marker.substring(marker.indexOf(":") + 1, marker.indexOf("_")));
        if (tmap == null)
        {
            tmap = new TreeMap();
            tmap.put(new Integer(pos), marker);
        }
        else
        {
            tmap.put(new Integer(pos), marker);
        }
    }
    iter1 = tmap.values().iterator();
    String blockid = new String();
    while (iter1.hasNext())
    {
        String nextblock = (String) iter1.next();
        String locid = nextblock.substring(0, nextblock.indexOf("."));
        if(blockid.indexOf(locid) != -1)
        {
            nextblock = nextblock.substring(nextblock.indexOf(".") + 1);
        }
        if(iter1.hasNext())
        {
            blockid = blockid + nextblock + ",";
        }
        else
        {
            blockid = blockid + nextblock;
        }
    }
}

```

```

        }

        //System.out.println("this is the blockid =" + blockid);
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("select * from grip2 dbo.HaplotypeBlock where HaplotypeID = '" + haploid +
and MarkerId = "' + blockid + "'");
        if (rs.next() == false)
        {
            rs = st.executeQuery("select count(MarkerId) as \"count\" from grip2 dbo.HaplotypeBlock where
HaplotypeID = '" + haploid + "'");
            rs.next();
            int types = rs.getInt("count");
            ++types;
            st.executeUpdate("insert into grip2 dbo.HaplotypeBlock values ('" + haploid + "','" + blockid +
"',0,'" + haploid + "_" + types + "')");
        }
        return blockid;
    }

    public static String defineUserHaplotype(String usertype, String server) throws Exception //used by JSP to define user
haplotypes
    {
        Connection con = pcdviewer.inbreedcoeff.establishCon(server);
        Statement st = con.createStatement();
        StringTokenizer str = new StringTokenizer(usertype, ",");
        String hapid = usertype.substring(0, usertype.indexOf(":"));
        ResultSet rs = rs = st.executeQuery("select count(HaplotypeID) as \"count\" from grip2 dbo.HaplotypeInfo
where HaplotypeID like '%" + hapid + "%'");
        rs.next();
        hapid = hapid + " " + rs.getInt("count");
        String blockid = new String();
        int killflag = 0;
        TreeMap tmap = new TreeMap();
        while (str.hasMoreTokens() && killflag < 5)
        {
            String thislocus = str.nextToken();
            thislocus = thislocus.substring(0, thislocus.lastIndexOf("_"));
            //System.out.println(thislocus);
            tmap.put(new Integer(Integer.parseInt(thislocus.substring(thislocus.indexOf(".") + 1))), thislocus);
            ++killflag;
        }
        Iterator iter = tmap.values().iterator();
        while (iter.hasNext())
        {
            String nextblock = (String) iter.next();
            String locid = nextblock.substring(0, nextblock.indexOf(":"));
            if(blockid.indexOf(locid) != -1)
            {
                nextblock = nextblock.substring(nextblock.indexOf(".") + 1);
            }
            if(nextblock.indexOf("_") != -1)
            {
                nextblock = nextblock.substring(0, nextblock.indexOf("_"));
            }
            if(iter.hasNext())
            {
                blockid = blockid + nextblock + ",";
            }
            else
            {
                blockid = blockid + nextblock;
            }
            ++killflag;
        }
        //System.out.println("user adding" + blockid);
        rs = st.executeQuery("select HaplotypeID from grip2 dbo.HaplotypeInfo where HaplotypeID = '" + blockid +
"');");
        if(rs.next() == false)
        {
            st.executeUpdate("insert into grip2 dbo.HaplotypeInfo values ('" + blockid + "','" + hapid + "_inf', null)");
        }
    }
}

```

```

        }
        return blockid;
    }

    public static double getMinorFreq(String locus, String server) throws Exception //this won't be useful on production server
    {
        Connection con = pedviewer.inbreedcoeff建立连接(server);
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("select Abs_allele1, Abs_allele2 from grip2.dbo.Genotypes where MarkerId = '" +
+ locus + "'");
        boolean next = rs.next();
        //System.out.println(next);
        double freq = 1;
        HashMap alleles = new HashMap();
        double counter = 0;
        while (next == true)
        {
            String thisallele = rs.getString("Abs_allele1");
            thisallele = thisallele.substring(thisallele.length() - 1);
            Double thiscount = (Double) alleles.get(thisallele);
            if(thisallele.equals("N"))
            {
            }
            else if(thiscount == null)
            {
                thiscount = new Double(1.0);
                alleles.put(thisallele, thiscount);
                ++counter;
            }
            else
            {
                alleles.put(thisallele, new Double(thiscount.doubleValue() + 1.0));
                ++counter;
            }
            thisallele = rs.getString("Abs_allele2");
            thisallele = thisallele.substring(thisallele.length() - 1);
            thiscount = (Double) alleles.get(thisallele);
            if (thisallele.equals("N"))
            {
            }
            else if(thiscount == null)
            {
                thiscount = new Double(1.0);
                alleles.put(thisallele, thiscount);
                ++counter;
            }
            else
            {
                alleles.put(thisallele, new Double(thiscount.doubleValue() + 1.0));
                ++counter;
            }
            next = rs.next();
        }
        Iterator iter = alleles.values().iterator();
        while (iter.hasNext())
        {
            Double thiscount = (Double) iter.next();
            double thisfreq = thiscount.doubleValue() / counter;
            if(thisfreq < freq)
            {
                freq = thisfreq;
            }
        }
        //System.out.println(locus + " " + freq);
        if (freq == 1.0)
        {
            freq = 0;
        }
        return freq;
    }
}

```

```

        }

    public static TreeMap getLociList(String locus, String server) throws Exception //gets the loci for a specific marker list for
the JSP
{
    Connection con = pedviewer.inbreedcoeff建立连接(server);
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("select MarkerId from grip2.dbo.Genotypes where MarkerId like '%" + locus +
"%'");

    boolean next = rs.next();
    double freq = 1;
    TreeMap alleles = new TreeMap();
    double counter = 0;
    while (next == true)
    {
        String thismarker = rs.getString("MarkerId");
        Double thiscount = new Double(getMinorFreq(thismarker, server));
        alleles.put(thismarker, thiscount);
        next = rs.next();
    }
    return alleles;
}

}

class fopener
{
    public FileReader getFr (String name) throws Exception
    {
        wait(10000);
        FileReader fr = new FileReader(name);
        return fr;
    }
}

```

haplosearcher.java

```

//Haplotype database searcher for ONPRC
//By Hollis Wright
//Release Version 1.0
//April 14, 2005
package haplotyper;
import java.io.*;
import java.util.*;
import java.sql.*;
import pedviewer.*;

public class haplosearcher
{
    public static ArrayList getHaplotypes(ArrayList individuals, ArrayList loci, boolean specific) throws Exception
    {
        Connection con = inbreedcoeff.establishCon("//localhost:1731");
        Statement st = con.createStatement();
        ResultSet rs = null;
        ResultSet rs2 = null;
        ResultSet rs3 = null;
        String allelelist1 = null;
        String allelelist2 = null;
        boolean next = true;
        String indivlist = new String();
        String genelist = new String();
        String locilist = new String();
        Iterator iter1 = null;
        String lc1 = "";
        String where = " where ";
        ArrayList results = new ArrayList();
        if(individuals == null)
        {
            indivlist = "";
        }

```

```

        }
    else
    {
        iter1 = individuals.iterator();
        indivlist = "AnimalId in (";
        while(iter1.hasNext())
        {
            String nextid = (String) iter1.next();
            if(iter1.hasNext())
            {
                indivlist = indivlist + "" + nextid + ",";
            }
            else
            {
                indivlist = indivlist + "" + nextid + ")";
            }
        }
        if(loci == null)
        {
            locilist = "";
        }
        else
        {
            iter1 = loci.iterator();
            locilist = "HaplotypeId like ";
            if(specific == true)
            {
                TreeMap tmap = new TreeMap();
                while (iter1.hasNext())
                {
                    String thislocus = (String) iter1.next();
                    //System.out.println(thislocus);
                    tmap.put(new
Integer(Integer.parseInt(thislocus.substring(thislocus.indexOf(":") + 1))), thislocus);
                }
                iter1 = tmap.values().iterator();
                while(iter1.hasNext())
                {
                    String nextblock = (String) iter1.next();
                    String locid = nextblock.substring(0, nextblock.indexOf(":"));
                    if(locilist.indexOf(locid) != -1)
                    {
                        nextblock = nextblock.substring(nextblock.indexOf(":") + 1);
                    }
                    if(iter1.hasNext())
                    {
                        locilist = locilist + nextblock + ",";
                    }
                    else
                    {
                        locilist = locilist + nextblock + "";
                    }
                }
            }
            if(individuals != null && loci != null)
            {
                lc1 = " and ";
            }
            if(individuals == null && loci == null)
            {
                where = "";
            }
            System.out.println(where + indivlist + lc1 + locilist + next);
            if(specific == true || loci == null)
            {
                rs = st.executeQuery ("Select * from grip2.dbo.Haplotypes" + where + indivlist + lc1 + locilist);
                next = rs.next();
            }
        }
    }
}

```

```

        while (next == true)
        {
            con = inbreedcoff.establishCon("//localhost:1731");
            st = con.createStatement();
            rs2 = st.executeQuery("Select MarkerId from grip2.dbo.HaplotypeBlock where BlockId
= " + rs.getString("BlockId1") + "");

            next = rs2.next();
            if (next == true)
            {
                allelelist1 = rs2.getString("MarkerId");
            }
            else
            {
                allelelist1 = "unknown";
            }
            rs2 = null;
            con = inbreedcoeff.establishCon("//localhost:1731");
            st = con.createStatement();
            rs3 = st.executeQuery("Select MarkerId from grip2.dbo.HaplotypeBlock where BlockId
= " + rs.getString("BlockId2") + "");

            next = rs3.next();
            if (next == true)
            {
                allelelist2 = rs3.getString("MarkerId");
            }
            else
            {
                allelelist2 = "unknown";
            }
            rs3 = null;
            results.add("<TR><TD>" + rs.getString("AnimalId") + "</TD><TD>" +
rs.getString("HaplotypeId") + "</TD><TD>" + rs.getString("BlockId1") + "</TD><TD>" + rs.getString("BlockId2") +
"</TD><TD>" + allelelist1 + "</TD><TD>" + allelelist2 + "</TD></TR>");
            next = rs.next();
        }
    }
    else
    {
        while(iter1.hasNext())
        {
            String thisone = (String) iter1.next();
            thisone = thisone.replace('.', '%');
            loclist = "HaplotypeId like " + thisone + "%";
            System.out.println(loclist);
            rs = st.executeQuery ("Select * from grip2.dbo.Haplotypes" + where + indivlist + lc1 +
loclist);
            next = rs.next();
            while (next == true)
            {
                con = inbreedcoeff.establishCon("//localhost:1731");
                st = con.createStatement();
                rs2 = st.executeQuery("Select MarkerId from grip2.dbo.HaplotypeBlock
where BlockId = " + rs.getString("BlockId1") + "");

                next = rs2.next();
                if (next == true)
                {
                    allelelist1 = rs2.getString("MarkerId");
                }
                else
                {
                    allelelist1 = "unknown";
                }
                rs2 = null;
                con = inbreedcoff.establishCon("//localhost:1731");
                st = con.createStatement();
                rs3 = st.executeQuery("Select MarkerId from grip2.dbo.HaplotypeBlock
where BlockId = " + rs.getString("BlockId2") + "");

                next = rs3.next();
                if (next == true)
                {

```

```

        allelelist2 = rs3.getString("MarkerId");
    }
    else
    {
        allelelist2 = "unknown";
    }
    rs3 = null;
    results.add("<TR><TD>" + rs.getString("AnimalId") + "</TD><TD>" +
    rs.getString("HaplotypeId") + "</TD><TD>" + rs.getString("BlockId1") + "</TD><TD>" + rs.getString("BlockId2") +
    "</TD><TD>" + allcclist1 + "</TD><TD>" + allelelist2 + "</TD></TR>");
    next = rs.next();
}
}
return results;//results are unsorted, with duplicates. Calling program needs to deal with that.
}
}

```

ProcessFileUpload.jsp

```

<%@ page contentType="text/html;charset=windows-1252"%>
<%@ page import="org.apache.commons.fileupload.DiskFileUpload"%>
<%@ page import="org.apache.commons.fileupload.FileItem"%>
<%@ page import="snploader.*"%>
<%@ page import="haplotyper.*"%>
<%@ page import="java.util.List"%>
<%@ page import="java.util.ArrayList"%>
<%@ page import="java.util.Iterator"%>
<%@ page import="java.io.File"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Process File Upload</title>
</head>
<%
    System.out.println("Content Type =" + request.getContentType());
    String fname = null;
    String expID = null;
    String remoteID = null;
    String source = null;
    DiskFileUpload fu = new DiskFileUpload();
    fu.setSizeMax(1000000);
    List fileItems = fu.parseRequest(request);
    Iterator itr = fileItems.iterator();
    ArrayList arr = new ArrayList();
    String mask = "false";
    while(itr.hasNext())
    {
        FileItem fi = (FileItem)itr.next();
        if(!fi.isFormField())
        {
            System.out.println("\nNAME: " + fi.getName());
            fname = fi.getName();
            if(fname.length() != 0)
            {
                fname = fname.substring(fname.lastIndexOf("\\") + 1);
                System.out.println("SIZE: " + fi.getSize());
                //System.out.println(fi.getOutputStream().toString());
                File fNew = new File("c:\\snpuploads\\" + fname);
                arr.add(fname);
                System.out.println(fNew.getAbsolutePath());
                fi.write(fNew);
            }
        }
    }
}

```

```

String check = fi.getFieldName();
    System.out.println(check);
    if (check.equals("RemoteID"))
    {
        remoteID = fi.getString();
    }
    else if (check.equals("ExpID"))
    {
        expID = fi.getString();
    }
    else if (check.equals("Source"))
    {
        source = fi.getString();
    }
    else if (check.equals("Mask"))
    {
        mask = fi.getString();
    }
}
}
}
String stupid1 = (String) arr.get(0);
String stupid2 = "nomask";
int checkflag = 0;
if (mask.equals("true"))
{
    stupid2 = (String) arr.get(1);
}
snploader.uploadSNPs("c:\\snpuploads\\" + stupid1, "//localhost:1731", "c:\\snpuploads\\" + stupid2, expID, remoteID,
source, mask);

%>
<body>

Upload Successful.
</body>
</html>

```

getloci.jsp

```

<%@ page language="java" import="java.util.*" import="java.io.*" import="haplotyper.*" %>
<HTML>
<HEAD>
<TITLE>Select Loci for Haplotyping</TITLE>
<BODY>

<%
String locus = request.getParameter("Locus");
TreeMap tmap = haplotyper.getLociList(locus, "//localhost:1731");
Iterator iter = tmap.keySet().iterator();
ArrayList arr = new ArrayList();
int size = tmap.keySet().size();
String thislocus = null;
FileWriter fout = new FileWriter("c:\\hapdata\\haploc.svg");
fout.write("<svg width = \"800\" height = \"50\" xmlns=\"http://www.w3.org/2000/svg\">\n");
fout.write("<g stroke = \"black\"><line x1 = \"0\" y1 = \"25\" x2 = \"800\" y2 = \"25\" /></g>\n");
while(iter.hasNext())
{
    thislocus = (String) iter.next();
    Double thisfreq = (Double) tmap.get(thislocus);
    String tagcolor = null;
    if (thisfreq.doubleValue() > .4)
    {
        tagcolor = "green";
    }
    else if (thisfreq.doubleValue() > .2)
    {
        tagcolor = "yellow";
    }
    else
    {

```

```

        tagcolor = "red";
    }
    int pos = Integer.parseInt(thislocus.substring(thislocus.indexOf(":") + 1, thislocus.indexOf("_")));
    fout.write("<g id = \"f" + thislocus + "\"><rect width = \"5\" height = \"5\" fill = \"\" + tagcolor + "\"
transform = \"translate(" + pos + ", 25)\"></rect></g>\n");
    fout.write("<g style=\"display:none\"><set attributeName=\"display\" from=\"none\" to=\"block\" begin=\"f" + thislocus + ".click\" fill = \"freeze\"/> <set attributeName=\"display\" from=\"block\" to=\"none\" begin=\"click\" fill = \"freeze\"/> <text opacity=\"1.0\" x=\"" + pos + "\" y = \"35\" font-size = \"8\">" + thislocus + " " + thisfreq + "</text> </g>\n");
    thislocus = thisfreq.doubleValue() + "</TD><TD>" + thislocus;
    arr.add(thislocus);
}
fout.write("</svg>");
fout.close();
%>
<H1></H1><embed src="c:\hapdata\haploc.svg" width="800" height="50" type="image/svg+xml" name="emap">
</embed>
<H2 id = "table"></H2>
<P>
Number of rows: <%=size %>
<FORM type = "post" action = "genuserhaplo.jsp">
<TABLE BORDER>
<TH>Select</TH><TH>Minor Allele Frequency</TH><TH>Locus</TH>
<%
Object[] arr2 = arr.toArray();
Arrays.sort(arr2);
size = arr2.length - 1;
while(size > -1)
{
    thislocus = (String) arr2[size];
    System.out.println(thislocus);
    %><TR><TD><Input type="checkbox" name = "loci" value = "<%=thislocus.substring(thislocus.lastIndexOf("<TD>") + 4)%>"></TD><TD><%= thislocus %></TD>
<%
--size;
}
%>
</TABLE>
<Input type = "submit">
</FORM>
</P>
</BODY>

svgsource.jsp

<%@ page language="java" import="java.util.*" import="java.io.*" import="haplotyper.*" %>
<%
String locus = request.getParameter("Locus");
TreeMap tmap = haplotyper.getLocList(locus, "//localhost:1731");
Iterator iter = tmap.keySet().iterator();
int sz = tmap.keySet().size() * 50;
String thislocus = null;
%>
<svg width = "<%=sz%>" height = "50" xmlns="http://www.w3.org/2000/svg">
<g stroke = "black"><line x1 = "0" y1 = "25" x2= "<%=sz%>" y2 = "25" /></g>
<rect width = "5" height = "5" fill = "green"></rect>
<%
while(iter.hasNext())
{
    thislocus = (String) iter.next();
    Double thisfreq = (Double) tmap.get(thislocus);
    int pos = Integer.parseInt(thislocus.substring(thislocus.indexOf(":") + 1, thislocus.indexOf("_")));
    %>
    <g id = "f<%=thislocus %>"><rect width = "5" height = "5" fill = "red" transform = "translate(<%=pos/5 %>, 25)"></rect></g>
    <g style="display:none"><set attributeName="display" from="none" to="block" begin="f<%=thislocus%>.click" fill = "freeze"/> <set attributeName="display" from="block" to="none" begin="click" fill="freeze"/> <text opacity="1.0" x="<%= pos/5 %>" y = "35" font-size = "8"><%=thislocus + " " + thisfreq%></text> </g>
    <%
}
%>
</svg>
```

genuscrhaplo.jsp

```
<%@ page language="java" import="java.util.*" import="java.io.*" import="haplotyper.*" import="pedviewer.*"
import="java.sql.*"%>
<HTML>
<HEAD>
<TITLE>Select Loci for Haplotyping</TITLE>
<BODY>
<%
String param[] = request.getParameterValues("loci");
int killflag = 0;
String usertype = new String();
while(killflag < param.length)
{
    usertype = usertype + param[killflag] + ",";
    ++killflag;
}
usertype = haplotyper.defineUserHaplotype(usertype, "//localhost:1731");
System.out.println("this is the usertype = " + usertype);
Connection con = inbrcdcoeff建立连接("//localhost:1731");
int happy = haplotyper.calculateHaplotypes("//localhost:1731", usertype, false);
Statement st = con.createStatement();
ResultSet rs = st.executeQuery("select * from grip2.dbo.Haplotypes where HaplotypeID like '%" + usertype + "%' order by
AnimalId");
boolean next = rs.next();
%> Number of rows = <%= rs.getFetchSize()%>
<TABLE BORDER>
<TH>AnimalID</TH><TH>HaplotypeId</TH><TH>BlockId1</TH><TH>BlockId2</TH>
<%
while(next == true)
{
    String thisrow = "<TR><TD>" + rs.getString("AnimalID") + "</TD><TD>" + rs.getString("HaplotypeId") +
"</TD><TD>" + rs.getString("BlockId1") + "</TD><TD>" + rs.getString("BlockId2") + "</TD></TR>";
    %> <%= thisrow %>
    <%
    next = rs.next();
}
%>
</BODY>
</HTML>
```

gethaplotypes.jsp

```
<%@ page language="java" import="java.util.*" import="java.io.*" import="haplotyper.*" %>
<HTML>
<HEAD>
<TITLE>Results</TITLE>
<BODY>
<%
ArrayList individuals = null;
ArrayList loci = null;
ArrayList results = null;
boolean specific = false;
String indv = request.getParameter("Individuals");
if (indv.length() > 0)
{
    StringTokenizer str1 = new StringTokenizer(indv);
    individuals = new ArrayList();
    while(str1.hasMoreTokens())
    {
        individuals.add(str1.nextToken());
    }
}
String locus = request.getParameter("Loci");
if (locus.length() > 0)
{
    StringTokenizer str2 = new StringTokenizer(locus);
    loci = new ArrayList();
    while(str2.hasMoreTokens())
    {
        loci.add(str2.nextToken());
    }
}
```

```

        {
            loci.add(str2.nextToken());
        }
    }
    String spec = request.getParameter("Specific");
    if(spec != null)
    {
        specific = true;
    }
    results = haplosearcher.getHaplotypes(individuals, loci, specific);
    if(results != null)
    {
        Object[] arr = results.toArray();
        Arrays.sort(arr);
        int killflag = 0;
        %>
        Number of rows = <%=arr.length%><BR>
        <TABLE BORDER>

        <TH>AnimalID</TH><TH>HaplotypeID</TH><TH>BlockId1</TH><TH>BlockId2</TH><TH>Block1Alleles</TH><
        TH>Block2Alleles</TH>
        <%
        while(killflag < arr.length)
        {
            if(killflag == 0)
            {
                %>
                <%=(String) arr[killflag]%>
                <%
            }
            else
            {
                %>
                <%=(String) arr[killflag]%>
                <%
            }
            ++killflag;
        }
        %
    }
    %>
</P>
</BODY>

```

APPENDIX 2: HapMap Test Data Set and Linkage File for Haplotype Inference Testing

HapMap Data Set (Alleles)											
SNPalleles	chrom	pos	NA06985	NA06991	NA06993	NA06994	NA07000	NA07019	NA07022	NA07029	NA07034
	NA07048	NA07055	NA07056	NA07345	NA07348	NA07357	NA10830	NA10831	NA10835	NA10838	NA10839
	NA10846	NA10847	NA10851	NA10854	NA10855	NA10856	NA10857	NA10859	NA10860	NA10861	NA10863
	NA11829	NA11830	NA11831	NA11832	NA11839	NA11840	NA11881	NA11882	NA11992	NA11993	NA11994
	NA11995	NA12003	NA12004	NA12005	NA12006	NA12043	NA12044	NA12056	NA12057	NA12144	NA12145
	NA12146	NA12154	NA12155	NA12156	NA12234	NA12236	NA12239	NA12248	NA12249	NA12264	NA12707
	NA12716	NA12717	NA12740	NA12750	NA12751	NA12752	NA12753	NA12760	NA12761	NA12762	NA12763
	NA12801	NA12802	NA12812	NA12813	NA12814	NA12815	NA12864	NA12865	NA12872	NA12873	NA12874
	NA12875	NA12878	NA12891	NA12892							
C/T	Chr10	26765	CT	CT	CT	CT	CC	CT	CC	CT	CC
	CC	CC	CT	CT	CT	CC	CT	CC	CT	CT	CT
	CC	CC	CT	CC	CC	CC	CC	CC	CT	CT	CC
	CC	CT	CC	CC	CC	CT	CC	CC	CC	CT	CT
	CC	CT	CC	CC	CT	CC	CC	CT	CC	CC	CC
	CC	CC	CC	CC	CT	CC	CC	CT	CC	CC	CC
	CT	CC	CC	CC	CC	CT	CC	CC	CT	CC	CC
	TT	CT	TT	CT	CC	CT	CC	CC	CT	CC	NN
	CC	CT	CT	CC							

C/T	Chr10	32380	TT									
	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	TT	TT	TT	TT	NN	NN	TT	TT	TT	TT	TT	NN
	NN	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
C/T	Chr10	49949	CC									
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
C/T	Chr10	52277	TT									
	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
C/T	Chr10	57317	CC									
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
A/C	Chr10	58178	CC									
	CC	CC	NN	CC								
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
C/T	Chr10	64658	CC									
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
A/G	Chr10	65955	AG	AG	AA							
	AA	AA	AA	NN	AA							
	AA	AA	AA	AA	AA	AA	AG	AA	AA	AA	AA	AA
	AA	AG	AA									
	AA	AG	AA	AG								
	AA	AG	AA	AA	AG	AA						
	AA	AG	AA	AG	AA							
	AG	AA	AA	AA	AA	AA	AA	AA	AG	AA	AA	AA
C/T	Chr10	67224	TT	TT	TT	CT	CT	CT	CT	CT	CT	TT
	CT	CC	TT	NN	CT	CC	TT	CT	TT	CT	TT	CT
	CT	TT	CT	CT	CT	TT	TT	CC	CT	TT	CC	CT
	CT	TT	TT	CC	TT	CT	CT	CC	CT	CC	CC	CT
	CT	TT	TT	CC	TT	CT	CT	CT	CT	CC	CC	TT
	TT	TT	CT	TT	CT	TT	CT	CT	CT	CT	TT	TT
	TT	TT	CC	CT	CT	CT	TT	CT	TT	TT	TT	TT

		GG	GG	GG	GG	GG	GG	GG	GG	GG	GG	GG
		GG	GG	GG	GG	GG	GG	GG	GG	GG	GG	GG
		GG	GG	GG	GG	GG	GG	GG	GG	GG	GG	NN
A/G	Chr10	88767	AG	GG	GG	AA	AG	AA	AG	AA	AG	GG
		AG	AA	AA	AA	AA	AG	AG	AG	GG	AG	AA
		AG	GG	AA	AG	AG	GG	AG	AA	AG	AG	AG
		AG	AG	GG	AA	GG	AG	AA	AA	AG	AA	AA
		AG	AG	GG	AA	AG	AG	AA	AA	AA	AA	GG
		GG	GG	AG	GG	AG	AA	AG	AG	AG	GG	GG
		AG	GG	AA	AG	AG	AA	GG	AG	AG	GG	GG
		AA	AG	AA	AG	GG	AG	AG	GG	AG	AG	AG
		GG	AG	AG	GG							
C/G	Chr10	97076	CG	CC	CC	CG	CC	CG	CC	CG	CC	CC
		CC	CC	CG	CG	CC	CG	CC	CC	CG	CG	CG
		CC	CC	CG	CC	CC	CC	CC	CC	CG	CC	CC
		CC	CG	CC	CC	CC	CG	CC	CC	CC	CG	CG
		CC	CG	CC	CC	CG	CC	CC	CG	CC	CC	CC
		CC	CC	CC	CC	GG	CC	CC	CC	CC	CC	CC
		CG	CC	CC	CC	CG	CC	CC	CG	CC	CC	CC
		CG	CG	CG	CC	CG	CC	CC	CC	CG	CC	CC
		CC	CG	CG	CC	CG	CC	CC	CC	CG	CC	CC
A/G	Chr10	102325	AA									
		AA	AA	AG	AA							
		AA	AA	AA	AA	AA	AA	AG	AA	AA	AA	AG
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
		AA	AA	AA	AA	AA	AG	AA	AA	AA	AA	AA
		AA	AA	AG	AA	AG						
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
C/T	Chr10	103173	CC									
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
C/T	Chr10	103465	CT	CT	TT							
		TT	TT	TT	TT	TT	TT	TT	TT	CT	TT	TT
		TT	TT	TT	TT	TT	CT	TT	TT	TT	TT	TT
		TT	CT	TT								
		TT	CT	TT	CT							
		TT	CT	TT	TT	CT	TT	TT	TT	CT	TT	TT
		TT	CT	TT	CT	TT						
		TT	TT	TT	TT	TT	TT	TT	CT	TT	TT	TT
G/T	Chr10	108552	TT									
		TT	TT	GT	TT							
		TT	TT	TT	TT	TT	TT	GT	TT	TT	TT	GT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	GT	TT	TT	TT	TT	TT
		TT	TT	GT	TT	GT						
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
A/T	Chr10	111441	TT	TT	TT	AT						
		AT	AT	TT	NN	AT	AA	TT	AT	TT	TT	AT
		AT	TT	AT	AT	AA	TT	TT	AA	AT	TT	AT
		AT	TT	AT	AA	TT	AT	AT	AA	TT	AA	AT
		AT	TT	TT	AA	TT	AT	AT	AA	AA	AA	TT
		TT	TT	AT	TT	AT	TT	AT	AT	AT	TT	TT
		TT	TT	AA	AT	AT	AT	TT	AT	TT	TT	TT
		TT	TT	TT	TT	TT	AT	AT	TT	AT	TT	AT
		TT	TT	TT	TT	TT	TT	AT	TT	AT	TT	AT
A/G	Chr10	113076	AG	AG	GG	AG						
		AG	AG	AG	AG	AA	AG	AG	AG	AG	GG	AG
		AG	GG	AG	AG	AA	AG	AG	AA	AG	GG	AA

		AG	AG	AG	AA	GG	AG	AG	AA	GG	AA	AG
		AG	AG	GG	AA	GG	AA	AG	AG	AA	AA	AG
		GG	AG	AA	GG	AA	GG	AG	AG	AA	AG	GG
		GG	AG	AA	AG	AG	AG	GG	AG	GG	AG	GG
		GG	GG	GG	GG	GG	GG	AG	AG	AG	GG	AG
A/G	Chr10	116271	GG									
		GG	GG	AG	GG							
		GG	GG	GG	GG	GG	GG	AG	GG	GG	GG	AG
		GG	GG	GG	GG	GG	GG	GG	GG	GG	GG	GG
		GG	GG	GG	GG	GG	GG	AG	GG	GG	GG	GG
		GG	GG	AG	GG	GG	GG	GG	GG	GG	AG	GG
		GG	GG	GG	GG	GG	GG	GG	GG	GG	GG	GG
		GG	GG	GG	NN	GG						
		GG	GG	GG	GG	GG	GG	GG	GG	GG	GG	GG
C/T	Chr10	117536	TT									
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
C/T	Chr10	117707	CC									
		CC	CC	CT	CC							
		CC	CC	CC	CC	CC	CC	CT	CC	CC	CC	CT
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CT	CC	CC	CC	CC
		CC	CC	CT	CC	CC	CC	CC	CC	CC	CT	CC
		CC	CC	CC	CC	NN	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
A/C	Chr10	118417	AA									
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
		AA	AA	AA	AC	AA						
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
C/T	Chr10	122814	TT									
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	NN
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
G/T	Chr10	123560	TT									
		TT	TT	GT	TT							
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	GT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	GT	TT	GT						
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
		TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
A/G	Chr10	125126	AA									
		AA	AA	AG	AA							
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AG
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
		AA	AA	AG	AA	AG						
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
		AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
C/T	Chr10	129653	CC	CC	CC	CT	CC	CC	CC	CC	CC	CC

		CT	CT	CC	CT	CT	CT	CC	CC	CC	CC	CT
		CC	CC	CC	CT	CC						
		CT	CC	CC	CT	CC	CT	CC	CT	CC	CC	CT
		CT	CC	CC	CT	CC	CC	CC	CT	CC	CC	CC
		CC	CC	CC	CC	NN	CC	CT	CT	CT	CC	CC
		CC	CC	TT	CT	CT	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CT	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CT	CC	CC	CC	CT
G/T	Chr10	132434	GG	GG	GT	GG	GG	GG	GT	GG	GG	GG
		GG	GG	GT	GG	GG	GT	GG	GT	GT	GG	GG
		GT	TT	GT	GT	GT	TT	GG	GT	GT	GT	GT
		GT	GG	GG	GT	TT	GT	GG	TT	GG	GG	GG
		GT	GG	TT	GG	GG	GT	GG	GT	GG	GT	GT
		GT	GT	GT	GT	GG	GT	GT	GT	GG	GT	TT
		GT	GT	GT	GT	GG	GT	GT	GT	GT	GG	TT
		GG	GT	GT	GG	GT	GT	GG	GT	GG	GG	GT
		GT	GT	GG	GT	GT	GG	GT	GG	GG	GT	GT
C/G	Chr10	136783	CC									
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
A/G	Chr10	140795	GG	GG	AG	GG	GG	GG	AG	GG	GG	GG
		GG	GG	AG	GG	GG	AG	GG	AG	AG	AG	GG
		AG	AA	AG	AG	AG	AG	AA	GG	AG	AG	AG
		AG	GG	GG	AG	AA	AG	GG	AA	GG	GG	GG
		AG	GG	AA	GG	GG	AG	AG	GG	AG	GG	AG
		AG	AG	AG	AG	NN	GG	AG	AG	GG	AA	AA
		AG	AG	GG	AG	AG	GG	AG	AG	GG	AA	AA
		GG	AG	AG	GG	AG	AG	GG	AG	GG	GG	AG
		AG	AG	GG	AG	AG	GG	AG	AG	GG	GG	AG
A/G	Chr10	142805	AA	AA	AA	AG	AA	AA	AA	AA	AA	AA
		AG	AG	AA	NN	AG	AG	AA	AA	AA	AA	AG
		AA	AA	AA	AG	AA						
		AG	AA	AA	AG	AA	AG	AA	AG	AA	AA	AG
		AG	AA	AA	AG	AA	AA	AA	AG	AA	AA	AA
		AA	AA	AA	AA	AA	AA	AG	AG	AG	AA	AA
		AA	AA	GG	AG	AG	AA	AA	AA	AA	AA	AA
		AA	AA	AA	AA	AA	AG	AA	AA	AA	AA	AA
		AA	AA	AA	AA	AA	AG	AA	AG	AA	AA	AG
C/T	Chr10	144851	CC									
		CC	CC	CT	CC							
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CT
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CT	CC							
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
		CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
A/G	Chr10	149100	GG									
		GG	GG	GG	NN	GG						
		GG	GG	GG	GG	GG	GG	GG	GG	GG	GG	GG
		GG	GG	GG	GG	GG	GG	GG	GG	GG	GG	GG
		GG	GG	GG	GG	GG	GG	GG	GG	GG	GG	GG
		GG	GG	GG	GG	GG	GG	GG	GG	GG	GG	GG
		GG	GG	GG	GG	GG	GG	GG	GG	GG	GG	GG
		GG	GG	GG	GG	GG	GG	GG	GG	GG	GG	GG
A/G	Chr10	156543	GG	GG	AG	GG	GG	GG	AG	GG	AG	AG
		GG	GG	AG	GG	GG	AG	AG	AG	AG	AG	GG
		AG	AA	AG	AG	AG	AG	AA	GG	AG	AG	AG
		AG	GG	GG	AG	AA	AG	GG	GG	AA	GG	GG
		AG	GG	AA	GG	GG	AG	AG	GG	AG	GG	AG
		AG	AG	AG	AA	GG	GG	AG	AG	GG	AA	AA
		AG	AG	GG	AG	AG	GG	AA	AG	AG	AG	AA
		GG	AG	AG	GG	AG						

	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	NN
A/G	Chr10	179659	GG	GG	AG	GG	GG	GG	AG	GG	AG
	GG	GG	AG	GG	GG	AG	AG	AG	AG	AG	GG
	AG	AA	AG	AG	AG	AA	GG	AG	AG	AG	AG
	AG	GG	GG	AG	AA	AG	GG	GG	AA	GG	GG
	AG	GG	AA	GG	AG	AG	AG	GG	AG	GG	AG
	AG	AG	AG	AA	GG	GG	AG	AG	GG	AA	AA
	AG	AG	GG	AG	AG	GG	AA	AG	AG	AG	AA
	GG	AG	AG	GG	AG						
	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG

HapMap Pedigree File

1420 9 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1420.09:1
urn:lsid:dcc.hapmap.org:Sample:NA12003:1

1420 10 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1420.10:1
urn:lsid:dcc.hapmap.org:Sample:NA12004:1

1420 1 9 10 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1420.01:1
urn:lsid:dcc.hapmap.org:Sample:NA10838:1

1420 11 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1420.11:1
urn:lsid:dcc.hapmap.org:Sample:NA12005:1

1420 12 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1420.12:1
urn:lsid:dcc.hapmap.org:Sample:NA12006:1

1420 2 11 12 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1420.02:1
urn:lsid:dcc.hapmap.org:Sample:NA10839:1

1344 12 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1344.12:1
urn:lsid:dcc.hapmap.org:Sample:NA12056:1

1344 13 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1344.13:1
urn:lsid:dcc.hapmap.org:Sample:NA12057:1

1344 1 12 13 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1344.01:1
urn:lsid:dcc.hapmap.org:Sample:NA10851:1

1350 10 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1350.10:1
urn:lsid:dcc.hapmap.org:Sample:NA11829:1

1350 11 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1350.11:1
urn:lsid:dcc.hapmap.org:Sample:NA11830:1

1350 1 10 11 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1350.01:1
urn:lsid:dcc.hapmap.org:Sample:NA10856:1

1350 12 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1350.12:1
urn:lsid:dcc.hapmap.org:Sample:NA11831:1

1350 13 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1350.13:1
urn:lsid:dcc.hapmap.org:Sample:NA11832:1

1350 2 12 13 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1350.02:1
urn:lsid:dcc.hapmap.org:Sample:NA10855:1

1362 13 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1362.13:1
urn:lsid:dcc.hapmap.org:Sample:NA11992:1

1362 14 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1362.14:1
urn:lsid:dcc.hapmap.org:Sample:NA11993:1

1362 1 13 14 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1362.01:1
urn:lsid:dcc.hapmap.org:Sample:NA10860:1

1362 15 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1362.15:1
urn:lsid:dcc.hapmap.org:Sample:NA11994:1

1362 16 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1362.16:1
urn:lsid:dcc.hapmap.org:Sample:NA11995:1

1362 2 15 16 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1362.02:1
urn:lsid:dcc.hapmap.org:Sample:NA10861:1

1408 10 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1408.10:1
urn:lsid:dcc.hapmap.org:Sample:NA12154:1

1408 11 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1408.11:1
urn:lsid:dcc.hapmap.org:Sample:NA12236:1

1408 1 10 11 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1408.01:1
urn:lsid:dcc.hapmap.org:Sample:NA10830:1

1408 12 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1408.12:1
urn:lsid:dcc.hapmap.org:Sample:NA12155:1

1408 13 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1408.13:1
urn:lsid:dcc.hapmap.org:Sample:NA12156:1

1408 2 12 13 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1408.02:1
urn:lsid:dcc.hapmap.org:Sample:NA10831:1

1345 12 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1345.12:1
urn:lsid:dcc.hapmap.org:Sample:NA07357:1

1345 13 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1345.13:1
 urn:lsid:dcc.hapmap.org:Sample:NA07345:1
 1345 2 12 13 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1345.02:1
 urn:lsid:dcc.hapmap.org:Sample:NA07348:1
 1340 9 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1340.09:1
 urn:lsid:dcc.hapmap.org:Sample:NA06994:1
 1340 10 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1340.10:1
 urn:lsid:dcc.hapmap.org:Sample:NA07000:1
 1340 1 9 10 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1340.01:1
 urn:lsid:dcc.hapmap.org:Sample:NA07029:1
 1340 11 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1340.11:1
 urn:lsid:dcc.hapmap.org:Sample:NA07022:1
 1340 12 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1340.12:1
 urn:lsid:dcc.hapmap.org:Sample:NA07056:1
 1340 2 11 12 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1340.02:1
 urn:lsid:dcc.hapmap.org:Sample:NA07019:1
 1349 13 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1349.13:1
 urn:lsid:dcc.hapmap.org:Sample:NA11839:1
 1349 14 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1349.14:1
 urn:lsid:dcc.hapmap.org:Sample:NA11840:1
 1349 2 13 14 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1349.02:1
 urn:lsid:dcc.hapmap.org:Sample:NA10854:1
 1346 11 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1346.11:1
 urn:lsid:dcc.hapmap.org:Sample:NA12043:1
 1346 12 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1346.12:1
 urn:lsid:dcc.hapmap.org:Sample:NA12044:1
 1346 1 11 12 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1346.01:1
 urn:lsid:dcc.hapmap.org:Sample:NA10857:1
 1334 10 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1334.10:1
 urn:lsid:dcc.hapmap.org:Sample:NA12144:1
 1334 11 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1334.11:1
 urn:lsid:dcc.hapmap.org:Sample:NA12145:1
 1334 1 10 11 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1334.01:1
 urn:lsid:dcc.hapmap.org:Sample:NA10846:1
 1334 12 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1334.12:1
 urn:lsid:dcc.hapmap.org:Sample:NA12146:1
 1334 13 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1334.13:1
 urn:lsid:dcc.hapmap.org:Sample:NA12239:1
 1334 2 12 13 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1334.02:1
 urn:lsid:dcc.hapmap.org:Sample:NA10847:1
 1375 11 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1375.11:1
 urn:lsid:dcc.hapmap.org:Sample:NA12264:1
 1375 12 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1375.12:1
 urn:lsid:dcc.hapmap.org:Sample:NA12234:1
 1375 2 11 12 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1375.02:1
 urn:lsid:dcc.hapmap.org:Sample:NA10863:1
 1358 11 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1358.11:1
 urn:lsid:dcc.hapmap.org:Sample:NA12716:1
 1358 12 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1358.12:1
 urn:lsid:dcc.hapmap.org:Sample:NA12717:1
 1358 1 11 12 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1358.01:1
 urn:lsid:dcc.hapmap.org:Sample:NA12707:1
 1463 15 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1463.15:1
 urn:lsid:dcc.hapmap.org:Sample:NA12891:1
 1463 16 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1463.16:1
 urn:lsid:dcc.hapmap.org:Sample:NA12892:1
 1463 2 15 16 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1463.02:1
 urn:lsid:dcc.hapmap.org:Sample:NA12878:1
 1454 12 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1454.12:1
 urn:lsid:dcc.hapmap.org:Sample:NA12812:1
 1454 13 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1454.13:1
 urn:lsid:dcc.hapmap.org:Sample:NA12813:1
 1454 1 12 13 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1454.01:1
 urn:lsid:dcc.hapmap.org:Sample:NA12801:1
 1454 14 0 0 1 urn:lsid:dcc.hapmap.org:Individual:CEPH1454.14:1
 urn:lsid:dcc.hapmap.org:Sample:NA12814:1
 1454 15 0 0 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1454.15:1
 urn:lsid:dcc.hapmap.org:Sample:NA12815:1
 1454 2 14 15 2 urn:lsid:dcc.hapmap.org:Individual:CEPH1454.02:1
 urn:lsid:dcc.hapmap.org:Sample:NA12802:1

1459 9 0 0 1 urn:lsid:dec.hapmap.org:Individual:CEPH1459.09:1
urn:lsid:dec.hapmap.org:Sample:NA12872:1
1459 10 0 0 2 urn:lsid:dec.hapmap.org:Individual:CEPH1459.10:1
urn:lsid:dec.hapmap.org:Sample:NA12873:1
1459 1 9 10 1 urn:lsid:dec.hapmap.org:Individual:CEPH1459.01:1
urn:lsid:dec.hapmap.org:Sample:NA12864:1
1459 11 0 0 1 urn:lsid:dec.hapmap.org:Individual:CEPH1459.11:1
urn:lsid:dec.hapmap.org:Sample:NA12874:1
1459 12 0 0 2 urn:lsid:dec.hapmap.org:Individual:CEPH1459.12:1
urn:lsid:dec.hapmap.org:Sample:NA12875:1
1459 2 11 12 2 urn:lsid:dec.hapmap.org:Individual:CEPH1459.02:1
urn:lsid:dec.hapmap.org:Sample:NA12865:1
1447 9 0 0 1 urn:lsid:dec.hapmap.org:Individual:CEPH1447.09:1
urn:lsid:dec.hapmap.org:Sample:NA12760:1
1447 10 0 0 2 urn:lsid:dec.hapmap.org:Individual:CEPH1447.10:1
urn:lsid:dec.hapmap.org:Sample:NA12761:1
1447 1 9 10 1 urn:lsid:dec.hapmap.org:Individual:CEPH1447.01:1
urn:lsid:dec.hapmap.org:Sample:NA12752:1
1447 11 0 0 1 urn:lsid:dec.hapmap.org:Individual:CEPH1447.11:1
urn:lsid:dec.hapmap.org:Sample:NA12762:1
1447 12 0 0 2 urn:lsid:dec.hapmap.org:Individual:CEPH1447.12:1
urn:lsid:dec.hapmap.org:Sample:NA12763:1
1447 2 11 12 2 urn:lsid:dec.hapmap.org:Individual:CEPH1447.02:1
urn:lsid:dec.hapmap.org:Sample:NA12753:1
1341 11 0 0 1 urn:lsid:dec.hapmap.org:Individual:CEPH1341.11:1
urn:lsid:dec.hapmap.org:Sample:NA07034:1
1341 12 0 0 2 urn:lsid:dec.hapmap.org:Individual:CEPH1341.12:1
urn:lsid:dec.hapmap.org:Sample:NA07055:1
1341 1 11 12 1 urn:lsid:dec.hapmap.org:Individual:CEPH1341.01:1
urn:lsid:dec.hapmap.org:Sample:NA07048:1
1341 13 0 0 1 urn:lsid:dec.hapmap.org:Individual:CEPH1341.13:1
urn:lsid:dec.hapmap.org:Sample:NA06993:1
1341 14 0 0 2 urn:lsid:dec.hapmap.org:Individual:CEPH1341.14:1
urn:lsid:dec.hapmap.org:Sample:NA06985:1
1341 2 13 14 2 urn:lsid:dec.hapmap.org:Individual:CEPH1341.02:1
urn:lsid:dec.hapmap.org:Sample:NA06991:1
1444 13 0 0 1 urn:lsid:dec.hapmap.org:Individual:CEPH1444.13:1
urn:lsid:dec.hapmap.org:Sample:NA12750:1
1444 14 0 0 2 urn:lsid:dec.hapmap.org:Individual:CEPH1444.14:1
urn:lsid:dec.hapmap.org:Sample:NA12751:1
1444 2 13 14 2 urn:lsid:dec.hapmap.org:Individual:CEPH1444.02:1
urn:lsid:dec.hapmap.org:Sample:NA12740:1
1347 14 0 0 1 urn:lsid:dec.hapmap.org:Individual:CEPH1347.14:1
urn:lsid:dec.hapmap.org:Sample:NA11881:1
1347 15 0 0 2 urn:lsid:dec.hapmap.org:Individual:CEPH1347.15:1
urn:lsid:dec.hapmap.org:Sample:NA11882:1
1347 2 14 15 2 urn:lsid:dec.hapmap.org:Individual:CEPH1347.02:1
urn:lsid:dec.hapmap.org:Sample:NA10859:1
1416 11 0 0 1 urn:lsid:dec.hapmap.org:Individual:CEPH1416.11:1
urn:lsid:dec.hapmap.org:Sample:NA12248:1
1416 12 0 0 2 urn:lsid:dec.hapmap.org:Individual:CEPH1416.12:1
urn:lsid:dec.hapmap.org:Sample:NA12249:1
1416 1 11 12 1 urn:lsid:dec.hapmap.org:Individual:CEPH1416.01:1
urn:lsid:dec.hapmap.org:Sample:NA10835:1

Generated Linkage File (Leading NA in HapMap ID replaced with leading 9. Loci represented are at positions 67224, 67934, and 73893, respectively)

1	912865 1	912874	912875	2	0	0	2	2	1	2	1
1	912875 1	0	0	2	0	0	2	2	1	2	1
1	912874 0	0	0	1	0	0	1	2	0	0	0
2	912864 2	912872	912873	1	0	0	1	2	1	2	1
2	912872 2	0	0	1	0	0	1	2	1	2	1
2	912873 1	0	0	2	0	0	2	2	1	2	1

3	912753 2	912762	912763	2	0	0	2	2	2	2	2	1
3	912763 2	0	0	2	0	0	2	2	2	2	2	1
3	912762 1	0	0	1	0	0	2	2	1	2	2	1
4	912815 1	0	0	2	0	0	2	2	1	2	2	1
4	912814 1	0	0	1	0	0	2	2	2	2	2	1
4	912802 1	912814	912815	2	0	0	2	2	1	2	2	1
5	912006 1	0	0	2	0	0	2	2	1	2	2	1
5	912005 2	0	0	1	0	0	1	1	1	1	1	2
5	910839 2	912005	912006	2	0	0	1	2	1	1	1	1
6	910856 1	911829	911830	1	0	0	2	2	1	2	2	1
6	911829 2	0	0	1	0	0	1	2	1	2	2	1
6	911830 1	0	0	2	0	0	2	2	1	1	1	1
7	912761 1	0	0	2	0	0	2	2	1	2	2	1
7	912752 2	912760	912761	1	0	0	1	2	1	1	1	1
7	912760 2	0	0	1	0	0	1	2	1	2	2	1
8	912056 2	0	0	1	0	0	1	2	1	1	1	1
8	912057 2	0	0	2	0	0	1	1	1	1	1	2
8	910851 2	912056	912057	1	0	0	1	2	1	1	1	1
9	912801 1	912812	912813	1	0	0	2	2	1	1	1	1
9	912812 1	0	0	1	0	0	2	2	1	1	1	1
9	912813 1	0	0	2	0	0	2	2	1	2	2	1
10	912892 1	0	0	2	0	0	2	2	1	2	2	1
10	912878 1	912891	912892	2	0	0	2	2	1	2	2	1
10	912891 1	0	0	1	0	0	2	2	1	2	2	1
11	910847 1	912146	912239	2	0	0	2	2	2	2	2	1
11	912146 1	0	0	1	0	0	2	2	2	2	2	1
11	912239 2	0	0	2	0	0	1	2	1	2	2	1
12	910854 2	911839	911840	2	0	0	1	2	1	2	2	1
12	911840 2	0	0	2	0	0	1	2	1	2	2	1
12	911839 1	0	0	1	0	0	2	2	2	2	2	1
14	910835 1	912248	912249	1	0	0	2	2	1	2	2	1
14	912248 2	0	0	1	0	0	1	2	1	2	2	1
14	912249 2	0	0	2	0	0	1	2	1	1	1	1
15	911992 2	0	0	1	0	0	1	2	1	2	2	1
15	911993 2	0	0	2	0	0	1	1	1	1	1	2

15	910860 2	911992	911993	1	0	0	1	2	1	2	1
16	910846 2	912144	912145	1	0	0	1	2	1	2	1
16	912145 1	0	0	2	0	0	2	2	1	2	1
16	912144 2	0	0	1	0	0	1	1	1	1	2
17	912717 1	0	0	2	0	0	2	2	1	2	1
17	912707 1	912716	912717	1	0	0	2	2	2	2	1
17	912716 1	0	0	1	0	0	2	2	1	2	1
18	906985 1	0	0	2	0	0	2	2	1	1	1
18	906993 1	0	0	1	0	0	2	2	1	2	1
18	906991 1	906993	906985	2	0	0	2	2	1	1	1
20	910857 1	912043	912044	1	0	0	2	2	1	1	1
20	912043 2	0	0	1	0	0	1	2	1	1	1
20	912044 2	0	0	2	0	0	1	2	1	1	1
21	907348 2	907357	907345	2	0	0	1	2	1	1	1
21	907357 2	0	0	1	0	0	1	1	1	1	2
21	907345 2	0	0	2	0	0	0	0	1	1	1
22	911882 2	0	0	2	0	0	1	1	1	1	2
22	911881 2	0	0	1	0	0	1	2	1	1	1
22	910859 2	911881	911882	2	0	0	1	1	1	1	2
23	911994 2	0	0	1	0	0	1	2	1	1	1
23	910861 1	911994	911995	2	0	0	2	2	1	2	1
23	911995 2	0	0	2	0	0	1	2	1	2	1
24	907034 1	0	0	1	0	0	2	2	2	2	1
24	907048 2	907034	907055	1	0	0	1	2	1	2	1
24	907055 2	0	0	2	0	0	1	1	1	1	2
25	912154 1	0	0	1	0	0	2	2	1	2	1
25	910830 1	912154	912236	1	0	0	2	2	1	2	1
25	912236 1	0	0	2	0	0	2	2	1	1	1
26	907019 2	907022	907056	2	0	0	1	2	1	1	1
26	907056 1	0	0	2	0	0	2	2	1	1	1
26	907022 2	0	0	1	0	0	1	2	1	2	1
28	911832 2	0	0	2	0	0	1	1	1	1	2
28	911831 1	0	0	1	0	0	2	2	1	2	1
28	910855 2	911831	911832	2	0	0	1	2	1	2	1
33	912264 1	0	0	1	0	0	2	2	1	2	1

33	910863 2	912264	912234	2	0	0	1	2	1	1	1
33	912234 2	0	0	2	0	0	1	2	1	1	1
34	910831 2	912155	912156	2	0	0	1	2	1	2	1
34	912155 2	0	0	1	0	0	1	2	1	1	1
34	912156 1	0	0	2	0	0	2	2	2	2	1
35	907029 2	906994	907000	1	0	0	1	2	1	1	1
35	906994 2	0	0	1	0	0	1	2	1	1	1
35	907000 2	0	0	2	0	0	1	2	1	2	1
36	912740 2	912750	912751	2	0	0	1	1	1	1	1
36	912750 2	0	0	1	0	0	1	2	1	2	1
36	912751 2	0	0	2	0	0	1	2	1	2	1
37	912003 1	0	0	1	0	0	2	2	1	1	1
37	912004 1	0	0	2	0	0	2	2	2	2	1
37	910838 1	912003	912004	1	0	0	2	2	1	2	1