5-24-2006

Identifying the largest complete data set from ALFRED

Mohamed Uduman

THESIS

IDENTIFYING THE LARGEST COMPLETE

DATA SET FROM ALFRED

by

Mohamed Uduman

Submitted in partial fulfillment of the requirements for the

Masters of Science degree

in

Bioinformatics

at

Rochester Institute of Technology

THESIS ADVISORY COMMITTEE

1. Committee Chair

   Dr. Michael V. Osier
   Assistant Professor
   Department of Biological Sciences
   College of Science
   Rochester Institute of Technology

2. Committee Member

   Dr. Carl Reynolds
   Visiting Professor
   Department of Computer Science
   B. Thomas Golisano College of Computing and Information Sciences
   Rochester Institute of Technology

3. Committee Member

   Dr. James Halavin
   Professor
   Department of Mathematics & Statistics
   College of Science
   Rochester Institute of Technology

ABSTRACT

ALFRED is a central and curated repository for allele frequency data for anthropologically defined human populations. To study and estimate the relationships and similarities between populations, researchers require a large and complete data set. However, the data set within ALFRED is not complete. Specifically, not all the populations in the database have been typed for all the polymorphisms.

Mining ALFRED for the largest complete data set is equivalent to the "Maximal Biclique" problem in graph theory. This is proven to be NP-Complete and no single algorithm can find the perfect solution in polynomial time. This project describes a heuristic (Largest Maximal Biclique Heuristic) which finds the largest complete data set from ALFRED, in real time. The program is compared to various other methods, including Wen-Chieh Chang's implementation of the "maximal biclique" algorithm proposed by Alexe *et.al.*

The algorithm efficiently mines ALFRED to extract the largest complete data set, and the results are made available for researchers in uniform data exchange format, through a Web site. Since ALFRED is updated frequently, the LMBH program is set up to mine ALFRED on a regular basis and provide researchers with the most up-to-date, largest complete data set from ALFRED.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

1. INTRODUCTION & BACKGROUND

The ALlele FREquency Database, ALFRED, is a publicly accessible database that provides allele frequency data on anthropologically well defined, both geographically and ethnically, human population samples and DNA polymorphisms [1, 2]. Initially ALFRED was developed to maintain data generated by the Kidd Lab in the Department of Genetics at Yale University School of Medicine. However, a large volume of data has been and is being published in various diverse journals. It would be very difficult and virtually impossible for researchers to find all of the available allele frequency data for a specific population. Taking this into consideration, ALFRED expanded to include curated data from literature in the broad areas of medicine, genetics, forensics and anthropology [2].

ALFRED is heavily curated and maintained by the Kidd Lab. While other polymorphism databases exist, such as dbSNP [3] and HGVBase [4], ALFRED is unique because it is curated. As such, it is considered a "gold standard" data set by researchers since the contents are robust and clearly defined. The data is useful for researchers such as molecular anthropologists, human geneticists, and forensic scientists studying genetic variation among populations. As of May 2006 the database has over 1600 polymorphisms, 490 populations and 47900 frequencies. However, the data in ALFRED is not complete. In other words, not all of the populations in the database have allele frequency values for each and every polymorphism present in the database. Correspondingly, not every polymorphism has been typed in samples of every population.

The incompleteness of the data in ALFRED reflects the nature of research in the field. Researchers usually study a relatively small set of populations and genetic polymorphisms. Some populations or polymorphisms are well studied and a plethora of

allele frequency data is available for them, whereas most other populations and polymorphisms are rarely studied, and only very limited data is available. Figure 1.1 shows a representation of the sparse distribution of data in ALFRED. This is also known as the "empty matrix problem" [2].

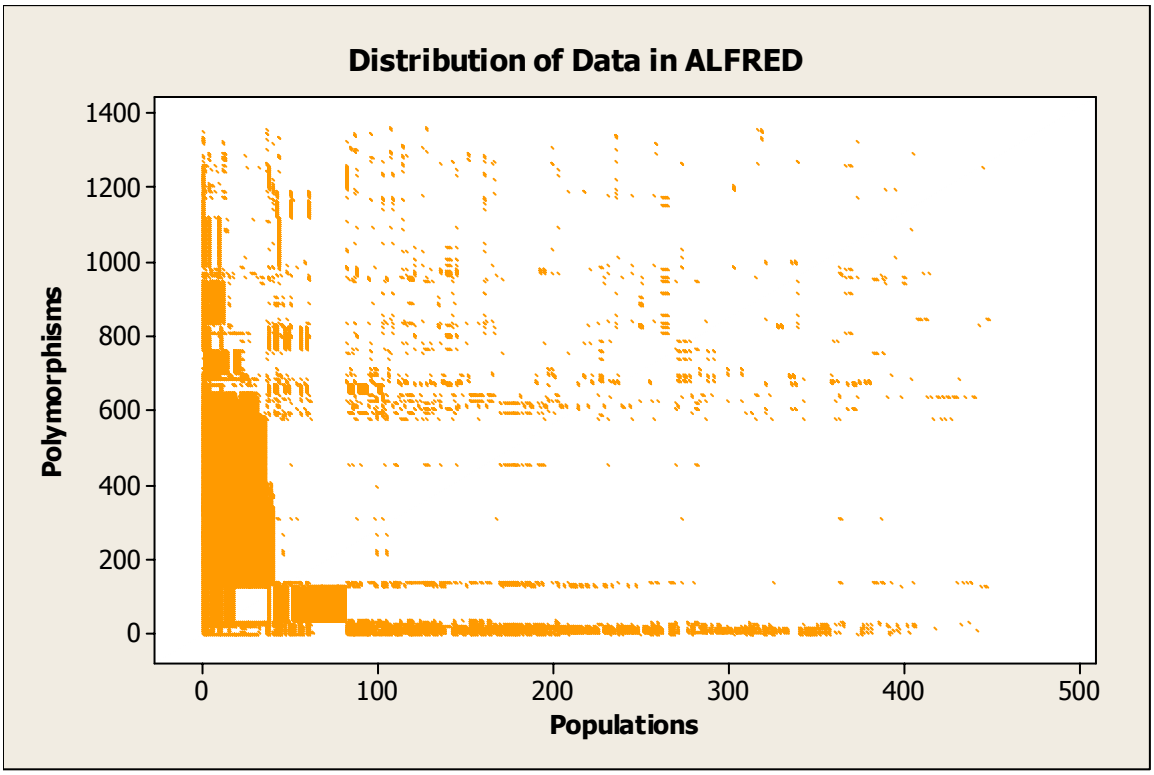

Figure 1.1: Distribution of data in ALFRED. Rows represent individual polymorphisms, columns individual populations.

To study and estimate the relationships and similarities between populations requires one to study as many polymorphisms as these populations have been commonly typed for [5]. Currently, many individual medical research labs only study a relatively small group of populations that they are interested in. Other groups, such as forensic scientists, require as large and complete a data set as possible in order to make educated choices when selecting the best markers to identify an individual. Physical anthropologists also require a large, complete data set when studying genetic similarities and evolutionary histories of human

populations. Finding the largest possible set of populations that have all been completely typed for the same set of polymorphisms, from polymorphisms databases such as ALFRED, would greatly facilitate such research.

The primary goal of this project was to extract from ALFRED the largest possible complete data set of populations that have all been typed for the largest possible set of polymorphisms (the largest complete data set). A complete subset of data is defined as a subset of ALFRED where all the populations have allele frequency information available for all the polymorphisms. There are multiple measures of size to appraise a complete subset. The measure used to assess a subset, in this project, was the area. The area of a complete subset is calculated as the number of populations multiplied by the number of polymorphism within the subset.

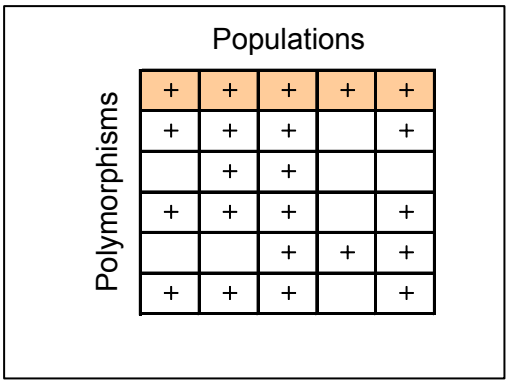

Figure 1.2: A hypothetical data matrix. Shaded area represents a complete data set.

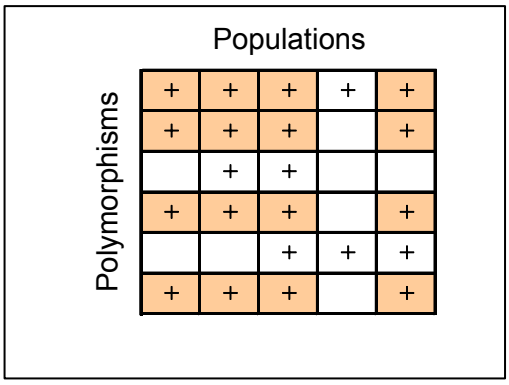

Figure 1.3: A hypothetical data matrix. Shaded area represents the largest complete data set.

To identify the maximal complete data set, the data in ALFRED can be represented as a matrix where each column and row corresponds to individual populations and polymorphisms respectively. Both Figure 1.2 and 1.3 illustrate a hypothetical data matrix, which also demonstrates the sparseness of the data in ALFRED. A "+" indicates that the population has been typed for the corresponding polymorphism. The shaded area in Figure

1.3 represents the largest possible complete set of populations that have all been typed for the largest possible set of polymorphisms, i.e. the largest complete data set in respect to total area. Even though 5 populations are available for study only 4 of them are typed for most polymorphisms. While a data set with all five populations could be chosen for the first polymorphism, as shown in Figure 1.2, it would not be appropriate for effective phylogenetic analysis due to reduced statistical power, i.e. it would provide little meaningful information about the populations.

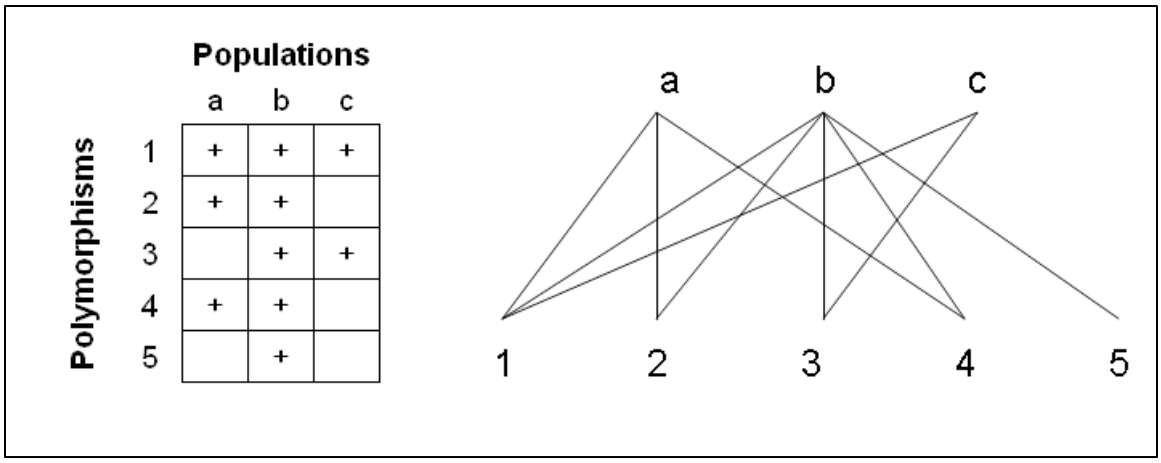

Figure 1.4: A hypothetical data matrix, and its corresponding bipartite graph.

The problem of finding the largest complete set from ALFRED is equivalent to the maximal biclique problem in graph theory [6]. The problem is also very similar in nature to the biclustering problem in bioinformatics that is used in microarray analysis [7]. A biclique is defined to be a sub-graph of the bipartite graph where all the nodes are connected. A graph can have many bicliques, however a maximal biclique is defined to be a biclique that cannot extend any further. In other words, that biclique cannot be a sub-graph of an even larger biclique. The largest maximal biclique would be the maximal biclique with the most number of nodes. The data-matrix from ALFRED can be represented as a bipartite graph, where the nodes are our individual populations and polymorphisms, and the connecting

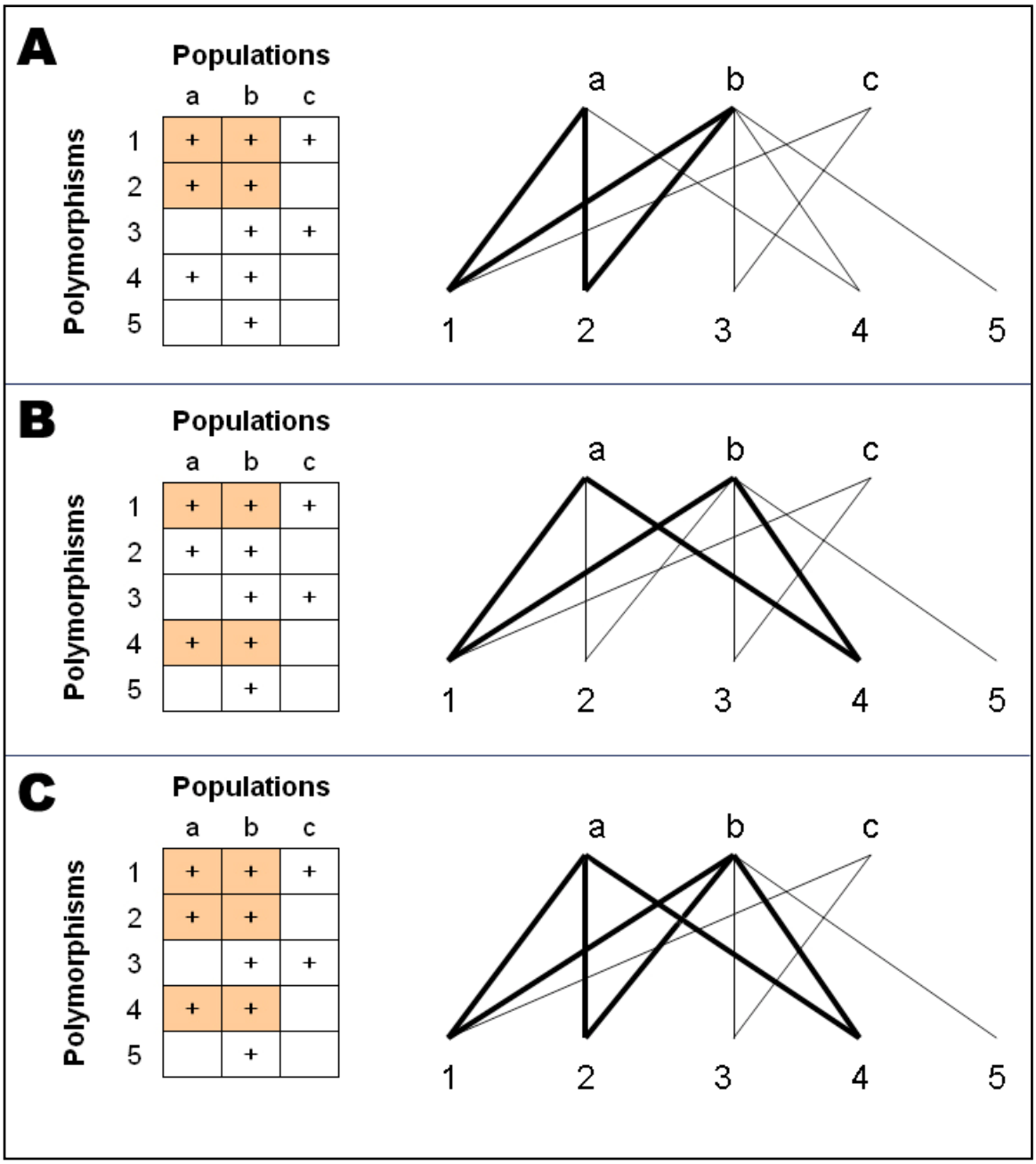

Figure 1.5: Bicliques shown shaded in the data matrix and bolded in the bipartite graph.

edges denote that a population has been typed for that polymorphism (Figure 1.4). Figure 1.5 shows three possible bicliques in the given data matrix. Bicliques A & B can be extended to be a larger biclique, however biclique C cannot be extend. Therefore, it is a maximal

biclique. In this case it is also the max-clique. The shaded area and the bolded lines in the sample data matrix and the graph respectively represent the corresponding bicliques.

The computational solution of the maximal biclique problem is proven to be NP-Complete [8, 9]. There is no single algorithm that can find the perfect solution in polynomial time. There is no single heuristic that one could use to solve NP-complete problems, and there is no existing heuristic that will give one the best answer. In recent years Genetic Algorithms (GA) have become popular solutions for NP-complete problems [10]. Hence, designing and implementing a GA to solve the problem is one possible solution to the problem. Based on a rough conception of evolution, Genetic Algorithms were introduced by John Holland in 1975 [11]. In evolution, a population begins with an initial set of genes, or traits. As generations progress, mating occurs within the population and the fittest genes survive and move on to the next generation. Mutation occurs randomly to produce newer genes which may or may not result in a better fitness. In principle the fittest of the population survive and pass on their genes to the subsequent generations, and the least fit are dropped out of the population. Computationally intensive problems are solved by iteratively "mating" potential solutions and checking to see how much more "fitter" the newer solutions can get. This school of thought is inspired by modern theories of evolution, hence the name Genetic Algorithm.

Given the nature of the biclique problem with the ALFRED data, an exponential time heuristic and a Genetic Algorithm were developed to solve this problem. To test the effectiveness of these algorithms, they were compared with Wen-Chieh Chang's implementation [12] of the "maximal biclique" algorithm [13]. Similar to evolution, the first step in designing a GA was to represent a solution to the problem as "chromosomes." These were not biological genetic chromosomes but mathematical entities. The process is

started with a set of initial chromosomes (or solutions). Simulating evolution, the individuals in the initial population are mated (crossed over) randomly and the resulting chromosomes are evaluated for fitness. If they yield a good solution, then they are added to the population. Otherwise, they are dropped from the population. To introduce newer solutions mutations are introduced randomly to the resulting population of solutions. This process is iterated over many generations until no further "fitter" solutions are obtained.

The measure used to determine the fitness of a solution was the area of the complete subset, i.e. the number of populations times the number of polymorphism present in the biclique. As mentioned earlier, there are multiple ways of assessing a complete subset – the fitness of the biclique. To better evaluate a subset favoring populations over loci, one can use a measure that places more emphasis on the number of populations and downplays the number of polymorphisms. This might be a better system of evaluating different subsets from a population geneticist's perspective.

With rapid accumulation of new polymorphism data, the ALFRED database is updated frequently; more distinct populations are added and new polymorphisms are typed for existing populations. As the data set from ALFRED grows, so does the potential for the largest complete data set to grow. Taking this into consideration, the project was extended to periodically and automatically mine the data from ALFRED to provide researchers with an up-to-date largest complete data set. Initially this was performed at regular calendar intervals. Now the project has been extended so that the ALFRED developers can automatically notify a Web Service at RIT to update the LMBH results.

The resulting largest complete data set is very valuable to physical anthropologists and other genetics researchers, who can now study the largest available set of populations with greater statistical confidence. This data set is presented in a Web site, freely accessible

to researchers, in both XHTML and XML formats. The program which performs this analysis is called Largest Maximal Biclique Heuristic (LMBH).

2. MATERIALS & METHODS

Data Analyzed

ALFRED can be accessed freely at http://alfred.med.yale.edu. The data from ALFRED is also freely available in XML [14] format. Extensible Markup Language (XML) is a metalanguage used for the easy interchange of data. To extract the data matrix, the XML is parsed using Java [15] and SAX (Simple API for XML) [16].

```
X X X X X X X X 0.911:0.089: X X
X X X X X X X X X X
X X X X X X X X X X
X X X X X X X X 0.901:0.099: X X
0.816:0.184: X 0.875:0.125: X X X X 0.683:0.317: X X
0.046:0.954: X 0.271:0.729: X X X X 0.500:0.500: X X
X X X X X X X X X X
```

Figure 2.1: A sample dataMatrix.txt file.

```
Athabaskan:PO000514K
Russians:PO000019K
Georgians:PO000342J
Bosnian:PO000593R
Ache:PO000410F
Toto:PO000338O
African Americans:PO000098R
Kogui:PO000170I
Tucano:PO000589W
Kabardinian:PO000406K
```

Figure 2.2: A sample populations.txt file. The population names are separated from their ALFRED ID by a colon.

SI001690Q
SI000249P
SI001111E
SI001364O
SI000792S
SI001038M
SI000213G
SI000466Q
SI001581P
SI001002D

Figure 2.3: Sample sites.txt file. Each row contains the site ID from ALFRED, for the corresponding row in the dataMatrix.txt file.

Once parsed, the data from the XML file is output by the Java program (XMLParser.java) as a tab-delimited text file (dataMatrix.txt) where the rows and columns represent the polymorphisms and the populations respectively. Figure 2.1 shows a sample dataMatrix.txt file. Two other separate text files identify the population (populations.txt) and polymorphism (sites.txt) information. Figure 2.2 and Figure 2.3 show the corresponding information files for the sample data matrix in Figure 2.1. The values are tab-delimited. Each of the columns and rows represents individual populations and polymorphisms respectively. An "X" means that the corresponding population has not been typed for the polymorphism. Colon separated values indicate the frequencies of the different alleles of a population at the corresponding locus. The source code for the XMLParser.java can be found at Appendix A.

Finding the Largest Complete Data Set

In order to find the largest maximal biclique in ALFRED, various methods were employed. Below are the different approaches undertaken, and descriptions of the statistics and methods used to analyze the efficiency of the different methods.

Wen-Chieh Chang's Maximal Biclique Algorithm

Alexe et al. (2002), describe a consensus algorithm for finding all maximal bicliques of a given graph. The algorithm has also proven to be efficient and polynomial in complexity for graphs with up to 2,000 vertices and 20,000 edges [6]. The algorithm exhaustively builds progressively larger bicliques, and lists all the bicliques in the given bipartite graph greater than a certain size. Given a bipartite graph G, with two sets of vertexes A and B, the algorithm examines every pair of vertices in one A, and calculates the intersections of their biclique set. This finds all the bicliques where the size of the vertex from set A is two. The next steps iteratively attempt to add the intersection sets of the remaining vertexes in set A to the biclique sets until no more vertexes can be added.

An implementation of the algorithm that is described by Alexe et al. (2002) can be found at Chang's website at http://www.cs.iastate.edu/~wcchang/biclique.html. Wen-Chieh Chang implemented the algorithm in C, and made the code available freely. The program accepts a bipartite graph as a text file listing the node pairs that form an edge, one node pair per line. A program in Java was written (MToB.java) to convert the data matrix from ALFRED into the bipartite graph representation, which can be then used as an input file for this Maximal Biclique Enumeration algorithm. Figure 2.4 shows the corresponding input file of the bipartite graph and data matrix shown in Figure 1.4. The source code for this program can be found in Appendix B.

```
0 0
0 1
0 2
1 0
1 1
2 1
2 2
3 0
3 1
4 1
```

Figure 2.4: Textual representation of a bipartite graph.

2D Sorting

A naïve approach to solving the problem is to sort the matrix in both dimensions and extract the densest quadrant. The matrix is first sorted in one dimension, e.g. the columns or the populations are sorted according to the number of polymorphism that they are typed for. The matrix is then sorted again in the other dimension, i.e. the rows or the polymorphisms are sorted according to the number of populations that they have been typed for. This sorts the matrix in both dimensions, providing a distribution where most of the data lies in a single quadrant.

A Java program (Sorting.java) was used to read in the data matrix, generated from mining ALFRED, and sort each dimension using the Bubble sort algorithm [17]. This is one of the most simple ways of sorting. Figure 2.5 shows a simplified data matrix and Figure 2.6 shows the same data matrix it has been sorted in both dimensions. The source code for the Sorting.java program can be found in Appendix C.

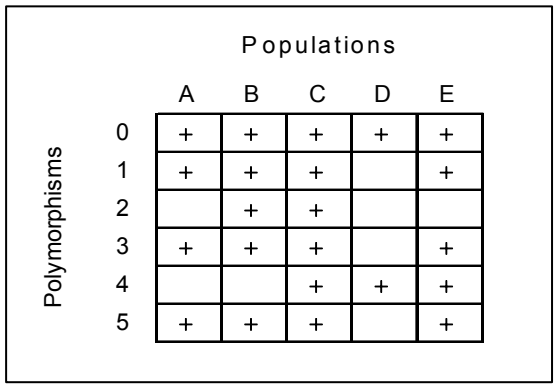

Figure 2.5: A hypothetical data matrix.

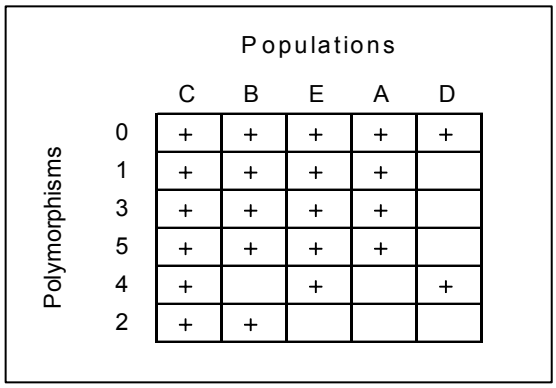

Figure 2.6: A hypothetical data matrix, 2D Sorted.

Genetic Algorithm

The solutions (bicliques) from Figure 1.5 can be represented by the chromosomes in Figure 2.7. The shaded and unshaded areas of the chromosome represent the population and loci respectively. A 1 means that the particular loci or population at that index is part of

the solution, and a 0 means that it is not included in the biclique. When a chromosome has been determined to be a true biclique, the fitness score of that chromosome is the area of the biclique that it represents (i.e. the product of the number of populations and loci). The fitness is zero if the chromosome is not a biclique. In the above figure, the fitnesses of the chromosomes A, B, and C, assuming that they are bicliques, are 4, 4, and 6 respectively.

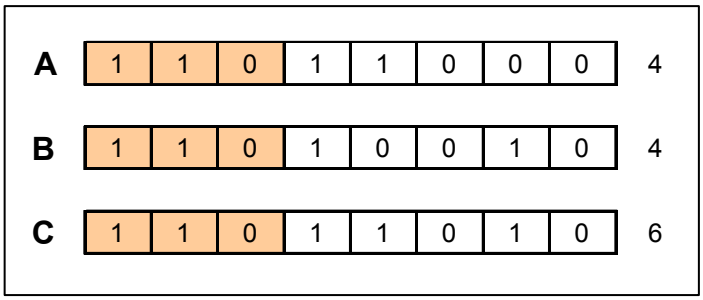

Figure 2.7: Chromosome representation of bicliques. Numbers to the right reflect the fitness of the chromosome.

As mentioned earlier, more than one fitness measure can be used, and each measure will result in a different set of solutions. Since the primary target audience is physical anthropologists and other genetics researchers, who need to study a very large set of populations, the largest complete data set should be evaluated with bias towards the number of populations. This can be achieved by calculating the fitness as $(P^2)*(L)$ or $(P^2)*(L/2)$, where P and L are the number of populations and polymorphisms in a given complete subset. The GA was tested with all of the above mentioned fitness scoring methods, and the results are discussed below in the "Results" section.

The GA was designed to start with an initial population size of 1000 chromosomes (solutions); in this case 1000 randomly selected bicliques (i.e. fitness > 0) from the ALFRED dataset. All of these initial set of solutions are mated (crossed over) at random. Mating results in the formation of two child chromosomes which might or might not be bicliques. Each resulting child chromosome is evaluated for fitness; if the fitness is not zero (i.e. the solution is a valid biclique) and the solution does not already exist in the population then it is added to the population.

At the end of one generation all of the parent chromosomes have mated at least once and the child chromosomes that were fit are now part of the new population. Unlike real life, the parent chromosomes do not die and are kept within the gene pool. Dropping parent chromosomes will most likely lead to the optimal solution stagnating at a local maxima and hinder the evolutionary process to find the real maximum. Another measure to introduce variation and potentially newer genes (or solutions) not present in the original population will be to randomly mutate genes. This is achieved by flipping one or more bits from 0 to 1 or vice versa in the child chromosome before evaluating it for fitness. As mentioned earlier this may or may not lead to a newer and better solution. However, it helps overcome the chance of the GA stagnating at a local maximum.

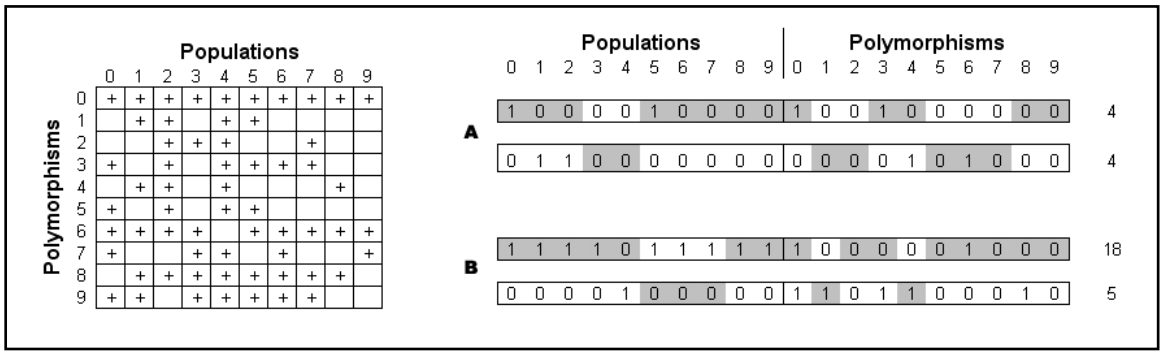

Figure 2.8: A hypothetical data matrix, and the mating of two set of parent chromosomes. A and B represents two sets of parents chromosomes ready to be crossed over at 6 random crossover points.

Mating or crossover is performed by the swapping of chromosomal information at a predefined number of random positions along the chromosomes of each set of parents. The number of crossover points is specified at the start of the GA, and each set of parents are crossed over at exactly the same number of points. However, the positions of these points vary from one set of parents to another.

Figure 2.8 shows two sets of parent chromosomes (A and B) ready to be crossed over. The four chromosomes represent solutions to the adjacent sample data matrix. The numbers to the right of the chromosomes represent the fitness. Both set of parents will be crossed over at 6 cross-over points. However, as the shaded area depicts, these positions vary for different sets of parents. Figure 2.9 shows the resulting chromosomes from the two crossovers in Figure 2.8. Mating does not always result in child chromosomes that are bicliques (i.e. have a fitness > 0), and if they do the child chromosomes are not necessarily fitter than the parents. The bit shaded blue in the child chromosome represents a mutation. The bit has been mutated from a 0 to a 1. In this case the mutation has introduced a variations that results in a chromosome that is more fit.

Generally in a GA, the maximum fitness curve grows exponentially at first. Then, as an optimal fitness is found, the curve flattens out. The iterative process of the GA continues until a plateau is reached for the optimal fitness. In our problem, the fittest chromosome represents the largest complete dataset from ALFRED. The average fitness of the population was calculated as the statistical mean of all the fitness scores of the individual chromosomes in the population. At the end of each generation the GA was designed to write out to a text file the fitness score of the fittest chromosome and the average fitness score of the entire population. The graph of the fitness curve is presented in Results (Figure 3.4).

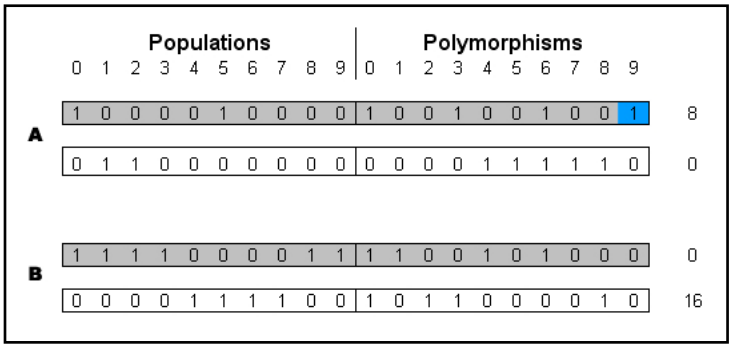

Figure 2.9: Resulting child chromosomes.

Largest Maximal Biclique Heuristic (LMBH)

The GA was initialized with chromosomes that were randomly selected bicliques from the ALFRED data set. The heuristic used to find 1000 random bicliques to initialize the GA looked so promising that it was modified and became the final approach used to extract the largest complete data set from ALFRED.

The algorithm randomly selects a polymorphism and examines the populations the selected polymorphism has been typed for. The algorithm then randomly selects a subset of these populations and looks for other polymorphisms that have been typed for all of the selected populations. This constitutes one biclique. The algorithm continues to find bicliques in this manner and adds it to the population, provided that it is not already in the population.

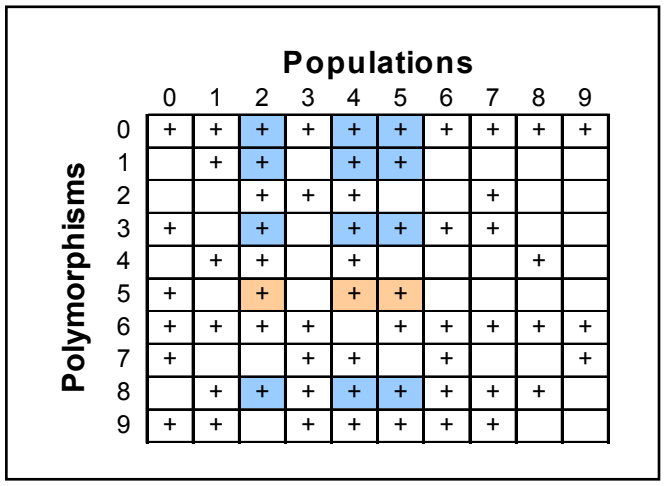

Figure 2.10: Biclique finding process of LMBH.

Figure 2.10 shows a sample data matrix. The heuristic randomly selects a polymorphism (polymorphism 5). This polymorphism has been typed for in populations 0, 2, 4 and 5. The algorithm randomly selects a random subset (it could potentially select the whole set as well). In the above example it has selected populations 2, 4 and 5 (shaded tan). Then it finds all the other polymorphisms that have been typed for in these populations (shaded blue). The shaded cells (both blue and tan) represent one biclique. The algorithm proceeds to find unique chromosomes in this fashion. This is a generally

applicable solution to the largest maximal biclique problem. The source code for the LMBH program can be found at Appendix D.

Tree Generation

A rough estimation of phylogeny is inferred from the largest complete data set. This is currently for visualization purposes only and it is by no means accurate. The tree is merely used to give the end users a visual conception of the currently identified maximal data set that can be compared to previous iterations. Over time, the user may be able to get an idea of how much the data have changed. The resulting tree is part of the presentation of the results in the website and is available in JPEG format.

There are multiple was of inferring phylogeny from genetic data. For the purpose of convenience (e.g. simplicity and easy of implementation) the CONTML package in PHYLIP [18, 19] is used. PHYLIP is available freely and has interactive menu features that allow one to customize and format the resulting tree. To automate the process, the menu input was redirected to the PHYLIP program from a pre-generated text file. This allows for the tree generation to be automated with the same set of parameters every time it is run, and without any human interaction.

Once the largest maximal biclique is found, the allele frequencies are extracted and passed to the CONTML program, within PHYLIP. The CONTML program estimates phylogenies by the restricted maximum likelihood method based on a Brownian motion model. The program assumes that all population divergence is due to genetic drift alone at each locus. Table 2.1 contains a complete data set for six African populations and three polymorphisms. Figure 2.11 shows the corresponding input file for the CONTML program. The result of the program is an unrooted tree with the length in evolutionary time in the

Newick Standard format [20]. The DRAWTREE program in the PHYLIP package accepts the unrooted tree and plots it, as shown in Figure 2.9.

	Locus	ADH1A		ADH1B		ADH1B	
	Polymorphism	C___8829387_10		E_rs1587264_10		C__27519856_10	
	Alleles	A	T	A	G	C	T
Populations	Biaka	0.939	0.061	0.858	0.142	0.515	0.485
	Hausa	0.973	0.027	0.744	0.256	0.355	0.645
	Ibo	0.968	0.032	0.552	0.448	0.333	0.667
	Mbuti	0.986	0.014	0.833	0.167	0.385	0.615
	Yoruba	0.954	0.046	0.667	0.333	0.391	0.609
	Chagga	0.944	0.056	0.878	0.122	0.433	0.567

Table 2.1: Subset of allele frequency data from ALFRED.

```
6 3
2 2 2
Biaka    0.939 0.858 0.515
Hausa    0.973 0.744 0.355
Ibo      0.968 0.552 0.333
Mbuti    0.986 0.833 0.385
Yoruba 0.954 0.667 0.391
Chagga 0.944 0.878 0.433
```

Figure 2.11: Allele frequency data, formatted for the COTML package.

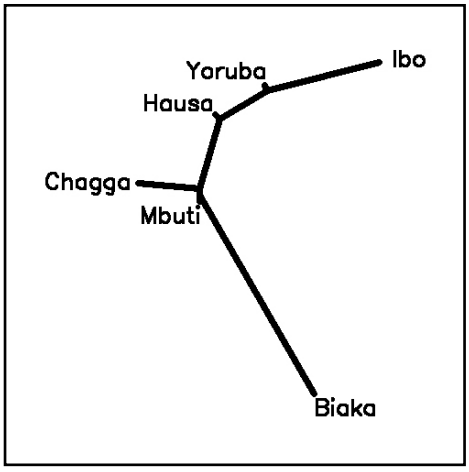

Figure 2.12: The phylogenetic tree depicting the six African populations, inferred from three polymorphic sites.

Benchmarking

The programs were analyzed in terms of the solution they derived, time taken to find the solution, and memory usage. To test and compare each method, the programs were run at least three separate times and the average values were taken. A random data set generated using a Java program was used to test all of the programs. The randomly generated matrix

(and the corresponding bipartite graph) had dimensions 800 by 800 (e.g. an equal number of populations and loci) and contained 42640 out of 640000 edges. The benchmarking was performed on a HP DL140 server with dual Xeon 3.2 GHz processors, 2 GB PC2100 DDR memory and Red Hat Enterprise Linux version 3.

The solution from each program was analyzed manually to calculate the size and contents of the largest complete data set returned. Time was measured using the "time" command in Linux. The command returns the time in seconds of the user time, system time, and real time. User time is the total number of CPU-seconds that the process used directly on behalf of the user, where CPU-seconds is defined as (the number of instructions * the number of clock cycles per instructions)/the clock cycles per second. System time is the total number of CPU-seconds used by the system on behalf of the process. Real time is the total time elapsed. The time value used to measure the amount of time required by the program was calculated by adding the user time and the system time.

Memory usage was somewhat more complicated to track. No programs to track memory usage are available for low cost. Linux does, however, provides a virtual file system located in "/proc" that contains up-to-date runtime system information for the different processes running on the machine. Inside the "/proc" directory there are multiple directories, each corresponding to a Process ID (PID). Each of these directories in turn contains multiple files. Each of these files contains different information pertaining to the process such as the command line arguments (/proc/PID/cmdline), values of the environmental variables (/proc/PID/exe), a link to the executable of this process (/proc/PID/fd), memory maps to executables and library files (/proc/PID/mem) and more. The file /proc/PID/status provides a summary of all the information about the

process in human readable form. This is perfect since a simple script can be written to parse and collect information about the process automatically and periodically.

```
Name:       emacs
State:      T (stopped)
SleepAVG:   88%
Tgid:       4505
Pid:        4505
PPid:       4476
TracerPid:  0
Uid:        501   501   501   501
Gid:        502   502   502   502
FDSize:     256
Groups:     502
VmSize:     11540 kB
VmLck:      0 kB
VmRSS:      4956 kB
VmData:     328 kB
VmStk:      416 kB
VmExe:      1320 kB
VmLib:      4168 kB
StaBrk:     08488000 kB
Brk:        093ce000 kB
StaStk:     bffaf000 kB
ExecLim:    08193000
Threads:    1
SigPnd:     0000000000000000
ShdPnd:     0000000000002000
SigBlk:     0000000000000000
SigIgn:     0000000000000000
SigCgt:     0000000051817efd
CapInh:     0000000000000000
CapPrm:     0000000000000000
CapEff:     0000000000000000
```

Figure 2.13: A sample /proc/PID/status file.

Figure 2.13 shows a sample status file for a running process. The VmSize is the total of memory used by the program and the VmRSS (Resident Set Size) is the amount actually in RAM; the rest is swapped out to disk. A Perl script was written (Appendix E) to execute the program and, using the PID, track and store the memory usage information every two seconds. The results were imported to an Excel spread sheet and plotted against time. When comparing the various methods together the memory usages of the programs were plotted against percent time (Figure 3.9), since the run time of the different programs varied. Percent time was calculated as the (time/total time to run the program) * 100.

3. RESULTS

Wen-Chieh Chang's Maximal Biclique Algorithm

This program very quickly finds all maximal bicliques in a given bipartite graph. The purpose of this algorithm was to solve the graph theory problem of finding the largest maximal biclique in a given bipartite graph. However, the algorithm does have worst case exponential runtimes. Unfortunately, the full ALFRED dataset is too large for the program and it fails to provide a solution. The bipartite graph from the ALFRED data set has over 2,000 vertices and 46,000 edges. Alexe et al. [6] concur that their algorithm is very efficient for graphs with up to 2,000 vertices and 20,000 edges. Since polymorphism data is not decreasing, this method will not be acceptable for the problem in question. It is, however, one method available to be used for the larger biclique problem.

Figure 3.1 shows the memory usage of the program while it attempts to resolve all maximal bicliques within the complete ALFRED dataset. The program uses up all system resources for over six hours until no more resources are available, at which time the program terminates. Before terminating, the program requires almost 3 GB of memory (e.g. the sum of physical and virtual memory).

Nevertheless the program is very efficient for relatively smaller datasets. The program was tested on randomly generated data sets. The results are shown below in Table 3.1. The graph in Figure 3.3 clearly demonstrates that the program scales in exponential time. In other words, as the complexity of the data set increases, so does the runtime. The complexity of the data set is not only determined by its dimension, but also by the number of edges present in the graph. Keeping this in mind all of the randomly generated test data sets were populated to have roughly the same nodes to edges ratio (15:1). Assuming a

constant ratio of nodes to edges, for every doubling in matrix area the run time increases exponentially.

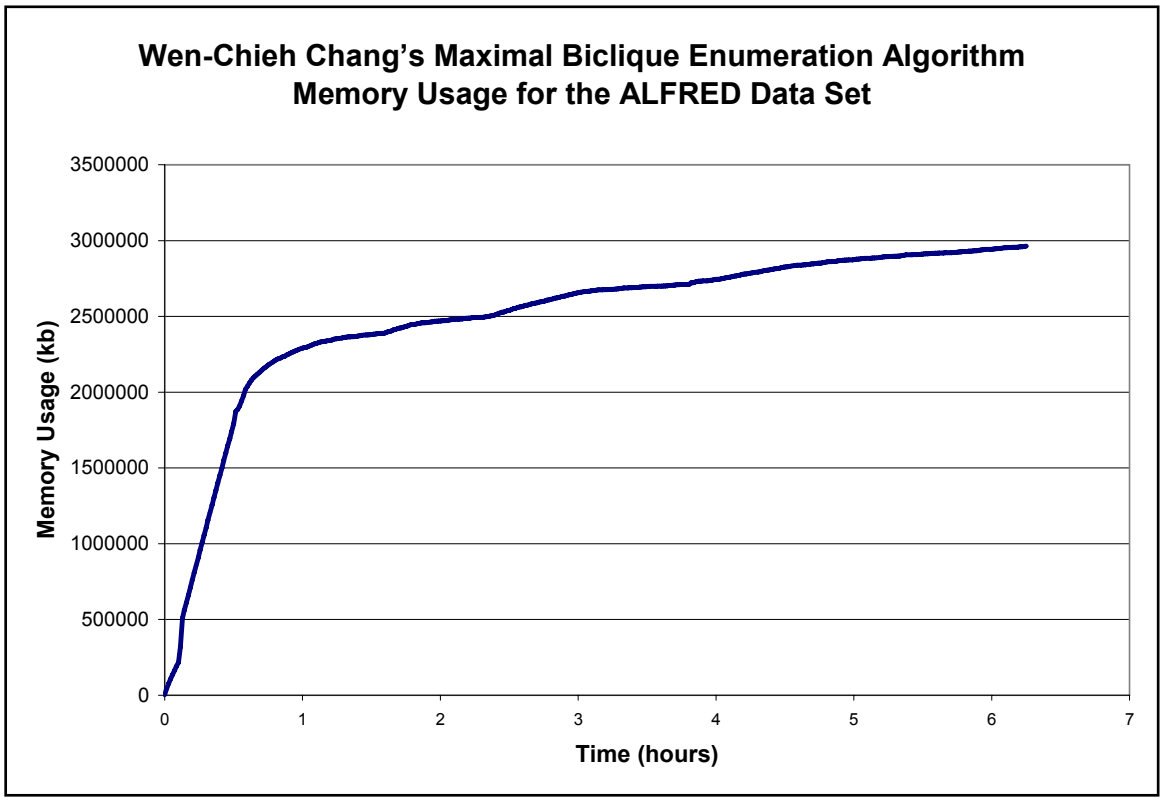

Figure 3.1: Memory usage of Wen-Chieh Chang's Maximal Biclique Enumeration Program for the complete ALFRED data set.

Sample	Matrix Size	Area	Edges	Time (seconds)	Time (minutes)
1	50 x 50	2500	161	0.013	0.00
2	100 x 100	10000	682	0.083	0.00
3	150 x 150	22500	1461	0.35	0.01
4	200 x 200	40000	2626	1.28	0.02
5	300 x 300	90000	5942	9.63	0.16
6	400 x 400	160000	10641	42.88	0.71
7	500 x 500	250000	16670	136.05	2.27
8	600 x 600	360000	24042	353.80	5.90
9	700 x 700	490000	32623	792.32	13.21
10	800 x 800	640000	42640	1629.48	27.16
11	900 x 900	810000	54138	Program Fails	Program Fails
12	1000 x 1000	1000000	67170	Program Fails	Program Fails

Table 3.1: Time taken by Wen-Chieh Chang's Maximal Biclique Enumeration program for data sets of different sizes.

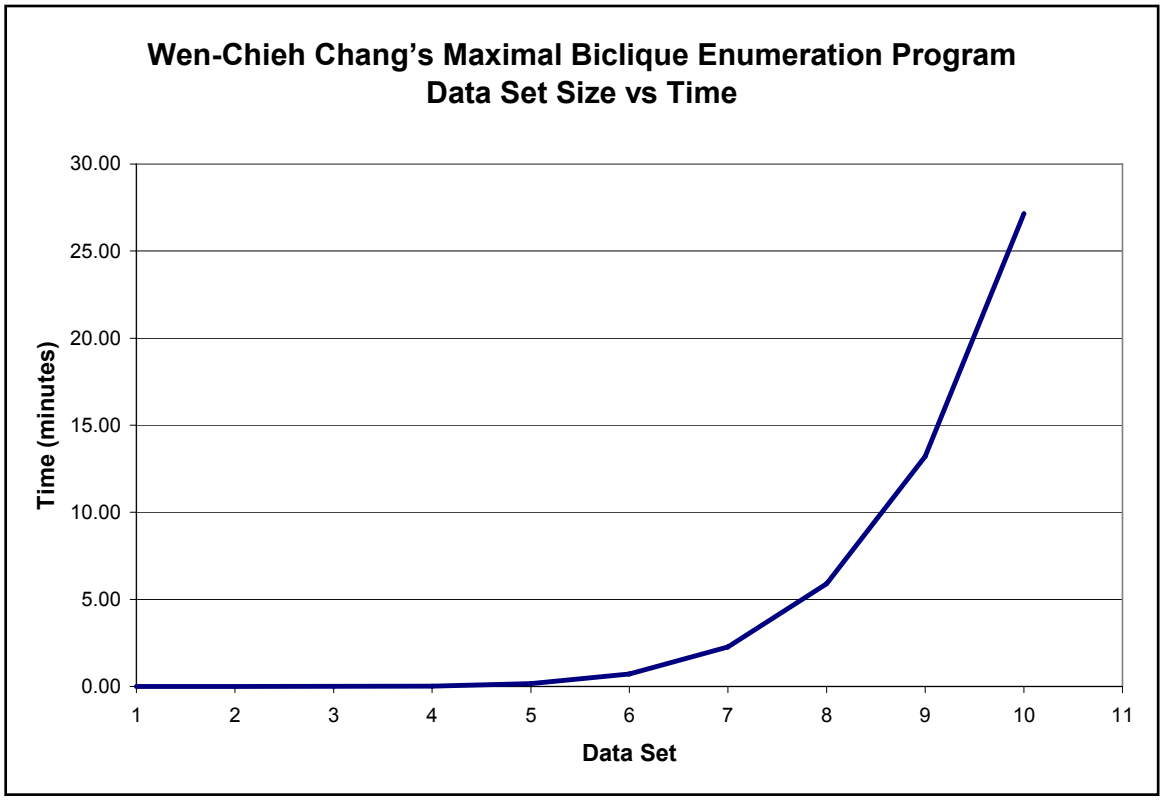

Figure 3.2: Data Set size vs. time for Wei-Chan's Maximal Biclique Enumeration program.

The program does not require as much memory as most of the other methods discussed here. Figure 3.3 shows the memory usage of the program during its 27 minute runtime, when run with an 800 by 800 data set. The algorithm progressively builds larger bicliques by iteratively attempting to add the intersection sets of bicliques from the previous iteration until no more nodes can be added. The jump in memory usage at around 15 minutes can be explained by a new iteration of the program, dealing with a biclique significantly larger in size than the previous iteration.

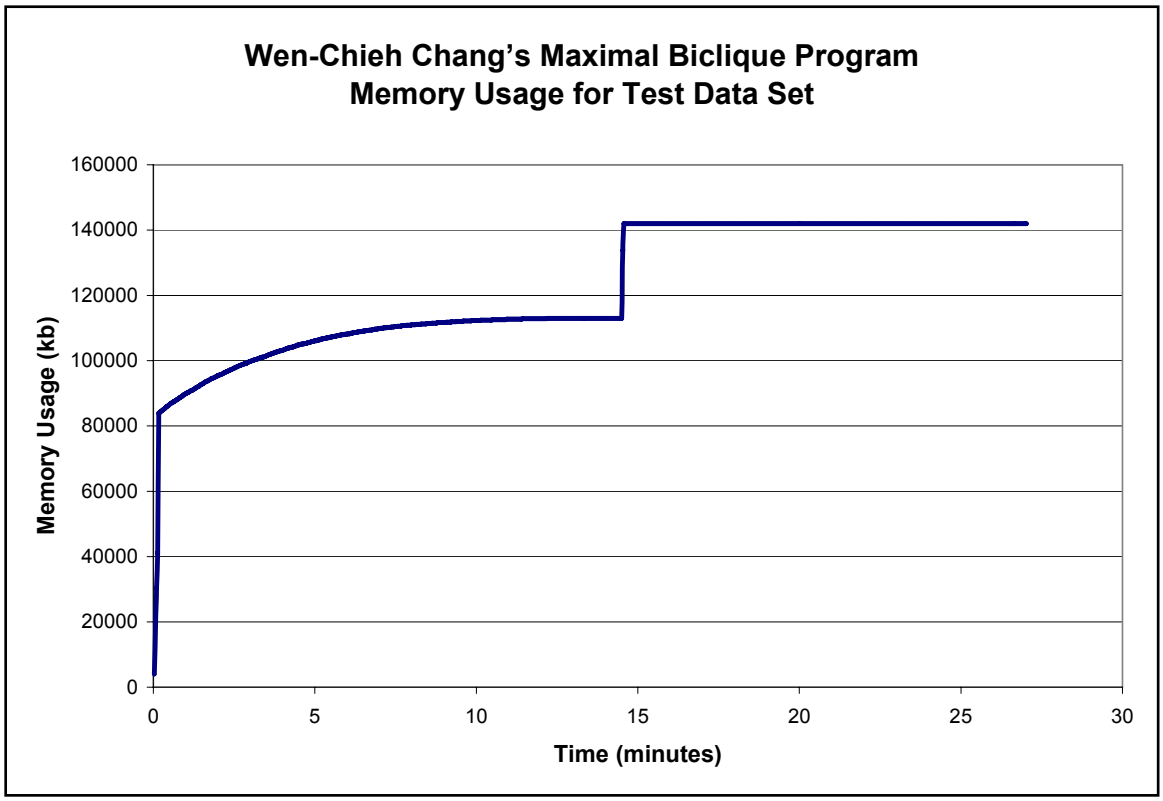

Figure 3.3: Memory usage of Wen-Chieh Chang's Maximal Biclique Enumeration Program for the test data set.

2D Sorting

One of the first attempts to solve the ALFRED largest maximal biclique problem turned out to be not so successful. The assumption was that if the data matrix was sorted in both dimensions by density of the data, the largest complete set would be the densest quadrant. Figure 1.1 shows the data-matrix, sorted in both dimensions. Clearly, while density does increase toward the lower-left quadrant, sorting the matrix in this manner does not yield a perfect quadrant which contains the largest complete dataset. This is because not all of the extensively studied populations in the database have been studied for the same set of polymorphisms, e.g. two populations can be typed for the same number or

polymorphisms and yet be two complete different sets of polymorphisms. Since this approach does not yield a biclique solution, benchmarking was not performed.

Genetic Algorithm

The heuristic (see below) used to initialize the GA finds the optimal solution to the problem. Hence, the GA fails to find a better solution. However, it does optimize the gene pool as a whole by finding more solutions and improves the average fitness of the population. It also serves as a means to further validate the results of the LMBH program. Since it successfully improves on the general fitness of the population yet fails to find a better solution, one can be somewhat assured that the results yielded by the LMBH program are valid.

The GA is a great tool to optimize the results obtained by any heuristic that generates bicliques. As shown in Figure 3.4 the fittest solution is identified in the initialization process and remains static throughout the subsequent generations. The average fitness however is constantly improved as the initial solutions are optimized. The fitness curve shows the values of the fittest solution (the largest maximal biclique) and the average fitness of all the solutions in each generation of the GA, when run with the complete ALFRED data set. The result supports the validity of the GA, since the fitness curve conforms to a pattern observed in a generally successful GA.

Figure 3.5 shows the memory usage of the GA. Once the population is initialized (the first 10 minutes) the program requires a constant amount of memory: almost 213 MB of memory (both in RAM and swap) for the complete ALFRED data set. The GA takes 3 hours and 24 minutes to complete 200 generations. Running the GA for longer (up to 1000 generations) does not improve the fittest solution, but does constantly improve the average

fitness. At 200 generations, the average population fitness was 99.2 percent of the maximum. The fittest solution (fitness = 16243) contains 37 populations and 439 polymorphisms. When compared to other problems solved with GAs, this GA is relatively fast; it only takes about 20 generations for the average population fitness to reach 98.4% of the maximum fitness. Over the next 180 generation or so, the average fitness is only improved by 0.8%.

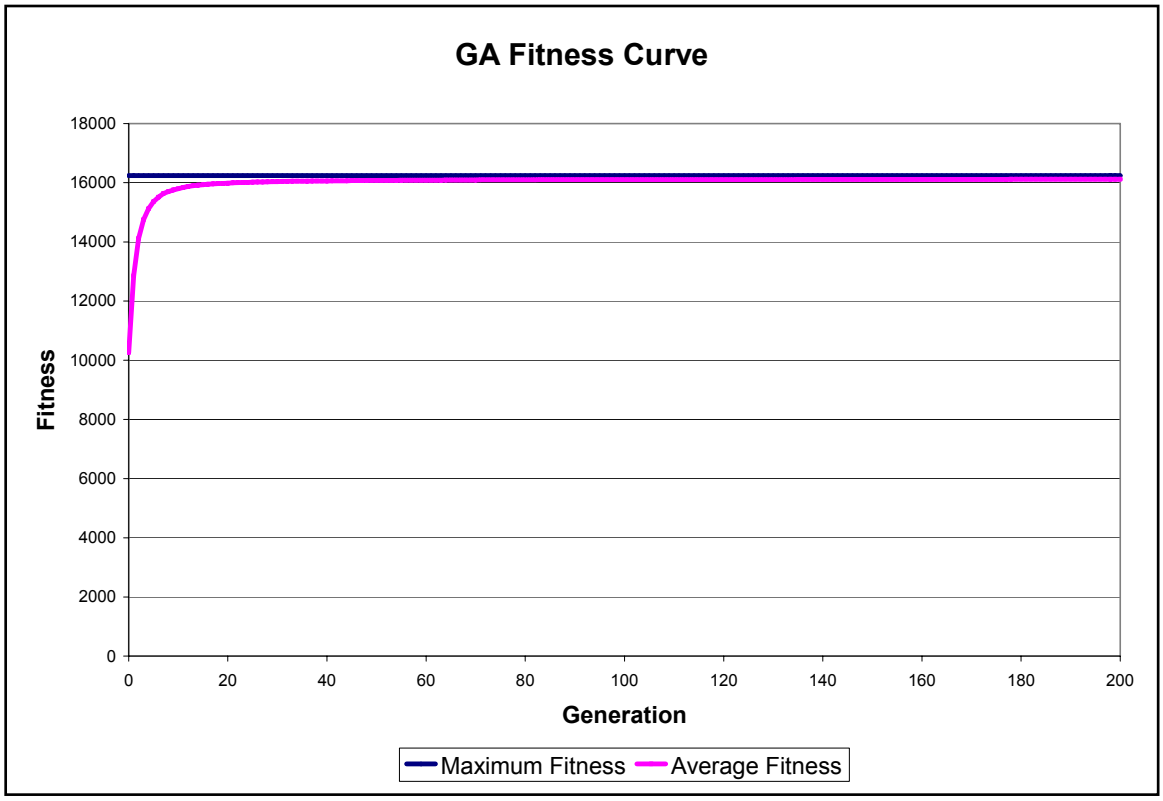

Figure 3.4: The fitness curve of the Genetic Algorithm, plotting the value of the fittest chromosome and the average fitness of the population at each generation.

Using a different measure of fitness, biased towards the number of populations, did not help find a different biclique from the current ALFRED data set. All the three fitness scores used, do not favor a biclique that exists within the ALFRED data consisting of 49 populations and 211 polymorphisms, which researches could be interested in. A better fitness measure needs to be investigated.

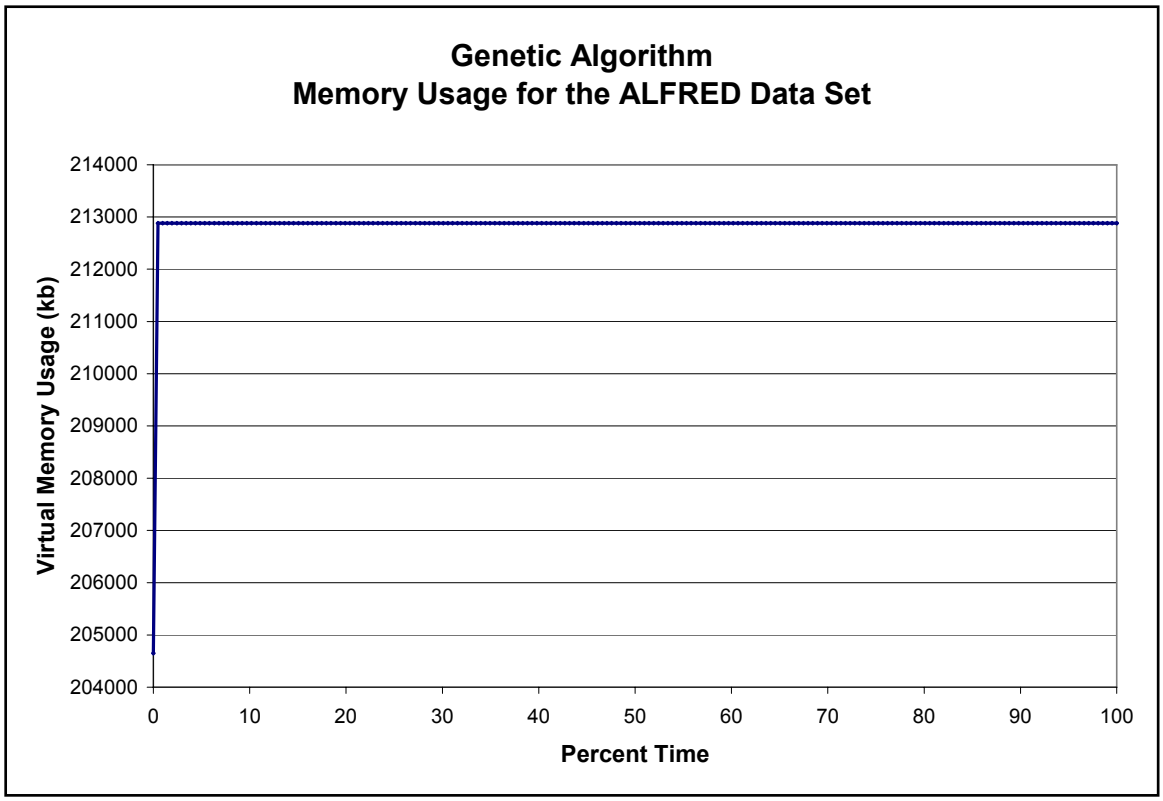

Figure 3.5: The memory usage of the Genetic Algorithm for the test data set.

Largest Maximal Biclique Heuristic (LMBH)

The heuristic, initially designed to simply initialize the set of populations for the genetic algorithm, turned out to find the optimal solution. The algorithm was tested on various datasets randomly generated and the results were verified with that of the Maximal Biclique Algorithm where applicable.

Table 3.2 shows that the program takes roughly the same amount of time for the different data sets even though they are considerably different in scale. Figure 3.6 shows the graph of how the program scales for the different data sets presented in the above table. The regression line (shown in red) demonstrates that the program scales linearly. The algorithm runs in linear time as the function time=0.099x + 16.1 where x is the size of the matrix and time is measured in minutes. The relatively small difference in time between the different data sets can be attributed to the

Sample	Matrix	Area	Edges	Time (minutes)
1	50 x 50	2500	161	16.43
2	100 x 100	10000	682	16.44
3	150 x 150	22500	1461	16.45
4	200 x 200	40000	2626	16.46
5	300 x 300	90000	5942	16.46
6	400 x 400	160000	10641	16.47
7	500 x 500	250000	16670	16.53
8	600 x 600	360000	24042	17.05
9	700 x 700	490000	32623	17.12
10	800 x 800	640000	42640	17.19
11	900 x 900	810000	54138	17.29
12	1000 x 1000	1000000	67170	17.37

Table 3.2: Time taken by the LMBH program for data sets of different sizes.

processing time involved in reading and handling a larger data matrix. The majority of run time is spent randomly searching for unique bicliques, which is nearly constant in respect to data size. As a result, increasing the data set size has little effect on run time. Therefore, the slope of the function of time is small. This heuristic also requires around 213 MB of memory and is static throughout the entire runtime (Figure 3.7), irrespective of the size of the data set analyzed. This is due to the nature of the data structures used in the code and the random searching nature of the algorithm.

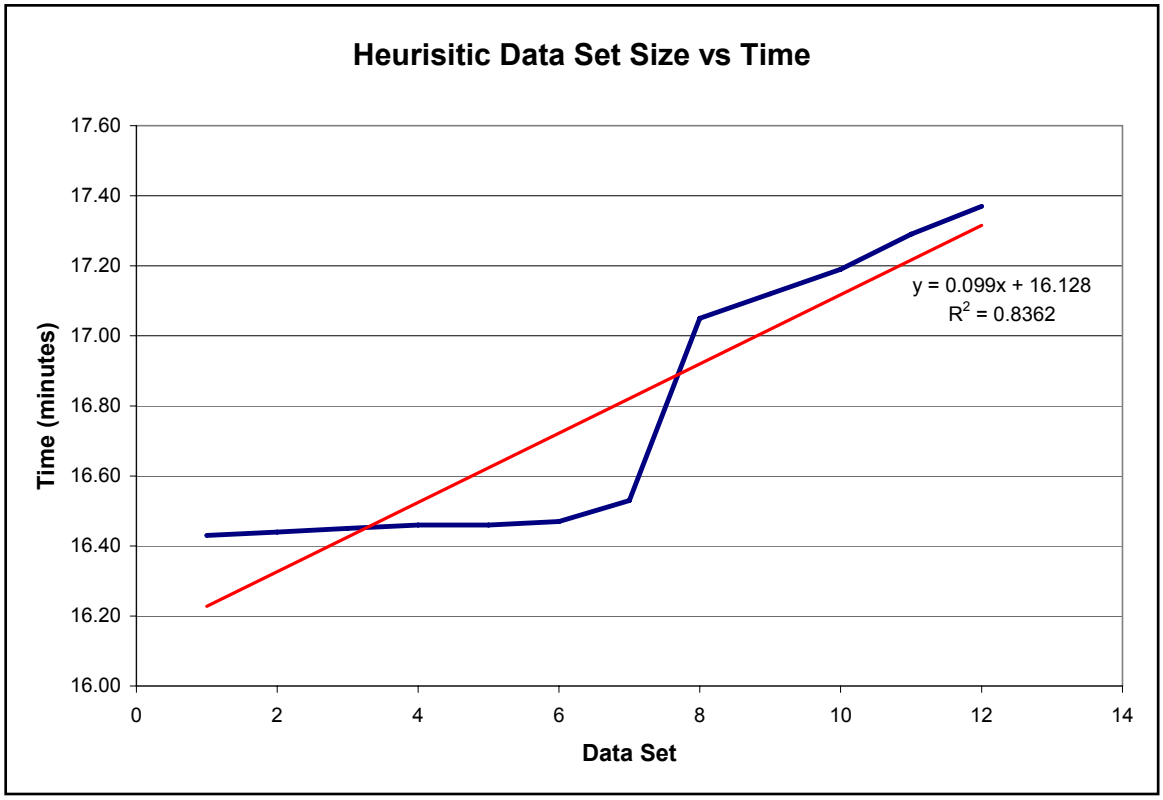

Figure 3.6: Data Set size vs. time for the LMBH program.

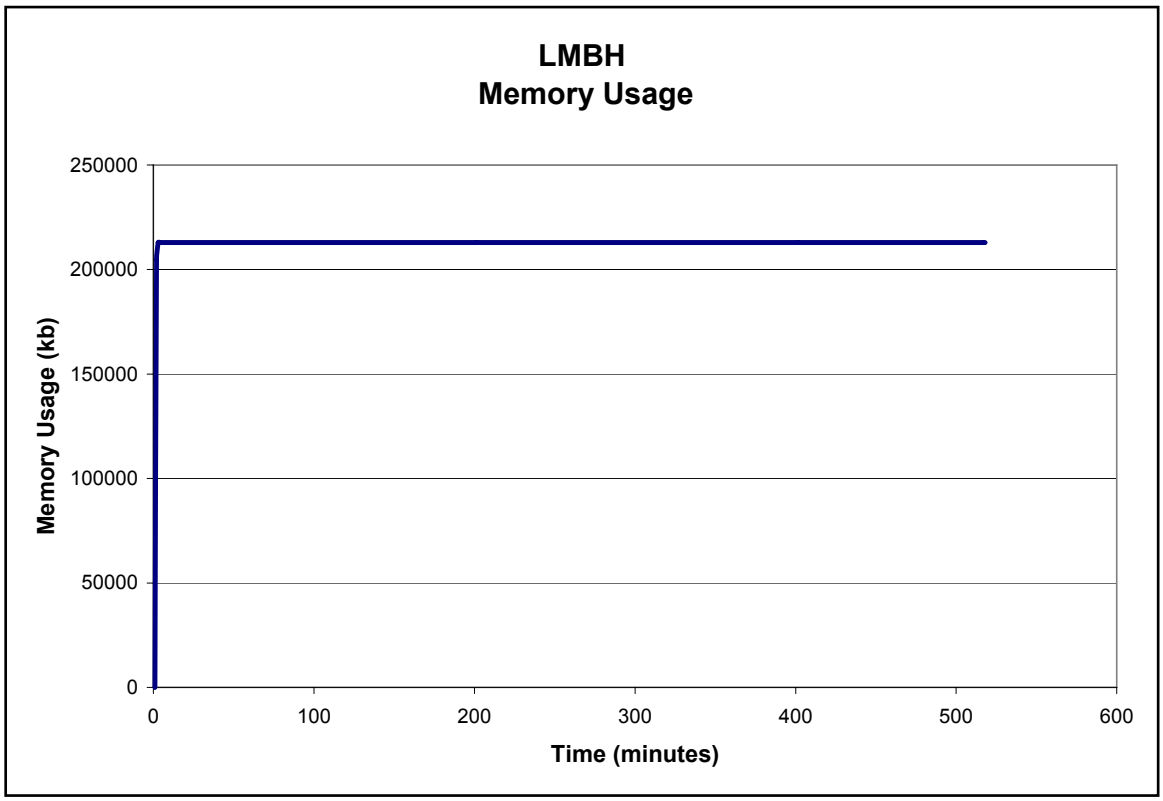

Figure 3.7: Memory usage of the LMBH program for the test data set.

Presentation of Results

As mentioned earlier, ALFRED continues to grow daily. Not only is the total area of the data matrix growing, but so is the allele frequency information for already existing populations and polymorphisms. Each time ALFRED is updated, the largest complete data set will almost certainly change to include more populations, polymorphisms, or both than are incorporated now. Researchers will want the most up-to-date data to better study the populations. Setting up an automated system through which the researchers can access the latest data would solve this problem. Taking this into consideration, this project has been extended to automatically mine ALFRED periodically, during which time the data have grown significantly, and make the largest complete data set available in an easily transferable

and both human and computer readable format (XHTML and XML). This allows for others to take the data and easily transform it into a format better suited for their purposes.

Based upon feedback by the current ALFRED designers, we have also created a Web Service [21] to shorten the time between updates to ALFRED and updates to the heuristic results. A platform independent service is run on the local server, waiting for the ALFRED designers to notify of updates. Even though the Web Service is implemented in Perl on the local server, the connecting client may use any platform or programming language to communicate with our Web Service. This is because the message sent and received by the Web Service client and server is translated to an XML standard, for platform independence.

The Web site presenting the latest largest complete data set from ALFRED can be found at http://dollyo.rit.edu/diaspora. Along with the largest complete data set, the website also includes a phylogenetic tree inferred from the data set. This tree is only for visualization purposes and is by no means meant to be an accurate depiction of human evolutionary history. Figure 3.8 is a screen shot of the main page from the website. The main page displays the results from the last run of the program. The user is shown A) the date the program was last run on, B) a thumbnail of the generated tree, C) the dimension of the data and D) the results in both XML and XHTML.

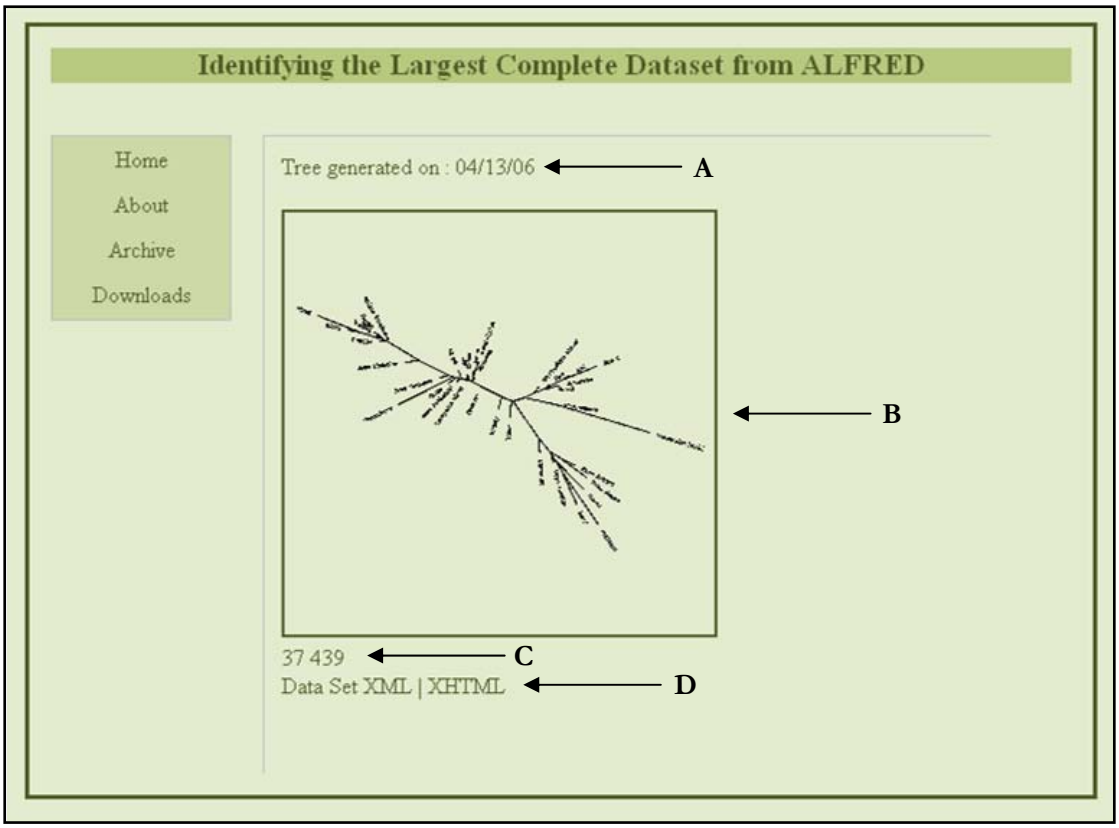

Figure 3.8: Screen shot of the project Web site.

Comparison of Methods

The Maximal Biclique Enumeration algorithm is quick and efficient for small data sets, but does not work for complex data sets and, more importantly for the problem at hand, with the ALFRED data set. The LMBH algorithm utilizes 1.5 times more memory but finds a solution to a problem solvable by both the programs (Figure 3.9). Even though the results from the LMBH algorithm cannot be proven to be the largest maximal biclique in ALFRED without exhaustive search, there are two indications that support it. The fact that the Genetic Algorithm could improve the average fitness of the population but not find a larger solution supports the result from the LMBH.

Further analyzing the 37 populations and the 439 polymorphisms that constitute the largest complete data set shows that all these data were generated from the Kidd Lab. The

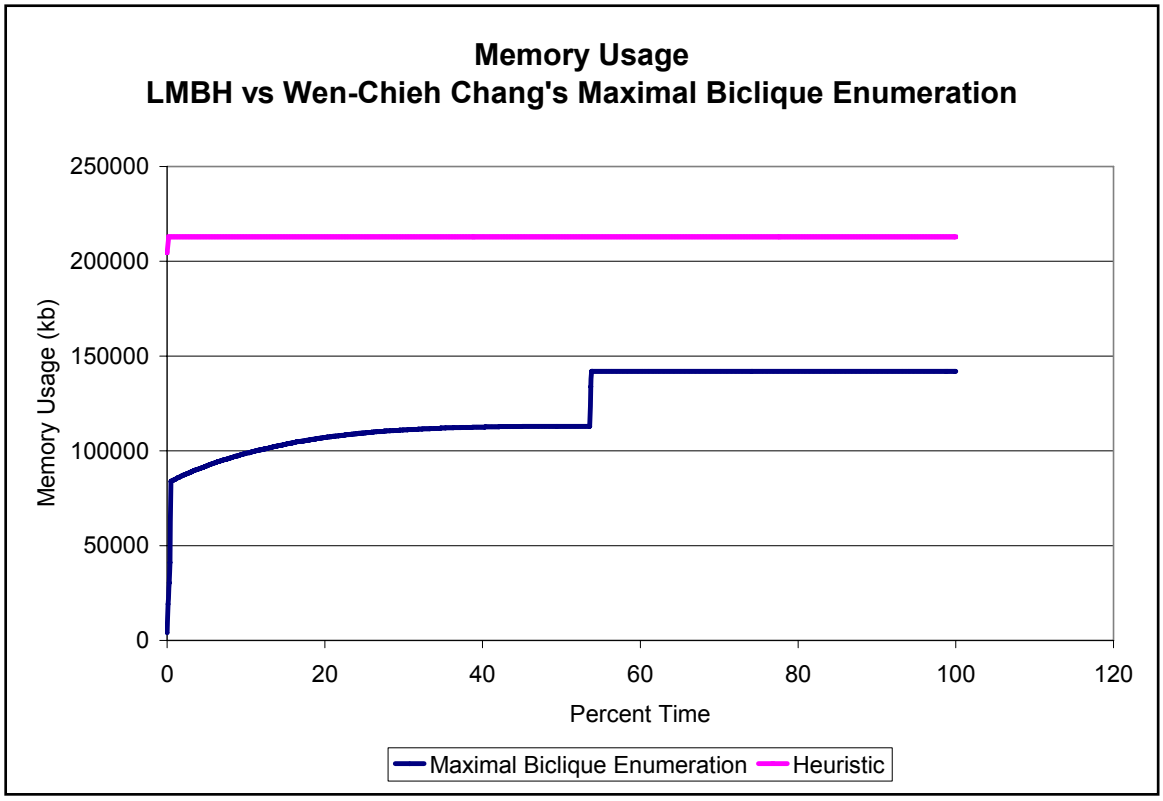

Figure 3.9: Comparison of memory usage by the two programs.

Kidd Lab studies allele frequencies of selected polymorphisms with an emphasis on those that have been studied in multiple populations. They maintain blood samples for over 37 populations and the lab has the resources and research interest in studying these populations over greater number of polymorphisms. Almost all data generated in the Kidd Lab is available through ALFRED. Most of the external data comes from publish work in journals. Unfortunately, as mentioned earlier, most researchers who generate data do not test a large number of populations over large number of polymorphisms. The fact that the constituents of the largest complete data set is exclusively from the Kidd Lab is, in hindsight, additional supporting evidence that the result is correct, since few other research groups are working on this "empty matrix" problem.

4. DISCUSSION

The LMBH algorithm described here provides a solution to the largest maximal biclique problem of finding the largest complete data set from ALFRED. Since no other freely available method can successfully find a solution, the heuristic was validated by comparing performance and results for a range of smaller data sets for which solutions could be calculated using the Wei-Chang's Maximal Biclique Enumeration algorithm. Additionally, the GA also supported and further validated the results of the heuristic. Based on these results, we are unable to find fault with the heuristic results. While an exhaustive test could be performed on the full ALFRED data set, it would not complete within a reasonable time. Given the results of the tests on smaller data sets, the heuristic therefore seems a plausible solution to identifying the largest complete data set from ALFRED.

An issue that still needs to be investigated is the scoring function used to evaluate the solution. The current scoring function resolves the graph theory problem of finding the largest maximal biclique defined by area. However, from a population geneticist's perspective this is merely one solution. A biclique of 1 by 500 nodes (fitness = 500) is clearly larger than a biclique of 10 by 40 nodes (fitness = 400). However, to a researcher interested in studying a large set of populations, a data set of 1 population typed for 500 polymorphisms is of less importance and value as a data set of 10 populations typed for 40 polymorphisms.

Currently, the heuristic is designed to favor the biclique with a larger population size if more than one bicliques have the same fitness score. Unfortunately, if the bicliques are similar, but not identical in fitness, the program chooses the biclique with the larger fitness score. Perhaps making the program more interactive, such as allowing the user to select from

a list of fitness functions depending on the nature of the data set might help the user to obtain a solution more appropriate for their needs.

Some researchers might only be interested in a specific set of polymorphisms or populations, and want a large and complete data set for these parameters. Another extension to this work would be to find the largest complete data set for a given list of populations or polymorphism. This work is currently being undertaken by Alex Haugh. Alex, working in conjunction with the ALFRED administrators, has set up a Web Service that allows the user to enter their own data and, using the LMBH algorithm, find the largest complete data set

The implementation of the LMBH algorithm could also be tuned to enhance performance. The scope of the variables must be limited to areas where the variables are required, since this will clear unneeded memory usage. Much more effective data structures could be used. Together, these steps could significantly decrease the run time and the amount of memory used. In other words, the algorithm could be profiled to make significant performance gains both in terms of runtime and memory.

The heuristic is also a general solution to the maximal biclique problem and can be used to solve the problem regardless of the domain, be it graph theory or bioinformatics. More specifically the heuristic can be applied to other polymorphism database such as such as the Single Nucleotide Polymorphism database (dbSNP) [15] and the Human Genome Variation Database (HGVbase) [16]. The Web site can be extended to include an interface to accept a bipartite graph (Figure 6), as input, compute the largest biclique using the LMBH program and output the result to users.

Finally, the phylogenetic tree used as a part of the results presentation could be made more accurate. It would be helpful to identify more robust statistics or programs that can be applied to the largest complete data set to generate a tree as easily as by PHYLIP, and yet be

more accurate. Now that the largest complete data set is available for other researchers, their work can be incorporated to this project to extend it by producing a robust tree, as opposed to being a supplemental visualization entity.

5. CONCLUSION

This project provides an algorithm that efficiently mines ALFRED to extract the largest complete data set. This subset of ALFRED is formatted and made available, in a universal data exchange format, to researchers primarily interested in studying genetic similarities and evolutionary history of a large set of human populations. Given the nature of the problem there is no means of verifying the answer, nonetheless the evidence presented validates and supports the effectiveness of the method, therefore increases the confidence given to the results.

The algorithm's efficiency is comparable to that of Wei-Chang's Maximal Biclique Enumeration algorithm for relatively small data sets, and surpasses it for much larger data sets. There is room for improving efficiency of the program and it can certainly be extended to solve variants of the largest maximal biclique problem. Despite some unresolved issues, the methodology discussed herein provides an extensible first step to solving the general largest maximal biclique problem, with specific application of extracting the largest complete data set from ALFRED.

BIBLIOGRAPHY

1. Cheung KH, Osier MV, Kidd JR, Pakstis AJ, Miller PL, Kidd KK. "ALFRED: an allele frequency database for diverse populations and DNA polymorphisms." Nucleic Acids Res. 28(1):361-3. 2000.
2. Osier MV, Cheung KH, Kidd JR, Pakstis AJ, Miller PL, Kidd KK. "ALFRED: an allele frequency database for Anthropology." Am J Phys Anthropol. 119:77-83. 2002.
3. Sherry ST, Ward M, Sirotkin K. 1999. dbSNP database for single nucleotide polymorphisms and other classes of minor genetic variation. Genome Res 9: 677-679.
4. Fredman D., Siegfried M., Yuan Y.P., Bork P., Lehvaslaiho H. and Brookes A.J. (2002) HGVbase: a human sequence variation database emphasizing data quality and a broad spectrum of data sources. Nucleic Acids Res., 30, 387–391.
5. Cavalli-Sforza L, Cavalli-Sforza F, Thorne S. The Great Human Diasporas: The History of Diversity and Evolution. Addison Wesley Publishing Company. 1996.
6. Alexe G et al. "Consensus algorithms for the generation of all maximal bicliques." DIMACS Technical Report 2002 - 52, Rutgers University, Piscataway, NJ, USA.
7. Cheng Y, Church GM. "Biclustering of expression data." Proc Int Conf Intell Syst Mol Biol. 2000;8:93-103.
8. Feige U et al. "Approximating clique is almost NP-complete." Foundations of Computer Science, 1991. Proceedings, 32nd Annual Symposium on 1-4 Oct. 1991 Pages:2-12.
9. René Peeters. "The maximum edge biclique problem is NP-complete." Discrete Applied Mathematics. Volume 131-3: 651-654. 2003.
10. Nossal R, Galla T. "Solving NP-Complete Problems in Real-Time System Design by Multichromosome Genetic Algorithms." In Proceedings of the SIGPLAN 1997

Workshop on Languages, Compilers, and Tools for Real-Time Systems, pages 68-76, ACM SIGPLAN, June 1997.

11. Holland JH. "Genetic Algorithms." Scientific American. July 1992, Pages 66-72.
12. Chang, Wen-Chieh. Maximal Biclique Algorithm Implementation. 16 June 2003. December 2005. < http://www.cs.iastate.edu/~wcchang/biclique.html >.
13. Alexe G et al. "Consensus algorithms for the generation of all maximal bicliques." DIMACS Technical Report 2002 - 52, Rutgers University, Piscataway, NJ, USA.
14. Yergeau F, Bray T, Paoli J, Sperberg-McQueen CM, Maler E. Extensible Markup Language (XML) 1.0 (Third Edition) W3C Recommendation 4th February 2004.
15. Sun Microsystems, Inc. Java Technology. December 2005. <http://java.sun.com/>.
16. Megginson, David. The official website for SAX. 27 April 2004. SourceForge. December 2005. < http://www.saxproject.org/ >.
17. Bubble Sort. Wikipedia, The Free Encyclopedia. 29 April 2006. Wikimedia Foundations, Inc. 29 April 2004 < http://en.wikipedia.org/wiki/Bubble_sort >.
18. Felsenstein, J. PHYLIP - Phylogeny Inference Package (Version 3.2). Cladistics 5: 164-166. 1989.
19. Felsenstein, J. PHYLIP (Phylogeny Inference Package) version 3.6. Distributed by the author. Department of Genome Sciences, University of Washington, Seattle. 2004. <http://evolution.genetics.washington.edu/phylip.html>.
20. Gary Olsen. Gary Olsen's Interpretation of the "Newick's 8:45" Tree Format Standard. 30 August 1990. < http://evolution.genetics.washington.edu/phylip/newick_doc.html>.
21. Web Services Activity. World Wide Web Consortium. 28 March 2006. April 2006. <http://www.w3.org/2002/ws/>.

APPENDIX A

```
/**
 * XMLParser.java
 * Parses the ALFRED XML data dump and generates a data matrix
 * @version 1.0, Wed. 4 May 2005
 * @author Mohamed Uduman
 */

import org.xml.sax.*;
import org.xml.sax.helpers.*;
import javax.xml.parsers.*;
import org.xml.sax.SAXException;
import java.io.*;
import java.util.*;

public class XMLParser extends DefaultHandler {

      /**********************Declaring Variables**********************/

      // SAX
      private String currentTag = "";
      private String parentTag = "";
      private StringBuffer strCharBuffer;
      // frequency
      private String allele_frequency = "";
      private String allele_id = "";
      private String typedsample_id = "";
      // site
      private String site_uid = "";
      private String site_name = "";
      // population
      private String population_uid = "";
      private boolean enterPopName = false;
      private String population_name = "";
      // hashtables
      private Hashtable ht_typedsample = new Hashtable();
      private Hashtable ht_site = new Hashtable();
      private Hashtable ht_population = new Hashtable();
      //matrix
      private String[][] matrix;
      private String[] population;
      private String[] site;
      //IO
      BufferedWriter bw;

/**********************        Main         **********************/

      public XMLParser(){
            super();
      }

      public static void main(String[] args) {
```

```
            // Create a JAXP SAXParserFactory and configure it
            SAXParserFactory spf = SAXParserFactory.newInstance();
            //spf.setValidating(validation);
            spf.setValidating(false);

            XMLReader xmlReader = null;
            try {
            // Create a JAXP SAXParser
            SAXParser saxParser = spf.newSAXParser();

            // Get the encapsulated SAX XMLReader
            xmlReader = saxParser.getXMLReader();
            } catch (Exception ex) {
            System.err.println(ex);
            System.exit(1);
            }

            // Set the ContentHandler of the XMLReader
            xmlReader.setContentHandler(new XMLParser());

            // Set an ErrorHandler before parsing
            xmlReader.setErrorHandler(new MyErrorHandler(System.err));

            try {
            // Tell the XMLReader to parse the XML document
            xmlReader.parse(convertToFileURL(args[0]));
            } catch (SAXException se) {
            System.err.println(se.getMessage());
            System.exit(1);
            } catch (IOException ioe) {
            System.err.println(ioe);
            System.exit(1);
            }

      } // main


      /**********************Handler Methods ************************/
      public void startDocument(){
      }


      public void startElement(String namespace,String localName,
String qName,
Attributes attrList ){

            currentTag = qName;
            strCharBuffer = new StringBuffer();

            if(currentTag.equals("frequency"))
                  parentTag = currentTag;
            if(currentTag.equals("site"))
                  parentTag = currentTag;
            if(currentTag.equals("population"))
                  parentTag = currentTag;
```

```
	} // startElement


	public void characters(char[] ch, int start, int len){
		strCharBuffer.append(ch,start,len);
	} // characters

	public void endElement(String namespace,String localName, String
qName){
		// if end of parent element clear parentTag
		if(qName.equals(parentTag))
			parentTag="";
		if(parentTag.equals("frequency"))

	frequency(currentTag,(strCharBuffer.toString()).trim());
		if(parentTag.equals("site"))
			site(currentTag,(strCharBuffer.toString()).trim());
		if(parentTag.equals("population"))

	population(currentTag,(strCharBuffer.toString()).trim());
	} // endElement


	public void endDocument(){
/*
In the XML files there are certain frequencies that have no samples
associated to them
e.g. typedsample_id = 24182, has frequencies but NO loci/population
There are certain typedsamples that specify a site that does not exist
any where else.
e.g. site_uid = SI000505K , SI001485S , SI001483Q
*/
//Itegrity check
	Enumeration enum_typedsample = ht_typedsample.keys();
	while( enum_typedsample.hasMoreElements()  ){
String htKey = (String)enum_typedsample.nextElement();
typedsample ts = (typedsample)ht_typedsample.get(htKey);
String _population_uid = ts.getPopulation_uid();
String _site_uid = ts.getSite_uid();
		if(_population_uid==null || _site_uid==null){
			ht_typedsample.remove(htKey);
			continue;
		}
		if(!ht_site.containsKey(_site_uid)){
			ht_typedsample.remove(htKey);
			continue;
		}
		if(!ht_population.containsKey(_population_uid)){
			ht_typedsample.remove(htKey);
			continue;
		}
	  }

//initializing matrix & info
	population = new String[ht_population.size()];
	site = new String[ht_site.size()];
	matrix = new String[ht_site.size()][ht_population.size()];
```

```
      //populating the above arrays
      Enumeration enum_population = ht_population.keys();
      int popCounter=0;
while( enum_population.hasMoreElements()  ){
            String htKey = (String)enum_population.nextElement();
            population[popCounter]=htKey;
            popCounter++;
        }
Enumeration enum_site = ht_site.keys();
      int siteCounter=0;
      while( enum_site.hasMoreElements()  ){
String htKey = (String)enum_site.nextElement();
site[siteCounter]=htKey;
siteCounter++;
        }
        for(int x=0;x<site.length;x++)
for(int y=0;y<population.length;y++)
matrix[x][y]="X";

//Filling matrix
enum_typedsample = ht_typedsample.keys();
        while( enum_typedsample.hasMoreElements()  ){
            String htKey = (String)enum_typedsample.nextElement();
            typedsample ts = (typedsample)ht_typedsample.get(htKey);
            String _population_uid = ts.getPopulation_uid();
            String _site_uid = ts.getSite_uid();
            String[] _alleles =
((site)ht_site.get(_site_uid)).getAlleles();
            String freq = "";
for(int i=0;i<_alleles.length;i++){
String alleleFreq = ts.getAllele_frequency(_alleles[i]);
                  if( (alleleFreq.trim()).equals("U") )
                        alleleFreq = "0.000";
                  freq+= alleleFreq + ":";
            } // for

            int x = xIndexOf(_site_uid);
            int y = yIndexOf(_population_uid);

            if(x>=0 && y>=0)
                  matrix[x][y] = freq;
            freq="";
        } // while

      //write matrix to file
try {
bw = new BufferedWriter(new
FileWriter("../DataMatrix/dataMatrix.txt"));
for(int x=0;x<site.length;x++){
for(int y=0;y<population.length;y++){
bw.write( matrix[x][y] + "\t");
bw.flush();
} // for
bw.write( "\n");
bw.flush();
} // for
bw.close();
```

```
}
      catch (IOException ioe){
      }

      //write population.txt
      try {
            bw = new BufferedWriter(new
FileWriter("../DataMatrix/populations.txt"));
            for(int x=0;x<population.length;x++){
                  String pid = population[x];
                  String pName = (String)ht_population.get(pid);
                  bw.write(pName + ":" + pid + "\n");
                  bw.flush();
            } // for
            bw.close();
      }
      catch (IOException ioe){
      }

      //write site.txt
      try {
            bw = new BufferedWriter(new
FileWriter("../DataMatrix/sites.txt"));
            for(int x=0;x<site.length;x++){
                  String sid = site[x];
                  bw.write(sid + "\n");
                  bw.flush();
            } // for
            bw.close();
      }
      catch (IOException ioe){
      }


      }     // endDocument


/**********************      Methods      ************************/

      public int xIndexOf(String value){
            int returnXVal=-1;
            for(int i=0;i<site.length;i++){
                  if(site[i].equals(value)){
                        returnXVal=i;
                        break;
                  }
            }
            return returnXVal;
      }

      public int yIndexOf(String value){
            int returnYVal=-1;
            for(int i=0;i<population.length;i++){
                  if(population[i].equals(value)){
                        returnYVal=i;
                        break;
                  }
```

```
            }
            return returnYVal;
      }

      public void frequency(String tagName, String characters){

            if(tagName.equals("allele_frequency")) {
                  allele_frequency = characters;
            } // if
            else if(tagName.equals("allele_id")) {
                  allele_id = characters;
            } // else if
            else if(tagName.equals("typedsample_id")) {
                  typedsample_id = characters;

                  if( !(ht_typedsample.containsKey(typedsample_id)) ){

ht_typedsample.put( typedsample_id , new
typedsample(typedsample_id,allele_id,allele_frequency) );
                  }
                  else {
                        typedsample ts =
(typedsample)ht_typedsample.get(typedsample_id);

      ts.putAllele_frequency(allele_id,allele_frequency);
                        ht_typedsample.put( typedsample_id , ts );
                  }

                  // clearing variables
                  allele_frequency = "";
                  allele_id = "";
                  typedsample_id = "";
            } // else if

      } // frequency

      public void site(String tagName, String characters){

            if(tagName.equals("site_uid")) {
                  site_uid = characters;
                  ht_site.put(site_uid, new site(site_uid));
            } // if
            else if(tagName.equals("name")) {
                    site_name = characters;
                    if(site_name.indexOf("haplotype")>-1)
                      if(ht_site.containsKey(site_uid)){
                        ht_site.remove(site_uid);
                      }
            }
            else if(tagName.equals("allele_id")) {
                if(ht_site.containsKey(site_uid)){
                  allele_id = characters;
                  site s = (site)ht_site.get(site_uid);
                  s.addAllele(allele_id);
                }

            } // else if
```

```
      }// site

      public void population(String tagName, String characters){
            if(tagName.equals("population_uid")) {
                  population_uid = characters;
                  enterPopName=true;
            } // if
            else if(tagName.equals("name") && enterPopName==true) {
                  population_name = characters;
                  population_name =
removeSpecialCharacters(population_name);
                  ht_population.put(population_uid,population_name);
                  enterPopName = false;
            } // else if
            else if(tagName.equals("typedsample_id")) {
                  typedsample_id = characters;
            } // else if
            if(tagName.equals("site_uid")) {
                  site_uid = characters;
                  typedsample ts =
(typedsample)ht_typedsample.get(typedsample_id);
                  ts.putPopulation_uid(population_uid);
                  ts.putSite_uid(site_uid);
                  ht_typedsample.put( typedsample_id , ts);

            } // else if

      }// site

      public String removeSpecialCharacters(String str){
            String newStr="";
            for (int i = 0; i < str.length(); i++) {
                  if (str.charAt(i) != ',')
                        newStr+= str.charAt(i);
            }
            return newStr;
      } // removeSpecialCharacters


/*****************************************************************
************************ Error Handling ************************
*****************************************************************/

      // Error handler
private static class MyErrorHandler implements ErrorHandler {

      private PrintStream out;

      MyErrorHandler(PrintStream out) {
this.out = out;
}


private String getParseExceptionInfo(SAXParseException spe) {
String systemId = spe.getSystemId();
if (systemId == null) {
```

```
systemId = "null";
}
String info = "URI=" + systemId +
" Line=" + spe.getLineNumber() +
": " + spe.getMessage();
return info;
}

// SAX ErrorHandler methods

public void warning(SAXParseException spe) throws SAXException {
out.println("Warning: " + getParseExceptionInfo(spe));
}

public void error(SAXParseException spe) throws SAXException {
            String message = "Error: " + getParseExceptionInfo(spe);
            throw new SAXException(message);
      }

      public void fatalError(SAXParseException spe) throws SAXException
{
            String message = "Fatal Error: " +
getParseExceptionInfo(spe);
            throw new SAXException(message);
      }
      }

} // class
```

APPENDIX B

```
/**
 * MToBG.java
 * Coverts a given data matrix to a bipartite graph
 * @version 1.0, Wed. 10 August 2005
 * @author Mohamed Uduman
 */

import java.util.*;
import java.io.*;
import java.lang.*;

public class MToBG{

      private static int[][] matrix;
      private static int ROWS=0;    //Numb of rows in our matrix
      private static int COLS=0;    //Numb of columns in our matrix


      /*************** MAIN ***************/

      public static void main(String[] args){

            // If the user does not pass a filename then
            // the default location of  the matrix is matrix.txt
            String matrixFile = "dataMatrix.txt";
            if(args.length>0)
                  matrixFile = args[0];

            //read the matrix into 2D Array
            try{
                  readMatrix(matrixFile); //reads the matrix file into
                                                        an array
            }catch (IOException e) {
                  System.out.println("Problem reading file!");
                  System.exit(-1);
            }

            //Convert the matrix to a bigraph (bigraph.txt)
            matrixToBigraph();
      }

      /************** METHODS **************/
      /* Reads the tab-delimitted matrix file into a 2D Array. */
      private static void readMatrix(String fileName) throws
                                          IOException{
            int rows=0;
            int cols=0;
            //count the numb of cols & rows in the matrix (need to
                                          initialize array)
            BufferedReader br =
            new BufferedReader (new FileReader(fileName));
```

```
        String line = br.readLine();
        String[] arrayCols = line.split("\t");
        cols = arrayCols.length;
        while (line != null){
        rows++;
        line = br.readLine();
        }
        br.close();

        System.out.println();
        System.out.println("Matrix dimensions: " + rows + " x " +
                                                        cols);
        System.out.print("...");
        matrix = new int[rows][cols];
        rows=0;
        cols=0;

        //storing the matrix in the 2D Array
        br =
        new BufferedReader (new FileReader(fileName));
        line = br.readLine();
        while (line != null){
        String[] arrayLine = line.split("\t");
        for(int i=0;i<arrayLine.length;i++){
              if(arrayLine[i].equals("X"))
              matrix[rows][cols]=0;
              else
              matrix[rows][cols]=1;
              cols++;
        }
        line = br.readLine();
        rows++;
        cols=0;
        }
        br.close();
        System.out.println("\nMatrix file read into 2D Array");

        ROWS = matrix.length;
        COLS = matrix[1].length;
}



//Converts the matrix to a bigraph
private static void matrixToBigraph(){
        //create a file for the bigraph
        try{
              createFile("bigraph.txt");
        }catch(IOException ioe){
              System.out.println("Problem creating the file for the
                                                        bigraph!");
              System.exit(-1);
        }

        for(int i=0;i<ROWS;i++){
              for(int j=0;j<COLS;j++){
                    //If there is an edge write it to the file
```

```
                        if(matrix[i][j]==1){
                              try{
                                    writeToFile("bigraph.txt",true,
                                                                  i,j);
                              }catch(IOException ioe){
                                    System.out.println("Problem writing
                                                      the bigraph!");
                                    System.exit(-1);
                              }
                        }
                  }
            }

      }

      /* Appends the bigraph to the "bigraph.txt" file. */
      private static void createFile(String fileName) throws
                                          IOException{
            BufferedWriter bw =
            new BufferedWriter(new FileWriter(fileName));
      }

      /* Appends the bigraph to the "bigraph.txt" file. */
      private static void writeToFile(String fileName, boolean append,
                        int x, int y) throws IOException{
            BufferedWriter bw =
                  new BufferedWriter(new FileWriter(fileName, true));
            bw.write(x + "\t" + y + "\n");
            bw.flush();
            bw.close();
      }
}
```

APPENDIX C

```
/**
 * sorting.java
 * 2D Sorting
 * Sorts the dataMatrix in 2 Dimensions
 * @version 1.0, Wed. 10 August 2005
 * @author Mohamed Uduman
 */

import java.util.*;

public class Sorting{

   public static void main(String[] args){

      //Load data matrix
      loadMatrix m = new loadMatrix();
      Matrix matrix = m.getMatrix();
      //Initializing arrays to hold the counts of each row and column
                                                          of the matrixes
      int X = matrix.getNumbRows();
      int Y = matrix.getNumbCols();;
      int[] xCount = new int[X];
      int[] yCount = new int[Y];

      //Sorting by X
      for(int y=0;y<Y;y++) {
         for(int x=0;x<X;x++) {
            if(array2D[x][y]!=null)
               xCount[x]++;
         }
      }

      int[] newXPos = sort(xCount);
      String[][] matXSorted = new String[X][Y];
      for(int x=(X-1);x>=0;x--)  {
         for(int y=0;y<Y;y++) {
            matXSorted[(X-1)-x][y] = array2D[newXPos[x]][y];
         }
      }

      //Sorting by Y
      for(int x=0;x<X;x++){
         for(int y=0;y<Y;y++) {
            if(matXSorted[x][y]!=null)
               yCount[y]++;
            }
      }

     int[] newYPos = sort(yCount);
      String[][] matYSorted = new String[X][Y];
      for(int x=0;x<X;x++) {
```

```
            for(int y=(Y-1);y>=0;y--) {
                  matYSorted[x][(Y-1)-y] = matXSorted[x][newYPos[y]];
            }
         }

      }

     //Bubble sorts an array of int
      public int[] sort(int[] counts) {
         int tmp;
       //the array with the original indexes of the row/column
         int[] index = new int[counts.length];
         for(int i=0;i<counts.length;i++) {
            index[i]=i;
         }
       //bubble sort
         for(int i=0;i<counts.length;i++) {
            for(int j=0;j<(counts.length - 1);j++) {
               if(counts[j+1]<counts[j]) {
                  tmp = counts[j];
                  counts[j] = counts[j+1];
                  counts[j+1] = tmp;
                  tmp = index[j];
                  index[j] = index[j+1];
                  index[j+1] = tmp;
               }
            }
         }
         return index;
      }

}
```

APPENDIX D

```
/**
 * LMBH.java
 * Largest Maximal Biclique Heuristic
 * Randomly searches for bicliques and adds them to the population.
 * @version 1.0, Wed. 15 August 2005
 * @author Mohamed Uduman
 */

import java.util.*;
import java.io.*;
import java.lang.*;
import java.text.*;
import java.math.*;

public class LMBH{


      // Declaration of variables

      private Vector population = new Vector();
      // the vector that holds the population ( the chromosomes)
      private Vector populationFitness = new Vector();
      // the vector that holds the fitness of each chromosome in the
            //population
      private int N;
      //the number of Chromosomes in our population
      private Matrix matrix;
      // the Matrix object which contains the data matrix
      private java.util.Random rnd = null;
      // Random number generator
      private int totalFitness=0;
      // the total fitness of the population at a given generation
      private double averageFitness=0.0;
      // the average fitness of the population at a given generation
      private int maxFitness=0;
      // the maximum fitness of the population at a given generation
      private BufferedWriter bw;
      // Buffered Writer to write to files

      private int SCORING_METHOD = 1;
      private boolean needsSorting = false;

      //Constructor
      public LMBH (Matrix m, int scoringMethod){
            matrix = m;
            N = 2000;
            SCORING_METHOD = scoringMethod;
            generateCompleteChromosomes();
            writeToFileFittest("fittestChromosome");
            removeTemp();
      }
```

```
      /**
       * Find N complete submatrixes populations and fill the
population
       */
      private void generateCompleteChromosomes(){

            rnd = new java.util.Random( );
            int rndLocus , rndNumbPop;
            int rndPop , rndNumbLoci;

            String popList=""; //the string that holds the indexes of
the
biclique
            String lociList ="";
            boolean[] c; //the formed chromsome
            double cf = 0.0; // fitness of the new chromosome

            int populationCount = 0;

            while( populationCount<N ){

                  //re-setting variables
                  popList="";
                  lociList="";

                  // By Locus
                  // select a random locus
                  rndLocus = rnd.nextInt(matrix.getNumbRows());

                  // select a random number of populations
                  rndNumbPop = rnd.nextInt(matrix.getNumbCols()-2) + 1;

                  // Check if that locus has at least that many
populations typed
                  if(matrix.getRowCount(rndLocus)>=rndNumbPop){
                        // get list of pops this locus had been typed
for
                        popList = matrix.getTypedPopulations(rndLocus);
                        // Of the pops this locus has been typed for
pick rndNumbPop
randomly
                        popList = getPopList(popList,rndNumbPop);
                        // find ALL loci that have the above
populations typed for
                        for(int i=0;i<matrix.getNumbRows();i++){
                              if
(matrix.areThesePopulationsTyped(i,popList)){
                                    lociList = lociList + ":" + i;
                              }
                        }
                        if(!lociList.equals(""))
                              lociList =
lociList.substring(1,lociList.length());
                        //From the selected populations and loci create
a chromosome
                        c = createChromosome(popList,lociList);
```

```
                        //cf = (int)( ((lociList.split(":").length)/2
)*(Math.pow(popList.split(":").length,4)) );
                        int numberOfLoci = lociList.split(":").length;
                        int numberOfPop = popList.split(":").length;
                        cf = computeFitness(numberOfPop,numberOfLoci);

                        //if Chromosome is unique add to population
                        if ( isUnique(c)==true ){
                              population.add(c);
                              populationFitness.add(new Double (cf));
                              populationCount++;
                              /*if(populationCount%50 == 0){

                                    sortPopulation();
                                    System.out.print("Count: " +
populationCount);
                                    System.out.print("\tMax: " +
(Double)populationFitness.get(0));
                                    double average =0.0;
                                    double sum=0.0;
                                    for(int
      x=0;x<populationFitness.size();x++)

      sum+=((Double)populationFitness.get(x)).doubleValue();
                                    average = sum/population.size();
                                    System.out.println("\tAverage: " +
average);
                                    }*/
                              sortPopulation();
                              //print out the fittest chromosome from
this generation (backup)

      //writeToFileFittest("currentFittestChromosome");
                        }

                  }
            }
      }

      /**
       * Check if the chromosome is unique.
       * Returns TRUE if it is. If the chromosome already exists
returns
FALSE.
       */
      private boolean isUnique(boolean[] c){
            boolean result  = true;
            Double fitness = new Double(getFitness(c));
            int pos;
            if(populationFitness.contains(fitness)){
                  pos = populationFitness.indexOf(fitness);
                  while(populationFitness.indexOf(fitness,pos)!=-1 &&
result){
                        boolean[] otherChromosome = (boolean[])
population.get(pos);
                        for( int i = 0; i < c.length; i++ ) {
```

```
                              if( c[i] != otherChromosome[i] ) {
                                    result = true;
                                    break;
                              }//if
                              if( i == c.length - 1 ) { result = false;
}
                        }//for
                        pos++;
                  }//while
            }//if others of equal fitness
            return result;
      }

      private double computeFitness(int p, int l){
            // 1. p * l
            // 2. (p^4) * (l/2)

            int totalNumberOfPop = matrix.getNumbCols();
            int totalNumberOfLoci = matrix.getNumbRows();
            double max=0.0;
            double fitness=0.0;
            double normal=0.0;

            if(SCORING_METHOD == 1){
                  //1.
                  max = (double)((totalNumberOfPop*2) *
totalNumberOfLoci);
                  fitness= (double)(p*l);
            } else if(SCORING_METHOD == 2){
                  //2.
                  max = (double)( (Math.pow(totalNumberOfPop,2)) *
(double)(totalNumberOfLoci/4));
                  fitness = (double)( (Math.pow(p,4)) * (double)(l/2));

            }

            normal = fitness/max;
            fitness = normal;
            return fitness;
      }

      /**
      * Given a list of population typed for a locus, find all the loci
      * with the same populations typed. i.e bicliques
      */
      private String getPopList(String popList, int rndNumbPop){

            String[] arrayPopList = popList.split(":");
            boolean[] arrayBoolean = new boolean[arrayPopList.length];
            Arrays.fill(arrayBoolean, true);
            int cnt=0;
            int rndIndex;

            //clearing popList
            popList="";

            while(cnt<rndNumbPop){
```

```
                  rndIndex = rnd.nextInt(arrayPopList.length);
                  if(arrayBoolean[rndIndex]==true){
                        popList+=":" + arrayPopList[rndIndex];
                        arrayBoolean[rndIndex]=false;
                        cnt++;
                  }
            }
            popList = popList.substring(1,popList.length());

            String[] arrayNewPopList = popList.split(":");
            int tmp;
            int[] arraySorted = new int[arrayNewPopList.length];

            for(int i=0;i<arrayNewPopList.length;i++)
                  arraySorted[i] =
Integer.valueOf(arrayNewPopList[i]).intValue();

            for(int i=0;i<arraySorted.length;i++) {
                  for(int j=0;j<(arraySorted.length - 1);j++) {
                        if(arraySorted[j+1]<arraySorted[j]) {
                              tmp = arraySorted[j];
                              arraySorted[j] = arraySorted[j+1];
                              arraySorted[j+1] = tmp;
                        }
                  }
            }
            popList="";
            for(int i=0;i<arraySorted.length;i++)
                  popList+=":" + arraySorted[i];
            popList = popList.substring(1,popList.length());
            return popList;
      }

      /**
      * Given a colon delimited string of pops & loci, return a
chromosome
      */
      private boolean[] createChromosome(String p, String l){

            boolean[] chromosome = new
boolean[matrix.getNumbCols()+matrix.getNumbRows() +1];

            String[] tempPop = p.split(":");
            String[] tempLoci = l.split(":");

            for(int i=0;i<tempPop.length;i++)
                  chromosome[Integer.parseInt(tempPop[i])]=true;

            for(int i=0;i<tempLoci.length;i++)

chromosome[Integer.parseInt(tempLoci[i])+matrix.getNumbCols()]=true;

            //printChromosome(chromosome);
            return chromosome;
      }

      /**
```

```
	 * Prints to screen the chromosome contents and its fitness
	 */
	public void printChromosomes(){

		boolean[] chromosome;
		for(int j=0;j<population.size();j++){
			System.out.println();
			chromosome = (boolean[])population.get(j);

			for(int i=0;i<matrix.getNumbCols();i++){
				if(chromosome[i]==true)
					System.out.print(i + ":");
			}
			System.out.println();
			for(int i=matrix.getNumbCols();i<chromosome.length-
1;i++){
				if(chromosome[i]==true)
					System.out.print((i-
matrix.getNumbCols())+":");
			}
			System.out.println();
			System.out.println(populationFitness.get(j) );
		}
	}

	/**
	 * Writes to file the fittest chromosome
	 * 1. Simple view
	 * 2. Phylip infile
	 * 3. HTML
	 */
	public void writeToFileFittest(String fileName){
		fileName = fileName +  SCORING_METHOD;
		String simpleFileName = "../Results/" + fileName + ".txt";
		boolean[] chromosome;
		chromosome = (boolean[])population.get(0);
		String chrmosomePop = "";
		for(int i=0;i<matrix.getNumbCols();i++){
			if(chromosome[i]==true)
				chrmosomePop+=(i + "\t");
		}
		String chrmosomeLoci ="";
		for(int i=matrix.getNumbCols();i<chromosome.length-1;i++){
			if(chromosome[i]==true)
				chrmosomeLoci+=((i-matrix.getNumbCols())+"\t");
		}
		String[] arraychrmosomePop = chrmosomePop.split("\t");
		String[] arraychrmosomeLoci = chrmosomeLoci.split("\t");

		// Simple view
		String fitness =
((Double)populationFitness.get(0)).toString() ;
		String nPop =
Integer.toString((chrmosomePop.split("\t")).length);
		String nLoci =
Integer.toString((chrmosomeLoci.split("\t")).length);
		//write to file
```

```
            try {
                  bw = new BufferedWriter(new
FileWriter(simpleFileName));
            }
            catch (IOException ioe){
                  System.out.println("Usage 'java IO filename'");
                  System.out.println("Be sure filename exists");
            }
            try{
                //bw.write( chrmosomePop + "\n" + chrmosomeLoci + "\n"
+
"Population: " + nPop + "\n" + "Loci      : " + nLoci +"\n" + "Fitness
: " +
fitness );
                bw.write( nPop + "\n" + nLoci );
                  bw.flush();
            }catch(IOException ioe){
                  System.out.println("Error writing to file");
            }
            try{
                  bw.close();
            }catch(IOException ioe){
                  System.out.println("Error closing file");
            }

            //Phylip infile
            String phylipFileName = "../Results/" + fileName +
".phylip" ;
            String lineOne = arraychrmosomePop.length + " " +
arraychrmosomeLoci.length;
            String lineTwo = "";
            for(int x=0;x<arraychrmosomeLoci.length;x++){
                String val =
matrix.getElement(Integer.parseInt(arraychrmosomeLoci[x]),0);
                val = val.trim();
                String[] arraySplitLoci = val.split(":");
                lineTwo+=arraySplitLoci.length + " ";
            }
            lineTwo = lineTwo.trim();
             //write to file
            try {
                  bw = new BufferedWriter(new
FileWriter(phylipFileName));
            }
            catch (IOException ioe){
                  System.out.println("Usage 'java IO filename'");
                  System.out.println("Be sure filename exists");
            }
            try{
                  bw.write( lineOne + "\n" + lineTwo);
                  bw.flush();
                  for(int x=0;x<arraychrmosomePop.length;x++){
                        int popIndex =
Integer.parseInt(arraychrmosomePop[x]);
                        String pop = matrix.arrayPopulations[popIndex];
                        String popInfo = "\n" +
pop.substring(pop.length()-9,pop.length());
```

```
                        String lociInfo = " ";
                        for(int y=0;y<arraychrmosomeLoci.length;y++){
                              int lociIndex =
Integer.parseInt(arraychrmosomeLoci[y]);
                              String value =
(matrix.getElement(lociIndex,popIndex)).trim();
                              String[] splitValue = value.split(":");
                              String newVal="";
                              for(int q=0;q<splitValue.length-1;q++)
                                  newVal+=splitValue[q]+ " ";
                              newVal=newVal.trim();
                              lociInfo+=" " + newVal;
                        }
                        bw.write( popInfo + lociInfo );
                        bw.flush();
                  }
            }catch(IOException ioe){
                  System.out.println("Error writing to file");
            }
            try{
                  bw.close();
            }catch(IOException ioe){
                  System.out.println("Error closing file");
            }

            //Chromosomes Lists
            simpleFileName = "../Results/" + fileName + ".chromosomes";

            //write to file
            try {
                  bw = new BufferedWriter(new
FileWriter(simpleFileName));
            }
            catch (IOException ioe){
                  System.out.println("Usage 'java IO filename'");
                  System.out.println("Be sure filename exists");
            }

            for(int x=0;x<population.size();x++){
                  //boolean[] chromosome;
                  int numberOfPop=0;
                  int numberOfLoci=0;

                  chromosome = (boolean[])population.get(x);

                  for(int i=0;i<matrix.getNumbCols();i++){
                        if(chromosome[i]==true)
                              numberOfPop++;
                  }

                  for(int i=matrix.getNumbCols();i<chromosome.length-
1;i++){
                        if(chromosome[i]==true)
                              numberOfLoci++;
                  }
```

```
                  // Simple view
                  fitness =
((Double)populationFitness.get(x)).toString() + "\t>" +
computeFitness(numberOfPop,numberOfLoci) ;

                  try{
                        bw.write( x + "\t" + numberOfPop  + " x " +
numberOfLoci  + "\t" +
fitness + "\n");
                        bw.flush();
                  }catch(IOException ioe){
                        System.out.println("Error writing to file");
                  }

            }

            try{
                  bw.close();
            }catch(IOException ioe){
                  System.out.println("Error closing file");
            }



            //XML
            String XHTMLFileName = "../Results/" + fileName + ".xml" ;
            //write to file
            try {
                bw = new BufferedWriter(new FileWriter(XHTMLFileName));
                bw.write("<?xml version=\"1.0\" encoding=\"ISO-8859-
1\"?>\n");
                bw.write("<!DOCTYPE dataset [\n");
                bw.write("<!ELEMENT dataset (population+)>\n");
                bw.write("<!ELEMENT population (site+)>\n");
                bw.write("<!ELEMENT site (allele_frequency+)>\n");
                bw.write("<!ELEMENT allele_frequency (#PCDATA)>\n");
                bw.write("]>\n");
                bw.write("<dataset>\n");
                bw.flush();
                for(int x=0;x<arraychrmosomePop.length;x++){
                  int popIndex =
Integer.parseInt(arraychrmosomePop[x]);
                  String pop = matrix.arrayPopulations[popIndex];
                  String[] popInfo = pop.split(":");
                  bw.write("<population id=\"" + popInfo[1] + "\"
name=\"" +
popInfo[0] + "\">\n");
                  for(int y=0;y<arraychrmosomeLoci.length;y++){
                      String sid =
matrix.arrayLoci[Integer.parseInt(arraychrmosomeLoci[y])];
                      bw.write("<site id=\"" + sid + "\">\n");
                      String alleleFreqs =
matrix.getElement(Integer.parseInt(arraychrmosomeLoci[y]),popIndex);
                      String[] arrayFreqs = alleleFreqs.split(":");
                      for(int q=0;q<arrayFreqs.length;q++){
                        bw.write("<allele_frequency>" + arrayFreqs[q] +
"</allele_frequency>\n");
```

```
                    bw.flush();
                }
                bw.write("</site>\n");
              }
              bw.write("</population>\n");
            }
            bw.write("</dataset>\n");
            bw.flush();
            bw.close();
        }catch(IOException ioe){
            System.out.println("Error writing XML file");
        }


        //XHTML
        XHTMLFileName = "../Results/" + fileName + ".html" ;
        //write to file
        try {
            bw = new BufferedWriter(new FileWriter(XHTMLFileName));
            bw.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0
Strict//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd\">\n");
            bw.write("<html xmlns=\"http://www.w3.org/1999/xhtml\"
xml:lang=\"en\" lang=\"en\">\n");
            bw.write("<head>\n<title>Finding the Largest Complete
Dataset
from ALFRED</title>\n</head>\n<body>\n");
            bw.flush();

bw.write("<table border=\"1\" cellspacing=\"0\"
cellpadding=\"5\">\n");
bw.write("  <caption>Largest Complete DataSet from
ALFRED</caption>\n");
bw.write("  <tr>\n");
bw.write("    <td colspan=\"2\" rowspan=\"2\"></td>\n");
bw.write("    <th colspan=\"" + arraychrmosomePop.length + "\"
align=\"left\">Populations</th>\n");
bw.write("  </tr>\n");
bw.write("  <tr>\n");
      for(int x=0;x<arraychrmosomePop.length;x++){
            int popIndex = Integer.parseInt(arraychrmosomePop[x]);
            String pop = matrix.arrayPopulations[popIndex];
            String[] popInfo = pop.split(":");
            bw.write("        <th>" + popInfo[0] + " - " + popInfo[1] +
"</th>\n");
            }
bw.write("  </tr>\n");
bw.write("  <tr align=\"center\">\n");
bw.write("    <th valign=\"top\" rowspan=\"" +
arraychrmosomeLoci.length + "\">Polymorphisms</th>\n");
    for(int y=0;y<arraychrmosomeLoci.length;y++){
      String sid =
matrix.arrayLoci[Integer.parseInt(arraychrmosomeLoci[y])];
      if(y>0)
          bw.write("    <tr align=\"center\">\n");
      bw.write("        <th>" + sid + "</th>\n");
      for(int x=0;x<arraychrmosomePop.length;x++){
```

```
          int popIndex = Integer.parseInt(arraychrmosomePop[x]);
            String alleleFreqs =
matrix.getElement(Integer.parseInt(arraychrmosomeLoci[y]),popIndex);
            alleleFreqs = alleleFreqs.substring(0,alleleFreqs.length()-
1);
                        bw.write("        <td>" + alleleFreqs +
"</td>\n");
      }
      bw.write("  </tr>\n");
    }
bw.write("</table>\n");
bw.write("<p>\n");
bw.write("  <a
href=\"http://validator.w3.org/check?uri=referer\"><img");
bw.write(" src=\"http://www.w3.org/Icons/valid-xhtml10\"");
bw.write(" alt=\"Valid XHTML 1.0 Strict\" height=\"31\" width=\"88\"
/></a>");
bw.write("</p>\n");
bw.write("</body>\n");
bw.write("</html>");


                bw.flush();
                bw.close();
            }catch(IOException ioe){
                System.out.println("Error writing XML file");
            }



      }

      public void removeTemp(){
            if((new
File("../Results/currentFittestChromosome.txt")).exists())
                  (new
File("../Results/currentFittestChromosome.txt")).delete();
            if((new
File("../Results/currentFittestChromosome.phylip")).exists())
                  (new
File("../Results/currentFittestChromosome.phylip")).delete();
            if((new
File("../Results/currentFittestChromosome.chromosomes")).exists())
                  (new
File("../Results/currentFittestChromosome.chromosomes")).delete();
      }

      /**
      * Returns the fitness score of a chromosome
      */
      private double getFitness(boolean[] chromosome){
            int numberOfPop=0;
            int numberOfLoci=0;

            for(int x=0;x<matrix.getNumbCols();x++){
                  if(chromosome[x]==true)
                        numberOfPop++;
```

```
            }

            for( int y=matrix.getNumbCols();y<chromosome.length-1;y++){
                  if(chromosome[y]==true)
                        numberOfLoci++;
            }

            double fitness = computeFitness(numberOfPop,numberOfLoci);

            return fitness;
      }

      /**
      * Sort the chromosomes in the population by thier fitness
(decending
fitness)
      */
      public void sortPopulation(){

            //an array of the original index positions
            int[] indexes = new int[population.size()];
            for(int i=0;i<indexes.length;i++){
                  indexes[i]=i;
            }

            double f1,f2,fTmp;
            boolean[] c1,c2,cTmp;

            //bubble sort the fitness
            for(int i=0;i<populationFitness.size();i++){
                  for(int j=0;j<(populationFitness.size() - 1);j++) {
                        f1= ( (Double)populationFitness.get(j)
).doubleValue();
                        f2= ( (Double)populationFitness.get(j+1)
).doubleValue();
                        if(f2>f1) {
                              fTmp = f1;
                              f1 = f2;
                              f2 = fTmp;
                              populationFitness.set(j,new Double(f1));
                              populationFitness.set(j+1,new
Double(f2));
                              c1 = ((boolean[])population.get(j));
                              c2 = ((boolean[])population.get(j+1));
                              cTmp = c1;
                              c1=c2;
                              c2=cTmp;
                              population.set(j,c1);
                              population.set(j+1,c2);
                        }
                  }
            }
            needsSorting = false;
      }

}
```

APPENDIX E

```
# memoryTrack.pl
# Memory tracking
# Runs the program passed as an argument, and tracks the memory usage
# of the program using the /proc/PIC/status file.
# @version 1.0, Thurs. 6 April 2006
# @author Mohamed Uduman

#!/usr/bin/perl
My $progName = $ARGV[0];
system ("/usr/java/j2sdk1.4.2_06/bin/java $progName &");
system ("ps -a | grep java > track");
open (IN, "track") or die "Error";
while(<IN>){
    chomp;
    @top = split(/ /,$_);
    $pid=$top[1];
    last;
}
close(IN);

print "PID=$pid\n";

$cmd= "cat /proc/$pid/status | grep Vm> track";
@args = ("$cmd");
system(@args);

$flag="true";
while($flag=="true"){
    open (IN, "track") or die "Error\n";
    while(<IN>){
      chomp;

      if(/^Vm/){
          $cmd= "cat /proc/$pid/status | grep Vm >> trackGA.txt";
          @args = ("$cmd");
          system(@args);
          last;
          $cmd= "cat /proc/$pid/status | grep Vm> track";
          @args = ("$cmd");
          system(@args);
      }
      else{
          $flag="false";
          last;
          print "false\n";
      }

    }
    close(IN);
    sleep(60);
}
```

```
@args = ("kill","0","$pid");
system(@args);
```