Thesis

Use of Traveling Salesman Problem Solvers in the Construction of Genetic Linkage Maps

Submitted by

Zachariah A. Allen

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2015

Master's Committee:

Advisor: Darrell Whitley

Ross M. McConnell
John McKay

Abstract

Use of Traveling Salesman Problem Solvers in the Construction of Genetic Linkage Maps

Construction of genetic linkage maps is an important tool for Biology. Researchers use a variety of laboratory techniques to extract genetic marker data from an experimental cross of a species of interest and then use these data to group the markers into chromosomes, and then construct maps of the locations of the markers within the chromosomes. This in turn allows them to determine which sections of the chromosomes are responsible for variation in agricultural, medical or other traits of interest. The current methods of constructing genetic linkage maps are tedious and time-consuming.

This thesis presents a method of utilizing the Hamiltonian path problem (HPP) to solve the problem of genetic linkage mapping. Since solvers already exist for the traveling salesman problem (LKH and Concorde), by casting the linkage mapping problem as a HPP we can use these solvers to efficiently find the solution. To do this, the recombination frequencies between genetic markers are treated as internode weights and the TSP solution gives the lowest-cost path through the markers. By adding a dummy marker with zero weight to all other markers, the TSP solution is made equivalent to a HPP.

The primary difficulty in constructing a linkage map is the fact that all data sets are noisy: errors in laboratory techniques create uncertainty in the relationships between genetic markers, so a straightforward HPP solution is not sufficient. This thesis describes a method of using the HPP to separate the raw data into clusters so that the researcher can be sure that the chromosomes are well-separated, the clusters can then be assembled into complete chromosomes using existing TSP solvers.

The results show that this method produces results which are equally as good as the prevalent software in the field, while drastically decreasing the time required to run an analysis.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Deoxyribonucleic acid, or DNA, is the molecule which encodes the genetic information within a living organism. Since the discovery of the double-helix structure in 1953, understanding the DNA structure of plants and animals has become a major area of research in the biological sciences.

Understanding the order and structure of the genome of any organism is of paramount importance when associating genotypes and phenotypes [1], or understanding the mechanisms of adaptation of a species [2]. By identifying the parts of DNA that control variation in traits of interest, agricultural researchers can exploit and strengthen these traits to improve the quality of animals, crops and microorganisms.

1.1. Genetic Linkage Mapping

DNA is organized into a chain of chromosomes, which in turn are comprised of chains of genes. It is difficult to directly identify the genes themselves, but there are genetic markers within the chromosomes which are more easily detected. These markers may or may not control the traits of the species (they are both within and between coding regions), but understanding the structure of the genetic markers allows us to indirectly understand where genes of interest are located [1].

Each genetic marker occupies a particular fixed location within the chromosome called a *locus* (plural *loci*). During meiosis, the chromosomes of one parent are crossed with those of the other parent such that the offspring have DNA comprised of various combinations of the original genes, while maintaining the loci of the genetic markers. This is what allows for genetic diversity in an offspring population. Without genetic crossover at the chromosomal

	Parent A	Parent B
Locus 1	a	b
Locus 2	a	b
Locus 3	a	b
Locus 4	a	b
Locus 5	a	b
Locus 6	a	b
Locus 7	a	b
Locus 8	a	b

FIGURE 1.1. A single chromosome from both parents.

level, all offspring of two parents would be identical to one of the parents (except for the possible presence of mutations).

The goal here is to find the correct order and pairwise distance between all markers. This is not a straightforward process: the process begins with two parent individuals, denoted A and B. The chromosomes of these plants are analyzed and the genetic markers are identified, which allows researchers to track from which parent a particular region of a chromosome is inherited. It is important to keep in mind that, while the genetic markers can be identified, their locations on the DNA strand is not known at the beginning of the analysis.

We can extract a sequence of markers, each at the appropriate locus, and each being known to come from parent A. The first marker is the marker that originated in parent A at locus 1, the second marker is the marker that originated in parent A at locus 2, etc. The sequence extracted from parent B will be represented the same way (see Figure 1.1).

The two parents are then interbred to create a large population of offspring, and the offspring population is then interbred for several generations in order to achieve homozygocity. At each locus of this chromosome, each plant in the final population will have inherited two copies (homozygous) of a marker from either parent A or parent B, as shown in Fig 1.2.

	Offspring 1	Offspring 2	Offspring 3	Offspring 4	Offspring 5	Offspring 6	Offspring 7	Offspring 8
Locus 1	a	b	a	b	b	b	b	b
Locus 2	a	b	a	b	b	a	b	b
Locus 3	a	a	b	b	b	a	b	b
Locus 4	a	a	b	b	b	a	a	b
Locus 5	a	a	b	b	b	a	a	b
Locus 6	b	a	b	b	a	a	a	b
Locus 7	b	a	a	b	a	a	a	b
Locus 8	b	a	a	a	a	a	a	b

FIGURE 1.2. A single chromosome from several offspring.

	Parent A	Parent B
Locus 1	a	b
Locus 2	a	b
Locus 3	a	b
Locus 4	a	b
Locus 5	a	b
Locus 6	a	b
Locus 7	a	b
Locus 8	a	b

Original chromosomes

	Offspring 1	Offspring 2
Locus 1	a	b
Locus 2	a	b
Locus 3	a	b
Locus 4	b	a
Locus 5	b	a
Locus 6	b	a
Locus 7	b	a
Locus 8	b	a

Offspring chromosomes

FIGURE 1.3. A single crossover. The crossover point is indicated by the heavy line between loci 3 and 4.

Here we see many various combinations of genetic markers from the two parents. Notice that in some cases, an offspring plant may have a chromosome which only contains markers from one parent or the other.

This phenomenon is known as *genetic recombination* and this forms the basis for the method of constructing genetic linkage maps.

1.1.1. GENETIC RECOMBINATION. Genetic recombination occurs during a process called *meiosis*, wherein the corresponding chromosomes of the two parents will form one or more temporary connections at arbitrary loci and then separate from each other so that each chromosome connects to the corresponding segment of the other chromosome at that locus. These points are known as *crossover points*, which are illustrated in Figure 1.3.

In the Boveri–Sutton chromosome theory, also known as the chromosome theory of inheritance, it is assumed that loci are randomly chosen to be crossover points, and the crossover points are evenly-distributed along the chromosome [3]. This leads to the hypothesis that if two genetic markers are close together on the chromosome, they are less likely to be separated by a crossover.

For example, consider the crossover shown in Fig 1.3. Since the crossover point occurs between loci 3 and 4, these two markers are separated in the offspring (offspring #1 has marker a in locus 3 and marker b in locus 4, and vice versa for offspring #2). However, if the crossover point had occurred anywhere else along the chromosome, these two markers would have ended up in the same offspring (one offspring would have inherited a markers at loci 3 and 4 and the other offspring would have inherited b markers at loci 3 and 4). So we see in this simple case of a single crossover event, there is only a $\frac{1}{7}$ chance that two consecutive markers will be separated due to recombination. Furthermore, as the distance between two markers increases, so does the probability that they will be separated by a crossover event. This leads to the concept of recombination frequency.

1.1.2. Recombination Frequency. As described above, the distance between two markers within a chromosome effects how frequently those two markers will be separated by a crossover event (meaning that the markers from a single parent end up in different offspring). The reason for creating a large offspring population now becomes apparent: by measuring how often two markers from the same parent end up in the same offspring, we can estimate how likely they are to be separated by a crossover event. This probability is known as the *recombination frequency*, or *rf*.

Consider the very simple example shown in Fig 1.4. We see that there are three offspring where the markers are inherited from different parents (shown in bold), out of eight total

	Offspring 1	Offspring 2	Offspring 3	Offspring 4	Offspring 5	Offspring 6	Offspring 7	Offspring 8
Marker 1	a	b	**a**	a	**b**	**b**	b	b
Marker 2	a	b	**b**	a	**a**	**a**	b	b

FIGURE 1.4. Two example markers, measured across 8 offspring. The locations where the markers are inherited from different parents are shown in bold. The recombination frequency of these two markers is $\frac{3}{8} = 0.375$.

offspring (in a real dataset there will be a much larger number of offspring). This gives a recombination frequency of

$$rf = \frac{\text{number of markers inherited from different parents}}{\text{total number of offspring}} = \frac{3}{8} = 0.375 \tag{1}$$

By computing the recombination frequency between all pairs of markers, we are given a clue as to which sets of markers reside on the same chromosome. The computation of the *rf* value is simple: given two markers, we take the ratio of the number of times they are inherited from different parents with the total number of offspring.

For two markers that are close to each other on the chromosome, the recombination frequency will be a small value. As the distance between the two markers increases, so does the probability of recombination. At very large distances, eventually we will be considering two markers which are either very far apart on the same chromosome or on separate chromosomes. In this case, recombination events are no longer correlated with the occurrence of markers in the offspring chromosomes and the recombination frequency is approximately 50%.

1.1.3. IDEAL DATA. An idealized dataset would give us a perfect picture of the genetic linkage map. Consider the *rf* matrix shown in Table 1.1. In the absence of any gaps or noise in the data we could obtain such values which would allow us to immediately conclude

TABLE 1.1. Table of rf values for an idealized data set. The neighbors of each marker can be immediately determined simply by looking for the smallest values. The markers at the ends of the chromosome (markers 1 and 50) have a single neighbor with $rf = 0.01$. All other markers have two neighbors with $rf = 0.01$. A simple greedy algorithm could be used to construct the genetic linkage map for such an ideal data set.

Marker	2	3	4	5	...	46	47	48	49	50
1	0.01	0.02	0.03	0.04	...	0.45	0.46	0.47	0.48	0.49
2		0.01	0.02	0.03	...	0.44	0.45	0.46	0.47	0.48
3			0.01	0.02	...	0.43	0.44	0.45	0.46	0.47
4				0.01	...	0.42	0.43	0.44	0.45	0.46
5					...	0.41	0.42	0.43	0.44	0.45
...										
46							0.01	0.02	0.03	0.04
47								0.01	0.02	0.03
48									0.01	0.02
49										0.01

which markers are adjacent to each other: marker 1 has a single neighbor with $rf = 0.01$, indicating that this marker is on the end of the chromosome. Marker 2 has two neighbors with $rf = 0.01$, both marker 1 and marker 3. Therefore, marker 2 would be between markers 1 and 3. In fact, for such data, every marker would have exactly two neighbors with $rf = 0.01$ (except for the markers on the ends of the chromosome) and a simple greedy algorithm could be used to construct the linkage map.

Additionally, it should be noted that, in keeping with recombination theory, as markers are located farther apart on the chromosome, the recombination frequency values get larger and larger. The rf values of marker 1 as compared with markers 48, 49, and 50 are all approximately 0.5. So even though these markers are located on the same chromosome, they are far enough apart that their inheritance is completely uncorrelated with each other. This will be an important phenomenon when we discuss missing data (Section 3.2).

1.1.4. REAL DATA. Table 1.2 shows some sample rf values from chromosomes #1 and #3 of the arabidopsis genetic linkage map. Arabidopsis is a small flowering plant. It was the

TABLE 1.2. Table of rf values for selected arabidopsis markers. Markers which are close to each other have lower recombination frequency values. As markers get farther apart, the *rf* value between them increases. The markers from different chromosomes have rf values of approximately 0.5, indicating no correlation between them.

Chromosome	1					3				
Locus	1	2	5	9	10	3	4	7	10	11
Marker Name	L1_1	L1_2	L1_5	L1_9	L1_10	L3_3	L3_4	L3_7	L3_10	L3_11
L1_1		0.028	0.063	0.08	0.10	0.537	0.542	0.545	0.545	0.544
L1_2			0.039	0.059	0.074	0.539	0.544	0.547	0.554	0.553
L1_5				0.030	0.045	0.531	0.535	0.540	0.553	0.552
L1_9					0.015	0.528	0.531	0.536	0.546	0.549
L1_10						0.521	0.520	0.526	0.537	0.542
L3_3							0.013	0.034	0.060	0.064
L3_4								0.022	0.049	0.052
L3_7									0.032	0.036
L3_10										0.011

first plant to have a fully-sequenced genome. Its genetic linkage map is well-established and therefore it makes a good example for comparison when investigating a new technique. Also, the arabidopsis dataset is fairly small (367 markers measured from 545 individual offspring plants) and mostly complete, so we avoid the problems that arise with missing data (see section 3.2).

The markers from chromosome #1 have low *rf* values with each other, but when compared with the markers from chromosome #3 the *rf* values are all approximately 0.5. This indicates that there is no correlation between the inheritance of the markers from different chromosomes, which is expected since the crossover points from one chromosome do not affect those from any other chromosome. Also, the markers from chromosome #3 are highly correlated with each other.

Furthermore, we see that the *rf* value between marker L1_1 and L1_2 is lower than the *rf* value between L1_1 and L1_5. This is consistent with the fact that L1_1 and L1_2 are

directly adjacent to each other (occupying loci 1 and 2, respectively), but L1_1 and L1_5 are farther apart (loci 1 and 5, respectively).

Also, the *rf* value between L1_9 and L1_10 is small, due to that fact that they also are directly adjacent to each other. The *rf* between L1_1 and L1_10, while still a relatively small value, is clearly larger than any of the other *rf* values within chromosome #1 which is consistent with what we expect.

The recombination frequencies between markers in chromosome 3 show similar relationships in terms of the distances between them.

Since the recombination frequency matrix is large, it can be useful to visualize it using a heat map. In this case the heat map is a 2D plot of a matrix which uses color to indicate the relative values within the matrix. For visualizing a recombination frequency matrix, each axis will represent the full array of markers. Each point on the plot therefore represents the *rf* value between the marker at the x and y positions of that point. The points along the diagonal $x = y$ are the recombination frequency values of each marker with itself, and therefore have the value 0 by definition. Values of low correlation (high recombination frequency) are shown in blue, progressing through the color spectrum to lower-*rf* values, where the highest correlation ($rf = 0$) is shown in red.

Figure 1.5 shows the heatmap of rf values for 295 markers of the arabidopsis species (after markers of 99% similarity were removed - see section 3.1). Here it is clear where the delineations are between the five chromosomes.

This heat map vividly shows the behavior of recombination frequency values, both within a chromosome and between chromosomes. Within each chromosome we see obvious areas of low-value rf, and beyond the limits of each chromosome there is no expectation of any meaningful linkage between any markers.

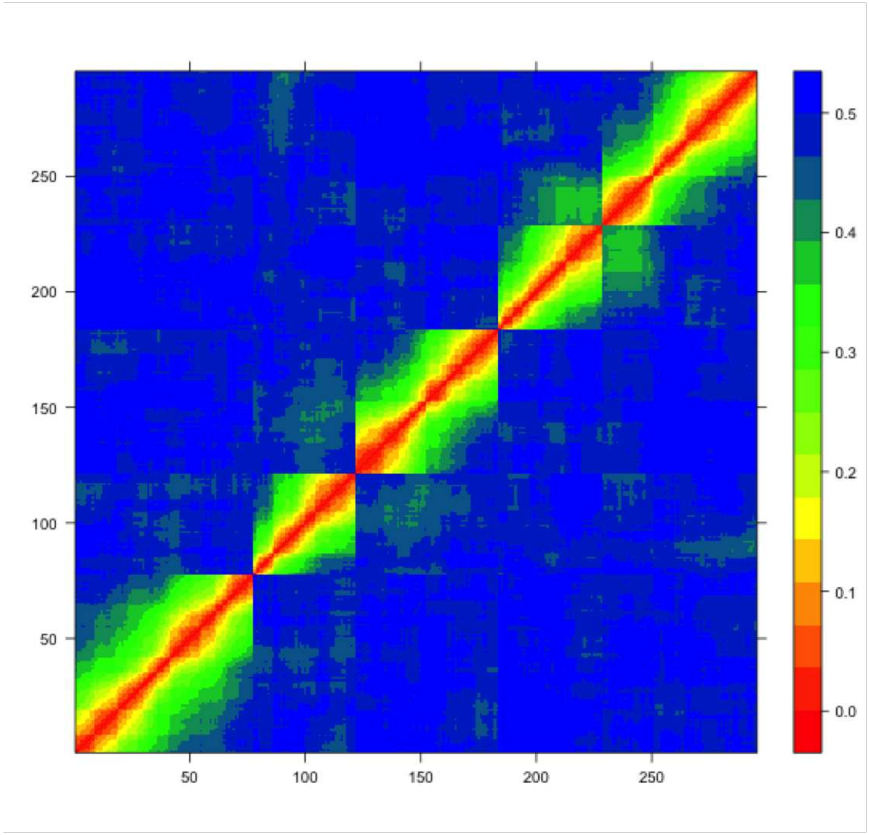


FIGURE 1.5. Heat map of recombination frequency values for the genetic linkage map of arabidopsis. The x and y axes represent the full array of markers and each point within the plot shows the recombination frequency between the marker at point x and the marker at point y. The diagonal represents the rf value of each marker with itself, which is always 0. High rf values are shown in blue, $rf = 0$ is shown in red. The five chromosomes can be identified by the five square-shaped areas of low-rf values along the diagonal, indicating that the markers within these regions have low recombination frequency values relative to each other (bright green, yellow, and red), and high recombination frequency values relative to the markers in other regions (dark green and blue).

Not all data sets are so clean however. Figure 1.6 shows the heat map for IR64B, an experimental mapping population of rice. The percentage of missing data is much higher in the IR64B dataset, and it is clear that the heat map is not so smooth as that for arabidopsis.

1.2. Travelling Salesman Problem

The travelling salesman problem (TSP) is a well-known computational problem which has been a major focus of optimization efforts for many years [8]. The TSP is formulated as

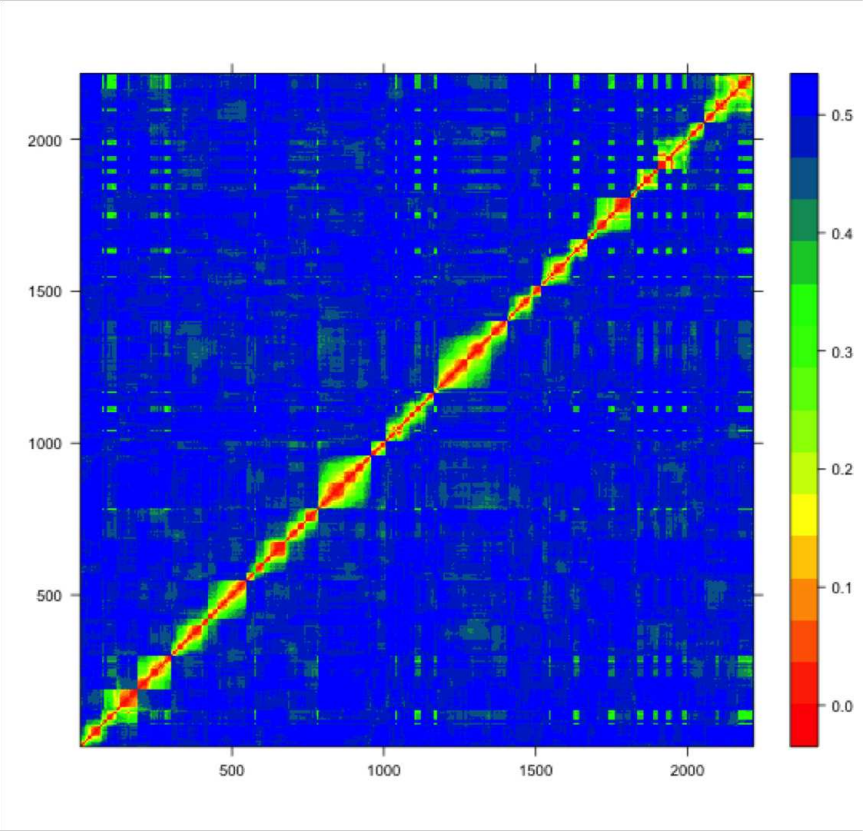
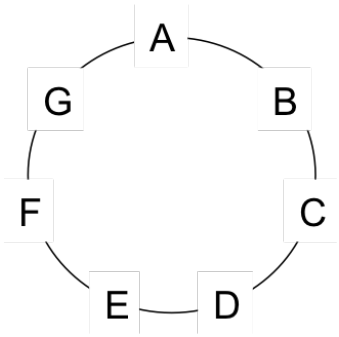


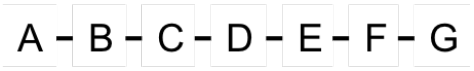
FIGURE 1.6. The heat map for the IR64B data set, which contains 12 chromosomes. The chromosomes are not clearly segregated, but it is possible to recognize a few square-shaped areas indicative of a grouping of highly-correlated markers. There are many areas far from the diagonal which show low recombination frequency values (shown in bright green), indicating relatively high correlation between markers that are located in different chromosomes.

a problem in which an agent wishes to visit a set of nodes at the lowest possible total cost. This method can be applied to many different kinds of problems, including vehicle routing, the order of drilling holes in PCB board manufacturing, and laying a network of fiber optic cables. In this project, the TSP method is applied to the genetic linkage mapping problem.

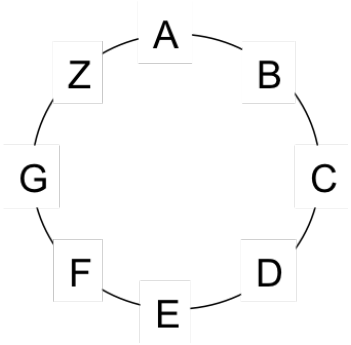
The similarity between the TSP and genetic linkage map construction has been known for quite some time [9]. By considering each marker to be a node in the TSP tour, the *rf* values between markers represent the weights between these nodes. This allows the recombination frequency matrix to serve as the weight matrix in the TSP, and the solution will give us the lowest-cost path through the markers.



(a) Standard TSP solution, a Hamiltonian cycle



(b) Basic Hamiltonian path



(c) Hamiltonian cycle with dummy node Z

(d) Hamiltonian path with dummy node Z

FIGURE 1.7. Hamiltonian cycle versus Hamiltonian path. The Hamiltonian cycle (a) visits every node and terminates at the beginning node A, so the weight of the edge $\overline{AG}$ is included in the total solution cost. A Hamiltonian path (b) visits each node but does not terminate at the beginning node, so the weight between nodes G and A does not affect the total cost. By adding a dummy node Z (c) which has zero weight between all other nodes, the edge cost of $\overline{AG}$ is removed from the total cost. Therefore, this configuration is equivalent to the HPP (d) because of the presence of the dummy node Z.

The two most prevalent TSP solvers in use today are Lin-Kernighan-Helsgaun (LKH), a heuristic solver, and Concorde, an exact solver.

1.2.1. TRAVELLING SALESMAN PROBLEM VERSUS HAMILTONIAN PATH PROBLEM. In the typical travelling salesman problem formulation, the solution is a Hamiltonian cycle (Figure 1.7a). That is, a tour in which each node is visited exactly once and ends at the beginning node, thus forming a complete cycle. In the case of a genetic linkage map, we have no need to consider the linkage between the last marker in the chromosome and the first. In

fact, allowing the TSP to run in this standard configuration will negatively impact the result since the algorithm will include the value of this last edge when attempting to minimize the total tour cost, and the *rf* value of the beginning node and ending node are expected to be large since they are so far apart on the chromosome.

To eliminate this effect we convert TSP problem into a Hamiltonian path problem (HPP, Figure 1.7b). To accomplish this, we introduce into the TSP weight matrix a dummy node (Figure 1.7c) which has a zero-weight connection to all other nodes. The inclusion of this node allows the TSP solver to connect the last node in the tour with the first without incurring the large weight penalty associated with the *rf* value between two distant nodes, in essence allowing the solver to construct a non-cyclic path [13]. Therefore, the Hamiltonian cycle shown in Figure 1.7c is equivalent to the Hamiltonian path in Figure 1.7d.

By doing this, we can make use of existing TSP solvers without making any modifications to their algorithms. Throughout this thesis, the terms "TSP" and "HPP" are used interchangeably since the conversion from TSP to HPP is incorporated into the algorithm.

1.2.2. Lin-Kernighan-Helsgaun (LKH). The basis of the Lin-Kernighan-Helsgaun solver is the Lin-Kernighan heuristic [10]. This heuristic works by rearranging subtours of a candidate TSP solution using a variable k-opt technique which allows the algorithm to perform a kind of iterated local search, while ensuring that the local search does not get stuck in local minima.

The LK heuristic was improved by K. Helsgaun [11]. LKH is an approximate method, which allows it to run in significantly less time than an exact solver, such as Concorde. However, the performance of LKH is still very good, and it is not unusual to obtain the optimal solution using LKH for smaller datasets ($n \leq 1000$). For the purposes of this project, LKH is used in the early stages of the process, mainly to identify and separate the

chromosome groups. Once these groups are identified, the final linkage map will be produced by Concorde.

1.2.3. Concorde. Concorde is an exact solver based on the branch-and-bound method [12]. Branch-and-bound is a technique used to prune the search space and limit unnecessary exploration of areas of the state space which are guaranteed not to produce improvement on an already existing result.

In the case of Concorde, as the algorithm explores the state space it keeps track of the best-known (i.e., lowest-cost) solution that it has yet produced. As the algorithm explores new areas of the state space, it checks to see if the newly-constructed partial candidate solution is determined to be worse than the current best solution. If so, there is clearly no point in continuing to explore further, since any solutions containing that portion will necessarily be higher-valued than the best known solution. Therefore, the algorithm eliminates the entire branch of the solution space, thus avoiding spending time in pointless computation.

Because Concorde is an exact method, it is guaranteed to find the optimal TSP solution. Because of this, its run time may be much longer than that of LKH, especially as the problem size grows.

1.3. Current Approaches

The goal of constructing a genetic linkage map is to recover the marker order within each chromosome from the recombination frequency values computed from the individual marker data. The order of chromosomes within the DNA does not matter, but it is important to note that the number of chromosomes will already be known prior to beginning the analysis.

1.3.1. JoinMap. The currently prevalent software tool for constructing genetic linkage maps from recombination frequency data is a program called JoinMap [4]. The JoinMap

technique takes place in two main steps: first the linkage groups are determined, and then the linkage map is constructed for each group [5].

JoinMap is largely based on a statistical approach to genetic linkage map construction, using a combination of four basic techniques to construct the linkage map: independence test logarithm of odds (LOD) score, linkage LOD score, independence test P-value, and recombination frequency.

LOD, or logarithm of odds, is a statistical technique that has been in use for genetic linkage mapping since 1947 [6]. The LOD score for a pair of markers is an estimate of the probability of observing the given dataset in the case the markers are uncorrelated. This is calculated as the base-10 logarithm of the ratio of the probability of the given data distribution in the case where the markers are linked with the probability of the given data distribution in the absence of a link between the markers.

Once the linkage groups have been constructed, the individual linkage maps are built up by adding markers one-by-one in a three-round process. In the first round, the algorithm chooses which marker to add based on the LOD scores of all unmapped markers. Each marker is added by computing a goodness-of-fit score at each possible position on the map, and the best result is chosen. A marker may be removed from the map at this stage if it leads to degradation in the solution quality.

The second round consists of an attempt to reintroduce the markers that were removed from the linkage map in step 1. Since this stage begins with a more complete linkage map (i.e., the map resulting from stage 1), there is the possibility that these markers may find a place in the map where they improve the solution. However, this is not guaranteed and markers which cannot be placed are discarded.

In the final round, all markers previously removed from the map are added back into the map in an attempt to gain some general knowledge about the location of the markers which the algorithm is unable to properly map. This step is not meant to produce a high-quality result.

It can be seen that the highly iterative nature of this process can be very time consuming, taking several days for a data set of approximately 1,000 offspring with 2,000 markers.. Also, removing duplicate markers is done manually in a tedious process. Because of this, it is desirable to construct a new technique that is faster and simpler.

1.3.2. MSTmap. MSTmap attempts to construct the genetic linkage map using a modified minimal spanning tree [7]. The theoretical underpinning of MSTmap is that, in the absence of noise the MST solution is identical to the TSP solution. On the other hand, when noise is present in the data, the MST produces branching clusters which the MSTmap algorithm then transforms into a linear path using iterative application of three heuristics: 2-opt, node relocation, and a block-optimization heuristic.

MSTmap converts the TSP clusters into Hamiltonian paths, then applies the 2-opt heuristic by breaking the path into 3 pieces and swapping two of the pieces until the lowest-cost permutation is found. Then, the node relocation heuristic operates by relocating each node in the cluster to every other possible position to see if any improvement can be made. In the final stage, MSTmap uses the block-optimization heuristic to break the clusters into blocks consisting of low-cost subtours and rearranges the blocks until the smallest overall cost is found.

MSTmap is geared for noiseless data, and uses an expectation-maximization (EM) algorithm to extrapolate values for missing data.

CHAPTER 2

Basic TSP Example

2.1. Clean Data - Arabidposis

The most basic approach is to simply run the *rf* matrix through the TSP solver to obtain the Hamiltonian path. The path for the arabidopsis data set is shown in Figure 2.1. Using the known genetic linkage map of arabidopsis, we can color the markers based on their true chromosome membership in order the verify this solution. The x-axis shows the position of each marker, in order, and the y-axis is the cumulative recombination frequency.

Since the goal of constructing a genetic linkage map is to find the marker order within each chromosome, we need a method to identify and separate the chromosomes. The number of chromosomes is known at the beginning of the process: in the case of arabidopsis, there are five chromosomes. Since we know that the *rf* values between chromosomes should be large, we identify the locations of the largest *rf* values in the Hamiltonian path and mark these locations on the plot as candidate inter-chromosomal cutpoints. Since the theoretical

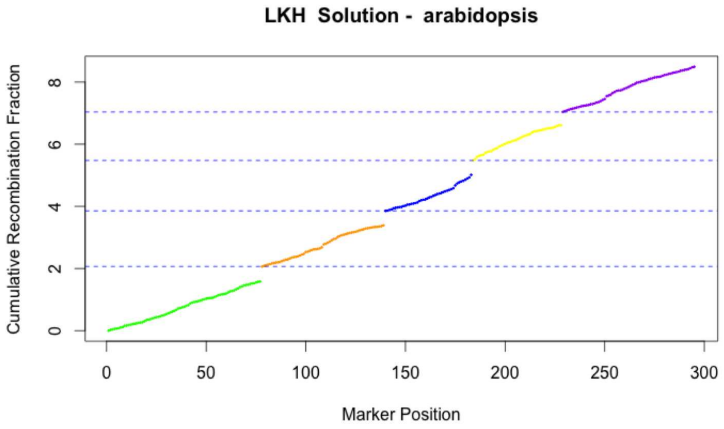


Figure 2.1. Hamiltonian path for arabidopsis, including cutpoints at the four largest *rf* values. The markers are color-coded by chromosome as given by the known genetic linkage map for this species. The cutpoints show the locations of the four largest *rf* values (see Table 2.1).

TABLE 2.1. Table of the 6 largest *rf* values in the HPP solution for the arabidopsis genetic linkage map. The four largest values occur at the breakpoints between chromosomes in the Hamiltonian path, shown as dotted lines in Figure 2.1. The fifth largest value is significantly smaller, indicating that it is clearly not a breakpoint between chromosomes.

Marker Position	Recombination Frequency
78	0.482
140	0.469
184	0.463
229	0.425
250	0.092
87	0.089

rf value of uncorrelated genetic markers is 0.5, the closer to this value the cutpoint values are, the more confidence we have that it is a true breakpoint between chromosomes. These cutpoints are shown in Figure 2.1 as dotted lines, and the *rf* cutpoint values are given in Table 2.1.

We see that the four largest *rf* values in Table 2.1 are all well above 0.4, and the next largest value is less than 0.1. This is a strong indication that the chromosomes are well-separated and no further processing is necessary in order to identify the chromosome groups.

2.2. Noisy Data - IR64B

When we apply the same technique to the IR64B dataset, we get the results shown in table 2.2 and figure 2.2. Here the results are not so clean. Because of the high levels of missing data, the *rf* values between markers are less reliable in many places. When we break apart the TSP solution into 12 groups, we see the first major problem that will be encountered in this kind of analysis: one chromosome has been split into two pieces.

Notice the red-colored group inside the solid circle. There is a dotted line passing through this group, indicating that one of the cutpoints from Table 2.2 occurs somewhere within the interior of these points. These red markers are part of single chromosome, but the raw data

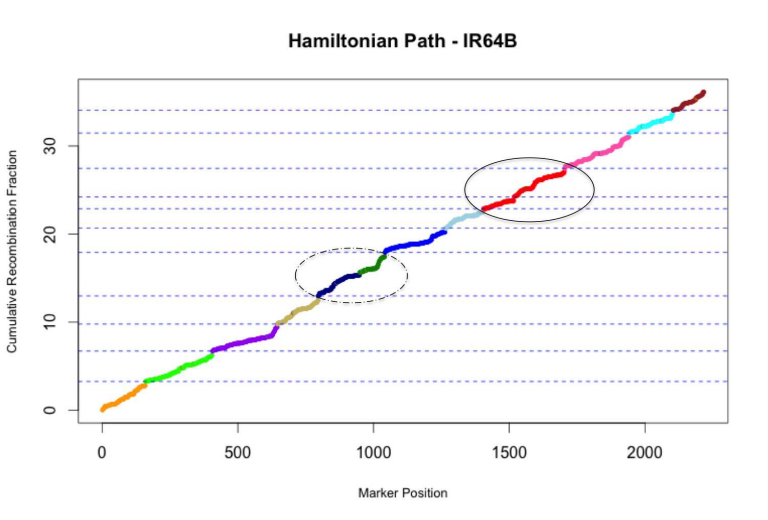


FIGURE 2.2. TSP solution for IR64B, including cutpoints at the 11 largest *rf* values, to create 12 groups. These cutpoints have not properly separated two of the chromosomes (solid circle) and erroneously left two other chromosomes in a single group (dotted circle).

is missing a large portion of markers in the middle of this chromosome (see Section 3.2). This leads to a large recombination frequency between the markers at the edges of this gap, which in turn prevents the algorithm from properly merging these two groups.

The second major problem that we encounter is a direct result of the first: since there is a large break within the red chromosome, we have mistakenly identified this break as one of the cutpoints that we need in order to produce 12 groups. Therefore, there is a true cutpoint, i.e. a true inter-chromosome break, that has not been identified, leading to a group which contains two different chromosomes (inside the dotted circle). This is the worse of the two errors, as it is more damaging to downstream efforts to map traits and genes to chromosomes if multiple chromosomes are fused than if a single chromosome is split.

To attempt to correct this problem we add another cutpoint at the next-largest *rf* value from the Hamiltonian path. There are now 12 cuts and 13 groups, as shown in Figure 2.3, and we see that the problem still remains: the new cutpoint cuts through the blue chromosome instead of breaking apart the two chromosomes within the dotted circle.

TABLE 2.2. Table of the largest 15 *rf* values in the TSP solution for the IR64B genetic linkage map. The 11 largest values (above the horizontal line) are chosen as cutpoints on Figure 2.2, separating the Hamiltonian path into 12 groups. There is no clear separation in this table between values which occur between chromosomes and values within chromosomes, which is a consequence of noise in the raw data.

Marker Position within Hamiltonian Path	Recombination Frequency
1405	0.487
300	0.484
1811	0.475
1164	0.468
957	0.466
1520	0.464
546	0.453
189	0.445
1743	0.393
1166	0.335
118	0.327
784	0.319
1943	0.319
76	0.266
573	0.266

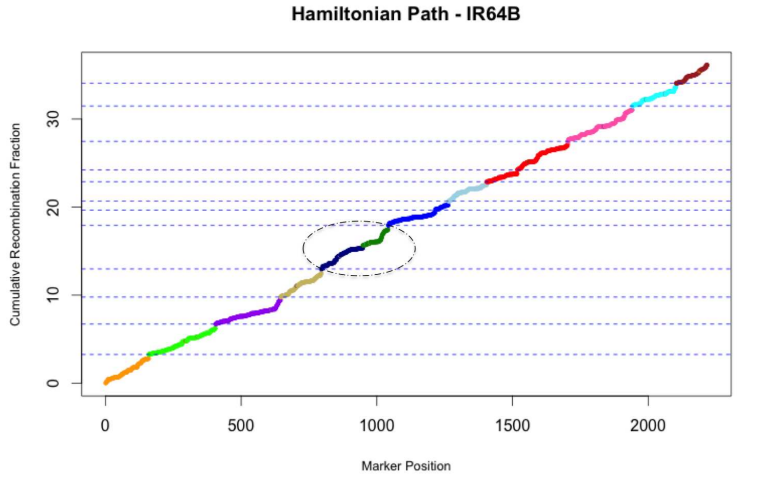


FIGURE 2.3. TSP solution for IR64B, including cutpoints at the 12 largest *rf* values, to create 13 groups. The addition of another cutpoint still has not separated the two circled chromosomes.

The conclusion is that the basic TSP approach is only viable for an exceedingly noiseless data set.

CHAPTER 3

Algorithm Overview

The approaches in this thesis uses the LKH and Concorde TSP solvers to form genetic linkage maps. The recombination frequency values are used as the inter-node weights, and therefore the solution of the Hamiltonian path problem is the solution with the lowest overall weight, which in turn is the best genetic linkage map since the markers will be placed in the optimal order in terms of *rf* value. Since the genetic map is comprised of several chromosomes, all of which are isolated from each other, this leads to a solution which is actually a grouping of independent, smaller Hamiltonian paths, each separated from the others buy a single, high-weight edge. This can be seen in Figure 1.5: since the markers in different chromosomes have such high *rf* values relative to each other, the TSP solvers will easily segregate each chromosome, leading to what is essentially a clustered travelling salesman problem (Figure 3.1).

The structure of the linkage map data allows us to use a divide-and-conquer approch by breaking the data into clusters

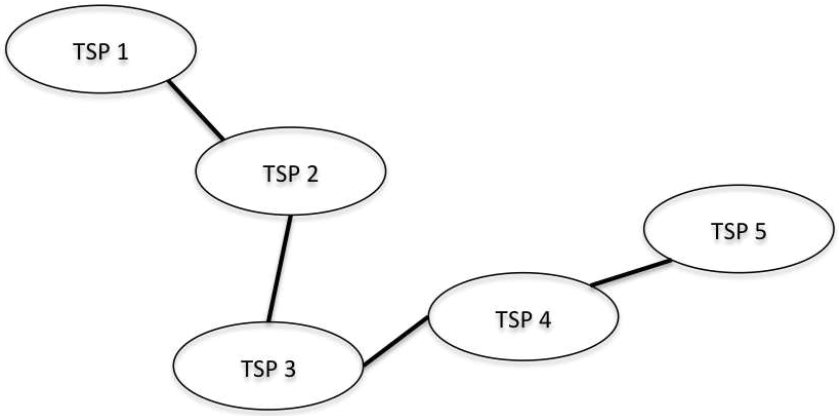


Figure 3.1. Example of a clustered traveling salesman problem. The clusters are separated by single, high-weight edges. This allows each cluster to be separated out and solved independently of the others, vastly decreasing the runtime of the TSP solver.

When applied to clean, well-behaved data, such as the arabidopsis dataset, we see the clear results that we expect (Figure 2.1). However, for noisy datasets, it is not always clear where the inter-chromosome breaks are (Figure 2.2). Because of this, the technique must be made more robust to account for the possibility that the solution will not be well-behaved.

The outline of the method of creating a genetic linkage map using TSP solvers is as follows (a diagram is shown in Figure 3.2.):

(1) Filter the raw data by removing duplicate markers and markers which are 99% similar.

(2) Compute recombination frequency values between all pairs of markers.

(3) Using the recombination frequency matrix, the unordered dataset is broken into large clusters which may contain markers from more than one chromosome.

(4) The clusters are subdivided further into smaller clusters so that no cluster contains markers from more than one chromosome.

(5) The small clusters are merged together until all the markers from a single chromosome are contained within a single cluster.

(6) The final clusters are processed using Concorde to produce the optimal linkage map for the given data.

There are two methods by which the unordered data set may be separated into small clusters (steps 3 and 4): the first utilizes an LKH-based method (described in Chapter 4.1), the other is based on a minimum spanning tree (Chapter 4.2).

3.1. Preprocessing and Data Filtering

Filtering the marker data is an essential part of genetic linkage map construction. An individual marker is not highly valuable in and of itself and it is not uncommon for a

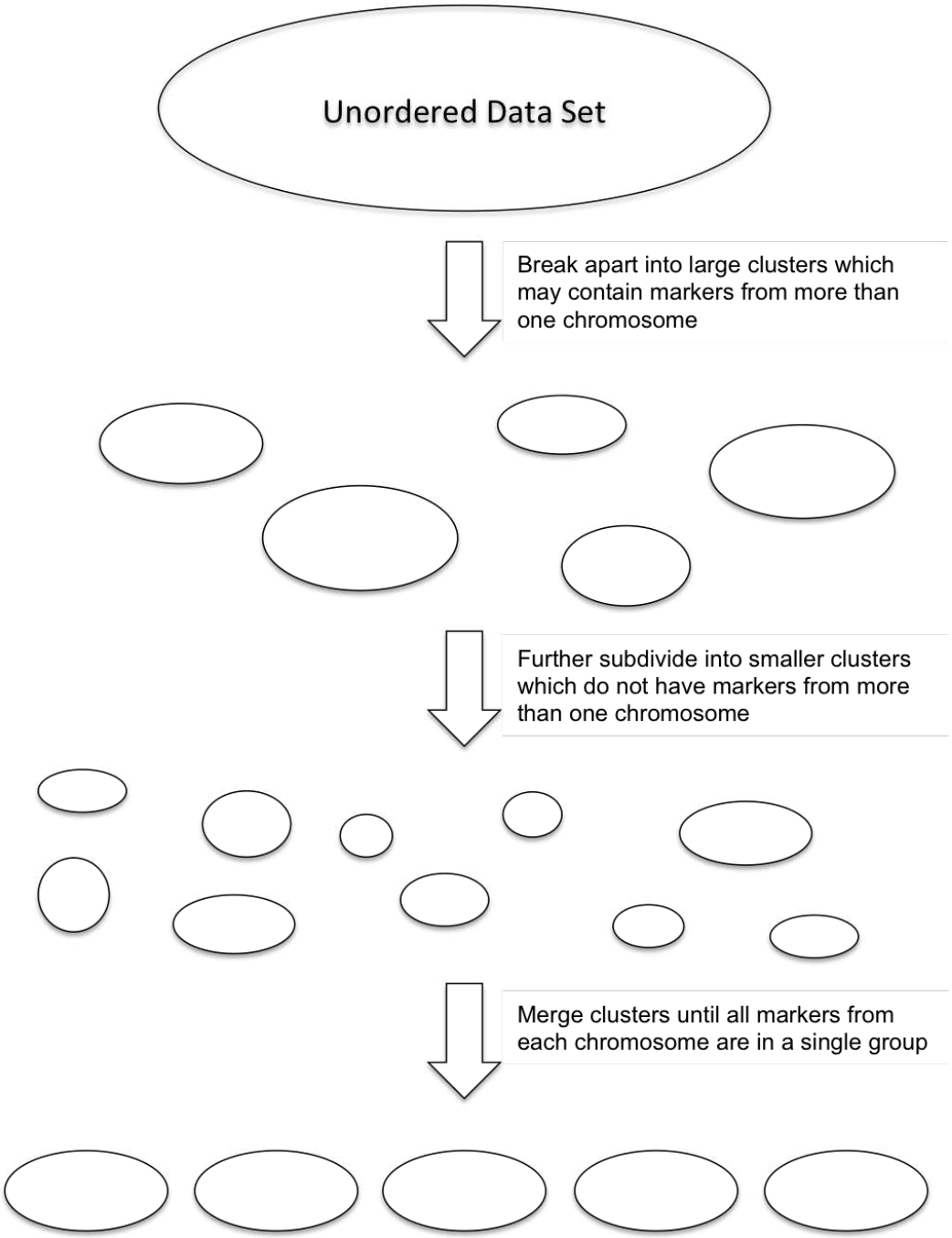


FIGURE 3.2. Diagram describing the overview of the construction of a genetic linkage map. First, the full data set is segregated into large clusters using either a TSP or MST method. These groups are further subdivided until no cluster contains markers from more than one chromosome. Finally, the clusters are merged together until each chromosome is fully contained within a single cluster.

researcher to filter out a large percentage of the raw data before constructing the linkage map, possibly as much as 90%. The IR64B dataset used in this thesis started out with over 4,000 markers but only 433 are used in the final analysis. The Hall's panicgrass dataset had over 3,500 markers in the raw dataset and only 880 made it to the final linkage map.

The more detailed aspects of filtering data is left to the agricultural sciences researchers as it depends on their judgment and experience. Only the most basic data filtering techniques, intended to remove redundant data, are included in this thesis.

3.2. Missing Data

In recent years, complexity reduction sequencing has emerged as a flexible and rapid method for genome-wide genotyping. Often called "Genotyping by Sequencing", this offers a way to get large numbers of genome-wide markers. However these methods are often implemented in a way that results in thousands to millions of individual markers, each of which may have a large proportion of missing data. This means that, within the dataset, we have to allow for the fact that an individual marker datum (known as a "call") may be missing for a particular offspring. In these cases, the simplest correction is to remove the offspring in question from the *rf* calculation so that the computation of the recombination frequency does not include the offending marker in the number of total offspring. Figure 3.3 shows an example where one of the calls from Marker 2 is missing. By removing offspring 5 from the *rf* calculation we are left with a total of 7 offspring, with two of them having markers from different parents. This results in a recombination frequency of $\frac{2}{7} = 0.286$.

By comparison with Figure 1.4 (where the *rf* value was 0.375) we see that the *rf* value between these two markers has now decreased, which would seem to indicate that the two markers are now more closely correlated. This is due to the fact that the missing call

	Offspring 1	Offspring 2	Offspring 3	Offspring 4	Offspring 5	Offspring 6	Offspring 7	Offspring 8
Marker 1	a	b	**a**	a	b	**b**	b	b
Marker 2	a	b	**b**	a	-	**a**	b	b

FIGURE 3.3. Example of computing the recombination frequency between two markers in the event of missing data. The locations where the markers are inherited from different parents are shown in bold. Offspring 5 is missing the call for marker 2, so this offspring will be omitted from the recombination frequency calculation, reducing the value of the total number of offspring from 8 to 7. The recombination frequency of these two markers is $\frac{2}{7} = 0.286$.

	Offspring 1	Offspring 2	Offspring 3	Offspring 4	Offspring 5	Offspring 6	Offspring 7	Offspring 8
Marker 1	a	b	**a**	a	**b**	**b**	b	b
Marker 2	a	b	**b**	-	**a**	**a**	b	b

FIGURE 3.4. Example of computing the recombination frequency between two markers in the event of missing data. The locations where the markers are inherited from different parents are shown in bold. Offspring 4 is missing the call for marker 2, so this offspring will be omitted from the recombination frequency calculation, reducing the value of the total number of offspring from 8 to 7. The recombination frequency of these two markers is $\frac{3}{7} = 0.429$.

represented a difference between Marker 1 and Marker 2 in the data shown in Figure 1.4. By losing this information, we have negatively impacted the *rf* value by underestimating it.

In contrast, the missing call could instead have been in one of the locations where the two markers originally were the same. Figure 3.4 shows an example of this, where the missing call represented a similarity in the original data of Figure 1.4. In this case, after removing offspring 4 from the *rf* calculation we are left with 7 total offspring which differ in 3 places, resulting in a recombination frequency of $\frac{3}{7} = 0.429$, now an overestimation.

From this we see that the method of removing an offspring from the *rf* calculation will always cause the value to be either overestimated or underestimated. Keeping in mind that real datasets have many more offspring than these simple calculations, the effects of a single missing marker will not be as drastic as these two examples show. Obviously, as the number of missing calls grows there will be more uncertainty in the *rf* values. This is unavoidable

in real-world situations and researchers are always faced with incomplete data sets. There is an additional complication arising from this issue, however, that specifically affects the TSP algorithm, which is addressed in the next section.

Another effect of missing data is that a single chromosome may be missing so many markers that it splits in two and cannot be reconciled into a single chromosome, since the markers of a single chromosome are uncorrelated when they are far enough apart (see Section 1.1.3). When this occurs, it is a problem with the raw data and cannot be rectified by any of the known algorithms.

3.3. Data Correction

As we have seen in section 3.2, missing data can cause recombination frequencies to be underestimated or overestimated, depending on whether or not the missing call represents a difference or similarity between the two markers.

Consider the case in Figure 3.5. If we use the method of removing offspring from the *rf* calculation, these markers will produce the recombination frequency values shown in Table 3.1. This gives a recombination frequency of zero between markers 1 and 3, and between markers 2 and 3 as well. However, the recombination frequency between markers 1 and 2 is 0.5, indicating that they are not correlated and should not be near each other within the linkage map. Since the TSP solver has no way of accounting for this, it will treat marker 3 as a zero-weight node between markers 1 and 2 and produce results where marker 1 is adjacent to marker 3, which itself is adjacent to marker 2. Since markers 1 and 2 are clearly not correlated ($rf = 0.5$), this will be a source of error in the TSP solution.

To correct for this, we consider the two extreme possibilities for the values of the missing calls in Marker 3, shown in Figure 3.6. Either the missing calls all match the calls in Marker

	Offspring 1	Offspring 2	Offspring 3	Offspring 4	Offspring 5	Offspring 6	Offspring 7	Offspring 8
Marker 1	a	a	a	a	a	a	a	a
Marker 2	a	a	a	a	b	b	b	b
Marker 3	a	a	a	a	-	-	-	-

FIGURE 3.5. Example of how missing data can cause the TSP algorithm to produce incorrect results. The *rf* value between Marker 1 and Marker 2 is 0.5, but the *rf* value between Markers 1 and 3, and Markers 2 and 3 are both 0. This can cause the TSP algorithm to produce a result where these three markers are linked, when clearly Markers 1 and 2 are uncorrelated.

TABLE 3.1. Table of rf values for the markers shown in Figure 3.5.

Marker	1	2	3
1		0.5	0
2			0

1, giving the smallest possible *rf* value, or they all differ, giving the largest possible *rf* value. The true value of the recombination frequency between these two markers must lie within this range.

To account for this uncertainty, recombination frequency values for markers with missing data are adjusted to lie at the midpoint of the possible range. Therefore, for the markers shown in Figure 3.5, the *rf* values between Markers 1 and 3 is adjusted to be

$$\frac{rf_{max} - rf_{min}}{2} = \frac{0.5 - 0.0}{2} = 0.25 \tag{2}$$

instead of 0 (likewise for the *rf* value between Markers 2 and 3) to avoid feeding artificially small *rf* values into the TSP solver (see Table 3.2).

It should be noted that the exact problem described here would not occur in a real analysis because Marker 3 would be removed from the data due to the fact that all of its information is contained within Marker 1. However, the principle is still valid and markers

	Offspring 1	Offspring 2	Offspring 3	Offspring 4	Offspring 5	Offspring 6	Offspring 7	Offspring 8
Marker 1	a	a	a	a	a	a	a	a
Marker 3	a	a	a	a	-	-	-	-
The extreme cases of the possible values of the missing calls in Marker 3:					a	a	a	a
					b	b	b	b

FIGURE 3.6. The two extreme possibilities for the missing calls for Marker 3 in Figure 3.5. In the first case, all missing calls match the calls in Marker 1, giving a minimum possible recombination frequency of $rf_{min} = 0$. In the second case, all missing calls differ from the calls in Marker 1, giving a maximum possible recombination frequency of $rf_{max} = 0.5$. The true value of the recombination frequency must lie within the range of these two values.

TABLE 3.2. Table of corrected rf values for the markers shown in Figure 3.5.

Marker	1	2	3
1		0.5	0.25
2			0.25

with missing data could still result in artificially low rf values, thus forcing the TSP solver to produce poor solutions.

3.4. Duplicate and Similar Markers

When comparing two markers and ignoring the individuals with missing calls, the remaining individuals may have the same calls at each loci (see Figure 3.7). In this situation, the marker which is missing more data is removed at the beginning of the analysis since it adds no information.

Furthermore, we can measure the similarity between two markers by calculating the percentage of calls which are the same between the two markers. Markers which are 99% similar or higher are omitted from the analysis. Other similarity thresholds may be applied, but here we use 99%.

Also, it is worth noting that it is possible that there are some elements within the data set that may merely be incorrect. The laboratory methods have a small margin of error,

	Offspring 1	Offspring 2	Offspring 3	Offspring 4	Offspring 5	Offspring 6	Offspring 7	Offspring 8
Marker 1	a	b	-	a	-	b	b	b
Marker 2	a	b	b	a	a	-	b	b

FIGURE 3.7. Example of duplicate markers. Offspring 3, 5, and 6 are excluded from the comparison since at least one of the individuals is missing data at these locations. Of the remaining calls, they are identical between the two markers. Marker 1 will be discarded from the analysis since it contains less data.

which means that it is possible that an individual marker can be misread within a given offspring. Ideally, filtering out duplicate markers helps to minimize this effect, but this can still represent a source of noise within the data and cannot be avoided.

3.5. Value Inflation

Our understanding of recombination frequency values tells us that as markers become farther apart, the *rf* value increases until it reaches a theoretical maximum of 0.5. From the heat map in figure 1.5, we can further see that *rf* values that are relevant to the construction of the linkage map are actually far below the 0.5 threshold. Therefore, we do not need to keep these relatively large values in the recombination frequency matrix when calling the TSP solver. In fact, it is detrimental to the performance of the TSP solver to allow these values to remain in the analysis.

The reason for this is that the TSP solver will attempt to find the best overall solution by optimizing all of the values it is given. As far as the TSP algorithms are concerned, any improvement to the Hamiltonian path is worthwhile, so for the relatively large weights in the path the solver will try to reduce these as much as possible, even though it does not matter for the genetic linkage map. Therefore, we simply choose a threshold and inflate all values above that threshold to an arbitrarily large value, such as 0.5. In this way, we relieve the TSP solver of the responsibility of optimizing these useless values, since they only serve

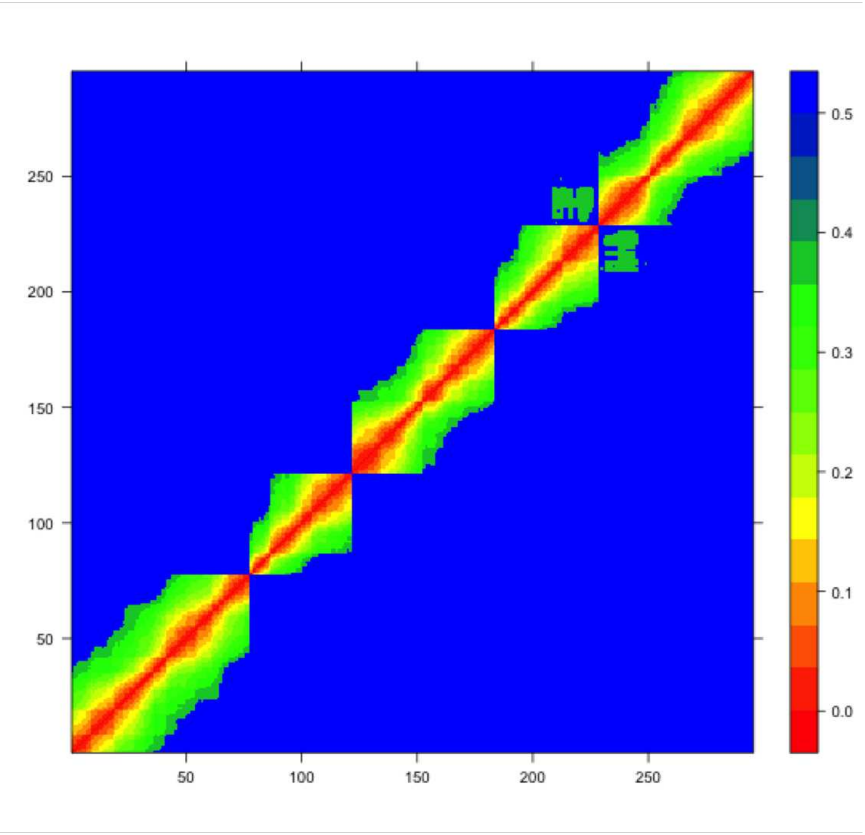


FIGURE 3.8. Arabidopsis heat map with rf value inflation. All recombination frequency above the 15th percentile were replaced with a value of 0.5.

to segregate the chromosome groups from each other and do not represent a meaningful improvement to the genetic linkage map.

Figure 3.8 shows the recomputed heat map for arabidopsis after value inflation. It is clear that, for this well-behaved data set, the technique of value inflation strongly simplifies the problem. All of the inter-chromosome variation in *rf* values from Figure 1.5 is gone, which will allow Concorde to perform its branch-and-bound operations much more efficiently.

For IR64B, the new heat map in Figure 3.9 shows improvement, but as we expect it is not as much as for arabidopsis. Because of the noise in this dataset, there are many low-value *rf* areas scattered throughout the data space, but nevertheless the improvement is still beneficial.

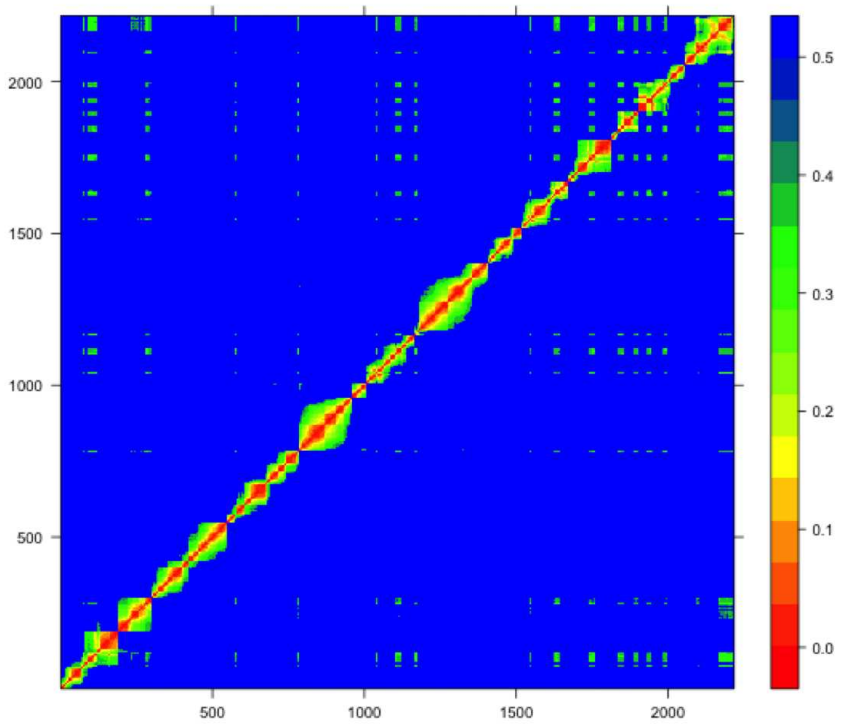


FIGURE 3.9. The heat map for the IR64B data set with value inflation. The green sections throughout the graph show areas of low recombination frequency between markers in different chromosomes. This indicates the presence of noise in the data.

Caution must be used at this stage, as setting the threshold too low can begin to affect the clustering and the final ordering of markers within the chromosome. This is especially true for noisy data sets, since meaningful *rf* values may be relatively high when there is a lot of noise in the data.

For the heatmaps in the previous section, the threshold for value inflation was chosen by choosing the average of the 15th percentile *rf* value for each marker. This value was chosen semi-empirically, and may need to be tailored to the data set in use.

3.6. Implementation

This project was implemented in R. However, due to R's inefficiency with iterated tasks, the algorithms which identify duplicate markers and compute the recombination frequency matrix were implemented in C and are called via R's built-in C interface.

CHAPTER 4

Cluster Separation

The primary goal of separating the data set into clusters is to ensure that the resulting clusters do not contain markers from more than one chromosome. There are two methods described here: the first utilizes the LKH solver, and the second utilizes a minimum spanning tree.

It is important to understand what we expect to see when we begin separating the data into clusters. We know that our end goal is a number of clusters equal to the number of chromosomes, so this process must produce at least that many clusters.

The size of the individual chromosomes is unknown at the beginning of the process. However, we assume that no chromosome should contain a number of markers that is exceedingly small relative to the total number of markers in the data. To help guide the initial cluster separation stage, a guess is made as to the expected minimum size of a chromosome. If m represents the number of markers and k represents the number of chromosomes then the expected minimum size s of a chromosome is defined to be half of the average cluster size, i.e.

$$s = \frac{m}{2k} \tag{3}$$

To begin, both methods break apart the data set into $1.5 \times k$ clusters (see the following sections for an explanation of how the breaks are made). Of these, there will most likely be several clusters which are very small, consisting of only a handful of markers. We would not consider such a small cluster to represent a significant portion of a chromosome. Therefore, to

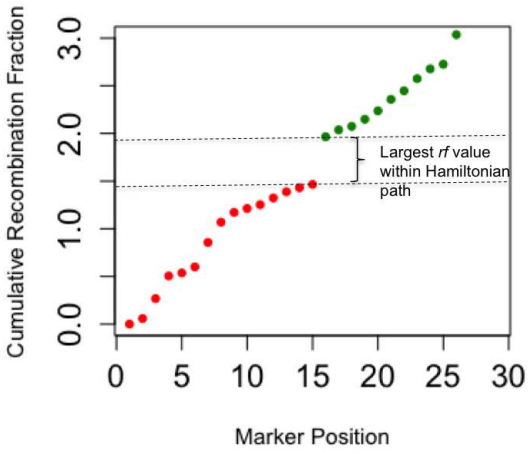


FIGURE 4.1. Cluster containing markers from two different chromosomes. The Hamiltonian path of this cluster contains an edge with a large recombination frequency value. This is indicative of the presence of markers from two different chromosomes. If this *rf* value meets or exceeds the user-defined threshold, this cluster will be separated into smaller clusters.

simply consider the number of clusters without also taking into account their size would not be wise. Instead, if a cluster meets or exceeds the size s, it is considered to be a reasonably-sized cluster and probably contains a significant portion of the markers from at least one chromosome. The cluster separation stage begins by producing a number of reasonably-sized clusters that is equal to the number of chromosomes.

It is likely that some of the larger clusters will contain markers from more than one chromosome. When this occurs, there will be a large recombination frequency value between the two chromosomes in the Hamiltonian path, as shown in Figure 4.1. A predetermined threshold is used to decide whether or not the cluster should be separated at this point (this threshold has a default value of 0.30, but can also be specified by the user). Each method applies this threshold to the clusters in a different way to insure that no clusters contain markers from more than one chromosome (see the following sections for more details).

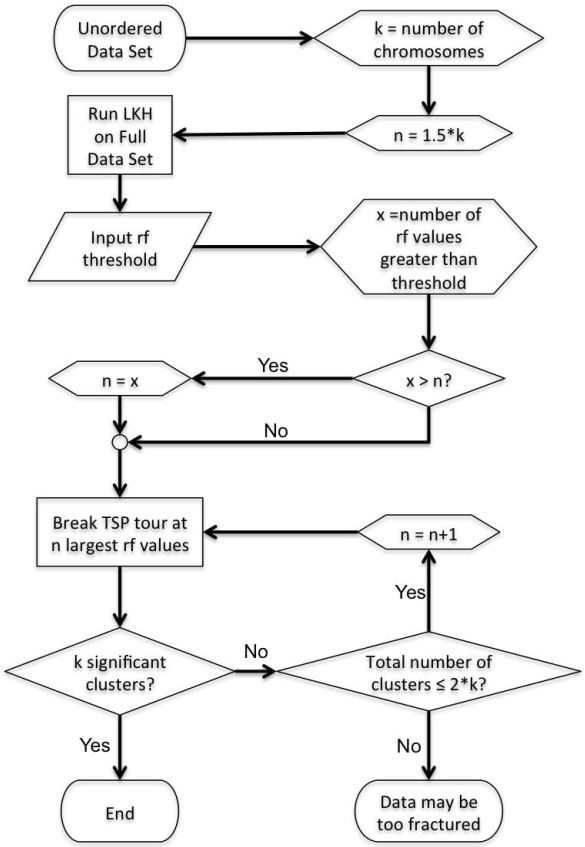


FIGURE 4.2. Flowchart of the cluster separation algorithm, TSP-based.

4.1. LKH-BASED

The method of using the LKH solver to separate the markers into clusters is outlined in Figure 4.2. The data are first processed with LKH to get the Hamiltonian path for the full dataset. All recombination frequency values in this path are sorted in descending order and all values above the *rf* threshold (provided by the user) are used as cutpoints. If this results in a number of cutpoints that is less than $1.5 \times k$, then the largest $1.5 \times k$ *rf* values are used instead. This results in the Hamiltonian path being separated as shown in Figure 4.3.

If the initial separation step results in at least k clusters of size s or greater, then the algorithm terminates and the clusters are considered to be well-separated. Otherwise, the next-largest *rf* value is added to the list of cutpoints until the condition is reached, or until

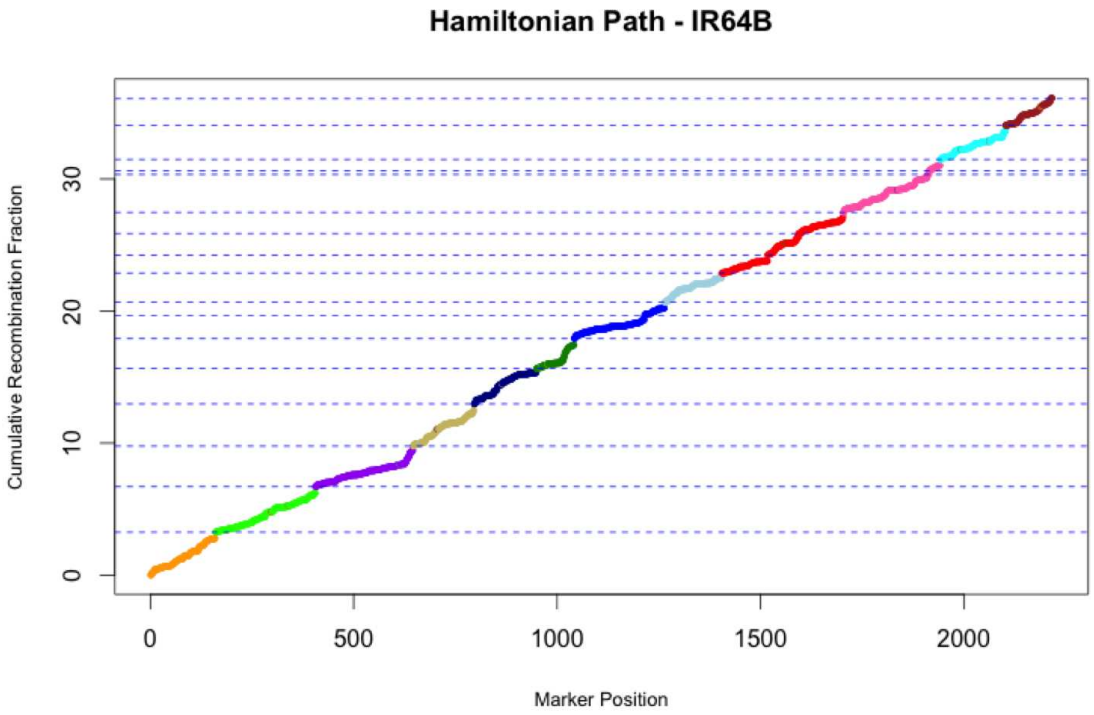


FIGURE 4.3. Hamiltonian path for IR64B dataset with 1.5*k cutpoints, where k is the number of chromosomes.

the total number of clusters reaches $2 \times k$. If the algorithm cannot produce enough significant clusters by this point, this may be an indication that the data set is too fractured to produce a reasonable result.

Since the Hamiltonian path has been created at the beginning of this method, we know with certainty whether or not all edges above the threshold have been cut, and the individual clusters do not need to be further subdivided.

4.2. Minimum Spanning Tree

The second method uses a minimum spanning tree to produce the clusters and is visualized in Figure 4.4. This method is broken into two stages. The first stage begins by constructing the minimum spanning tree and breaking it apart at the largest $1.5 \times k$ recombination frequency values. If this does not produce k significant clusters, the next-largest rf

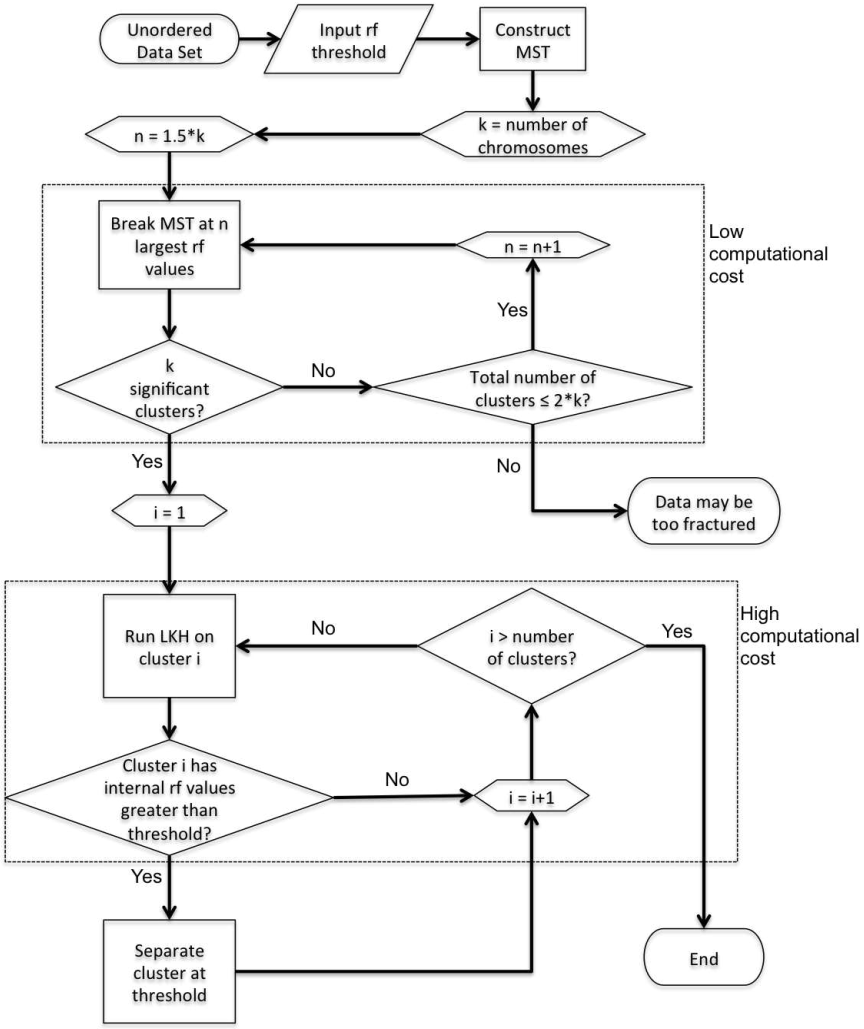


FIGURE 4.4. Flowchart of the cluster separation algorithm, MST-based.

value is added to the cutpoint list. This is repeated until k significant clusters have been produced.

The second stage involves running each cluster through LKH and checking for rf values that exceed the threshold, similar to the approach described in the previous section. When such a value is detected, the cluster is broken apart to make sure that the final clusters do not contain markers from more than one chromosome.

4.2.1. FIRST STAGE - COARSE CUTS. Figure 4.5 shows the MST for the arabidopsis data set, color-coded by chromosome. In this case, the markers from each chromosome cluster

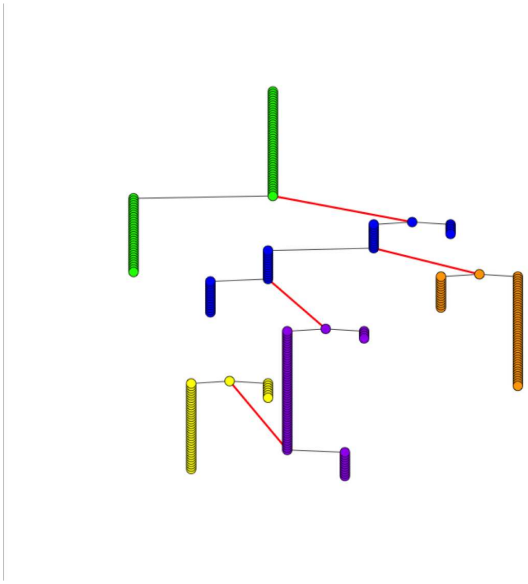


FIGURE 4.5. The minimum spanning tree for the arabidopsis data set, color-coded by chromosome. The largest-weight edges are marked in red. For these data, cleaving the MST at the red edges will completely separate the chromosomes into individual groups. This occurs because the arabidopsis data set has very little noise.

together due to the absence of noise within this data set, and the highest-weight edges are the connecting edges between these clusters (shown in red). Therefore, for a dataset that is very clean and complete, breaking the MST into k groups is enough to cleanly separate the chromosomes.

On the other hand, when dealing with noisy data, it is possible that some of these large-weight cutpoints will not be true breaks between chromosomes. In other words, it is possible that some intra-chromosome rf values may be larger than one or more of the inter-chromosome cutpoints.

As an example, see the minimal spanning tree for the IR64B genome, shown in Figure 4.6. The IR64B dataset is a fairly dirty dataset, with many individuals missing data. Because of this, it is a challenging dataset to process and gives a good example of the kinds of difficulties researchers will encounter while using this new processing technique. Figure 4.6 shows the

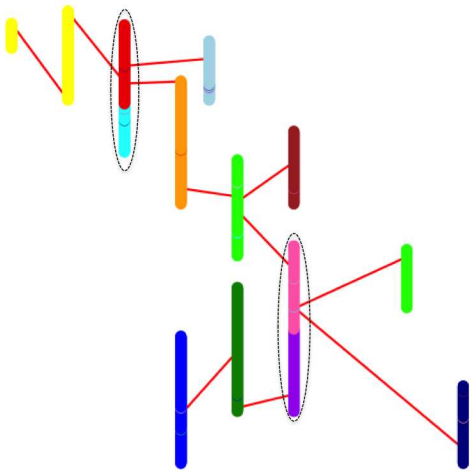


FIGURE 4.6. The minimum spanning tree for the IR64B data set, color-coded by chromosome. The largest-weight edges are marked in red. The symbol borders have been eliminated so that they do not cover the interior colors, due to the number of data points. The circled groups contain markers from multiple chromosomes. Cleaving the MST at the red edges will not cleanly separate all chromosomes into individual groups, due to the amount of noise in this data set.

MST separated into 12 groups by exploding the 11 highest-weight edges. The reason 12 groups are shown is because IR64B contains 12 chromosomes.

In Figure 4.6, two groups are circled in black. These two groups contain markers from two different chromosomes, as evidenced by the different colors in the cluster. This behavior is the reason that the clusters are run through the second stage (described below), which will analyze individual clusters and break them apart at large *rf* values.

4.2.2. SECOND STAGE - FINE CUTS. After the first stage, we will have several clusters which may or may not be properly separated according to chromosome. In order to determine whether or not this is the case, we do TSP runs on each cluster using LKH, since it is the faster solver (see section 1.2.2). The goal here is to see if any of the individual clusters

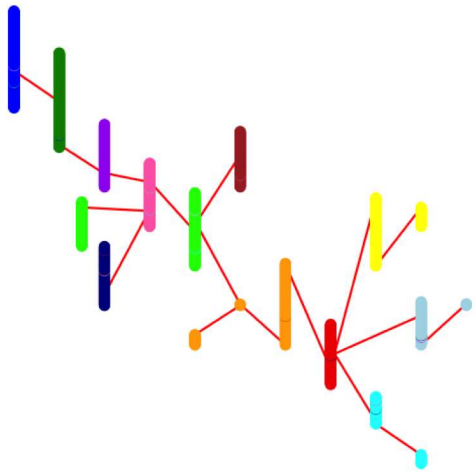


FIGURE 4.7. The minimum spanning tree for the IR64B data set when divided into 18 groups (1.5 times the number of chromosomes). No single group contains markers from more than one chromosome.

produce a TSP tour with markers from more than one chromosome in it. The Hamiltonian path for each cluster is checked for any recombination frequency values which exceed the user-defined *rf* threshold. When these values are found, the cluster is broken apart at that point (see Figure 4.1).

CHAPTER 5

Cluster Merging

Once the markers have been broken into well-separated clusters, the next step is to merge clusters which belong together. Clusters are processed in order of increasing size, since a small cluster is more likely to need to be merged with a larger cluster (two large clusters will probably not merge with each other, since they most likely contain markers from different chromosomes). Each cluster is processed through three stages to determine which cluster it should be merged with, each stage representing a higher computational cost than the last.

The first stage involves direct examination of the *rf* matrix to see if the cluster can be directly merged with another without any further analysis (Section 5.1). The second stage involves combining clusters pairwise and processing them with LKH to see if the Hamiltonian path clearly indicates whether the clusters should be merged (Section 5.2). Finally, if a cluster does not pass the criterion for being merged in stage 2, the third stage attempts to verify which cluster it belongs to using reciprocal matching technique of the most likely candidate cluster (Section 5.2). If a cluster does not meet any of these criteria, it is consider to be a complete chromosome.

5.1. Merge Clusters by Shortest Connecting Edges

We would like to directly merge clusters at a very low computational cost. To do this, we find the smallest *rf* value between any marker in one cluster to any marker in every other cluster, as shown in Figure 5.1. To determine whether or not two clusters can be merged at this phase, the smallest of these values (x) is compared to the second-smallest (y). If $x \leq \frac{1}{2}y$, then we merge the clusters without any further testing since this indicates that x is significantly smaller than all other values.

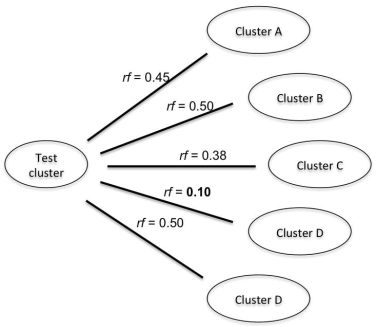


FIGURE 5.1. Example of testing for the smallest edge connections between clusters. The connection between the test cluster and cluster D is significantly smaller than all other values, so these two groups can be merged without further testing. Since this only involves examining the recombination frequency matrix, the computation cost is very low.

Consider the example values shown in Table 5.1. Here we wish to merge the clusters A, B, and C with the clusters 1, 2, 3, 4, and 5. The values in the table shown the smallest edge weights between any marker pair in these clusters.

For cluster A, the values between cluster A and clusters 1, 3, and 5 are all 0.50, indicating that there is no recombination frequency between the markers in these clusters with any of the markers within cluster A that is less than 0.50. This allows us to eliminate these clusters as possible candidates by simple inspection of the *rf* matrix. For clusters 2 and 3, the smallest recombination frequency value to cluster A is $x = 0.10$ and the second-smallest is $y = 0.22$. Since $x \leq \frac{1}{2}y$, clusters A and 2 will be merged with no further testing.

Cluster B we have $x = 0.15$ and $y = 0.22$. Here the $x \leq \frac{1}{2}y$ condition is not satisfied.

For cluster C, all values in the table are 0.50 except for the edge to cluster 1 ($x = 0.25$). Again we have the case where $x \leq \frac{1}{2}y$ so these clusters will be merged. Since the recombination frequency matrix was capped at a value of 0.50, the largest intercluster *rf* value that could cause two clusters to be merged at this stage is 0.25.

For cluster D, the smallest value is above 0.25, so it will not be merged at this stage.

TABLE 5.1. Example values of the smallest connecting edges between several clusters. If the smallest recombination frequency value in a row is less than half of the second-smallest value, the two clusters are merged.

Cluster	1	2	3	4	5
A	0.50	**0.10**	0.45	0.22	0.50
B	0.40	0.42	0.50	0.15	0.22
C	**0.25**	0.50	0.50	0.50	0.50
D	0.50	0.50	0.36	0.50	0.50

Since this process only involves looking up values in the recombination frequency matrix, the computational cost is very low.

5.2. Cluster Merging with LKH

The second stage involves merging pairs of clusters and processing them with LKH. This is essentially the reverse of the technique described in Figure 4.1 - if we merge two clusters that belong to the same chromosome then the resulting Hamiltonian path will not exhibit any large jumps. An example of this stage is shown in Figure 5.2.

To choose which cluster is the candidate to be merged with another cluster, we again start with the smallest. We combine the candidate cluster in turn with every other cluster and run them through LKH. The largest *rf* values from the Hamiltonian path resulting from each combination are compared, and we again use the $x \leq \frac{1}{2}y$ criterion to determine if the clusters should be merged.

In Figure 5.2, the largest *rf* value from each Hamiltonian path is printed beneath the combined plots. The smallest is $x = 0.18$ and the largest is $y = 0.39$. In this case the condition is satisfied and the groups will be merged.

5.3. Reciprocal Cluster Merging

The third stage is the final attempt to merge a cluster into another cluster. At this stage, the cluster which has the smallest value in the Hamiltonian path from stage 2 is used as the

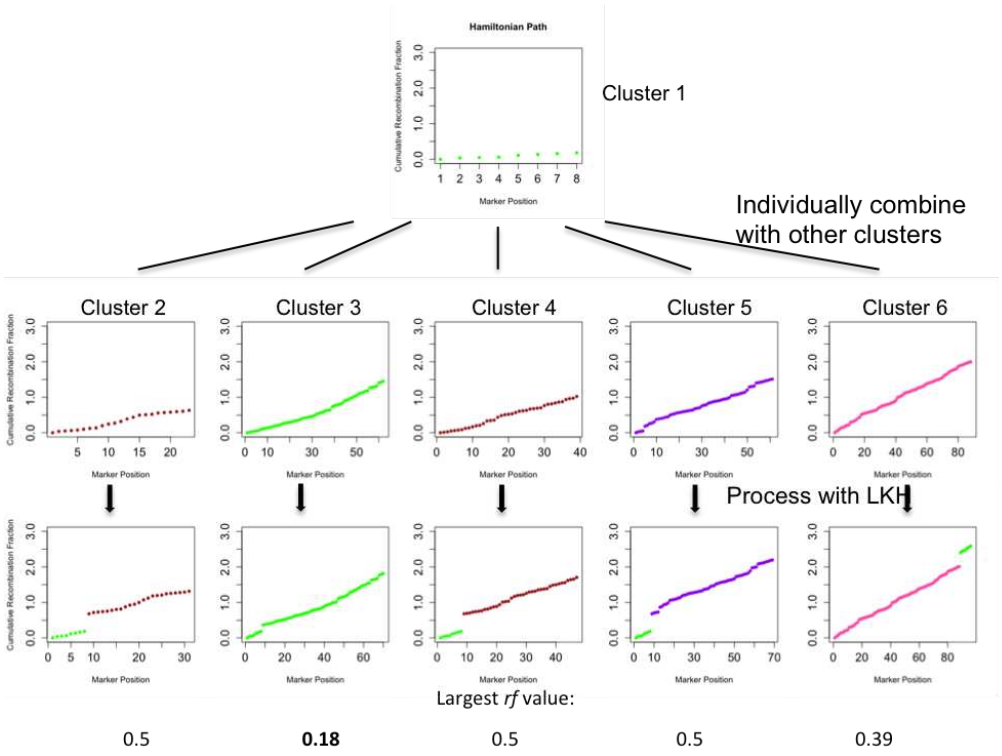


FIGURE 5.2. Cluster merging using LKH. A given cluster is separately merged with each of the other clusters and run through LKH. The largest rf value of the resulting Hamiltonian path indicates how closely related the two clusters are. Since the smallest of these values is less than half of the next largest value, the two clusters are merged.

new candidate solution (see Figure 5.3). This cluster is then put through the same process, and if the result indicates that the original cluster has the lowest maximum rf value in the combined Hamiltonian path, then the two clusters are merged.

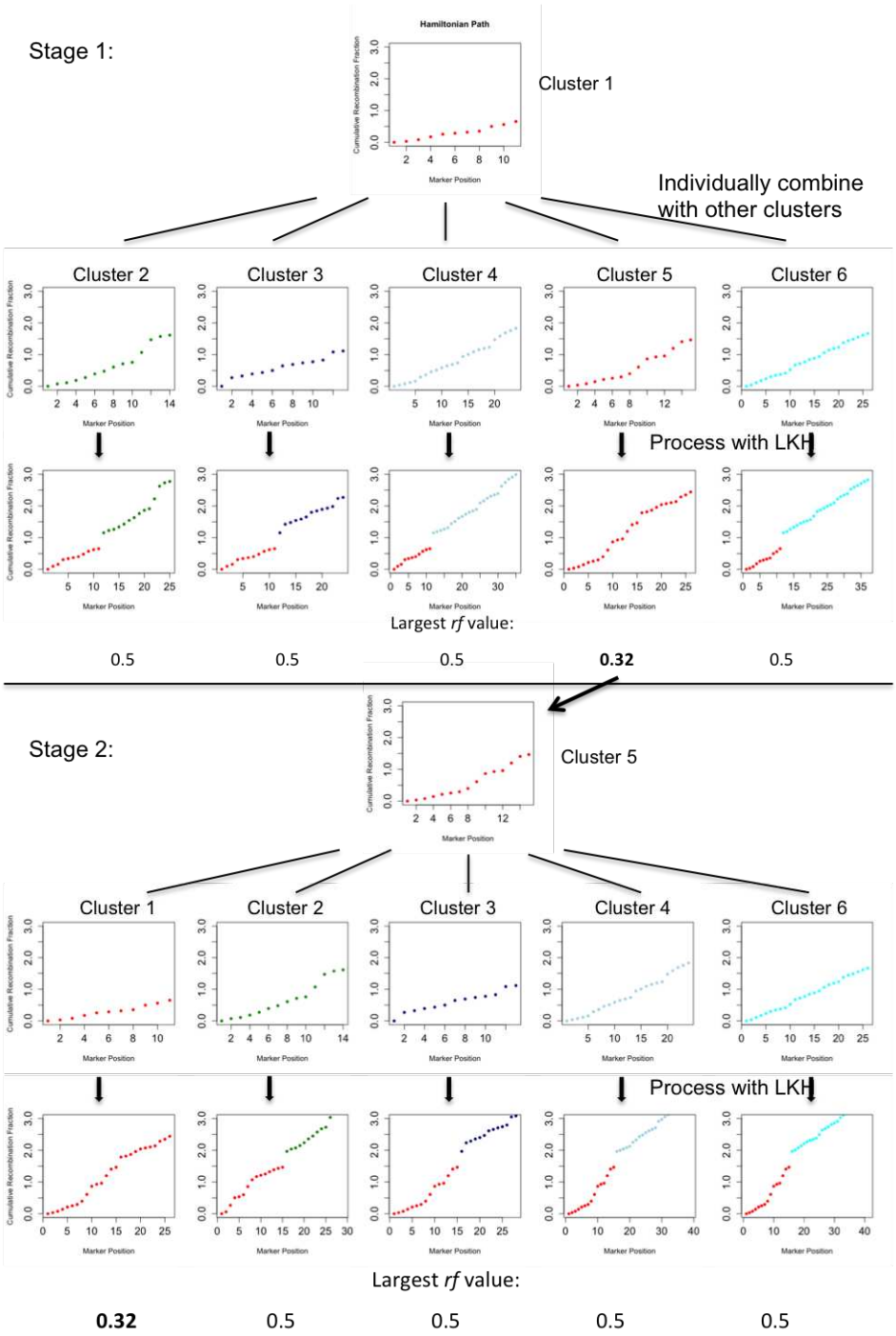


FIGURE 5.3. Cluster merging using LKH, in the event that no cluster is clearly the best match. Cluster 1 is separately merged with each of the other clusters and processed with LKH (top half of the figure). The largest *rf* value within the Hamiltonian path indicates how closely the two clusters are related. The process is repeated for the cluster which has the closest relationship to the original cluster (in the bottom half of the figure). If this step indicates the original cluster (cluster 1), the two clusters are merged.

CHAPTER 6

Final TSP Runs with Concorde

Once we are satisfied that we have merged the groups such that each group contains a single, complete chromosome, we find the final Hamiltonian path for each individual chromosome using Concorde. Since Concorde is a complete solver, we are assured that the solution for each group will be the best possible Hamiltonian path for that group of markers. Assuming that the filtering and grouping processes have correctly merged all markers for a single chromosome, this will be the best genetic linkage map that can be produced from the overall data set.

CHAPTER 7

Results

To evaluate the quality of the final solution of each chromosome, we compare the order of the markers to the order in the solution produced by the JoinMap program. Since JoinMap is the industry standard, this is the best indicator we have of the solution expected by plant sciences researchers. The comparison is made by constructing a plot of the marker order in the solution versus the marker order in the JoinMap solution. When the two solutions match perfectly, the plot will be a straight line.

Even though the chromosomes have been separated and independently processed with Concorde, for simplicity all chromosomes are shown on a single plot. The vertical lines show the breakpoints between chromosomes.

To quantitatively measure the quality of a solution, the number of erroneous pairs E is computed. This number is the number of pairs of markers which appear in reversed order as compared to some reference solution [7].

7.1. Simulated Data

Datasets were simulated using the R/qtl package [14]. All simulated datasets contain a total number of markers m of 1,000, 5,000, or 10,000 evenly distributed across five chromosomes.

Various values of genotype error rates (η) and the rate of missing genotypes (γ) were used to evaluate the performance of the current algorithm and MSTmap, shown in Table 7.1. Five datasets were created for each configuration using 300 individuals. Markers are evenly divided among five chromosomes, but not evenly spaced.

Table 7.1 shows several important results. First, the runtimes of the current algorithm are almost always significantly less than those of MSTmap. The exception to this is seen in the 10000 marker datasets, but only for very low values of γ when $\eta = 0.0$.

Second, the number of erroneous markers E is lower for the current algorithm in almost every case. Therefore, even if the user were to accept the longer runtime required by MSTmap, the solution quality is almost guaranteed to be worse.

Finally, the MSTmap results show significant difficulty with producing the correct number of chromosomes c. In situations where the number of groups produced is far less than the actual number of chromosomes, the E value could not be calculated, denoted by an asterisk. The current algorithm produced the correct number of chromosomes in every case except $\eta = 0.10$ and $\gamma = 0.10$.

TABLE 7.1. Average runtimes for simulated datasets with various genotyping error rates (η) and rates of missing data (γ). $\overline{E}$ is the average number of erroneous pairs, $\overline{c}$ is the average number of clusters in the result, $\overline{m}$ is the average number of markers in the data after removing duplicates. An asterisk shows that E could not be evaluated because the result did not produce the correct number of chromosomes. Runtimes do not include removal of duplicate markers since this was done before both algorithms were run.

				MST-based Method			MSTmap		
Problem Size	$\overline{m}$	γ	η	Runtime (s)	$\overline{E}$	$\overline{c}$	Runtime (s)	$\overline{E}$	$\overline{c}$
1000	925.2	0.00	0.00	8.4	9.2	5.0	12.7	175	4.6
	1000	0.00	0.01	11.4	61.0	5.0	15.5	210	5.0
	1000	0.00	0.05	14.1	149.2	5.0	15.6	236.2	5.0
	1000	0.00	0.10	17.0	329.8	5.0	20.1	381.0	5.0
	1000	0.05	0.00	9.9	78.8	5.0	12.0	193.0	5.0
	1000	0.05	0.01	9.8	96.2	5.0	18.0	285.6	5.0
	1000	0.05	0.05	15.9	217.8	5.0	25.0	332.8	4.2
	1000	0.05	0.10	16.1	365.2	5.0	24.6	421.2	5.0
	1000	0.10	0.00	12.7	90.4	5.0	11.6	204.0	5.0
	1000	0.10	0.01	11.8	106.4	5.0	15.2	157.4	4.2
	1000	0.10	0.05	16.8	207.8	5.0	25.0	380.8	5.0
	1000	0.10	0.10	16.7	403.2	5.0	32.7	529.4	5.0
5000	4626	0.00	0.00	176.1	28.6	5.0	241.0	732.0	5.0
	4998.5	0.00	0.01	251.0	316.5	5.0	336.9	903.8	5.0
	5000	0.00	0.05	409.3	759.5	5.0	384.2	1049.5	4.6
	5000	0.00	0.10	506.1	1538.2	5.0	689.1	1857.0	4.0
	5000	0.05	0.00	325.9	316.0	5.0	314.9	915.0	4.0
	5000	0.05	0.01	327.5	379.0	5.0	436.7	1229.0	4.0
	5000	0.05	0.05	325.3	819.0	5.0	2161.5	*	1.0
	5000	0.05	0.10	428.5	1704.5	5.0	1563.0	*	3.0
	5000	0.10	0.00	324.0	440.0	5.0	314.5	851.0	4.0
	5000	0.10	0.01	326.7	536.0	5.0	1135.9	*	1.0
	5000	0.10	0.05	534.2	1020.0	5.0	3303.5	*	1.0
	5000	0.10	0.10	323.0	2052.5	5.0	6264.1	*	1.0
10000	9252.5	0.00	0.00	1195.3	51.3	10.0	697.0	1435.3	9.2
	9997.3	0.00	0.01	1553.4	624.3	10.0	966.5	1926.5	9.4
	10000	0.00	0.05	1762.6	1444.0	10.0	1155.7	2175.5	8.0
	10000	0.00	0.10	2922.2	1581.6	10.0	1547.0	3614	9.0
	10000	0.05	0.00	1353.7	643.0	10.0	2707.5	*	3.0
	10000	0.05	0.01	1327.7	846.0	10.0	2788.0	*	4.0
	10000	0.05	0.05	1390.4	1656.0	10.0	10275.7	*	1.0
	10000	0.05	0.10	1439.6	3509.0	10.0	16608.2	*	2.0
	10000	0.10	0.00	1440.4	863.0	10.0	4063.9	*	1.0
	10000	0.10	0.01	1949.4	1059.0	10.0	5399.5	*	1.0
	10000	0.10	0.05	1639.0	2097.8	10.0	18976.4	*	2.0
	10000	0.10	0.10	2479.8	3969.3	10.2	43394.4	*	1.6

7.2. Real Data

Table 7.2 shows a comparison of the runtimes for the current algorithm and MSTmap. Runtimes for JoinMap are difficult to accurately report, since JoinMap can only construct one chromosome at a time. This means that the user has to set the program to run on one chromosome, then return to start the process on the next chromosome. For a dataset consisting of a few hundred markers (arabidopsis, barley, clean IR64B) the overall process takes several hours. For a larger dataset (dirty IR64B) the entire process can take a full day.

Table 7.2. Run times for the current algorithm (both MST-based and TSP-based methods) and MSTmap, shown in seconds. JoinMap runtimes are not reported as they are vastly larger (on the order of hours to days). m is the number of markers in the dataset, E is the number of erroneous pairs when compared to the JoinMap solution.

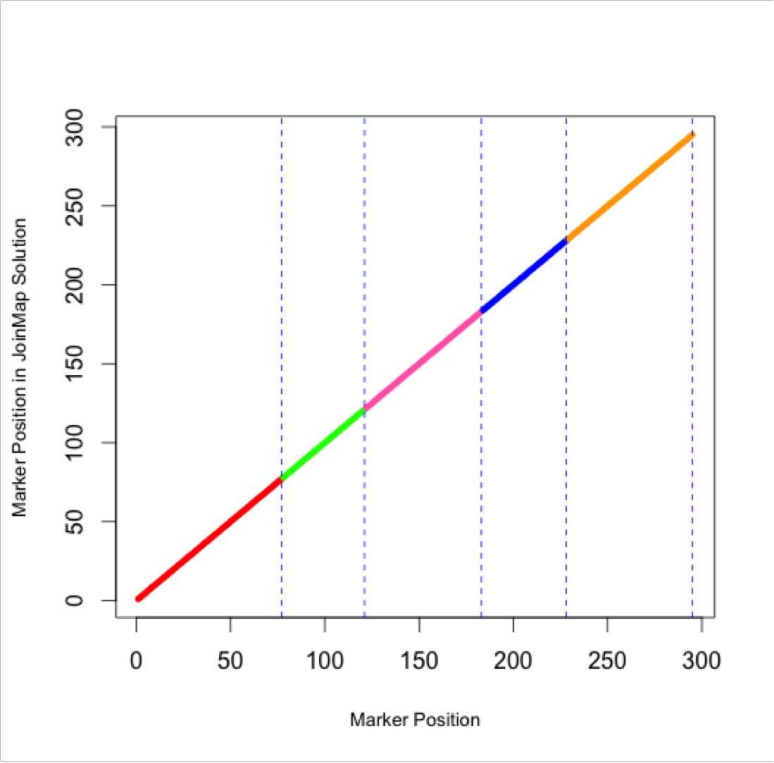
		MST-based	TSP-based		MSTmap	
Species	m	Runtime	Runtime	E	Runtime	E
Arabidopsis	295	2	2	0	5	13
Barley	481	4	4	2	3	23
Hall's Switchgrass	880	12	10	5	10	275
IR64B (clean)	433	19	18	0	13	177
IR64B (dirty)	2,217	218	260	5294	460	9001

The reported runtimes include detecting and removing duplicate/similar markers, and computing the recombination frequency matrix. There is no difference in the solution results between the TSP-based cluster separation method and the MST-based cluster separation method.

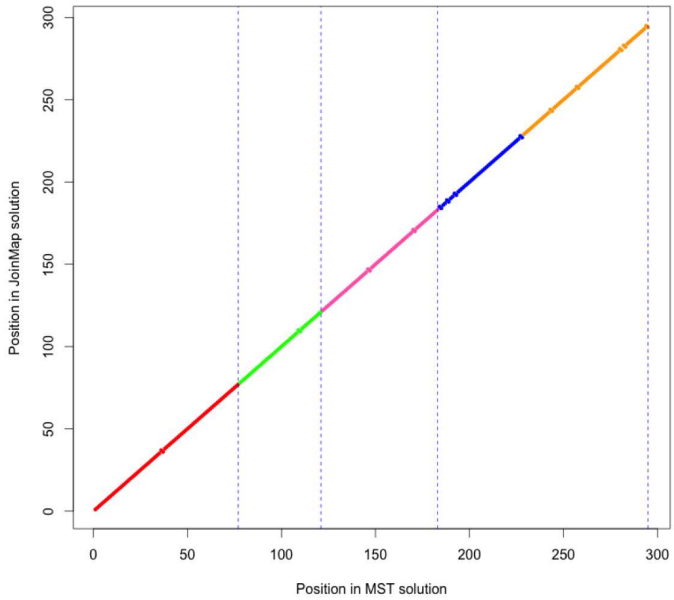
7.3. Arabidopsis

The arabidopsis data set consists of 295 markers. The current algorithm produces a linkage map which is identical to that produced by JoinMap (Figure 7.1a).

The MSTmap result (Figure 7.1b) is also very close to the JoinMap result, producing only 13 erroneous pairs of markers.



(a) Current algorithm compared to JoinMap result. $E = 0$, runtime = 2 seconds.



(b) MSTmap result compared to JoinMap result. $E = 13$, runtime = 5 seconds.

FIGURE 7.1. Arabidopsis results.

7.4. Barley

The barley data set consists of 481 markers. The current algorithm produces a linkage map which nearly identical to that produced by JoinMap ($E = 2$, Figure 7.2a).

The MSTmap result (Figure 7.2b) is slightly worse, producing 23 erroneous pairs.

7.5. Hall's Switchgrass

The Hall's switchgrass data set consists of 880 makers. The linkage map is nearly identical to that produced by JoinMap ($E = 5$, Figure 7.3a). This dataset was provided by Dr. John Lovell at the University of Texas.

The MSTmap result (Figure 7.3b) shows a moderate degradation in solution quality, producing 275 erroneous marker pairs.

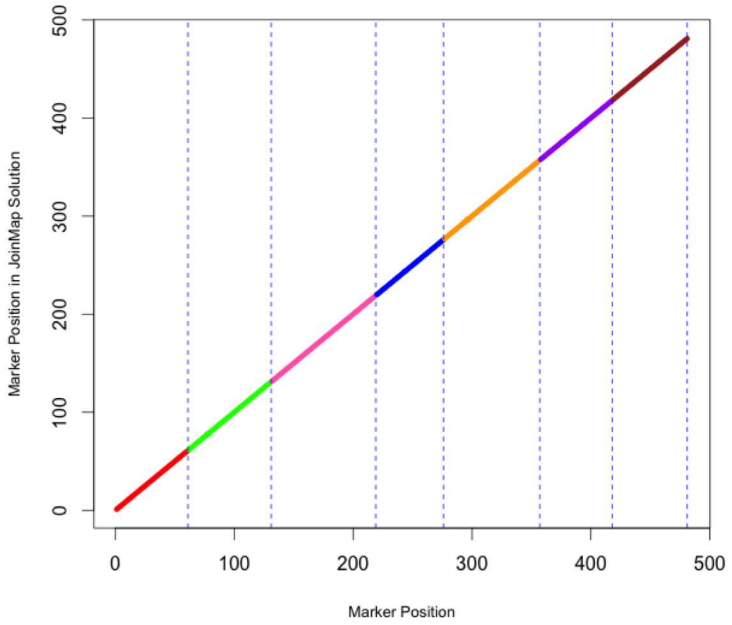
7.6. IR64B (Clean Dataset)

This is the result after fully filtering the IR64B dataset (filtered with techniques not included in this thesis), leaving 433 markers. The IR64B linkage map is identical to that produced by JoinMap (Figure 7.4a). This dataset was provided by Drs. John McKay and Paul Tanger at Colorado State University.

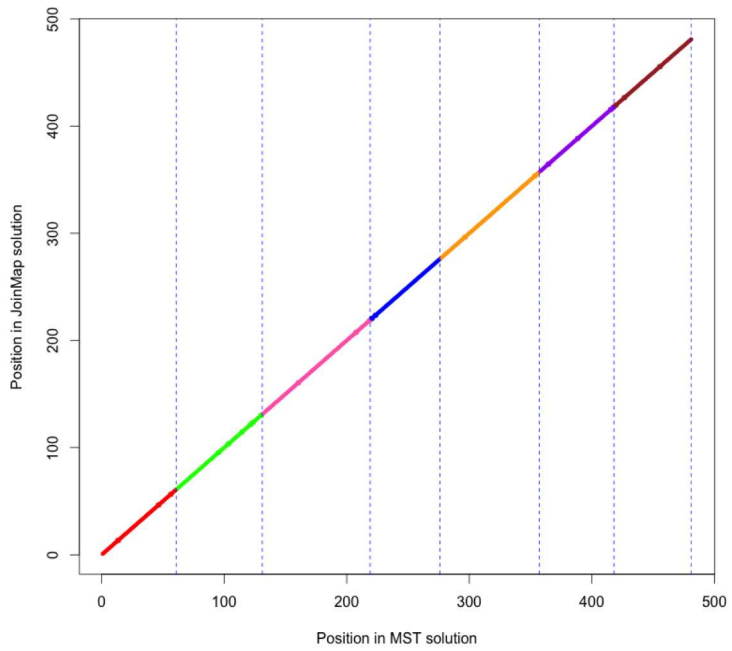
Here we see that MSTmap has not produced the correct number of chromosomes (Figure 7.4b). Three chromosomes are merged into a single group. However, within this group the markers of each chromosome are in relatively good order. To compute E, each chromosome was manually separated so that the E value was not affected by the improper grouping.

7.7. IR64B (Dirty Dataset)

This linkage map was constructed from a less-strictly filtered version of the IR64B dataset, containing 2,217 markers. This is done to show the capability of the current algorithm in

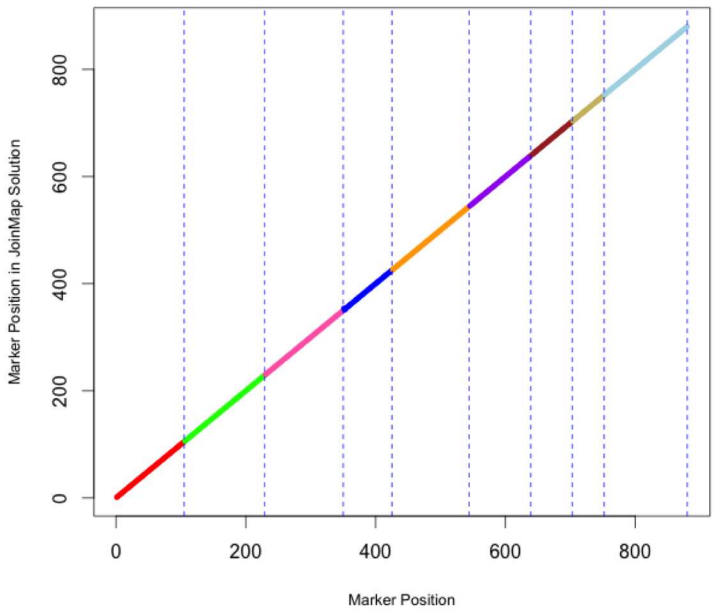


(a) Current algorithm compared to JoinMap result. $E = 2$, runtime = 4 seconds.

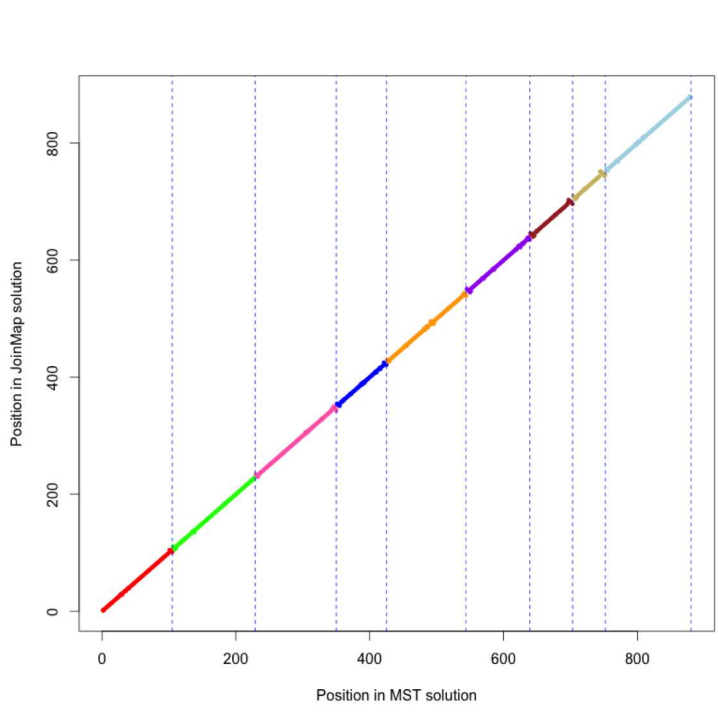


(b) MSTmap result compared to JoinMap result. $E = 23$, runtime = 3 seconds.

FIGURE 7.2. Barley results.

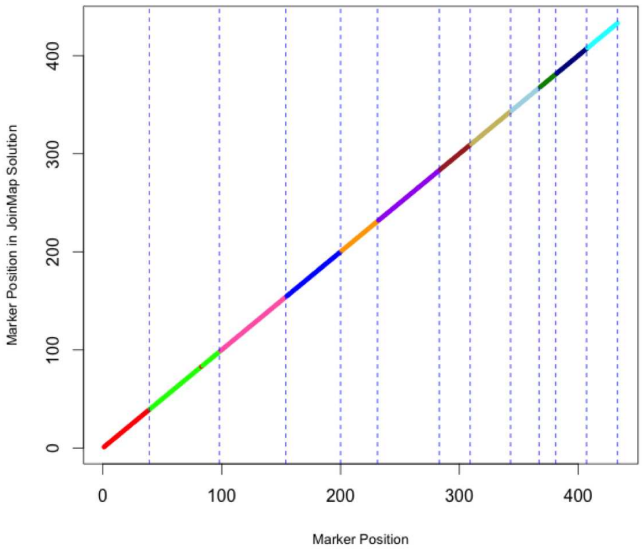


(a) Current algorithm compared to JoinMap result. $E = 5$, runtime = 12 seconds.

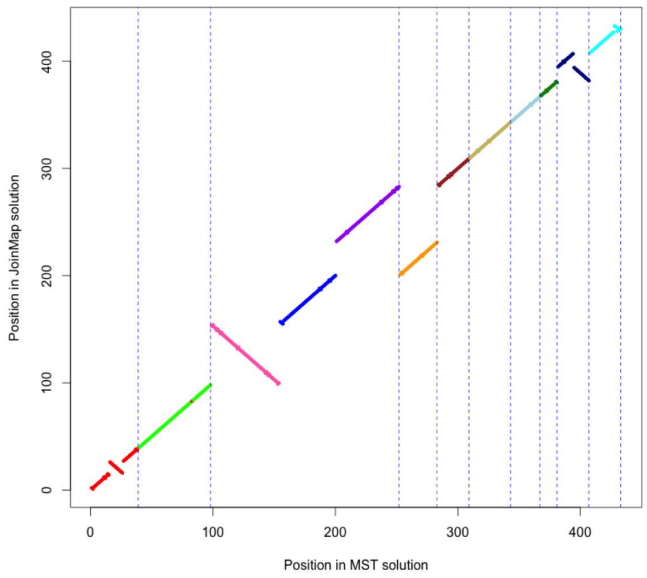


(b) MSTmap result compared to JoinMap result. $E = 275$, runtime = 10 seconds.

FIGURE 7.3. Hall's switchgrass results.



(a) Current algorithm compared to JoinMap result. $E = 0$, runtime = 19 seconds.



(b) MSTmap result compared to JoinMap result. $E = 177$, runtime = 13 seconds.

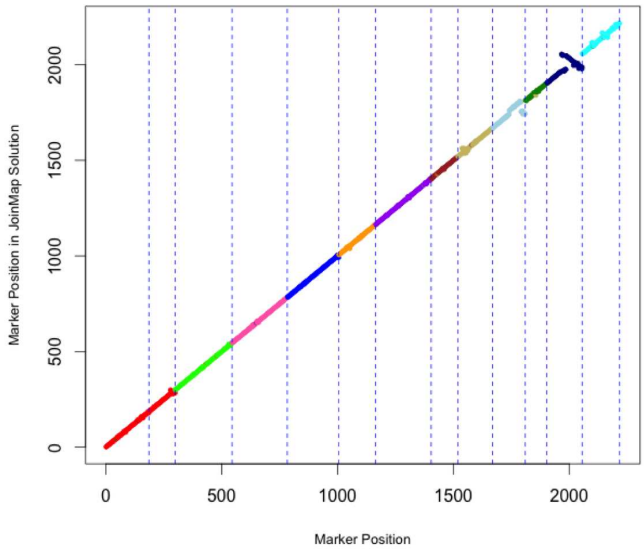
FIGURE 7.4. IR64B result (clean dataset) results. Note that MSTmap did not produce the correct number of chromosomes as there is a single group containing 3 chromosomes. However, within that group the markers of each chromosome are in relatively good order.

processing a larger dataset in much less time as compared to JoinMap, and to show that the results are similar to the JoinMap result.

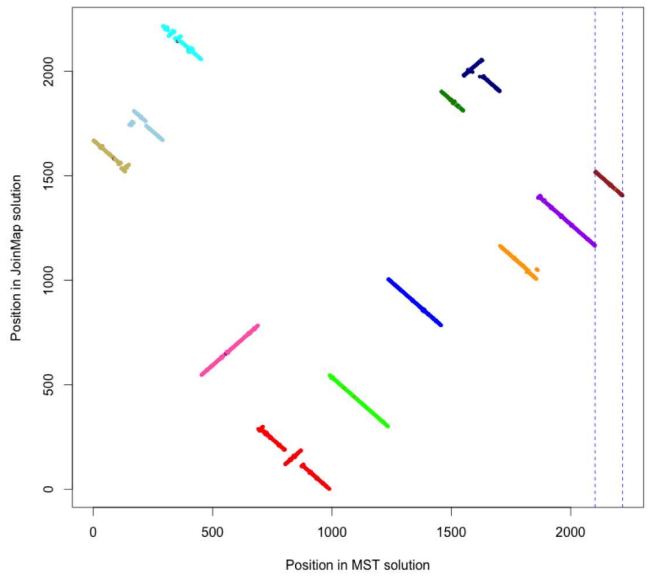
There are several differences between the current algorithm and the JoinMap solutions (Figure 7.5a), mostly in the gold, light blue, and dark blue chromosomes. It should be noted that there is no way to verify whether or not JoinMap produced a correct solution, so these differences can not be used to state which algorithm produces a more correct solution. The important thing to note is that both algorithms produce results which overall are very similar, so there is not a huge difference in solution quality.

There was a large chunk of missing markers in the middle of one of the red chromosome, so the analysis was run with 13 chromosomes instead of 12. This was a known problem with the dataset and the JoinMap linkage map produced 13 chromosomes as well.

The MSTmap result (Figure 7.5b) is again not properly separating the chromosomes. As in Section 7.6, each chromosome was manually separated in order to compute E. Also, the MSTmap runtime is more than double that of the current algorithm.



(a) Current algorithm compared to JoinMap result. $E = 5295$, runtime = 218 seconds.



(b) MSTmap result compared to JoinMap result. $E = 9001$, runtime = 460 seconds.

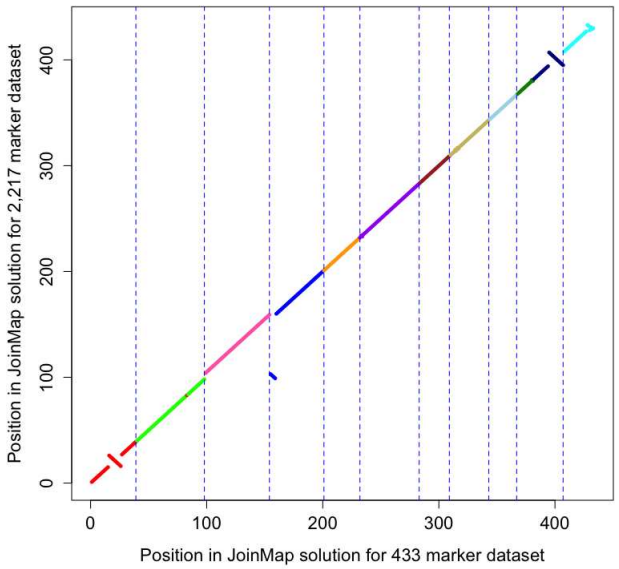
FIGURE 7.5. IR64B result (dirty dataset) results. This linkage map was constructed on 13 chromosomes due to a large number of markers missing from the interior of the red chromosome.

7.8. Evaluation of Filtering on JoinMap Solution Quality

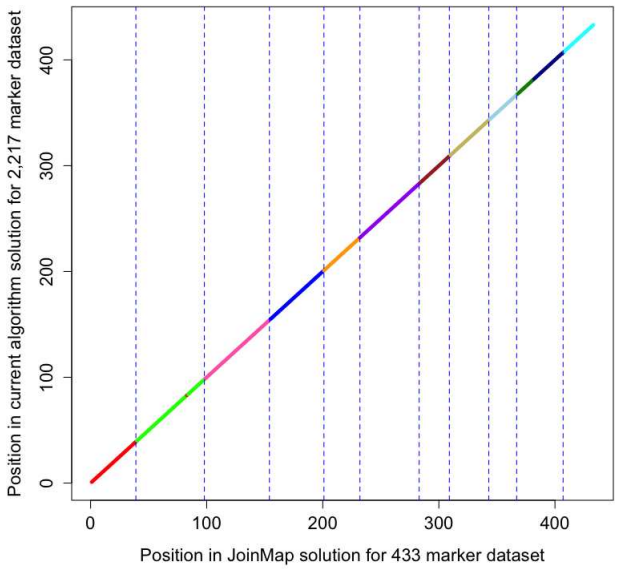
Figure 7.5a shows a comparison between the linkage maps produced by the current algorithm and JoinMap for the dirty IR64B dataset (2,217 markers). There are several significant differences, but keeping in mind that we do not know the true solution, this raises the question of which solution is causing the discrepancies.

To investigate this, Figure 7.6 shows both solutions compared to the JoinMap solution on the more highly-filtered data set (433 markers). The top plot shows the JoinMap solution for the dirty dataset compared to the JoinMap solution for the clean dataset. The bottom plot show the current algorithm's solution for the dirty dataset also compared to the JoinMap solution for the clean dataset. Note that only markers which appear in the smaller dataset are included in the plot.

The top plot indicates that the JoinMap solutions for the two datasets show disagreement, indicating that the less-filtered dataset causes JoinMap to produce a linkage map which does not agree with the linkage map produced by the more highly-filtered dataset. However, the lower plot shows that the current algorithm does not exhibit this problem. The dirty dataset produces a result which shows perfect agreement with the JoinMap solution for clean data, suggesting that the discrepancies shown in Figure 7.5a might be due more to JoinMap rather than the results produced by the current algorithm.



(a) JoinMap solution (dirty data) vs JoinMap solution (clean data). $E = 438$.



(b) Current algorithm (dirty data) vs JoinMap (clean data). $E = 0$.

FIGURE 7.6. Comparison of results for dirty data compared to the JoinMap solution for clean data (IR64B).

CHAPTER 8

Conclusions

8.1. Summary of Results

The method presented here produces results which are very similar to the results of JoinMap, and often identical. Since JoinMap is the standard software used by agricultural science researchers, this new method could be used instead with no loss of solution quality.

Furthermore, the runtime of the new method is drastically less than that of JoinMap. Genetic linkage maps can now be constructed in seconds rather than hours. Also, since the new method can construct the linkage map for all chromosomes without constant user intervention, the usability is much greater than that of JoinMap.

A more practical concern is that JoinMap is closed-source, proprietary software. Since the software in this thesis has been written in R and is available on the internet, researchers can tailor it to their specific needs. Also, since JoinMap was written in the 1990's its scalability is limited, and the time is rapidly approaching that the larger and larger datasets which researchers want to use will be too large for JoinMap to handle. The new method can handle a vastly larger number of markers than JoinMap.

8.2. Future Work

Future work would largely focus on adding advanced data filtering techniques to the software package. One of the issues that could be addressed immediately is the detection of markers which have high correlation to multiple chromosome groups. This kind of detection is clearly impossible to do before the chromosome groups are formed, and since the procedure is so labor-intensive in JoinMap it is time-prohibitive. Since approximate chromosome groups

can be formed quickly using LKH, these kind of markers could be detected very quickly and removed before running a full linkage map construction.

Bibliography

[1] B. Collard, M. Jahufer, J. Brouwer, and E. Pang, "An introduction to markers, quantitative trait loci (qtl) mapping and marker-assisted selection for crop improvement: the basic concepts," *Euphytica*, vol. 142, no. 1-2, pp. 169–196, 2005.

[2] J. Ågren, C. G. Oakley, J. K. McKay, J. T. Lovell, and D. W. Schemske, "Genetic mapping of adaptation reveals fitness tradeoffs in arabidopsis thaliana," *Proceedings of the National Academy of Sciences*, vol. 110, no. 52, pp. 21077–21082, 2013.

[3] J. W. Van Ooijen and J. Jansen, *Genetic mapping in experimental populations*. Cambridge University Press, 2013.

[4] P. Stam, "Construction of integrated genetic linkage maps by means of a new computer package: Join map," *The Plant Journal*, vol. 3, no. 5, pp. 739–744, 1993.

[5] J. Van Ooijen, "Joinmap® 4, software for the calculation of genetic linkage maps in experimental populations," *Kyazma BV, Wageningen*, vol. 33, 2006.

[6] J. Cheema and J. Dicks, "Computational approaches and software tools for genetic linkage map estimation in plants," *Briefings in bioinformatics*, vol. 10, no. 6, pp. 595–608, 2009.

[7] Y. Wu, P. Bhat, T. J. Close, and S. Lonardi, "Efficient and accurate construction of genetic linkage maps from noisy and missing genotyping data," in *Algorithms in Bioinformatics*, pp. 395–406, Springer, 2007.

[8] E. L. Lawler, J. K. Lenstra, A. R. Kan, and D. B. Shmoys, *The traveling salesman problem: a guided tour of combinatorial optimization*, vol. 3. Wiley New York, 1985.

[9] T. Schiex and C. Gaspin, "Cartagene: Constructing and joining maximum likelihood genetic maps," in *Proceedings of the fifth international conference on Intelligent Systems for Molecular Biology*, 1997.

[10] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.

[11] K. Helsgaun, "An effective implementation of the lin–kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[12] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, *Finding cuts in the TSP (A preliminary report)*, vol. 95. Citeseer, 1995.

[13] G. L. Thompson and S. Singhal, "A successful algorithm for the undirected hamiltonian path problem," *Discrete applied mathematics*, vol. 10, no. 2, pp. 179–195, 1985.

[14] K. W. Broman, H. Wu, Ś. Sen, and G. A. Churchill, "R/qtl: Qtl mapping in experimental crosses," *Bioinformatics*, vol. 19, no. 7, pp. 889–890, 2003.

APPENDIX A

Basic Usage of R Package

This code demonstrates the most basic functionality of the R package.

```
# ---------- Get the Package working ----------

# Do ONE of these two options)
# after you have done one of these once, you can skip this, and
# just load the library (unless the package is updated)

# OPTION 1 IF YOU HAVE R >= 3.1.2
# You only need to do this once, or if package is updated.
system("R CMD INSTALL TSPmap")
dyn.load("TSPmap/src/TSPmap.so")
# You need to do this every time you start R
library(TSPmap)

# OPTION 2 IF YOU HAVE R < 3.1.2
# You need to do this every time you start R
# go to the right directory with the c code
setwd("TSPmap/src/")
system("R CMD SHLIB rf.c")
```

```
dyn.load("../src/rf.so")

# then would need to source files..
# now change to the directory with the R code
setwd("../R/")
source('cutpoints.R')
source('duplicatemarkers.R')
source('readfile.R')
source('rfmatrix.R')
source('TSPinterface.R')

setwd("../../")

#---------- General Stuff ----------------------------

# everything assumes you are in the working directory for the repo
# if you aren't change it here:
# setwd("/path/to/TSPmap/")

#---------- Define variables -------------------------

# Define your path to the data directory
# this is where input and output files are located
```

```
# We'll use the joinmap demo data..
datapath = "JM20/"

# within this directory, we recommend the following:
# /rawdata/
# /results/LKH
# /results/Concorde
# /TSPfiles/

# LKH location
# there are two versions of the LKH and Concorde executables,
# one for mac, one for linux:
# so you have to pick one of these:
if(Sys.info()['sysname'] == "Darwin")
{
  # mac
  LKHpath = "TSPmap/exec/LKH_mac"
  Concordepath = "TSPmap/exec/concorde_mac"
}else{
  # linux
  LKHpath = "TSPmap/exec/LKH_linux"
  Concordepath = "TSPmap/exec/concorde_linux"
}
```

```
#-----------------------------------------------------------
# 1 read in data file
#-----------------------------------------------------------

# format can be either comma separated or tab delimited
# rows are individuals in the population, columns are markers
# If the second column isn't a "Classification" column for
# joinmap, then set the flag to false:
rawdata = readFile(paste0(datapath,
        "rawdata/JM20Demo_rawinput.tsv"),
        classificationFlag=F)

#-----------------------------------------------------------
# 2 check and remove duplicate markers
#-----------------------------------------------------------

# If you don't want to save the duplicate list to a file:
# first, define the list of possible calls
# the first 3 items in the list can be anything, in any order
# the last item is what you use to define missing data
calllist = c("a", "b", "", "-")
```

```
duplicates = finddups(rawdata, calllist, threshold = 99)

# if you also want a list of duplicates exported, you can do this
# instead:

# define where you want it saved
filepath = "results/"

duplicates = finddups(rawdata, calllist, threshold = 99,
                      filename=paste0(datapath, filepath,
                      paste("JM20dups","tsv", sep=".")))

# Remove duplicate markers from rawdata.
rawdata = removedups(rawdata, duplicates)

#------------------------------------------------------------
# 3 calculate recombination frequency matrix
#------------------------------------------------------------

# Compute recombination frequency matrix.
rfmat = computeRFmat(rawdata, calllist)

# it is easier to look at if you clean it up:
```

```
rfmatclean = cleanrfmat(rfmat)
rfmatclean = addMarkerNames(rfmatclean, rawdata)

# OPTIONAL: Write out the RF matrix after running computeRFmat.
# define what you want to call the file
# filename = addStampToFilename("JM20RFmatrixFromTSPmap")
filepath = "results/"

# you can't use the rfmatclean here, though obviously you could
# write rfmatclean to a file as well
writeRFmat(paste0(datapath, filepath,
          paste("JM20RFmatrixFromTSPmap",
          "csv", sep="."))), rawdata, rfmat)
# ------------------------------------------------------------
# 4 create TSP file for the solver (either LKH or Concorde)
# ------------------------------------------------------------

# filename = addStampToFilename("TSPfile")
filename = "JM20_TSPfile"
filepath = "SolverFiles/"

# This converts the rf matrix to an integer matrix that the TSP
# solvers need and saves it as a file
```

```
pathAndNameOfTspFile = createTSPFile(rawdata, rfmat, filename,
                        paste0(datapath, filepath))
# pathAndNameOfTspFile keeps track of where that file is.

#-------------------------------------------------------------
# 5 run the solver (depending on size of dataset this could take
# some time)
#-------------------------------------------------------------

# first define where the output file will be saved
outputPath = paste0(datapath, "results/LKH/")

# run LKH and time it in seconds
# this is the only step that should take some time
ptm <- proc.time()
pathAndNameOfLKHSolutionFile = callLKH(LKHpath,
          pathAndNameOfTspFile, paste0(datapath, filepath))
proc.time() - ptm

# pathAndNameOfLKHSolutionFile keeps track of output file LKH
# created

#-------------------------------------------------------------
```

```
# 6 save the results
#-----------------------------------------------------------------

# read in the solution, get the rf and calculate cumulative rf
# this is the marker order, but no groups have been defined yet
# we'll define groups by looking for large rf values

LKHMarkerlist = processLKHOutput(pathAndNameOfLKHSolutionFile,
                rawdata, rfmat)

# this isn't super useful, but if you need to check something you
# can save this

filename = "JM20_LKHmarkerlist"
filepath = "results/LKH/"

writeOutput(paste0(datapath, filepath, filename, ".csv"),
        ShiftedMarkerlist)

#-----------------------------------------------------------------
# 7 sort the results
#-----------------------------------------------------------------
```

```
# In order to process the results, we need to sort the rf matrix
# so that it is in the same order as the solution.
sortedMatrix = sortRFmatrix(rfmat, ShiftedMarkerlist)


#-----------------------------------------------------------
# 8 define number of groups you expect (generally the number of
# chromosomes)
#-----------------------------------------------------------

# Choose number of cutpoints
numberOfCutpoints = 5

#-----------------------------------------------------------
# 9 cut into the defined number of groups
#-----------------------------------------------------------

# Find list of cutpoints (use either Concorde or LKH result).
cutpoints = findcuts(ShiftedMarkerlist, numberOfCutpoints)

#-----------------------------------------------------------
# 10 examine whether these groups make sense
#-----------------------------------------------------------
```

```
# In this plot, you should see big jumps in the y axis
# corresponding to large rf values
# these are the likely breaks between groups
# remember, the solution wraps around, so part of one group may
# appear at the beginning and end of the x axis
plotTSPsoln(ShiftedMarkerlist, "JM20", "LKH")
# for the JM20 data, you should see 5 groups

# to make it easier to visualize, you can color the groups

# here are 15 colorblind safe colors:
colors = c("#C0C0C0","#808080","#000000","#FF0000","#800000",
           "#FFFF00","#707030","#00EE00","#009000","#00EEEE",
          "#00A0A0","#0000FF","#000080","#FF00FF","#900090")

# pick randomly enough colors for each cutpoint
colorvector = sample(colors, numberOfCutpoints)

# create an index for each marker of what color it should be
colorindex =  ColorGroupsinPlot(ShiftedMarkerlist, colorvector)

# plot with colors
```

```
plotTSPsoln(ShiftedMarkerlist, "JM20", "LKH", colorindex)

# you can add lines to identify each cutpoint:
for(i in cutpoints[,2])
{
  abline(h=ShiftedMarkerlist[i,4],col="blue",lty=2)
}

# or just look at the Markerlist object and compare with your
# marker names for us, our marker names tell us what
# chromosome on the reference genome they belong

#-----------------------------------------------------------
# 11 try steps 8-10 until you are satisfied
#-----------------------------------------------------------

#-----------------------------------------------------------
# 12 compute genetic distances
#-----------------------------------------------------------

# haldane
LKHHaldanelist = computeHaldane(ShiftedMarkerlist, cutpoints)
```

```
# kosambi
LKHKosambilist = computeKosambi(ShiftedMarkerlist, cutpoints)

# if a group is in the wrong order, you can reverse it:
# here, we are reversing group 4
reversedMarkerData = reverseGroup(ShiftedMarkerlist, cutpoints, 4,
        rfmat)


#-----------------------------------------------------------
# 13 save results
#-----------------------------------------------------------

# filename = addStampToFilename("LovellsConcordeOutput")

write.table(ShiftedMarkerlist, paste0(datapath, filepath,
        "JM20LKHfinalresults.tsv"), sep="\t", row.names=F, na="")

# for the JM20 data, you can compare to the joinmap results here:
setwd = "JM20/results/JoinMap/"
```

APPENDIX B

Advanced Usage

This code demonstrates the usage of the advanced features of the R package.

```
# ---------- General Stuff ----------

# everything assumes you are in the working directory for the repo
# if you aren't change it here:
# setwd("/path/to/TSPmap/")

# ---------- Define variables ----------

# Define your path to the data directory
# this is where input and output files are located
# We'll use the joinmap demo data..
datapath = "JM20/"

# within this directory, we recommend the following:
# /rawdata/
# /results/LKH
# /results/Concorde
# /TSPfiles/
```

```
# LKH location
# there are two versions of the LKH and Concorde executables,
# one for mac, one for linux:
# so you have to pick one of these:
if(Sys.info()['sysname'] == "Darwin")
{
  # mac
  LKHpath = "TSPmap/exec/LKH_mac"
  Concordepath = "TSPmap/exec/concorde_mac"
}else{
  # linux
  LKHpath = "TSPmap/exec/LKH_linux"
  Concordepath = "TSPmap/exec/concorde_linux"
}



#-------------------------------------------------------------
# 1 read in data file
#-------------------------------------------------------------

# format can be either comma separated or tab delimited
# rows are individuals in the population, columns are markers
```

```
# If the second column isn't a "Classification" column for joinmap,
# then set the flag to false:
rawdata = readFile(paste0(datapath,
        "rawdata/JM20Demo_rawinput.tsv"),
        classificationFlag=F)

#------------------------------------------------------------
# 2 find and remove duplicate markers
#------------------------------------------------------------

# If you don t want to save the duplicate list to a file:
# first, define the list of possible calls
# the first 3 items in the list can be anything, in any order
# the last item is what you use to define missing data
calllist = c("a", "b", "", "-")
duplicates = finddups(rawdata, calllist, threshold = 99)

# If you also want a list of duplicates exported, you can do
# this instead:

# define where you want it saved
filepath = "results/"
```

```
duplicates = finddups(rawdata, calllist, threshold = 99,
                      filename=paste0(datapath, filepath,
                      paste("JM20dups","tsv", sep=".")))

# Remove duplicate markers from rawdata.
rawdata = removedups(rawdata, duplicates)

#------------------------------------------------------------
# 3 calculate recombination frequency matrix
#------------------------------------------------------------

# Compute recombination frequency matrix.

# We will use the optional correction flag parameter.  This flag
# slightly modifies the recombination frequency matrix to
# avoid situations where markers with missing calls might
# act as a bridge between other markers, resulting in false
# correlations between markers which should not be
# related.
# This does not affect runtime but may affect solution quality,
# especially in data sets with many missing calls.

rfmat = computeRFmat(rawdata, calllist, corrFlag=T)
```

```
# it is easier to look at if you clean it up:
rfmatclean = cleanrfmat(rfmat)
rfmatclean = addMarkerNames(rfmatclean, rawdata)

# OPTIONAL: Write out the RF matrix to a file after running
# computeRFmat.
# define what you want to call the file
# filename = addStampToFilename("JM20RFmatrixFromTSPmap")
filepath = "results/"

# you can't use the rfmatclean here, though obviously you could
# write rfmatclean to a file as well
writeRFmat(paste0(datapath, filepath,
        paste("JM20RFmatrixFromTSPmap",
        "csv", sep="."))), rawdata, rfmat)
# Now we set an upper limit on the meaningful rf values.  Any
# recombination frequency values above this threshold are too
# large to contribute to the formation of the linkage groups, so
# they are effectively equivalent and will be replaced with a
# value of 0.5.  This allows the TSP solver to run more
# efficiently in order to reduce runtime. If this threshold is set
# too low, useful information may be lost and the linkage
```

```
# groups may be incorrect.  If this threshold is set too high, the
# TSP solver will take longer to run.

rfCap = 0.4
cappedrfmat = capRFmat(rfmat, rfCap, 0.5)

# The capped RF matrix will be used in all computations from this
# point onward.

#------------------------------------------------------------------
# 4 linkage group formation
#------------------------------------------------------------------

# define how many chromosomes in your species
numChromosomes = 5

# Break into small clusters.  This is the first step in forming
# linkage groups and ensures that the markers from
# different linkage groups are all well-separated.

# The internalRfThreshold variable sets the upper limit on the
# recombination frequency values that will be allowed within a
# cluster of markers. If this value is set too high, the linkage
```

```
# groups may not be well-separated and the linkage groups
# will be incorrect.  If this value is set too low, many small
# clusters will be formed resulting in longer run times.

rawclusters = autoClusterMST(rawdata, cappedrfmat, numChromosomes,
        LKHexec, internalRfThreshold=0.4)
# warning is ok:
# Warning message:
#   In clusterOrder[i] = which(clusterSizes == sortedSizes[i]) :
#   number of items to replace is not a multiple of replacement
#   length

# then merge the clusters into the final linkage groups.  This will
# create a number of clusters equal to the value of numChromosomes.

mergedclusters = autoMergeClusters(rawclusters, cappedrfmat,
        LKHexec, numChromosomes, rfCap)
# for each linkage group, compute the final linkage map.

finalMap = list()
for(i in 1:length(mergedclusters))
{
  finalMap[[i]] = createMap(rawdata, cappedrfmat,
```

```
        mergedclusters[i], Concordepath)
}

# remove empty elements, if necessary
finalMap = finalMap[!sapply(finalMap, is.null)]

# also within this loop it calculates the genetic distance. can
# do kosambi instead, if desired.
newdf = data.frame()
for(i in 1:length(finalMap)){
  # print (i)
  temp = as.data.frame(computeHaldane(finalMap[[i]]))
  temp$groupname = i
  newdf = rbind(newdf, temp)
}



#-----------------------------------------------------------
# 5 examine whether these groups make sense
#-----------------------------------------------------------

# if a group is in the wrong order, you can reverse it:
# here, we are reversing group 4
```

```
finalMap[[4]] = reverseCluster(finalMap[[4]], rfmat)

# Define a list of colors to be used on plots, if desired.
colors = c("red1", "green", "lightblue", "blue", "orange")

# These variables will be used to create an overall plot of all
# linkage groups.
fullMap = list()
fullColorVec = list()

# Create individual plots of the cumulative rf for each linkage
# group.
for(i in 1:length(finalMap))
{
  # Create a color vector for each plot:
  colorVec = rep(colors[i], length(finalMap[[i]][,1]))

  plotTSPsoln(finalMap[[i]], "JM20", "Concorde", colorVec)

  # This saves the plots.
  filename = paste0("JM20/results/JM20",i,".png")
  dev.copy(png,filename,width=8,height=8,units="in",res=100)
  dev.off()
```

```
  # Update the lists for the overall plot.
  fullMap = c(fullMap, as.numeric(finalMap[[i]][,4]))
  fullColorVec = c(fullColorVec, list(colorVec))
}

fullMap = unlist(fullMap)
fullColorVec = unlist(fullColorVec)

# Create the overall plot.
plot(fullMap, cex=.5, col = fullColorVec, xlab = "Marker Position",
        ylab = "Marker Position in JoinMap Solution", pch = 19)
# ------------------------------------------------------------
# 6 save results
# ------------------------------------------------------------

filename = addStampToFilename("finalist", "robject")
save(newdf, file=paste0("JM20/results/",filename))

filename = addStampToFilename("finalist", "tsv")
writeOutput(file=paste0("JM20/results/",filename), newdf)
```