Consensus Genetic Maps: A Graph Theoretic Approach

Benjamin N Jackson
Dept. of Electrical and Computer Engineering
Iowa State University
Ames, IA 50010
zbbrox@iastate.edu

Srinivas Aluru
Dept. of Electrical and Computer Engineering
Iowa State University
Ames, IA 50010
aluru@iastate.edu

Patrick S Schnable
Center for Plant Genomics
Iowa State University
Ames, IA 50010
schnable@iastate.edu

Abstract

A genetic map is an ordering of genetic markers constructed from genetic linkage data for use in linkage studies and experimental design. While traditional methods have focused on constructing maps from a single population study, increasingly maps are generated for multiple lines and populations of the same organism. For example, in crop plants, where the genetic variability is high, researchers have created maps for many populations. In the face of these new data, we address the increasingly important problem of generating a consensus map—an ordering of all markers in the various population studies. In our method, each input map is treated as a partial order on a set of markers. To find the most consistent order shared between maps, we model the partial orders as directed graphs. We create an aggregate by merginging the transitive closure of the input graphs and taking the transitive reduction of the result. In this process, cycles may need to be broken to resolve inconsistencies between the inputs. The cycle breaking problem is NP-hard, but the problem size depends upon the scope of the inconsistency between the input graphs, which will be local if the input graphs are from closely related organisms. We present results of running the resulting software on maps generated from seven populations of the crop plant Zea Mays.

1 Introduction

Genetic maps are a useful tool for researchers studying genetic linkage and genome structure and evolution. They have been used in designing experiments and for scaffolding during genome assembly. While traditionally researchers have focused on creating a single reference map from a single population, recently, with the push towards comparative genetics, researchers have been gathering data from multiple populations and lines of the same species.

This trend has identified issues in interpreting a reference map; in some organisms rearrangements occur even with little evolutionary distance. For example, a recent study identifies significant gene rearrangements between different lines of maize [2]. This knowledge does not remove the need for a reference map, but it does introduce the need for care in both creating the map and interpreting it correctly for a specific population.

We address the need to create a consensus map from these diverse input sources. To this end, we must deal with two types of inconsistencies in the input. First we wish to eliminate inconsistent orders resulting from experimental error. As we discuss below, the method by which these maps are constructed is much more likely to produce local errors in marker order than global errors. Second, we wish to identify and effectively address more significant inconsistencies in order. We will accomplish both of these goals through the use of graph algorithms, as described in Section 2.

1.1 Genetic Maps

Diploid organisms contain two copies of each chromosome, termed a pair of homologous chromosomes. Each member of a pair carries a copy (allele) of each gene. If the two alleles are the same the organism is said to be homozygous for the gene. Conversely, if the two alleles are different the organism is said to be heterozygous for the gene.

During the sexual reproduction of a diploid, genetic material from each member of homologous chromosomes mixes to create recombinant daughter chromosomes through a mechanism known as crossing over. A conceptual

mathematical model of crossing over treats the creation of the recombinant chromosome from a pair of parental chromosomes as follows. Each chromosome is modeled as a sequence of alleles. The recombinant chromosome is created by scanning the sequences of the two members of the pair of homologous parental chromosomes in parallel, copying one of the sequences into the sequence of the recombinant chromosome. At some point copying switches to the second sequence. This event is called a crossover. In the course of creating the recombinant chromosome, multiple crossovers can occur.

Given the genotype of the parent and the progeny and a large rate of heterozygocity, one can deduce if the alleles of two genes on a recombinant chromosome came from the same or different chromosomes in the parent. The frequency at which the alleles of two genes came from different chromosomes among the individuals in the mapping population is called the recombination fraction [18]. It is worth noting that two close genes will have a small recombination fraction, while two distant genes will have a fraction of about .5, as an odd or even number of crossovers is equally likely to occur between the two genes.

Many programs exist to handle the task of constructing a genetic map, and they all make use of the ideas presented above. Of interest to us is that given a pair of markers, it is easy to label them as close or distant. By applying this judgment universally, it is apparent that each marker will be placed fairly certainly in the proper neighborhood while its exact position may be incorrect.

The maximum likelihood method of map generation characterizes the probability of an ordering and then attempts to find the ordering with the maximum probability [5, 10, 11, 14, 18, 19]. The method produces an order by looking at the data in a holistic way, which aids in accuracy. However, as the number of markers mapped or the size of the population in the studies increases, the runtime of the method increases prohibitively.

Newer methods formulate the marker ordering problem as the traveling salesman problem (TSP) [13]. Using the pairwise distances between each marker, they attempt to find an ordering of the markers with minimum total length. The TSP problem is well studied, and while it is NP-hard, good approximation algorithms exist. These have been applied to the map generation problem [4, 15, 16]. They run quickly and allow larger problem instances to be approximately solved.

1.2 Consensus Maps

The first software to combine data from multiple mapping studies was JoinMap [20, 22]. JoinMap uses a statistical, data pooling approach.

Our method uses combinatorics in solving the problem. We consider a genetic map to be a partial order and model it as a directed acyclic graph (DAG). Nodes in the graph correspond to markers in the map, while edges correspond to the ordering of markers in the map. We combine DAGs from different studies into a single DAG that represents the consensus map, making use of the property that the input graphs will likely only disagree locally.

The first paper to model genetic maps as DAGs was [23]. Their goal was to aggregate data, while we are interested in conflict resolution and coming to a consensus. Thus they do not explicitly treat the maps as partial orders, use the transitive reduction/closure operations as we describe, and use a different method of graph merging. [17] uses similar ideas to reason about partial orders in the context of artificial intelligence.

Our model provides a number of benefits when applied to the problem.

1. As long as the inputs disagree on a local scope, the software is able to find the combinatorially best consensus.
2. Modeling the input as a partial order instead of a linear order allows for flexibility in specifying the input maps.
3. Because it processes finished maps, the method finishes within seconds or minutes.

In addition, the software described presents a number of useful features.

1. The input to the software is a .xml description of the maps.
2. The resulting map displays any discrepancies between the inputs.
3. The resulting map is annotated with information about which input maps contain which marker, which is useful for analysis.
4. The software outputs the input maps and the consensus map in .gdl format, which can be read by interactive map drawing software that is free for academic use.

2 Approach

We model a genetic map as a partial order on a set of markers. We represent the data as a directed acyclic graph (DAG) $G = \{V, E\}$. The vertex set V of the graph corresponds to the set of markers in the map. An edge (u, v) exists in E for nodes u and v if u comes before v in the input map and there is no other node $k \in V$ with edges $(u, k) \in E$ and $(k, v) \in E$. We say that v follows u or v is reachable from u if there exists a path in the graph from u to

v, denoted $\langle u, ..., v\rangle \in G$. The set of edges in our graph is the minimum size set of edges that captures all the ordering information for the set of markers. This graph is also known as a transitively reduced graph because of this property. In this paper we will make implicit use of the connection between a partial order and a DAG by referring to the order of the DAG and the graph of the order.

Given a set of DAGs $\{G_1, G_2, ...G_n\}$, we wish to create a DAG, G_C, whose order is the consensus order most consistent with the input orders, which might be inconsistent. For example, one map might have the path $\langle a, ..., b\rangle$, while another map might have $\langle b, ..., a\rangle$. If no inconsistencies exist, then the resulting order can be easily calculated as the superposition of the input graphs. However, given experimental data this is not likely.

In fact, if dealing with maps from different lines of the same plant, both inputs might be "correct" in that the genes in question might have different orders. However, for the purposes of this paper, we wish to create a resulting order by choosing between alternatives as much as possible. We will present to the user information about which decisions were made in reaching the consensus. In this way, large scale inconsistencies can be interpreted and dealt with.

We propose a four stage method for generating the consensus. First, we run a transitive closure algorithm on the input graphs. Next, we combine these graphs into a single graph that may contain cycles. Third, we break cycles in the aggregate. Finally, we run a transitive reduction algorithm to produce the result.

2.1 Transitive Closure

A transitive closure of a graph $G = \{V, E\}$ is a graph $G' = \{V', E'\}$ such that $V' = V$ and E' consists of all edges $\{(u, v)|\langle u, ..., v\rangle \in G\}$ [8].

For the purposes of our algorithm, we will a weight to each pair of nodes, corresponding to our confidence in their ordering. We define the the weight $w(u, v)$ as the shortest path from u to v in G. We define $w(v, u) = -w(u, v)$ and $w(u, v) = 0$ if neither $\langle u, ..., v\rangle \in G$ nor $\langle v, ..., u\rangle \in G$. Our weighting scheme comes directly from our knowledge that local mistakes in order are much more likely than global mistakes. If v directly follows u, we are less confident of the order than if v follows u with several markers in between.

We calculate all the pairwise shortest paths between nodes using the classic Floyd-Warshal $O(n^3)$ algorithm [6]. The pairwise shortest paths algorithm creates the transitive closure by reporting a positive shortest path for any pair of nodes u and v with v reachable from u. After the algorithm has finished, negative weights are assigned to the symmetric pair for each pair with a positive weight.

2.2 Aggregate Graph

The nodes in each input graph correspond to markers in the global set of markers U. For some node v in G_i, there could exist some equivalent node v' in G_j with v and v' corresponding to the same marker in U. Therefore, we must create a global index for identifying markers. Let the set of all markers in the input maps $U = V_1 \cup V_2 \cup ... \cup V_n$ be the global marker index of size $\|U\| = t$. Then we can create a mapping of any node v, $f(v) = i, 1 \leq i \leq t$.

We represent the graph G'_i using a $t \times t$ size matrix M_i such that $M[f(u), f(v)] = w(u, v)$. We can construct a global map G_A by taking the summation $M_A = \sum_{i=1}^{n} G'_i$. The nodes of G_A are all markers in the universe. An edge (u, v) exists in G_A if $M[u, v] > 0$. M_A can be thought of as the results of a vote. Each map has a weighted vote it casts for the ordering of nodes u and v. After tallying the votes of each map, the result in G_A is that u comes before v or v comes before u, or they exist on alternate paths through the graph.

2.3 Cycle Breaking

The voting scheme used when creating the aggregate G_A deals with direct pairwise inconsistencies between the input maps by allowing a weighted majority to decide the ordering of each pair of markers. However, other inconsistencies can arise transitively. A majority of input maps might say that v follows u, w follows v, and u follows w. In this case G_A contains a cycle denoted $\langle u, v, w, u\rangle$ and no final consensus order can be calculated before the cycle is broken.

We use the weights assigned to each edge in finding the optimal way to break cycles. The weight corresponds to the confidence we have in a particular ordering of markers, and we wish to remove those orderings that we have the least confidence in. To optimally break cycles, we remove that set of edges E_c such that the sum of all weights on those edges is minimum and all cycles are broken. The cycle breaking problem is known to be NP-hard [7].

We will model this problem as the set cover problem. Given a universal set $U = \{e_1, e_2, ...e_n\}$ and a set of sets of elements from U, $S = \{s_1, s_2, ...s_m\}$, find a minimum sized subset S' such that $\bigcup_{s_i \in S'} s_i = U$. The weighted version of the set cover problem additionally weights each set $w(s_i)$, and asks to find the subset S' that covers U with $\sum_{s_i \in S'} w(s_i)$ minimized.

Let U_C be the set of all simple cycles in G_A and $C(u, v)$ be the set of cycles of the form $\langle a, ..., u, v, ..., a\rangle$, that is all cycles containing the edge (u, v). In order to break all the cycles, we must find a set of edges E_c to remove from the graph, such that that $\bigcup_{(u_i,v_j)\in E_c} C(u_i, v_j) = U_C$. In order to break cycles optimally, we want $\sum_{(u_i,v_j)\in E_c} w(u_i, v_j)$ to be minimized. Obviously, if we consider the set of all

cycles to be U, let S be the set of all $C(u,v)$ and let the weight $w(C(u,v)) = w(u,v)$, this formulation of the cycle breaking problem is equivalent to the weighted set cover problem as described above.

The first step in breaking cycles is to enumerate the simple cycles in the graph. We will use the method proposed by Johnson [9] which finds the cycles in $O(C(V+E))$ time, where C is the number of simple cycles in the graph. The details are omitted here.

The number of cycles in the graph could be exponential, but is limited by the size of each strongly connected component. Therefore the size of each strongly connected component must be small for any solution involving enumerating cycles to be feasible. Data reduction methods reduce the input size by removing both sets and elements from the universe. We use the following three rules to reduce the problem size [21].

1. If $s_i \subseteq s_j$ and $w(s_i) \geq w(s_j)$, then s_i can be removed from consideration. In this case, under any condition that one might choose s_i, one could choose s_j and be no worse off.
2. If for all $s_i, e_k \in s_i \Rightarrow e_l \in s_i$, then e_k can be removed from consideration. Any set that covers e_l also covers e_k.
3. If element e_k only appears in set s_i, then set s_i must be selected, and all elements $e_l \in s_i$ should be removed from consideration, as they have been covered by the selection of s_i.

These three rules are repeatedly applied to the input data until no more reduction occurs. A solution to the problem will be an assignment of inclusion or exclusion to each edge. We search the solution space using a basic branch and bound technique [3].

Each strongly connected component of the cyclic graph G_A is considered independently. As long as the disagreement among input graphs is local, the size of these strongly connected components and number of cycles will be small, and solving the problem exactly is feasible. On the other hand, if an input disagrees with the general consensus because of large rearrangement, then we must deal with larger connected components.

The greedy $log(n)$ approximation of the set cover problem works as follows. Initialize the solution to be empty. Next, repeatedly choose the set M that contains the most uncovered elements, and add it to the solution, covering all uncovered elements in M until all elements are covered. This approximation will be used to break apart large connected components.

We cannot enumerate the cycles because the number of cycles can be exponential in the size of the component. Instead, we will approximate the ratio of cycle count on each edge to the total number of cycles in the graph by repeatedly selecting a cycle in the connected component and incrementing a counter on each edge along that cycle. To randomly select a cycle, we first randomly select some starting node s. Then we perform a randomized depth first search through the graph until we reach s. The path through the DFS tree $\langle s, ..., s \rangle$ is taken as a random cycle in the graph, and we accumulate a value along the edges of that cycle.

After many repetitions, we scan the component to find the edge with maximum accumulation, and remove this edge from the graph. Removing this edge reduces the size of the connected component, and may break the connected component into smaller connected components.

Thus the flow of the cycle breaking phase is to initially enumerate the connected components and place them in a queue. While the queue is not empty, deque the next component and optimally break it if it is small. If it is large, use the greedy approximation to remove a single edge, rerun the connected component algorithm on the subgraph, and place any new connected components in the queue. When the queue becomes empty, all connected components in the graph have been broken.

The algorithm combines approximation with the exact solution to find a good solution quickly, invoking the approximation algorithm only if necessary. If the input maps are reasonably consistent, the approximation scheme is never used. We will further discuss implications of this in Section 4.

2.4 Transitive Reduction

Once the cycles are broken in the input graph, the final step of the algorithm is to run a transitive reduction. This can easily be done in $O(n^3)$ time using the conceptual inverse of the Floyd-Warshal algorithm [1]. The details are omitted here.

2.5 Drawing the Map

The result of this process is a directed acyclic graph with the minimum number of edges required to capture the order information contained therein. To display this graph, we make use of interactive graph drawing software aiSee (www.aisee.com) which is free for academic use. The output of our software is a .gdl file which is input to aiSee. An image of the resulting graph is shown in Figure 1.

3 Results

We implemented the described method as a Java program (available upon request). We ran this software on synthetic data and experimental data. The synthetic data sets were designed to test both the software's ability to run efficiently in

the face of local inconsistencies among input sets as well as its ability to correct for those errors by finding the best order given conflicting data. The approximation scheme was not needed for this data. The real data set consisted of data gathered from seven populations of the crop plant *Zea mays*.

We will first discuss the synthetic data. As argued earlier, maps tend to be more correct in the ordering of genes at larger distances. Local mistakes are likely as the reversal of two adjacent markers only minimally changes the distance between each of these markers and the global set. Therefore we created synthetic data in which errors occur at a local level, simulating experimental error.

We make the assumption that a true ordering exists, and the actual ordering of genes on the synthetic chromosome is said to be the natural ordering of numbers between 1 and n. To create a partial map M_i of this chromosome, we choose a subset S_i of size s_i of the n numbers as the markers ordered in M_i. To introduce error, we perform x_i random alterations of the map, each alteration corresponding to a swapping of adjacent markers.

We wish to ascertain if software halts in reasonable time with an answer (in other words, that the number of cycles generated in the consensus graph is small), and we also wish to measure the quality of the resulting order. The quality is measured by counting the edges in the resulting DAG. Any time a node i precedes a node j, but $i > j$ using the natural ordering of numbers, we count this ordering as incorrect.

By counting both the edges of transitively reduced digraph n and the incorrect edges in the digraph b, we can calculate a score for the result by taking the ratio $\frac{b}{n}$. We then vary the number, completeness, and correctness of input maps and measure the resulting accuracy of the output map.

In the synthetic data, the number of maps generated, which varied between 3 and 9, did not have an effect on the correctness of the output. We are not worried about completeness when looking at the synthetic data. In real data, the number of maps is important as they each will often have exclusive markers and information not found in others. However, given the nature of the synthetic map generation, each map is as likely as any other to contain some bit of information. Therefore, five maps were generated for the analysis.

The two other variables, completeness and correctness did have a substantial effect on the quality of the results. The completeness of the maps effects the amount of overlap between two maps, and the more markers are shared between maps, the more likely errors can be corrected by the voting scheme. The number of errors introduced in the inputs more directly affects the number of errors in the output.

For all inputs, the resulting cycle breaking problems turned out to be small enough to be solved directly, and the program ran in a matter of minutes or seconds, depending upon the data. The results are presented in Figure 2. Each point represents an average of 5 randomly generated inputs.

The maize mapping data used in the study was assembled at the Center for Plant Genomics at Iowa State University in collaboration with the laboratory of Abraham Korol at the University of Haifa. Six populations of maize were mapped, labeled population A-F, with each map sharing a large number of markers. The maps also shared markers with the previously mapped IBM population [12].

The software used to develop the individual map for each line is presented in [16]. There are two versions of the genetic map generated by this software. The skeletal map is less complete but more accurate as it consists of the set of markers whose order remains invariant after applying the statistical technique of bootstrapping. The second more complete map adds additional markers by attaching each additional marker to the skeletal marker to which it is considered closest.

The maize genome has 10 chromosomes, and a consensus map for each chromosome was assembled by the software from the complete maps. For chromosomes 3,4,5,7,9, and 10, the mapping software produced a result without having to do any approximations: all errors were local. However, for chromosomes 1,2, and 6, map B showed a large scale disagreement with the consensus order. For chromosome 8, map F showed a large disagreement. By removing the offending maps from the input, consensus orders could again be easily produced without resorting to the approximation scheme. Figure 1 shows the graph visualization software aiSee. Figure 3 shows the consensus map of chromosome 1 with map B included, while Figure 4 shows the consensus map with map B excluded from the input.

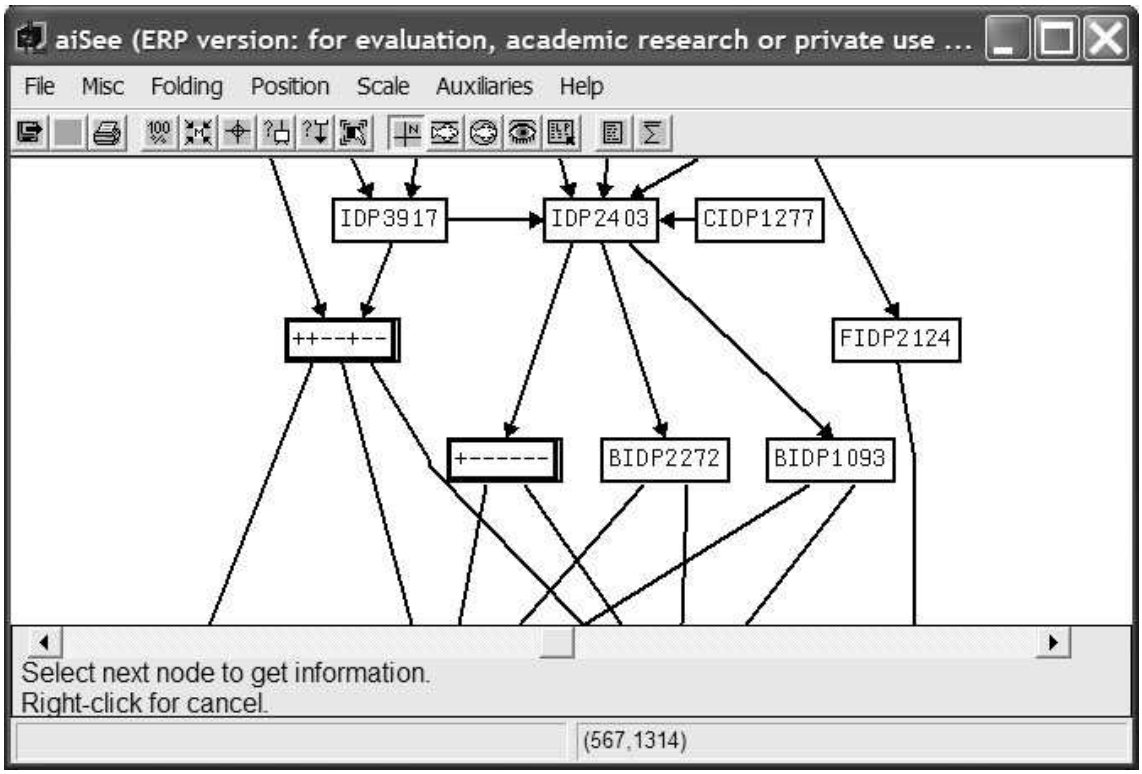

Figure 1. The graph viewing software aiSee.

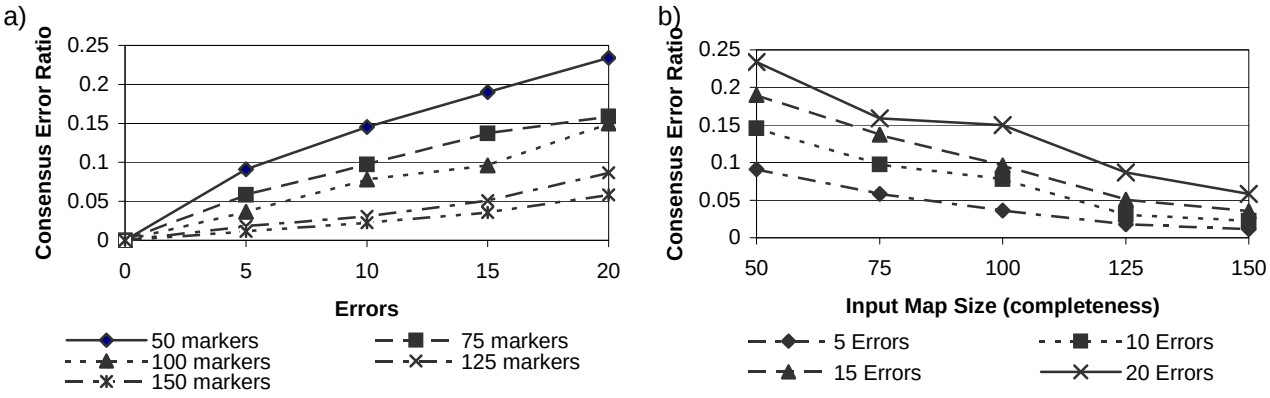

Figure 2. a) The error ratio in the consensus map versus input error rate on synthetic data. b) The error ratio in the consensus map versus input completeness synthetic data. There are 200 total markers on the synthetic chromosome.

4 Discussion

The creation of genetic maps is a heavily studied area in computational biology. While many methods of generating maps from raw data have been explored, the main software available to join multiple maps has been JoinMap, which uses a statistics based approach and data pooling.

We use graph algorithms to solve this problem. While finding the exact solution is NP-hard, the problem sizes of the NP-hard subproblems depend on the inconsistency in the input. As long as the inconsistency is local, the problem remains exactly solvable. This is because the size of the cycle-breaking problem is limited by the size of the graph's strongly connected components.

A more complete software can handle large scale differences by using an approximation scheme. Our software will switch into an approximation mode if faced with such a situation.

The resulting consensus map embodies the entire set of gene order information, as it combines the data from various inputs. The consensus map graph is transitively reduced, and in addition to the order, provides three pieces of information. First, it shows in red the edges removed during cycle breaking to reach the consensus. Second, it shows the ratio of 'yes' votes to total votes as edge line thickness. Finally, it tags each maker with the set of input maps that marker occurred in, shown as a string of pluses and minuses (see Figure 1). For example, a tag of "$--+-++-$" would indicate that a marker was found in inputs 3, 5, and 6.

This graph could be used in identifying those areas of the chromosome with a large amount of ambiguity. If resulting graph is very wide or there exist two long independent paths between nodes u and v, then there is a lot of missing information about the relative ordering of nodes on different paths. We can use the graph to prioritize the design of laboratory experiments to fill in the missing information.

The technique is designed for creating aggregate maps from closely related organisms. The technique is less useful the more evolutionarily disparate the two organisms are, as it is more likely that large scale rearrangements will have occurred during evolution. The technique can locate such differences, but they must be dealt with outside of the software for the consensus to be meaningful.

In our experiment, we chose to remove from the aggregate those maps that showed a rearrangement when compared to the general consensus. Alternatively, one could envision aligning such maps with the consensus through a series of reversals of subgraphs. By performing the inverted set of reversals on the consensus, we could infer a likely consensus for the differing maps. This would require future research.

This method could be used in map creation. An alternative approach to making a map from all the data would be to

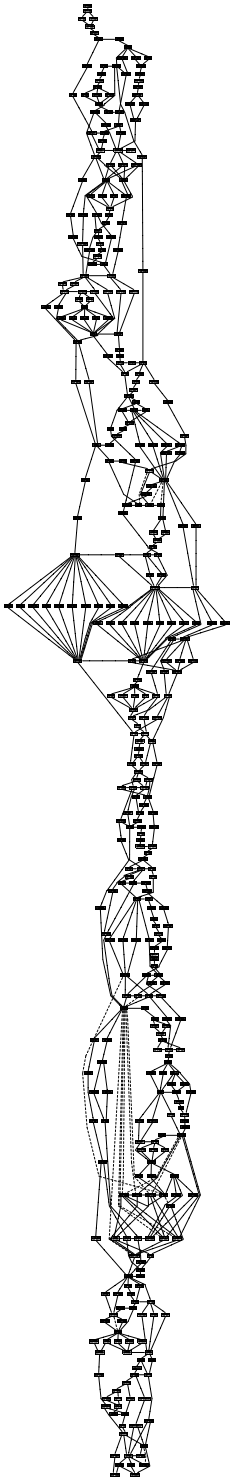

Figure 3. Consensus map for chromosome 1. Dashed edges correspond to edges removed during cycle breaking.

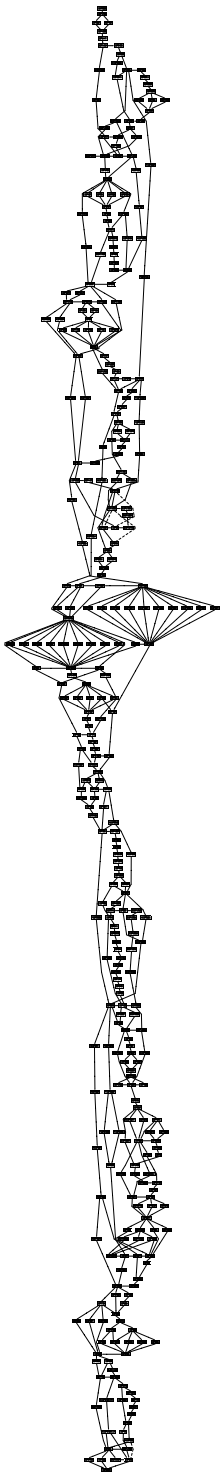

Figure 4. Consensus map for chromosome 1 without population B.

divide the data from a mapping experiment into overlapping subsets, use maximum likelihood techniques on these subsets, and combine the results into a consensus map. It would be interesting to compare this method of map generation to a TSP method.

An extension of the method would allow flexibility in the weighting the edges of the input graphs. Individual edges could be weighted based on some criteria, perhaps based on the genetic distance between markers. Additionally, one could weight each map based on perceived quality (e.g. the number of individuals in the mapping population). The goal of these weighting schemes would be to produce a higher quality consensus.

The presented method does not assign distances to the consensus. Unfortunately, genetic distances do not directly compare between studies or even along the chromosome in the same study. Some sort of complex normalization would be needed to address this issue.

5 Conclusion

We presented a method for creating consensus genetic maps from different populations and experiments for a species. This method solves the consensus map problem by modeling the inputs as inconsistent partial orders and uses combinatoric ideas and a graph theoretic approach in reaching an optimal solution. Despite containing an NP-hard subproblem, the software runs quickly on actual data, as the data exhibits the property of disagreements being limited to a local scope. We validated the solution on synthetic and experimental data and constructed consensus maps for an important crop plant, maize.

References

[1] A.V. Aho, M.R. Garey, and J.D. Ulman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

[2] S. Brunner, K. Fengler, M. Morgante, S. Tingey, and A. Rafalski. Evolution of dna sequence nonhomolgies among maize inbreds. *The Plant Cell*, 17:343–360, 2005.

[3] T.H. Corman, C.E. Leiserson, R.L Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. 2003.

[4] R. Doerge. Constructing genetic maps by rapid chain delineation. *Genetics Research*, 69:35–43, 1996.

[5] C.T. Falk. Preliminary ordering of multiple linked loci using pairwise linkage data. *Journal of Quantum Trait Loci*, 2, 1992.

[6] R.W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[7] M.R. Gary and D.S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. 1979.

[8] A. Goralcikova and K. Koubek. A reduct-and-closure algorithm for graphs. *Mathematical Foundations of Computer Science*, 74:301–307, 1979.

[9] D.B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[10] S. Knapp, C. Echt, and B.H. Liu. Genome mapping with non-inbred crosses using gmendel 2.0. *Maize Genetics Cooperation Newsletter*, 66:22–79, 1992.

[11] E. Lander, P. Green, J. Abrahamson, A. Barlow, M.J. Daly, S.E. Lincoln, and L. Newburg. An interactive computer package for constructing primary genetic linkage maps of experimental and natural populations. *Genomics*, 1:174–181, 1997.

[12] M. Lee, N. Sharopova, W.D. Beavis, D. Grant, M. Katt, D Blair, and A. Hallauer. Expanding the genetic map of maize with intermated b73 mo17 (ibm) population. *Plant Molecular Biology*, 48:453–461, 2002.

[13] B.H. Liu. The gene order problem, an analog of the traveling salesman problem. *Plant Genome 95*, 1995.

[14] T.C. Matice, M. Perlin, and A. Chakravert. Multimap: An expert system for automated genetic linkage mapping. In *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology*, pages 260–265, 1993.

[15] D. Mester and O. Braysy. Fast and high precision algorithms for optimization in large-scale genomic problems. *Computational Biology and Chemistry*, 28:281–289, 2004.

[16] D. Mester, E. Ronin, E. Nevo, and A. Korol. Constructing large scale genetic maps using evolutionary strategy algorithm. *Genetics*, 165:2269–2282, 2003.

[17] A. Montanary and F. Massimo. Pairing transitive closure and reduction to efficiently reason about partially ordered events. In *Proceedings of the Congress of the Italian Association for Artificial Intelligence*, pages 208–217, 1999.

[18] J. Ott. *Analysis of Human Genetic Linkage*. 1985.

[19] T. Schiex and C. Gaspin. Carthagene: constructing and joining maximum likelihood genetic maps. *Proceedings, The International Conference on Intelligent Systems in Molecular Biology*, 48:453–461, 1997.

[20] P. Stam. Construction of integrated genetic linkage maps by means of a new computer package: Joinmap. *The Plant Journal*, 3(5):739–744, 1993.

[21] M. Syslo, N. Deo, and J. Kowalik. *Discreet Optimization Algorithms and Pascal Programs*. 1983.

[22] J.W. VanOoigen and R.E. Voorrips. Joinmap 3.0 software for the calculation of genetic linkage maps. Technical report, Plant Research International, 2001.

[23] I.V. Yap, D. Schneider, J. Kleinberg, D. Matthews, S. Cartinhour, and S.R. McCough. A graph-theoretic approach to comparing and integrating genetic physical and sequence-based maps. *Genetics*, 165:2235–2247, 2003.