Accurate Construction of Consensus Genetic Maps via Integer Linear Programming

Yonghui Wu, Timothy J. Close, and Stefano Lonardi

Abstract—We study the problem of merging genetic maps, when the individual genetic maps are given as directed acyclic graphs. The computational problem is to build a *consensus map*, which is a directed graph that includes and is consistent with all (or, the vast majority of) the markers in the input maps. However, when markers in the individual maps have ordering conflicts, the resulting consensus map will contain cycles. Here, we formulate the problem of resolving cycles in the context of a parsimonious paradigm that takes into account two types of errors that may be present in the input maps, namely, local reshuffles and global displacements. The resulting combinatorial optimization problem is, in turn, expressed as an integer linear program. A fast approximation algorithm is proposed, and an additional speedup heuristic is developed. Our algorithms were implemented in a software tool named MERGEMAP which is freely available for academic use. An extensive set of experiments shows that MERGEMAP consistently outperforms JOINMAP, which is the most popular tool currently available for this task, both in terms of accuracy and running time. MERGEMAP is available for download at http://www.cs.ucr.edu/~yonghui/mgmap.html.

Index Terms—Linear programming, constrained optimization, algorithms, biology and genetics.

1 INTRODUCTION

GENETIC linkage maps represent the relative positions of genetic markers along a chromosome. The distance between markers is associated with the frequency at which two genetic loci become separated during chromosomal recombination.

The problem of building genetic linkage maps from genotyping data can be traced back to the beginning of the last century when life scientists started to investigate the recombinational nature and cellular behavior of chromosomes. In his pioneering work, Sturtevant studied in 1913 the first genetic linkage map of chromosome X of *Drosophila melanogaster* [1]. Early genetic linkage maps had just a few tens of phenotypic markers obtained one by one by recording biochemical and morphological variations of the organism under study, mainly following mutation. With the introduction of DNA-based markers (i.e., RFLPs, RAPDs, SSRs, and AFLPs), genetic maps have become much more densely populated. The number of markers has increased recently well above a thousand in a number of organisms with the adoption of DArT, SFP, and especially SNP markers, the latter providing avenues to hundred of thousands to millions of markers per genome. High-density genetic maps are the cornerstone of a variety of biological studies including map-based cloning, association genetics, and map-assisted breeding. Because they are relatively inexpensive compared to whole genome sequencing, high-density genetic linkage maps are currently of great interest, in particular for organisms with large genomes.

Traditionally, scientists have focused on building genetic map for a single mapping population, a task for which a wide variety of software tools are available and have satisfactory performance, e.g., JOINMAP [2], CARTHAGENE [3], ANTMAP [4], RECORD [5], TMAP [6], and our own MSTMAP [7], [8].

In recent years, the rapid adoption of high-throughput genotyping technologies has been paralleled not only by an increase in the map density but also by an expansion in the variety of marker types. It is increasingly common to find multiple genetic maps available for the same organism, usually for different sets of genetic markers and genotyping technologies. Notable examples are genetic linkage maps based on microsatellites in human [9] and cattle [10], and maps based on sequence length polymorphism in mouse [11] and rat [12]. In the case of maize (*Zea mays*), seven distinct mapping populations have been analyzed [13].

When multiple genetic maps are available, they typically share a subset of common markers. In this case, it is often desirable to construct a bigger map (hereafter called *consensus map*) that includes and is consistent with all (or, the vast majority of) the markers in the individual maps. A consensus map is desirable because it provides a higher density of markers, and therefore, a greater genome coverage than the individual maps. However, building a consensus map that is consistent with the individual maps is not always possible because genotyping errors are likely to introduce ordering conflicts between the individual maps. Due to the way individual genetic maps are assembled from genotyping data, two types of errors can be observed, namely, local reshuffles and global displacements. Local reshuffles refer to inaccuracies in the order of nearby markers, whereas global displacements refer to the

- *Y. Wu is with Google, Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043. E-mail: yonghui@google.com.*
- *T.J. Close is with the Department of Botany and Plant Sciences, University of California, Riverside, CA 92521. E-mail: timothy.close@ucr.edu.*
- *S. Lonardi is with the Department of Computer Science and Engineering, University of California, Engineering Building II, Riverside, CA 92521. E-mail: stelo@cs.ucr.edu.*

Manuscript received 1 Mar. 2009; revised 26 Sept. 2009; accepted 19 Jan. 2010; published online 20 Apr. 2010.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-2009-03-0027. Digital Object Identifier no. 10.1109/TCBB.2010.35.

cases where a few markers are placed at positions far from the correct ones. We are not the first ones to observe the presence of global displacement errors in the individual maps. Jackson et al. [14] observe that by removing problematic markers (called *outliers*) from the individual maps, the task of building a consensus map is greatly eased. When addressing the conflicts in the consensus maps, however, both types of errors need to be accounted for.

Several systematic approaches have been proposed to construct consensus maps (see, e.g., [13], [14], [15], [16], [17], [18]). The method adopted by Beavis and Grant [15] for the integration of maize maps is to pool together the genotyping data from the individual mapping populations, and then, rely on traditional mapping algorithms to build the consensus map. This pooling strategy is commonly used, but it has several shortcomings. First, it cannot be used in all circumstances. For example, when the data are obtained from different populations (e.g., one data set obtained from a double haploid population and another from an F5 recombinant inbred lines population), then they cannot be merged and treated uniformly downstream. Second, the pooling method results in a large number of missing observations, and the percentage of missing data increases with the number of data sets to be combined. For instance, combining two data sets with 80 percent shared markers will result in 16.67 percent missing observations. Combining three data sets with 80 percent shared markers will increase the percentage of missing observations to 28.57 percent. A large amount of missing observations combined with the limited tolerance to missing data by existing mapping algorithms inevitably deteriorates the quality of the consensus map.

An alternative approach, like the one used in JOINMAP [16], is to first obtain the consensus estimates of pairwise genetic distances by weighting for population structure and size. Then, one searches for a map that minimizes an objective function that measures the fit of the map to the distance estimates and the overall quality of the map. The drawbacks of this approach are twofold. First, distance estimates are not very accurate when based on a small sample of recombination events. Construction of genetic maps based on these estimates will result in inaccuracies in the ordering between nearby markers. Second, the computational problem of searching for an optimal map with respect to the objective function being used is very time-consuming. For instance, the most recent version of JOINMAP occupied a single PC workstation for *three months* in order to construct a consensus map from three individual maps of barley containing about 1,800 markers (markers were divided into seven linkage groups of roughly equal sizes). The same job was carried out by our method in about 5 minutes. The difficulties of constructing integrated maps by JOINMAP have also been discussed at length by Wenzl et al. [19]. Despite these drawbacks, JOINMAP is still the most popular software package available to build consensus maps.

There are other less known commercial tools like MULTIPOINT (http://www.multiqtl.com/) or CARTEBLANCHE (http://www.keygene.com/). The approach of MULTIPOINT to build consensus maps is to perform a reprocessing of the initial genotyping data rather than merging the individual maps [17]. The problem that MULTIPOINT needs to solve is computationally very hard, which drastically limits the number of markers in the maps (see [7], [8] for a discussion on the computational disadvantages of casting this problem as a version of the Traveling Salesman Problem).

In general, the fact that most of the available tools to build consensus maps (e.g., JOINMAP, MULTIPOINT, and CARTEBLANCHE) are proprietary commercial software tools hinders the ability of many academic labs to carry out this task. At the same time, it limits the ability of comparing competitive algorithmic approaches—as a consequence, we decided to compare our method only with JOINMAP and the pooling method [15]. We should note here that the latest version of CARTHAGENE [3] also offers some limited abilities of merging two genetic maps.

The most recent approach relies on graph theory and was initially proposed by Yap et al. [18] and later extended by Jackson et al. [13], [14]. Yap et al. [18] use directed acyclic graphs (DAGs) to represent maps from individual populations. The set of DAGs is then merged into a consensus graph on the basis of their shared vertices. A directed cycle in the resulting graph indicates an inconsistency among the individual maps with regard to the order of the markers involved. In order to resolve the inconsistencies, Jackson et al. [13], [14] proposed to break cycles by removing a minimum weight set of feedback edges. This objective function is reasonable when dealing with local reshuffles. However, in the presence of global relocations, it is not appropriate because too many edges need to be deleted in order for all the cycles to be broken. An alternative approach is to remove a minimum weight feedback vertex set from the graph. The obvious drawback of this method is that markers corresponding to those deleted vertices will be excluded from the consensus map.

Our contribution. We follow the graph theoretical paradigm outlined in [13], [14], [18] and represent individual genetic maps as DAGs. Individual maps are combined into a single directed graph according to their shared vertices. Any ordering conflict among the individual maps generates cycles in the combined graph. Here, we propose to resolve the cycles by removing the smallest set of (feedback) marker occurrences. We need to emphasize that we are not deleting markers, but marker occurrences. While a specific genetic marker may be shared by multiple individual maps, a marker occurrence refers to the appearance of a marker in a particular individual map. The deletion of a marker occurrence will not affect the occurrences of the same marker in other maps. Also, notice the difference between our objective function and that of [13], [14] which tries to delete a minimum number of feedback arcs to resolve all conflicts. Overall, the strategy of identifying and eliminating a small number of marker occurrences has less destructive effects when dealing with errors in the individual maps.

Our parsimonious strategy is first cast in a combinatorial optimization framework via integer linear programming, and then, a polynomial-time approximation algorithm is proposed to solve it. When the size of the problem to be solved is too large compared to the computing resources available, a heuristic can be used to decompose the original problem into smaller subproblems that can be solved independently.

Our own experience using the directed acyclic graph approach to represent the consensus map has shown that the resulting consensus DAGs are usually very complex and convoluted (see Fig. 1a for an example). In order to

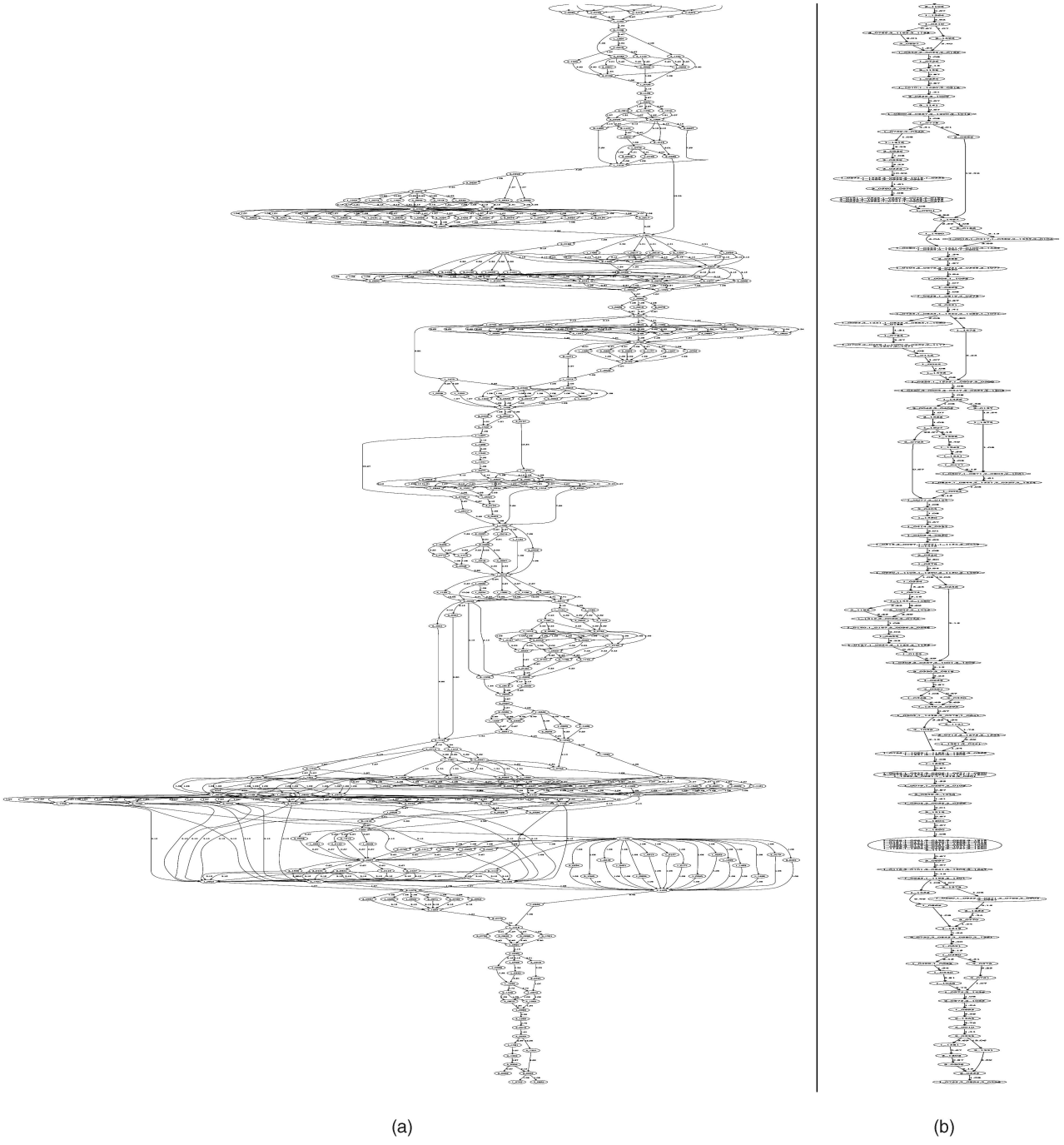


Fig. 1. A side by side comparison of (b) a simplified DAG and (a) an original DAG of barley chromosome 5H. In the simplified DAG, markers are condensed into supermarkers. Each supernode is represented as a single ellipse in the figure.

allow geneticists to visualize and make use of the consensus map, once all cycles in the consensus map are resolved, we postprocess the resulting directed acyclic graph by removing redundant edges and merging nodes without reintroducing conflicts (see Fig. 1b for the corresponding simplified map). The resulting simplified consensus map can be directly used by geneticists in downstream applications. If a linear ordering is needed, we employ another novel algorithm that produces a linear order of the markers which is consistent with the consensus graph.

The last two steps in our algorithmic pipeline, namely, simplifying and linearizing the DAG, further distinguish our approach from those in [13], [14], [18]. The final output of our workflow is a linear order of sets of markers which is a format geneticists are accustomed to. The output of the methods by Yap et al. [18] and Jackson et al. [13], [14] is instead a tangled DAG, which is often so complex and convoluted that it may not be useful to geneticists.

In order to assess strengths and weaknesses of our method, an extensive set of experiments both on synthetic data and real barley genotyping data was carried out. The evaluations were mainly concerned with comparing our tool to JOINMAP [16] which is, to the best of our knowledge, the most commonly used tool to build consensus maps. Our

approach produces consistently better results than JOINMAP, both in terms of accuracy and running time. Our method also outperforms the method of pooling together genotyping data from individual maps.

2 Methods

Our approach consists of four sequential steps. In the first step, the individual maps are compared with each other to determine a consistent orientation (details in Section 2.2). In the second step, the maps are merged and the conflicts among the individual maps are resolved by deleting a minimum number of marker occurrences (see Section 2.3). In the third step, the consensus DAG resulting from the previous step is simplified. The details of this step are presented in Section 2.4. In the fourth last step, the simplified DAG is "linearized" to produce the consensus map (see Section 2.5).

2.1 Preliminaries and Notations

A *genetic linkage map* represents the linear order and the pairwise distance of markers on a chromosome (the latter usually expressed in centimorgans, abbreviated as cM). A set of markers for which no recombination is detected is called a *bin*. Each pair of markers in the same bin have their relative pairwise orders undetermined. In the rest of the paper, we will assume that a genetic map is composed of a sequence of bins (of markers) and the distances between them. We also assume that the genetic distance when expressed in cMs is additive, i.e., the distance between bin A and bin C is the sum of the distance between bin A and bin B and the distance between bin B and bin C if the bins are ordered as $[A\ B\ C]$.

We use square brackets to delimit a genetic map, and we will use round parentheses to enclose a bin. For example, map $\Pi = [(m_1)\ 2\ (m_2, m_3)\ 2\ (m_4)\ 1\ (m_5, m_6)]$ consists of four bins, where the first and the third bin are both singletons (i.e., contain only one marker). Marker m_1 precedes both markers m_2 and m_3, which are followed by marker m_4. The relative order between m_2 and m_3 is undetermined. The distance between the first bin and the second bin is 2 cM. Sometimes, the distances can be omitted if one is concerned only with the order of the markers, e.g., $[(m_1)\ (m_2, m_3)\ (m_4)\ (m_5, m_6)]$.

We reserve the symbol Π to denote a genetic linkage map, and M_Π to denote the set of markers included in Π. Given a set of maps $\Omega = \{\Pi_1, \Pi_2, \ldots, \Pi_K\}$, we define M_Ω to be the *universe* of all the markers, i.e., $M_\Omega = \cup_{i=1}^{K} M_{\Pi_i}$. Given a map Π, we define $G_\Pi = (M_\Pi, E_\Pi)$ to be the directed weighted graph *induced* by Π, where the set of edges E_Π is defined as $E_\Pi = \{(m_i, m_j) |\ m_i$ is in the bin immediately preceding the bin containing $m_j\}$ and the weight of an edge (m_i, m_j) is set to the genetic distance between the corresponding bins. The notion of induced graph can easily be extended to a set of maps. We set $G_\Omega = (M_\Omega, E_\Omega)$ as the directed weighted graph induced by Ω, where $E_\Omega = \cup_{i=1}^{K} E_{\Pi_i}$. The weight of an edge in G_Ω is set to be the average of the weights of the corresponding edges in the original maps.

We use m_i to refer to a generic marker, and m_i^j to refer to the occurrence of marker m_i in map Π_j. We further define N_Ω to be the set containing all the marker occurrences. If we select a set $R \subseteq N_\Omega$, a *submap* $\Pi(R)$ of Π with respect to R is obtained by deleting the occurrences of all markers not in R from map Π. The set of submaps $\Omega(R)$ for the original set of maps Ω restricted to R is defined as $\Omega(R) = \{\Pi_i(R) | \Pi_i \in \Omega\}$.

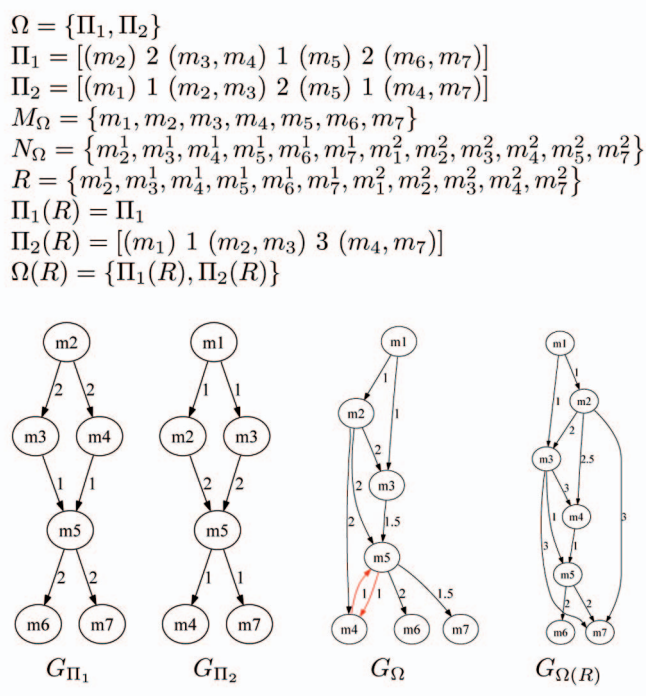


Fig. 2. Two simple genetic linkage maps, along with the corresponding notations used in this paper. Maps Π_1 and Π_2 both consist of four bins (enclosed in parentheses). The numbers in between adjacent bins indicate the distances between them. Maps Π_1 and Π_2 are not consistent with each other because there is a cycle in G_Ω between m_4 and m_5. Removing m_5 from Π_2 resolves the conflict.

Fig. 2 illustrates the notations Π, Ω, G_Π, G_Ω, M_Ω, N_Ω, $\Pi(R)$, and Ω_R for a small example.

2.2 Determining the Orientation of Individual Maps

When a map for a specific linkage group (i.e., a chromosome) is constructed from genotyping data, its canonical orientation (i.e., long arm of the chromosome on top) is usually unknown. The purpose of this step is to reverse the orientations of a subset of the input maps in Ω so that overall the resulting maps will be in a consistent orientation.

We employ Kendall's τ statistic to determine whether two maps are in a consistent orientation or not. Given two maps Π_i and Π_j, Kendall's τ statistic is defined as $\tau(\Pi_i, \Pi_j) = 2\frac{\#\text{ concordant marker pairs}}{\text{total }\#\text{ of marker pairs}} - 1$, where a marker pair is said to be *concordant* if they are in the same order in both maps. When computing $\tau(\Pi_i, \Pi_j)$, we restrict our attention to the common markers in $M_{\Pi_i} \cap M_{\Pi_j}$. If $\tau > 0$, then the two maps are in a consistent orientation, $\tau < 0$ otherwise.

In order to determine the consistent orientation, we first construct an undirected graph $H = (\Omega, E)$ as follows: Each individual map in Ω is represented by a vertex in H. Two vertices are connected by an edge if the corresponding maps share at least t markers.[1] An edge is colored red if τ for the corresponding pair is negative; otherwise, it is colored black. Without loss of generality, we can assume

1. The choice for t depends on the quality and the size of the maps. According to our experiments, when the number of common markers shared by two maps exceeds 10, the τ statistic is very reliable.

that H is connected; otherwise, we can solve each connected component of H. The problem of determining the set of maps to be reversed is equivalent to the problem of identifying a subset S of Ω that satisfies the following two conditions: 1) for every red edge, exactly one of the end vertices is in S and 2) for every black edge, either both of the end vertices are in S or none of them is in S. This problem can be solved with a relatively simple BFS-based algorithm as shown in Algorithm 1 (Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB. 2010.35).

2.3 Resolving Ordering Conflicts

Let $\Omega = \{\Pi_1, \Pi_2, \ldots, \Pi_K\}$ be the set of input maps in a consistent orientation. The problem of merging maps $\Pi_1, \Pi_2, \ldots, \Pi_K$ into a consensus DAG is straightforward when there are no conflicts. If some of the markers in the input maps have conflicting orders, then the induced graph G_Ω will not be acyclic. In order to resolve cycles, we propose to delete the smallest set of marker occurrences that can make G_Ω acyclic.

In order to capture the confidence associated with specific genotyping calls, we allow weight to be assigned to each individual marker occurrences. In practice, we assign weights to individual maps to represent their quality (i.e., high weight is associated with high quality/confidence). Once the weights are assigned, the computational problem is to delete the minimum-weight set of marker occurrences so that the resulting consensus map is acyclic. Formally, the optimization problem that emerges from this strategy is the following.

Minimum-weight feedback marker occurrence set (MWFMOS).

- **Input:** Ω and w, where Ω is a set of individual maps from which one would like to build a consensus map, and w is the associated weight function on N_Ω where $w(m_i^j)$ is the weight of marker occurrence m_i^j. Without loss of generality, we assume that $w(m_i^j) > 1$ for all $m_i^j \in N_\Omega$.
- **Objective:** Identify a set $\mathcal{D}$ of minimum total weight so that the subproblem restricted to $N_\Omega - \mathcal{D}$ is conflict-free (i.e., the graph induced by the subproblem, $G_{\Omega(N_\Omega - D)}$, is acyclic).

It is relatively easy to prove that MWFMOS is NP-complete when the number of maps is unbounded. The proof is a simple reduction from the minimum feedback edge set problem. We still do not know whether MWFMOS is still NP-complete when the number of maps is bounded by a constant, but we suspect it is.

The solution to the MWFMOS problem with input (Ω, w) can be obtained by solving MWFMOS for the nonoverlapping subproblems corresponding to the strongly connected components in G_Ω. For example, if we have $\Omega = \{[(m_1)\ (m_2)\ (m_3)\ (m_4)], [(m_2)\ (m_1)\ (m_4)\ (m_3)]\}$, there are two strongly connected components in G_Ω. The corresponding subproblems are $\Omega_1 = \{[(m_1)\ (m_2)], [(m_2)\ (m_1)]\}$ and $\Omega_2 = \{[(m_3)\ (m_4)], [(m_4)\ (m_3)]\}$. The optimal solution to the original problem is simply the concatenation of the optimal solutions to the subproblems. In the following, we will be focusing on solving MWFMOS for one of the connected components of G_Ω.

The algorithm that we propose requires to 1) express the problem as an Integer Linear Program (ILP), 2) relax the ILP to a Linear Program (LP) and solve it, and 3) use randomized rounding to convert the LP solutions to integral solutions. In practice, it turns out that the linear program contains too many variables (a high-order polynomial in the size of the input) to be easily tractable, so we propose to solve it with linear relaxation and rounding.

2.3.1 An LP-Based Algorithm

Let $\mathcal{I} = \{F_1, F_2, \ldots, F_K\}$ be a subproblem of Ω corresponding to a strongly connected component in G_Ω. A submap F_i is hereafter called a *fragment* since it is a contiguous piece of an individual map from Ω. Each fragment F_i has the same format as Π_i. Throughout this paper, we use Ω to denote the original problem, and $\mathcal{I}$ to denote a subproblem of Ω.

A conflict in $\mathcal{I}$ is characterized by a path $m_{i_1}^{j_1} \rightarrow m_{i_2}^{j_1}$, $m_{i_2}^{j_2} \rightarrow m_{i_3}^{j_2}, \ldots, m_{i_k}^{j_n} \rightarrow m_{i_1}^{j_n}$ (not to confused with a path in $G_\mathcal{I}$), wherein $m_{i_1}^{j_1} \rightarrow m_{i_2}^{j_1}$ indicates that marker m_{i_1} precedes marker m_{i_2} in fragment F_{j_1} (markers m_{i_1} and m_{i_2} do not have to be in adjacent bins). Observe that the path starts and ends with the same marker in two different fragments. Let P be the set of all such paths.

Given an instance of P, we formulate MWFMOS as an ILP as follows:

$$\begin{aligned} \text{Min} \quad & \sum x_i^j w(m_i^j) \\ \text{S.T.} \quad & \sum_{m_i^j \in p} x_i^j \geq 1 \quad \forall p \in P \\ & x_i^j \in \{0,1\}, \end{aligned} \tag{1}$$

where x_i^j is the binary variable associated with the marker occurrence m_i^j which is set to 1 if m_i^j needs to be deleted, 0 otherwise. The LP relaxation of the above ILP is straightforward.

The number of constraints of the LP relaxation of (1) is $|P|$, which is at most $O(K!|M_\mathcal{I}|^K)$, where K is the number of fragments in $\mathcal{I}$ and $|M_\mathcal{I}|$ is the total number of distinct markers in $\mathcal{I}$. The number of constraints is polynomial when the size of the input K is constant. The dual for the LP relaxation of (1) is the following program:

$$\begin{aligned} \text{Max} \quad & \sum y_p \\ \text{S.T.} \quad & \sum_{p \ni m_i^j} y_p \leq w(m_i^j) \quad \forall m_i^j \in N_\mathcal{I}, \\ & y_p \geq 0 \qquad \forall p \in P, \end{aligned} \tag{2}$$

where y_p is the associated variable with path $p \in P$, and $N_\mathcal{I}$ is the set containing all the marker occurrences in $\mathcal{I}$. The following LP is equivalent to (2):

$$\begin{aligned} \text{Min} \quad & \lambda \\ \text{S.T.} \quad & \sum_{p \ni m_i^j} y_p \leq \lambda w(m_i^j) \quad \forall m_i^j \in N_\mathcal{I}, \\ & \sum_{p \in P} y_p = 1, \\ & y_p \geq 0 \qquad \forall p \in P. \end{aligned} \tag{3}$$

The optimal solution to (3) is the reciprocal of the solution of (2). To simplify the notation, we can rewrite (3) in the matrix representation:

$$\begin{array}{ll} \text{Min} & \lambda \\ \text{S.T.} & A\vec{y} \le \lambda\vec{w} \\ & \vec{y} = 1 \text{ and } \vec{y} \ge 0. \end{array} \quad (4)$$

Each row of A corresponds to a marker occurrence in $N_{\mathcal{I}}$ and each column of A refers to a path in P. We have $A[r,c] = 1$ if and only if $m_i^j \in N_{\mathcal{I}}$ corresponding to the rth row of A is on the path corresponding to the cth column of A. With $\vec{y} = 1$, we mean $\sum_{p\in P} y_p = 1$.

Due to the large number of variables, solving optimally (4) can be very time-consuming. In the following, we show how to achieve an $(1+\epsilon)$ optimal (or simply ϵ optimal) solution.[2] To find such an approximate solution, we follow the method proposed by Plotkin et al. [20]. Let $\vec{z}$ be the dual variables associated with (4), and let us define $C(\vec{z}) = \min_{\vec{y}|\vec{y}=1} \vec{z}^t A\vec{y}$.

Consider an error parameter $0 < \epsilon < 1/6$, a feasible primal solution $(\vec{y}, \lambda)$, and a feasible dual solution $\vec{z}$. Then, λ is 6ϵ optimal if the following two relaxed optimality conditions are met:

$$(1-\epsilon)\lambda\vec{z}^t\vec{w} \le \vec{z}^t A\vec{y}, \quad (5)$$

$$\vec{z}^t A\vec{y} - C(\vec{z}) \le \epsilon(\vec{z}^t A\vec{y} + \lambda\vec{z}^t\vec{w}). \quad (6)$$

A sketch of the algorithm to find a 6ϵ optimal solution is presented as Algorithm APPROXSOLVE. Algorithm APPROXSOLVE converges within $O(\frac{1}{\epsilon^3\lambda^*}\log(|N_{\mathcal{I}}|\epsilon^{-1}))$ iterations, where λ^* is the optimal solution to (4). The performance guarantee of our algorithm APPROXSOLVE is formally presented as Theorem 1, and the time complexity analysis is presented as Theorem 2.

Algorithm 1. APPROXSOLVE$(\vec{y_0}, \epsilon)$

1: $\vec{y} \leftarrow \vec{y_0}$; $\lambda \leftarrow \max_r \vec{a_r}^t\vec{y}/w_r$; $\alpha \leftarrow 4\ln(2|N_{\mathcal{I}}|\epsilon^{-1})/(\lambda\epsilon)$; $\sigma \leftarrow \epsilon/(4\alpha)$;
{*$\vec{a_r}$ is the transpose of the r^{th} row vector of matrix A. $|N_{\mathcal{I}}|$ is the number of rows in A*}
2: **for** $r = 1, \ldots, |N_{\mathcal{I}}|$ **do**
3: $z_r \leftarrow e^{\alpha\vec{a_r}^t\vec{y}/w_r}/w_r$
4: **while** $(\vec{y}, \lambda, \vec{z})$ does not satisfy (6) **do**
5: $\widetilde{y} \leftarrow \text{argmin}_{\vec{y}|\vec{y}=1}\vec{z}^t A\vec{y}$
6: $\vec{y} \leftarrow (1-\sigma)\vec{y} + \sigma\widetilde{y}$
7: **if** $\max_r \vec{a_r}^t\vec{y}/w_r \le \lambda/2$ **then**
8: $\lambda \leftarrow \max_r \vec{a_r}^t\vec{y}/w_r$; $\alpha \leftarrow 4\ln(2|N_{\mathcal{I}}|\epsilon^{-1})/(\lambda\epsilon)$; $\sigma \leftarrow \epsilon/(4\alpha)$;
9: **for** $r = 1, \ldots, |N_{\mathcal{I}}|$ **do**
10: $z_r \leftarrow e^{\alpha\vec{a_r}^t\vec{y}/w_r}/w_r$
11: $\lambda \leftarrow \max_r \vec{a_r}^t\vec{y}/w_r$;
12: **return** $\vec{y}, \lambda, \vec{z}$

Lemma 1. *Let $(\vec{y}, \lambda)$ and $\vec{z}$ be feasible primal and dual solutions that satisfy both condition (5) and (6). Then, $(\vec{y}, \lambda)$ is a $(1+6\epsilon)$ optimal solution.*

Proof. This lemma corresponds to [20, Lemma 2.1]. To be self-contained, we present the proof here.

From (5) and (6), we have $C(\vec{z}) \ge (1-\epsilon)\vec{z}^t A\vec{y} - \epsilon\lambda\vec{z}^t\vec{w} \ge (1-\epsilon)^2\lambda\vec{z}^t\vec{w} - \epsilon\lambda\vec{z}^t\vec{w} \ge (1-3\epsilon)\lambda\vec{z}^t\vec{w}$. Hence, $\lambda \le (1-3\epsilon)^{-1}C(\vec{z})/\vec{z}^t\vec{w} \le (1-3\epsilon)^{-1}\lambda^* \le (1+6\epsilon)\lambda^*$. □

Theorem 1. *Algorithm APPROXSOLVE returns a $(1+6\epsilon)$ optimal solution to (4).*

Proof. The theorem follows from [20, Lemma 2.2]. To be self-contained, we present the proof here.

According to Lemma 1, in order to prove Theorem 1, we only have to show that condition (5) and (6) are both satisfied when Algorithm APPROXSOLVE stops. Since condition (6) is ensured by the while loop at line 4, we only have to show that (5) is satisfied when the algorithm stops.

We first show that when $\alpha \ge \alpha_0 = \frac{2\ln(2|N_{\mathcal{I}}|\epsilon^{-1})}{\lambda\epsilon}$, $\vec{z}$ as assigned by the "for" loops at lines 3 and 12 in algorithm APPROXSOLVE will satisfy condition (5).

Let $I = \{i : (1-\epsilon/2)\lambda w_i \ge \vec{a_i}^t\vec{y}\}$. Let $j \in I$. We have

$$\begin{aligned} \lambda z_j w_j &= \lambda e^{\alpha\vec{a_j}^t\vec{y}/w_j} \le \lambda e^{\alpha(1-\epsilon/2)\lambda} = \lambda e^{\alpha\lambda}e^{-\alpha\epsilon\lambda/2} \\ &\le \lambda e^{\alpha\lambda}e^{-ln(2|N_{\mathcal{I}}|\epsilon^{-1})} \le \frac{\epsilon}{2|N_{\mathcal{I}}|}\lambda e^{\alpha\lambda} \le \frac{\epsilon}{2|N_{\mathcal{I}}|}[\lambda\vec{z}^t\vec{w}]. \end{aligned}$$

Consequently,

$$\begin{aligned} \lambda\vec{z}^t\vec{w} &= \sum_{i\in I}\lambda z_i w_i + \sum_{i\notin I}\lambda z_i w_i \le \sum_{i\in I}\lambda z_i w_i + \sum_{i\notin I}\frac{1}{1-\epsilon/2} z_i\vec{a_i}^t\vec{y} \\ &\le \sum_{i\in I}\lambda z_i w_i + \frac{1}{1-\epsilon/2}\vec{z}^t A\vec{y} \le \frac{\epsilon}{2}\lambda\vec{z}^t\vec{w} + \frac{1}{1-\epsilon/2}\vec{z}^t A\vec{y}. \end{aligned}$$

Therefore, we have $(1-\epsilon)\lambda\vec{z}^t\vec{w} \le \vec{z}^t A\vec{y}$.

Note that α is initialized to be $2\alpha_0$ and whenever $\max_r \vec{a_r}^t\vec{y}/w_r \le \lambda/2$, α gets recomputed. Therefore, condition (5) is satisfied throughout the execution of Algorithm APPROXSOLVE. □

Lemma 2. *Let $(\vec{y_1}, \lambda)$ and $\vec{z_1}$, where $\vec{z_1} = \{\frac{1}{w_r}e^{\alpha\vec{a_r}^t\vec{y_1}/w_r}\}_{r=1}^{|N_{\mathcal{I}}|}$, be primal and dual solutions that do not satisfy (6). Let $(\vec{y_2}, \lambda)$ and $\vec{z_2}$ be the solutions in the next iteration, i.e., $\vec{y_2} = (1-\delta)\vec{y_1} + \delta\widetilde{y}$, and let α, δ, and ϵ be defined in Algorithm APPROXSOLVE. Let $\Phi_1 = \vec{z_1}^t\vec{w}$, $\Phi_2 = \vec{z_2}^t\vec{w}$. Then, $\Phi_1 - \Phi_2 > \lambda\epsilon^2\Phi_1/4$.*

Proof. We have

$$\begin{aligned} \Phi_2 = \vec{z_2}^t\vec{w} &= \sum_i e^{\alpha\vec{a_i}^t\vec{y_2}/w_i} = \sum_i e^{\alpha\vec{a_i}^t((1-\delta)\vec{y_1}+\delta\widetilde{y})/w_i} \\ &= \sum_i e^{\alpha\vec{a_i}^t\vec{y_1}/w_i}e^{\alpha\delta\vec{a_i}^t(\widetilde{y}-\vec{y_1})/w_i}. \end{aligned}$$

Since $w_i > 1$, $\vec{y_1} = 1$ and $\widetilde{y} = 1$, it follows that $|\vec{a_i}^t(\widetilde{y} - \vec{y_1})/w_i| < 1$. Since $\delta = \frac{\epsilon}{4\alpha}$, it follows that $|\alpha\delta\vec{a_i}^t(\widetilde{y} - \vec{y_1})/w_i| < \epsilon/4 < 1/4$. According to Taylor's expansion, $e^x < 1 + x + 2x^2$ for $|x| < 1/4$. By plugging in $x = \alpha\delta\vec{a_i}^t(\widetilde{y} - \vec{y_1})/w_i$, we get

$$\begin{aligned} e^{\alpha\delta\vec{a_i}^t(\widetilde{y}-\vec{y_1})/w_i} &< 1 + (\alpha\delta\vec{a_i}^t(\widetilde{y} - \vec{y_1})/w_i) + 2(\alpha\delta\vec{a_i}^t(\widetilde{y} - \vec{y_1})/w_i)^2 \\ &< 1 + (\alpha\delta\vec{a_i}^t(\widetilde{y} - \vec{y_1})/w_i) + \frac{\epsilon}{2}(\alpha\delta\vec{a_i}^t(\widetilde{y} + \vec{y_1})/w_i). \end{aligned}$$

Therefore, $\Phi_2 = \sum_i e^{\alpha\vec{a_i}^t\vec{y_1}/w_i}e^{\alpha\delta\vec{a_i}^t(\widetilde{y}-\vec{y_1})/w_i} < \sum_i e^{\alpha\vec{a_i}^t\vec{y_1}/w_i} + \alpha\delta(C(\vec{z_1}) - \vec{z_1}A\vec{y_1}) + \frac{\epsilon}{2}\alpha\delta(C(\vec{z_1}) + \vec{z_1}A\vec{y_1})$. Consequently, $\Phi_2 < \Phi_1 + \alpha\delta(C(\vec{z_1}) - \vec{z_1}A\vec{y_1}) + \frac{\epsilon}{2}\alpha\delta(C(\vec{z_1}) + \vec{z_1}A\vec{y_1})$. It follows that $\Phi_1 - \Phi_2 > \alpha\delta(\vec{z_1}A\vec{y_1} - C(\vec{z_1})) - \epsilon\alpha\delta\vec{z_1}A\vec{y_1}$. Due to the fact that $(\vec{y_1}, \lambda)$ and $\vec{z_1}$ do not satisfy (6), we

2. A solution λ is said to be $(1+\epsilon)$ optimal if $\lambda < (1+\epsilon)\lambda_{\text{opt}}$, where λ_{opt} is the optimal solution. A $(1+\epsilon)$ optimal solution is sometimes simply called an ϵ optimal solution.

have $\Phi_1 - \Phi_2 > \lambda\epsilon\alpha\delta\vec{z}_1{}^t\vec{w}$. According to the choice of δ, we have $\Phi_1 - \Phi_2 > \lambda\epsilon^2\Phi_1/4$. □

Theorem 2. *Algorithm APPROXSOLVE converges in $O(\frac{1}{\epsilon^3\lambda^*}\log(|N_{\mathcal{I}}|\epsilon^{-1}))$ iterations, where λ^* is the optimal solution to (4).*

Proof. Note that during the execution of Algorithm APPROXSOLVE, λ is a monotonically decreasing sequence with $\lambda_i > 2\lambda_{i+1}$. Let the sequence of λ be $\lambda_0, \lambda_1, \lambda_2, \ldots, \lambda_n$, where $\lambda_n > \lambda^*$ is the final output. When $\lambda = \lambda_k$, then $e^{\alpha\lambda_k/2} \leq \Phi \leq |N_{\mathcal{I}}|e^{\alpha\lambda_k}$.

Due to Lemma 2, it takes at most $O(\frac{1}{\epsilon^3\lambda_k}\log(|N_{\mathcal{I}}|\epsilon^{-1}))$ iterations to cut λ from λ_k to λ_{k+1}. Since $\lambda_i > 2\lambda_{i+1}$, the overall time complexity is determined by the last step. Hence, the overall running time is $O(\frac{1}{\epsilon^3\lambda^*}\log(|N_{\mathcal{I}}|\epsilon^{-1}))$. □

Step 5 in algorithm APPROXSOLVE can be solved by running all-pairs shortest path algorithm, which takes time $O(|N_{\mathcal{I}}|^3 \log |N_{\mathcal{I}}|)$. The vector $\vec{y}$ does not have to be stored in memory explicitly since all we need is $A\vec{y}$ which takes space $O(|N_{\mathcal{I}}|)$. Combining the running time for each iteration with the upper bound on the number of iterations, the overall time complexity of APPROXSOLVE is $O(\frac{1}{\epsilon^3\lambda^*}\log(|N_{\mathcal{I}}|\epsilon^{-1})|N_{\mathcal{I}}|^3 \log |N_{\mathcal{I}}|)$. Note that the time complexity does not depend on $|P|$.

Given the near-optimal solution $\vec{z}$ to the dual of (4), the near-optimal solution to the LP relaxation of (1) is $\vec{x} \leftarrow \vec{z}/C(\vec{z})$. In our algorithm, we apply two types of rounding to convert the fractional solution $\vec{x}$ to an integral solution, and then, choose the best. The first method is randomized. The randomized rounding algorithm progressively deletes marker occurrences until all the conflicts are resolved. In each step, the method samples a marker to be deleted according to a probability distribution proportional to $\vec{x}$. The solution obtained is further reduced to a minimal solution by removing redundant marker occurrences. The second rounding method employs a greedy strategy. The markers occurrences in $N_{\mathcal{I}}$ are sorted into descending order according to their associated probabilities, then we delete just enough marker occurrences to resolve all the conflicts. Again, the solution is further reduced to a minimal solution by removing redundancies. The pseudocode for the rounding step is presented as Algorithm 2 (Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2010.35).

As suggested by one of the anonymous reviewers, it may be possible to find an exact solution to the ILP (1) by designing a combinatorial algorithm that solves the separation problem on the path inequalities. While we have not explored this approach yet, it has the potential to lead to a comparatively fast method that finds truly exact solutions.

2.3.2 A Speedup Heuristic

Our LP-based algorithm works well when either the size of the subproblem is small (i.e., $|N_{\mathcal{I}}|$ is small) or the number of markers to be deleted is small (i.e., $1/\lambda*$ is small), the latter of which is usually the case in practice. However, if both $|N_{\mathcal{I}}|$ and $1/\lambda*$ are large, the LP-based algorithm can be still too slow. In this case, we advise to employ an heuristic algorithm which breaks a large subproblem $\mathcal{I}$ into even smaller sub-subproblems.

Our heuristic algorithm uses the notion of node betweenness [21]. Recall that the *betweenness centrality* of a node in a graph is equal to the number of shortest paths that go through it. The intuition is that nodes with high betweenness usually correspond to hubs, and their deletion will likely break the graph into disconnected components.

Now, let $m_i^{j_1}$ and $m_i^{j_2}$ be a pair of occurrences of the same marker in two individual maps. A path between $m_i^{j_1}$ and $m_i^{j_2}$ is the *shortest* if it traverses the smallest number of marker occurrences. Let Q be the set of all such pairwise shortest paths. If there are multiple shortest paths between a pair, we arbitrarily choose one to be included in Q. Observe that Q is a subset of P in the ILP (1).

We define the *weighted betweenness centrality* of a marker occurrence m_i^j as the number of shortest paths in Q that go through node m_i^j divided by its weight $w(m_i^j)$. The higher the weighted betweenness centrality for a marker occurrence, the higher is the likelihood that marker occurrence should be deleted. Our heuristic algorithm works by computing the weighted betweenness centrality for every marker occurrences, and then, iteratively deleting the ones with the highest value. The step is repeated until the sizes of the subproblems are all small enough to be handled by our LP-based algorithm. The pseudocode of our heuristic algorithm is presented as Algorithm 3 (Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2010.35).

2.4 Simplifying the Consensus Map

Having resolved the conflicts in Ω, the graph G_Ω is now acyclic. As it turns out, in practice, however, the graph G_Ω is overly complex to be useful for geneticists. In this phase, we propose two effective simplification steps. The first one addresses the problem of reducing the number of nodes, whereas the second focuses on decreasing the number of edges.

Recall that a bin represents a set of markers for which the relative orders are undetermined. In this first step, we aim to simplify G_Ω by condensing markers into bins. In order to differentiate the bins constructed in this step from the bins in the original maps, we refer to the former ones as *supermarkers*. The rationale for combining markers into supermarkers is the following. If two markers always appear paired in the same bin in the original individual maps, then we cannot determine the relative order between them and the two markers should be grouped as a single supermarker. Based on this observation, we generalize the notion of cosegregating markers as follows: Given a set of maps $\Omega = \{\Pi_1, \Pi_2, \ldots, \Pi_K\}$, two markers (m_i, m_j) are said to be *cosegregating* (denoted as $m_i \sim m_j$) if they satisfy the following two conditions: 1) m_i and m_j belong to the same bin in at least one of the maps in Ω and 2) there is no path from m_i to m_j or from m_j to m_i in G_Ω. The first condition is intended to ensure that the markers to be condensed into a supermarker are indeed close. The second condition makes sure that the relative order between the markers to be condensed into a supermarker is undetermined.

The cosegregation relation is not an equivalent relation, because it does not satisfy the transitivity property. Consider, for example, $\Omega = \{[(m_1, m_2, m_3)], [(m_1)\ (m_4)],\ [(m_4)\ (m_3)]\}$, then $m_1 \sim m_2$ and $m_2 \sim m_3$, but $m_1 \not\sim m_3$ since there is a path from m_1 to m_3 in G_Ω. When we group markers into supermarkers, we must be careful not to introduce new conflicts. For example, consider $\Omega = \{[(m_1, m_2)], [(m_1)\ (m_3)], [(m_3, m_4)], [(m_4)\ (m_2)]\}$, then (m_1, m_2) and (m_3, m_4) are both cosegregating pairs. But if they are both condensed into supermarkers, a new conflict will result.

In order to address these issues, we employ a greedy iterative algorithm to carry out a maximal decomposition of the markers into supermarkers. In each step, we condense one pair of cosegregating markers into a supermarker. The original problem Ω is transformed into a new problem Ω', which has one less marker than Ω. For example, if we have $\Omega = \{[(m_1, m_2, m_3)], [(m_1)\ (m_3)]\}$, after condensing the cosegregating pair (m_1, m_2) into supermarker m_s, the original problem becomes $\Omega' = \{[(m_s, m_3)], [(m_s)\ (m_3)]\}$. We keep repeating the iterative process until no cosegregating markers can be found. The final set of maps at the end of the process is denoted by Ω^f (G_{Ω^f} is the corresponding induced DAG).

In the second simplification step, we concentrate on reducing the number of edges from G_{Ω^f}. We define a directed edge (m_i, m_j) to be *redundant* if there exists an alternative (distinct) path from m_i to m_j in G_Ω. The removal of redundant edges is a transitive reduction [22], which is commonly used to untangle a graph. We denote with DAG_Ω the final graph obtained by removing redundant edges.

Each vertex in DAG_Ω represents a supermarker in Ω^f, which, in turn, represents a set of markers from the original problem Ω. In Theorem 3, we prove that the in-degree and out-degree of the vertices in DAG_Ω are at most K, where K is the number of maps. The entire simplification process is summarized in Algorithm 4 (Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2010.35).

Fig. 1 illustrates a side-by-side comparison between the original DAG and the corresponding simplified DAG for barley chromosome 5H. Observe that the original DAG is much more complex and tangled than the simplified DAG. The major benefit of this last step is that the simplified DAG can be used directly by geneticists and represents a viable alternative to the classical linear order representation for genetic maps.

Theorem 3. *The in-degree and out-degree of the vertices in DAG_Ω are at most K, where K is the number of maps.*

Proof. Let $\Pi^f \in \Omega^f$ be one of the final individual maps. Let M_{Π^f} be the set of supermarkers contained in Π^f. Consider any two supermarkers m_i and m_j from M_{Π^f}. If m_i and m_j belong to different bins in Π^f, then m_i and m_j are ordered (meaning that either m_i is before m_j or vice versa). On the other hand, if m_i and m_j belong to the same bin, since m_i and m_j do not form a cosegregating pair (due to the greediness of our algorithm), there must be a path from either m_i to m_j or from m_j to m_i in DAG_Ω. Therefore, if we restrict our attention to a single map $\Pi^f \in \Omega^f$, DAG_Ω defines a total order on the set of supermarkers M_{Π^f}. As a result, each supermarker can have at most one immediate predecessor and one immediate successor from one individual map. Since each supermarker can appear in at most K maps, the theorem follows. □

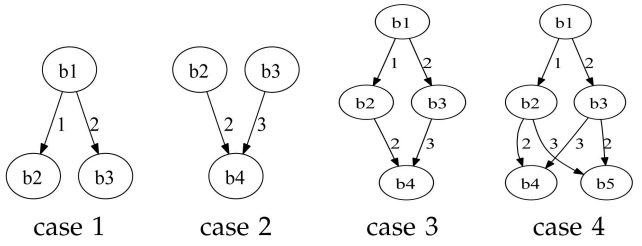


Fig. 3. A few cases to consider when estimating the distance between b2 and b3.

2.5 Linearizing the Consensus Map

In the fourth and last step of the workflow, we process DAG_Ω to produce a linear order of the bins (supermarkers). The objective is to compute the linear order that is consistent with the partial order of the bins, i.e., if there is a path from bin b_i to bin b_j in DAG_Ω, then b_i should precede b_j in the linear order. This problem is similar to topological sorting [23], but the genetic distances need to be taken into account as well. Also, when there is no path between a pair of bins, we have to impute the order of the two bins as well as the distance between them.

The main idea in our linearization procedure can be explained on the examples in Fig. 3. First, consider case 1, where there is no path between b_2 and b_3, but they share a common ancestor. The distances from b_2 and b_3 to the common ancestor are 1 and 2, respectively. In this case, it is reasonable to infer that b_3 follows b_2 and the distance between them is 1. For the same reason, in case 2, it is reasonable to infer that the linear order is $[b_3, b_2, b_4]$. In case 3, the situation is more complex. If we order b_2 and b_3 by only relying on their distances from b_1, then b_3 should be after b_2. However, if we order b_2 and b_3 based on their distances to b_4, then b_3 should be ordered before b_2. To resolve this problem, we order b_2 and b_3 based on the information from the pair b_1 and b_4. The average distance from b_1 and b_4 over the two paths is 4. Bin b_2 lies between b_1 and b_4 and is $1/3$ away from b_1. Similarly, b_3 is $2/5$ away from b_1. Based on that information, we order b_3 after b_2 and set the distance from b_3 to b_2 to be $(2/5 - 1/3)4 = 4/15$. In case 4, the situation is even more complicated. When estimating the order and distance between b_2 and b_3, we need to aggregate the distance information from all common ancestor and successor pairs.

This strategy can be formalized as follows: Let us define $D[b_i, b_j]$ to be the distance from bin b_i to bin b_j in DAG_Ω. If there is only one path from b_i to b_j, then $D[b_i, b_j]$ is trivially assigned the length of that path. If there are multiple paths from b_i to b_j, we set $D[b_i, b_j]$ to be the average length of all paths from b_i to b_j, which can be efficiently computed by dynamic programming as shown in Algorithm 5 (Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2010.35). Now, let b_i and b_j be two bins that are not ordered in DAG_Ω. Our algorithm

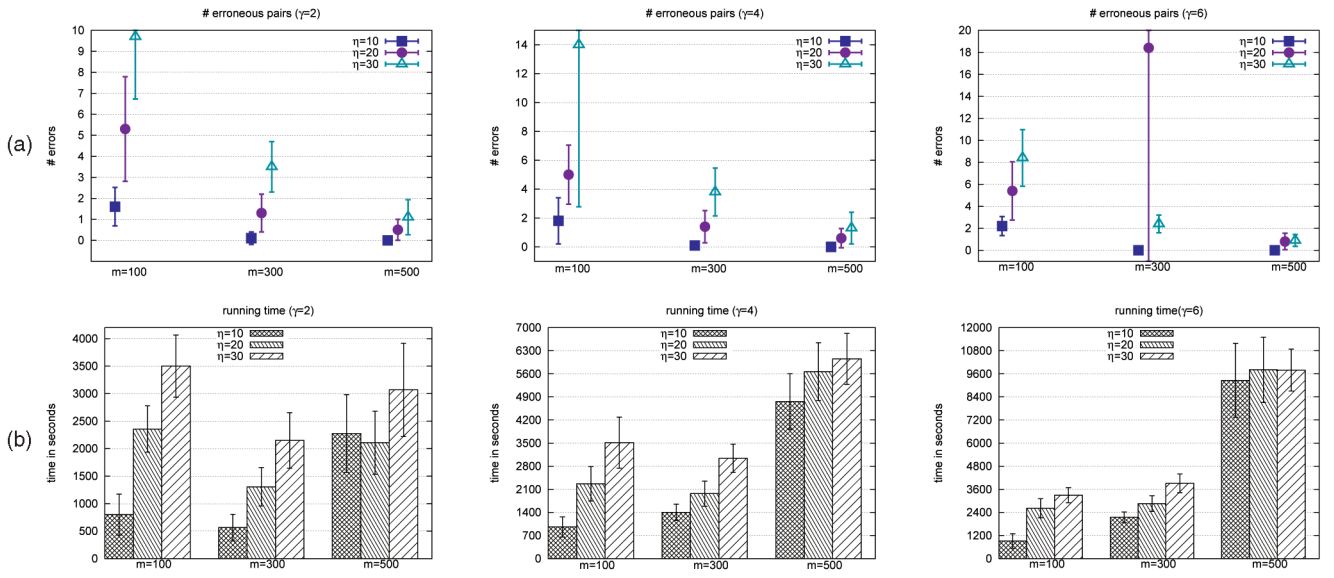


Fig. 4. (a) The number of erroneous marker pairs obtained with MERGEMAP and (b) the average running time for various choices of m, η, and γ. The parameter m is the number of markers in each individual map. The η swaps model local reshuffles, while the γ relocations model global displacements. Each point in the figure is an average of the results obtained from 10 independent data sets. Standard deviation for the corresponding statistic is represented as the error bars.

determines the relative order between b_i and b_j as follows: Consider these cases.

- Both b_i and b_j have common ancestors and common successors. Let A be the set of common ancestors and S be the set of common successors. Let $p \in A$ be one of the ancestors and $s \in S$ be one of the successors. We define the distance from bin b_i to bin b_j with respect to the common ancestor and successor pair (p, s) as $\Delta^{(p,s)}[b_i, b_j] = D[p, s](\frac{D[p,b_j]}{D[p,b_j]+D[b_j,s]} - \frac{D[p,b_i]}{D[p,b_i]+D[b_i,s]})$. The final distance $\Delta[b_i, b_j]$ is averaged over all (p, s) pairs, i.e., $\Delta[b_i, b_j] = \sum_{p\in A, s\in S} \Delta^{(p,s)}[b_i, b_j]/(|A|\,|S|)$. If $\Delta[b_i, b_j]$ is positive, then we order b_i before b_j. Otherwise, we order b_j before b_i.
- Both b_i and b_j have only common successors. Let S be the set of successors and let $s \in S$ be one of the successor. The distance from bin b_i to bin b_j with respect to s is defined as $\Delta^s[b_i, b_j] = D[b_i, s] - D[b_j, s]$. The final distance $\Delta[b_i, b_j]$ is again averaged over all successors, i.e.,

$$\Delta[b_i, b_j] = \frac{\sum_{s\in S} \Delta^s[b_i, b_j]}{|S|}.$$

- Both b_i and b_j have only common ancestors. $D[b_i, b_j]$ is similarly computed as in the previous case.

Once the distances in D are computed, the algorithm that linearizes DAG_Ω is similar to topological sorting [23]. Let T be the list of ordered bins ($T = \emptyset$ initially). At each iteration, our algorithm determines the next marker s to be ordered. If s is uniquely determined under the partial order given by DAG_Ω, then we simply append s to the end of T. Otherwise, if S is the set of multiple choices, s is chosen so that $\sum_{t\in S, t\neq s} \Delta_{s,t}$ is maximized. The details are presented in Algorithm 6 (Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2010.35).

3 Results and Discussions

We implemented our algorithmic workflow in C++ and carried out extensive evaluations on both real and synthetic data sets. Our software tool, called MERGEMAP, is available at http://www.cs.ucr.edu/~yonghui/mgmap.html.

3.1 Evaluating the Conflict Resolution Algorithm

The purpose of this first set of experiments is to assess the effectiveness and efficiency of our conflict resolution algorithm. Each data set in this experiment consists of six individual maps, all of which are noisy variants of one single *true* map. The *true* map is just a permutation of m markers, where the parameter m ranges from 100 to 500 (representing a spectrum of maps from medium size to large size). The distances between adjacent markers are fixed to be 1 cM. To generate an individual map from the true map, we first swap η randomly chosen adjacent pairs, and then, relocate γ randomly chosen markers to a random position. The η swaps model local reshuffles, while the γ relocations model global displacements. As said before, swaps and relocations are the two types of errors that may be present in a genetic map. In our experiments, η ranges from 10 to 30 and γ ranges from 2 to 6.

For each data set, a consensus map was constructed by MERGEMAP by running the conflict resolution module, followed by the simplification and the final linearization. The consensus map was compared with the true map and the number of erroneous marker pairs were counted. We called a pair of markers *erroneous* when their order in consensus map differs from the order in the true map. When the consensus map is identical to the true map, the number of erroneous marker pairs is zero. On the other hand, when the consensus map is the reverse of the true

map, the number of erroneous markers is equal to $m(m-1)/2$. For each choice of m, η, and γ, 10 independent random data sets were generated. For each data set, the number of erroneous marker pairs and the running time were collected. The mean and standard deviation for both performance measures were computed, and are summarized in Fig. 4.

As Fig. 4 illustrates, MERGEMAP is very accurate in detecting the problematic markers and removing them before merging the individual maps. In most cases, the number of erroneous marker pairs in the final map is less than 10, and in a few cases, the number of erroneous pairs is equal to zero. As η and γ increase, the problem becomes harder and the quality of the consensus map deteriorates. Vice versa, as m increases the number of erroneous pairs decreases.[3]

The running time of MERGEMAP increases as m or η or γ increases, but in most practical instances, it remains within reasonable bounds. For the largest data set with $m = 500$ markers, $\eta = 30$, and $\gamma = 6$, MERGEMAP finishes within 2-3 hours. In comparison, JOINMAP takes several **weeks** to assemble maps with about 300 markers.

3.2 Comparing with JOINMAP on Synthetic Genotyping Data

The objective of the second set of experiments is to evaluate the entire process of building consensus maps from "scratch" (i.e., starting from synthetic genotyping data). The synthetic genotyping data sets were generated according to a procedure which is controlled by six parameters. We attempted to model the genotyping process to be as realistic as possible. The parameters are the number K of mapping populations, the number m of markers, the number R of "bad markers" on each mapping populations, the genotyping error rate η, and the missing rate γ. The sixth parameter x controls the percentage of the markers shared by two individual maps. The latter is used to model the fact that the data for individual maps only represent a *subset* of the universe of genetic markers.

The entire procedure to generate a synthetic genotyping data set can be divided into four steps. In the first step, a "skeleton" map is produced with m markers. The markers on the skeleton map are spaced at a distance of 0.5 cM plus a random distance according to a Poisson process with a mean of 2 cM. The "skeleton" map serves the role of the *true* map.

Following the generation of the skeleton map, in the second step, the raw genotyping data for the K mapping populations are then generated sequentially. Here, we assume that the mapping populations are all of the double haploid (DH) type, and that each population consists of 100 individuals. The genotypes for the individuals are generated as follows: The genotype at the first marker is generated at random with probability 0.5 of being A and probability 0.5 of being B. The genotype at the next marker depends upon the genotype at the previous marker and the distance between them on the skeleton map. If the distance between the current marker and the previous marker is d cM, then with probability $d/100$, the genotype at the current locus will be the opposite of the one at the previous locus, and with probability $1-(d/100)$, the two genotypes will be the same. Finally, according to the specified error rate and missing rate, the genotype state is flipped to model the introduction of a genotyping error or is simply deleted to model a missing observation.

3. The only outlier in Fig. 4 is the case $m = 300$, $\eta = 20$, and $\gamma = 6$. We examined the raw data and found that the high mean and standard deviation are due to one single data set, for which our algorithm failed to place one single marker in the right place. This single bad marker contributed 172 erroneous marker pairs in total. When averaged over the 10 runs, the single bad marker contributed 17 to the average number of erroneous pairs.

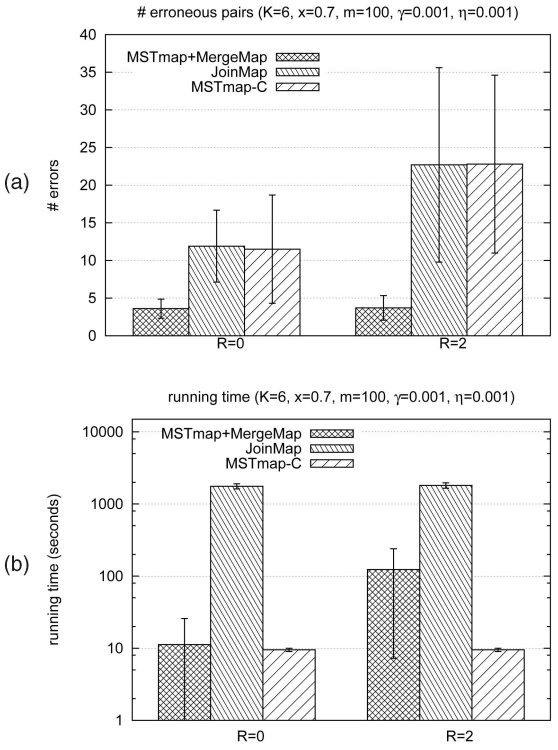


Fig. 5. Comparison between MSTMAP+MERGEMAP, JOINMAP, and MSTMAP-C in terms of (a) the number of erroneous marker pairs and (b) running time for $R = 0$ and $R = 2$, respectively. The rest of the parameters are as shown in the title of the figures. The parameter γ is the missing observation rate, η is the genotyping error rate, R is the number of "bad" markers, and m is the number of markers in each individual map. In these experiments, we have $K = 6$ mapping populations obtained by removing a fraction $x = 0.7$ of markers from each individual map. Each bar represents an average of 10 runs and the error bar indicates the standard deviation.

In the third step, "bad markers" are added to each mapping population. To do so, R markers are first selected at random from each population. The genotypes for those chosen markers across all the 100 individuals are flipped with probability 0.3. Due to the very high error rate introduced for these markers, their positions in the individual genetic maps will be unpredictable. We note that R is small relative to m, and therefore, the probability that two individual populations share a common bad marker is very small. When they do, we discard the entire data set and generate a new one.

The fourth step of generation procedure involves removing a fraction of markers from each individual map. A random subset of $(1-x)m$ markers is deleted from each mapping population, where x varies from 0.35 to 0.7 in our experiments. As a result, two mapping populations share x^2m markers on average.

For each data set, individual genetic maps were assembled by our tool MSTMAP [7], [8] with error correction disabled. The individual maps were then fed

TABLE 1
Comparison between MSTMAP+MERGEMAP and MSTMAP-C for $K = 6, x = 0.7$

	MSTMAP+MERGEMAP # erroneous pairs				MSTMAP-C # erroneous pairs			
$\gamma = \eta \rightarrow$	0.001	0.005	0.01	0.02	0.001	0.005	0.01	0.02
$R = 0$								
$m = 100$	3.6	10.0	16.3	17.9	11.5	15.1	18.0	21.0
$m = 300$	13.7	25.4	34.4	48.6	29.4	29.1	42.1	59.2
$m = 500$	20.9	43.0	56.0	86.9	42.2	56.2	74.3	99.3
$R = 2$								
$m = 100$	3.2	8.4	13.4	18.5	15.3	38.5	32.9	34.0
$m = 300$	11.0	27.6	37.2	55.8	36.9	45.3	48.7	64.9
$m = 500$	19.6	45.0	62.8	81.6	54.1	68.8	84.1	120.1
$R = 4$								
$m = 100$	3.3	12.0	10.6	16.4	24.4	32.1	37.0	44.1
$m = 300$	12.3	23.8	36.2	50.7	39.3	54.6	63.8	69.0
$m = 500$	18.4	46.8	61.2	76.8	59.0	75.2	89.2	120.9
$R = 6$								
$m = 100$	4.1	8.2	10.2	17.7	25.8	24.4	36.4	49.4
$m = 300$	9.6	22.1	31.3	46.4	40.9	52.4	64.6	78.2
$m = 500$	16.2	43.3	56.9	77.6	59.6	73.5	88.9	125.2

Each number in the table is the average of number of erroneous pairs obtained from ten independent runs. The parameter γ is the missing observation rate, η is the genotyping error rate, R is the number of "bad" markers, and m is the number of markers in each individual map. In these experiments we have $K = 6$ mapping populations obtained by removing a fraction $x = 0.7$ of markers from each individual map.

into MERGEMAP to build the consensus map. We denote this approach of building the consensus maps as MSTMAP+MERGEMAP.

Here, we compare the performance of MSTMAP+MERGEMAP against JOINMAP. To the best of our knowledge, JOINMAP is the most popular tool for building consensus map among geneticists. However, due to the fact that JOINMAP is GUI-based (nonscriptable) and becomes extremely slow when the number of markers exceeds 150, we collected results for only a few relatively small data sets. As mentioned in Section 1, an alternative approach to the problem of constructing consensus maps is to pool the genotype data for all the individual populations, and then, apply any existing genetic mapping algorithms by treating the pooled data set as a single population. When pooling individual data sets, a large number of missing observations have to be introduced. According to this strategy, we constructed a consensus map with MSTMAP by first combining the raw mapping data from multiple populations into a pooled data set. We call this latter approach MSTMAP-C.

We considered two parameter sets, which we believed to be realistic. In the first, the parameters are $m = 100$, $K = 6$, $x = 0.7$, $\eta = 0.001$, $\gamma = 0.001$, and $R = 0$. In the second, we set $R = 2$ and left the rest of the parameters untouched. For each choice of the parameters, 10 random data sets were generated, and the number of erroneous marker pairs and the running time was recorded. The results for the two parameters set are presented in Fig. 5.

Fig. 5b shows that MSTMAP+MERGEMAP is orders of magnitude faster than JOINMAP (the y-axis is in log-scale). The difference in running time becomes more apparent when

TABLE 2
Comparison between MSTMAP+MERGEMAP and MSTMAP-C for $K = 8, x = 0.5$

	MSTMAP+MERGEMAP # erroneous pairs				MSTMAP-C # erroneous pairs			
$\gamma = \eta \rightarrow$	0.001	0.005	0.01	0.02	0.001	0.005	0.01	0.02
$R = 0$								
$m = 100$	8.4	12.9	17.3	21.4	33.8	46.0	40.6	49.4
$m = 300$	20.5	37.5	42.7	73.4	113.1	111.0	124.2	140.1
$m = 500$	28.7	63.0	87.0	110.0	173.3	160.6	179.4	229.0
$R = 2$								
$m = 100$	7.4	13.3	17.7	22.4	42.8	43.1	48.1	67.7
$m = 300$	23.1	34.9	54.6	76.6	101.7	101.9	125.2	152.2
$m = 500$	31.7	63.4	89.1	109.1	167.0	178.1	238.7	255.8
$R = 4$								
$m = 100$	6.3	16.3	16.9	20.3	52.9	68.5	59.4	61.4
$m = 300$	19.2	38.0	51.8	81.1	127.2	108.7	123.6	145.7
$m = 500$	31.4	63.1	76.8	111.2	153.6	195.7	209.8	249.2
$R = 6$								
$m = 100$	6.3	12.2	13.5	22.6	48.8	57.4	62.0	63.0
$m = 300$	45.2	41.5	45.4	63.0	120.0	152.5	130.1	167.1
$m = 500$	36.2	65.4	71.7	125.6	181.0	233.1	181.0	246.8

Please refer to the caption of Table 1 for explanations of the notations used in the table.

TABLE 3
Comparison between MSTMAP+MERGEMAP and MSTMAP-C for $K = 10, x = 0.4$

	MSTMAP+MERGEMAP # erroneous pairs				MSTMAP-C # erroneous pairs			
$\gamma = \eta \rightarrow$	0.001	0.005	0.01	0.02	0.001	0.005	0.01	0.02
$R = 0$								
$m = 100$	10.3	15.9	21.1	29.6	54.9	84.3	58.7	70.5
$m = 300$	30.4	49.4	61.6	78.6	175.0	164.9	191.5	231.3
$m = 500$	50.3	77.7	97.7	130.8	301.1	324.5	334.8	414.1
$R = 2$								
$m = 100$	7.6	13.7	22.1	31.8	75.8	68.5	94.5	89.5
$m = 300$	28.1	48.0	66.1	79.6	186.1	176.7	199.4	265.6
$m = 500$	45.6	72.9	97.3	142.9	341.6	294.8	373.3	366.8
$R = 4$								
$m = 100$	11.6	12.1	20.0	29.9	56.7	82.9	97.9	93.9
$m = 300$	25.5	53.6	57.6	93.5	207.5	193.8	235.4	214.4
$m = 500$	46.3	81.5	93.3	138.8	283.2	353.4	375.7	412.5
$R = 6$								
$m = 100$	11.9	17.3	16.7	21.8	95.9	78.2	89.7	114.6
$m = 300$	27.8	50.4	57.0	83.1	194.6	193.6	182.2	234.1
$m = 500$	44.3	82.1	97.2	136.4	331.0	302.4	303.0	475.6

Please refer to the caption of Table 1 for explanations of the notations used in the table.

TABLE 4
Comparison between MSTMAP+MERGEMAP and MSTMAP-C for $K = 12, x = 0.35$

	MSTMAP+MERGEMAP # erroneous pairs				MSTMAP-C # erroneous pairs			
$\gamma = \eta \rightarrow$	0.001	0.005	0.01	0.02	0.001	0.005	0.01	0.02
$R = 0$								
$m = 100$	8.5	14.9	19.5	31.9	97.6	82.9	74.2	97.0
$m = 300$	32.0	46.2	60.7	85.2	221.7	310.9	284.7	245.1
$m = 500$	55.4	83.7	105.7	134.1	361.4	848.0	442.8	466.2
$R = 2$								
$m = 100$	13.4	19.3	22.4	31.0	79.0	88.2	82.0	110.0
$m = 300$	31.9	50.0	67.0	102.7	204.6	254.3	257.7	291.2
$m = 500$	55.4	86.9	108.7	138.0	365.6	407.5	731.6	480.2
$R = 4$								
$m = 100$	14.6	15.9	21.0	28.8	96.7	74.9	88.6	129.3
$m = 300$	33.6	51.1	64.1	86.9	218.4	227.4	254.7	304.9
$m = 500$	56.8	86.6	116.3	142.0	376.9	392.8	392.3	475.1
$R = 6$								
$m = 100$	12.6	15.6	23.3	27.7	82.4	116.4	70.8	140.2
$m = 300$	32.5	47.6	120.4	80.9	251.0	234.1	264.9	347.8
$m = 500$	55.7	74.3	101.3	134.0	381.2	366.6	416.7	399.5

Please refer to the caption of Table 1 for explanations of the notations used in the table.

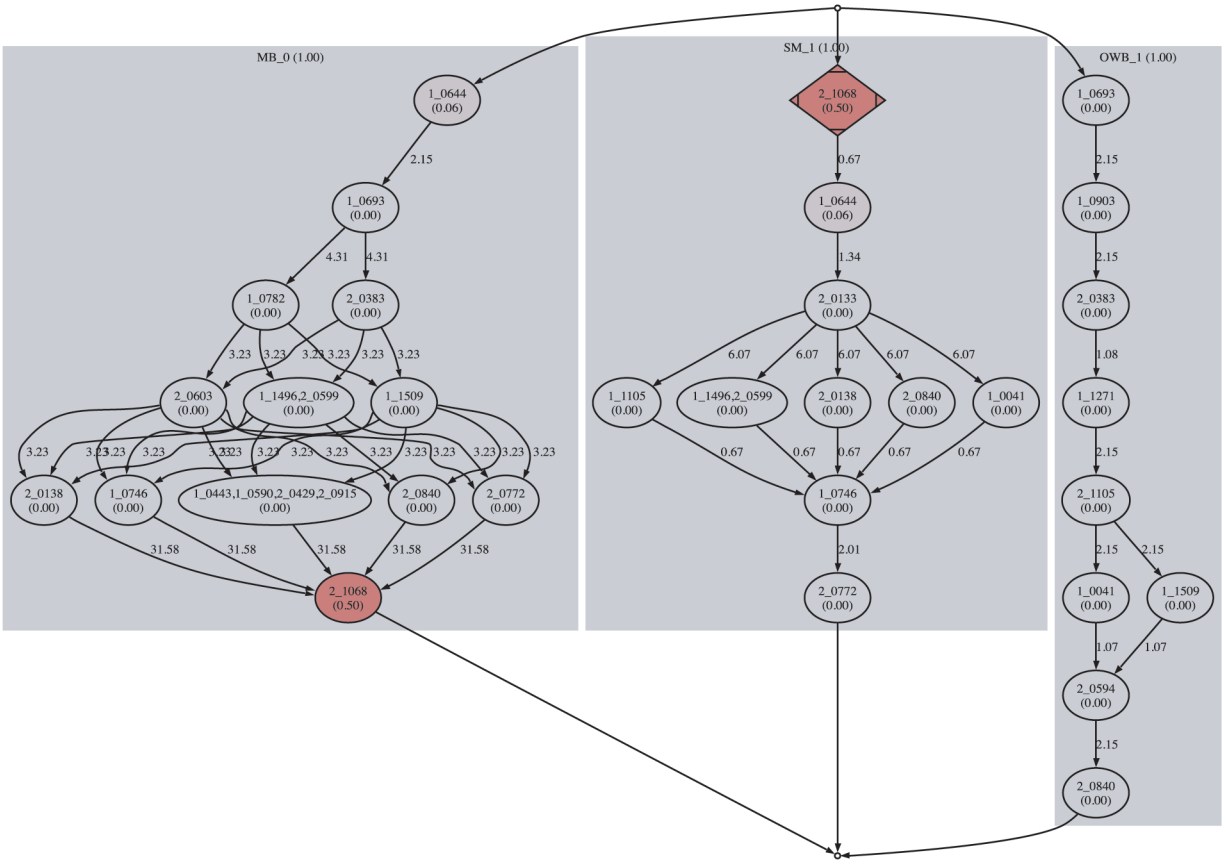


Fig. 6. A portion of the graph produced by MERGEMAP highlighting the conflicts among the OWB, SM, and MB map that emerged while building the consensus map for chromosome 1H of barley. Each individual map is framed in a shaded block, nodes correspond to marker, and the numbers on the edges indicate genetic distances. Markers at the same horizontal level belong to the same bin. The numbers enclosed in the parentheses inside the nodes are the probabilities of deletion computed by our algorithm. Each node is filled with a color whose saturation is proportional to the associated probability, which allows the user to quickly spot the problematic markers. The integral solution obtained by the rounding step would delete the marker occurrence enclosed in diamonds.

m is large. Also, observe that MSTMAP-C can be faster than MSTMAP+MERGEMAP. Fig. 5a shows that: 1) the consensus maps obtained by MSTMAP+MERGEMAP are significantly more accurate than the ones produced by JOINMAP or MSTMAP-C and 2) MSTMAP-C have comparable accuracy to JOINMAP. We believe that the same conclusions can be derived for larger data sets.

In order to investigate the extent of the advantages brought upon by MERGEMAP, we performed an extensive comparison between MSTMAP-C and MSTMAP+MERGEMAP for a variety of parameter settings. For example, Table 1 summarizes the results for $K = 6, x = 0.7$. For this choice of parameters, it is clear that MSTMAP+MERGEMAP outperforms MSTMAP-C for each choice of the parameters. The running time for MSTMAP+MERGEMAP is comparable with those presented in Fig. 4, whereas the running time for MSTMAP-C is always very short, within a few minutes regardless of the size of the input. Similar results were obtained for the cases where $K = 8$, $K = 10$, and $K = 12$ (see Tables 2, 3, and 4).

3.3 Comparing with Join Map on Real Genotyping Data

The real genotyping data were obtained in the context of an ongoing mapping project for the genome of *Hordeum vulgare* (barley). In total, we had three mapping populations under study, all of which are DH populations. The first mapping population (called OWB hereafter) is the result of crossing Oregon Wolfe Barley Dominant with Oregon Wolfe Barley Recessive (see http://barleyworld.org/oregonwolfe.php). The OWB data set consists of 1,020 markers genotyped on 93 individuals. The second mapping population (called SM hereafter) is the result of a cross of Steptoe with Morex (see http://wheat.pw.usda.gov/ggpages/SxM/), which consists of 800 markers genotyped on 149 individuals. The third mapping population (called MB hereafter) is the result of a cross of Morex with Barke.[4] The MB mapping population contains 1,068 markers on 93 individuals. The three data sets as a whole provide a coverage of 1,853 markers in total. The genotyping data were collected using the Illumina GoldenGate Assay platform.

The individual genetic maps for the three mapping populations of barley were assembled with MSTMAP [7], [8]. Each individual genetic map contains seven linkage groups corresponding to the seven chromosomes of barley, which are conventionally named 1H-7H. A consensus map was built with MERGEMAP from the OWB, SM, and MB maps of each of the seven chromosomes. We observed no conflicts among the three input maps for chromosomes 2H, 3H, 4H, 6H, and 7H. However, the consensus maps for

4. This latter cross was recently developed by Stein and colleagues at the Leibniz Institute of Plant Genetics and Crop Plant Research (IPK).

chromosome 1H and 5H generated conflicts involving a large number of markers.

MERGEMAP is able to produce a graphical view of the conflicts among the individual maps. By solving the LP relaxation of the linear integer program (1), MERGEMAP associates a probability with every marker occurrence. The higher the probability, the more likely that the marker occurrence is responsible for the conflicts. By inspecting the graph produced for 1H (the portion of the graph that highlights the conflicts is shown in Fig. 6), it was clear to us that marker 2_1068 was the one causing the conflict. It turned out that marker 2_1068 was placed at the telomere in the MB map, while in the SM map, it was placed somewhere in the middle of the chromosome. Observe that removing marker 2_1068 from either the MB map or the SM map would have resolved all the conflicts; therefore, both marker occurrences are correctly associated with a probability of 0.5 for deletion. Following this analysis, we revisited the raw genotype data for marker 2_1068, and noticed that the quality of the genotype call in the MB population was quite low. We deleted marker 2_1068 in the MB map and rebuilt the consensus map for chromosome 1H, this time observing no conflicts among the individual maps.

Similarly, in chromosome 5H, the algorithm identified marker 2_0029 as problematic by assigning it a high probability for deletion. When we revisited the Illumina GoldenGate Assay workspace, we confirmed that marker 2_0029 was a low-quality call in the MB map. After deleting marker 2_0029 from the MB map, the consensus map for chromosome 5H was conflict-free.

While we were implementing our algorithms and developing our map integration software, we were also processing the same barley data set with JOINMAP on a 3.3 GHz Pentium processor workstation with 2 GB memory. JOINMAP finished merging the seven linkage groups after about three months of uninterrupted execution. The same job was carried out in less than 5 minutes by MERGEMAP. When we compared the consensus map generated by JOINMAP to the original individual maps, in 174 places, the consensus maps were not consistent with the marker order in the individual maps. In contrast, after the removal of the two problematic markers, the consensus maps by MSTMAP+MERGEMAP were 100 percent consistent with the individual maps. According to this observation, we can conclude that the consensus maps generated by MERGEMAP are significantly more reliable than the one generated by JOINMAP.

ACKNOWLEDGMENTS

This project was supported in part by the US National Science Foundation (NSF) CAREER IIS-0447773, NSF DBI-0321756, and USDA CSREES Barley-CAP (visit http://barleycap.org/ for more information on this project).

REFERENCES

[1] A.H. Sturtevant, "The Linear Arrangement of Six Sex-Linked Factors in Drosophila, as Shown by Their Mode of Association," *J. Experimental Zoology,* vol. 14, pp. 43-59, 1913.

[2] J. Jansen, A.G. de Jong, and J.W. van Ooijen, "Constructing Dense Genetic Linkage Maps," *Theoretical and Applied Genetics,* vol. 102, pp. 1113-1122, 2001.

[3] T. Schiex and C. Gaspin, "CARTHAGENE: Constructing and Joining Maximum Likelihood Genetic Maps," *Proc. Int'l Conf. Intelligent Systems for Molecular Biology (ISMB),* pp. 258-267, 1997.

[4] H. Iwata and S. Ninomiya, "AntMap: Constructing Genetic Linkage Maps Using an Ant Colony Optimization Algorithm," *Breeding Science,* vol. 56, pp. 371-377, 2006.

[5] H.V. Os, P. Stam, R.G.F. Visser, and H.J.V. Eck, "RECORD: A Novel Method for Ordering Loci on a Genetic Linkage Map," *Theoretical and Applied Genetics,* vol. 112, pp. 30-40, 2005.

[6] D.A. Cartwright, M. Troggio, R. Velasco, and A. Gutin, "Genetic Mapping in the Presence of Genotyping Errors," *Genetics,* vol. 174, pp. 2521-2527, 2007.

[7] Y. Wu, P.R. Bhat, T.J. Close, and S. Lonardi, "Efficient and Accurate Construction of Genetic Linkage Maps from Noisy and Missing Genotyping," *Proc. Workshop Algorithms in Bioinformatics (WABI),* pp. 395-406, 2007.

[8] Y. Wu, P.R. Bhat, T.J. Close, and S. Lonardi, "Efficient and Accurate Construction of Genetic Linkage Maps from the Minimum Spanning Tree of a Graph," *PLoS Genetics,* vol. 4, p. e1000212, Oct. 2008.

[9] C. Dib, S. Faure, C. Fizames, D. Samson, N. Drouot, A. Vignal, P. Millasseau, S. Marc, J. Kazan, E. Seboun, M. Lathrop, G. Gyapay, J. Morissette, and J. Weissenbach, "A Comprehensive Genetic Map of the Human Genome Based on 5264 Microsatellites," *Nature,* vol. 380, pp. 152-154, 1996.

[10] N. Ihara, A. Takasuga, K. Mizoshita, H. Takeda, M. Sugimoto, Y. Mizoguchi, T. Hirano, T. Itoh, T. Watanabe, K.M. Reed, W.M. Snelling, S.M. Kappes, C.W. Beattie, G.L. Bennett, and Y. Sugimoto, "A Comprehensive Genetic Map of the Cattle Genome Based on 3802 Microsatellites," *Genome Research,* vol. 14, pp. 1987-1998, 2004.

[11] W.F. Dietrich, J.C. Miller, R.G. Steen, M. Merchant, D. Damron, R. Nahf, A. Gross, D.C. Joyce, M. Wessel, R.D. Dredge, A. Marquis, L.D. Stein, N. Goodman, D.C. Page, and E.S. Lander, "A Genetic Map of the Mouse with 4,006 Simple Sequence Length Polymorphisms," *Nature Genetics,* vol. 7, no. 2S, pp. 220-245, 1994.

[12] R.G. Steen, A.E. Kwitek-Black, C. Glenn, J. Gullings-Handley, W. Van Etten, O.S. Atkinson, D. Appel, S. Twigger, M. Muir, T. Mull, M. Granados, M. Kissebah, K. Russo, R. Crane, M. Popp, M. Peden, T. Matise, D.M. Brown, J. Lu, S. Kingsmore, P.J. Tonellato, S. Rozen, D. Slonim, P. Young, M. Knoblauch, A. Provoost, D. Ganten, S.D. Colman, J. Rothberg, E.S. Lander, and H.J. Jacob, "A High-Density Integrated Genetic Linkage and Radiation Hybrid Map of the Laboratory Rat," *Genome Research,* vol. 9, no. 6, pp. AP1-8, 1999.

[13] B.N. Jackson, S. Aluru, and P.S. Schnable, "Consensus Genetic Maps: A Graph Theoretic Approach," *Proc. Computational Systems Bioinformatics Conf. (CSB),* pp. 35-43, 2005.

[14] B.N. Jackson, P.S. Schnable, and S. Aluru, "Consensus Genetic Maps as Median Orders from Inconsistent Sources," *IEEE/ACM Trans. Computational Biology and Bioinformatics,* vol. 5, no. 2, pp. 161-171, Apr.-June 2008.

[15] W.D. Beavis and D. Grant, "A Linkage Map Based on Information from Four f_2 Populations of Maize (Zea Mays L.)," *Theoretical and Applied Genetics,* vol. 82, pp. 636-644, Oct. 1991.

[16] P. Stam, "Construction of Integrated Genetic Linkage Maps by Means of a New Computer Package: Joinmap," *The Plant J.,* vol. 3, pp. 739-744, 1993.

[17] D.I. Mester, Y.I. Ronin, M.A. Korostishevsky, V.L. Pikus, A.E. Glazman, and A.B. Korol, "Multilocus Consensus Genetic Maps (mcgm): Formulation, Algorithms, and Results," *Computational Biology and Chemistry,* vol. 30, no. 1, pp. 12-20, 2006.

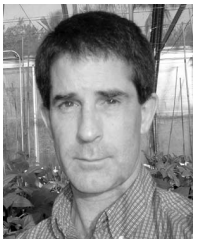
[18] I.V. Yap, D. Schneider, J. Kleinberg, D. Matthews, S. Cartinhour, and S.R. McCouch, "A Graph-Theoretic Approach to Comparing and Integrating Genetic, Physical and Sequence-Based Maps," *Genetics,* vol. 165, pp. 2235-2247, Dec. 2003.

[19] P. Wenzl, H. Li, J. Carling, M. Zhou, H. Raman, E. Paul, P. Hearnden, C. Maier, L. Xia, V. Caig, J. Ovesn, M. Cakir, D. Poulsen, J. Wang, R. Raman, K.P. Smith, G.J. Muehlbauer, K.J. Chalmers, A. Kleinhofs, and E.H.A. Kilian, "A High-Density Consensus Map of Barley Linking DArT Markers to SSR, RFLP and STS Loci and Agricultural Traits," *BMC Genomics,* vol. 7, 2006.

[20] S.A. Plotkin, D.B. Shmoys, and E. Tardos, "Fast Approximation Algorithms for Fractional Packing and Covering Problems," *Proc. Ann. Symp. Foundations of Computer Science (FOCS),* pp. 495-504, 1991.

[21] M. Girvan and M.E.J. Newman, "Community Structure in Social and Biological Networks," *Proc. Nat'l Academy of Sciences USA,* vol. 99, pp. 7821-7826, June 2002.
[22] A. Aho, "The Transitive Reduction of a Directed Graph," *SIAM J. Computing,* vol. 1, no. 2, pp. 131-137, 1972.
[23] A.B. Kahn, "Topological Sorting of Large Networks," *Comm. ACM,* vol. 5, no. 11, pp. 558-562, 1962.

Yonghui Wu received the PhD degree from the Department of Computer Science and Engineering, University of California, Riverside, in August 2008. His advisor was Dr. Stefano Lonardi. He is now a full time research and software engineer with Google. His research interests are in bioinformatics, computational biology, algorithm, in general, and data mining.



Timothy J. Close received the PhD degree in genetics from the University of California, Davis, in 1982. He is a professor of genetics in the Department of Botany and Plant Sciences at the University of California, Riverside. In 1992, he received the Divisional Young Investigator Award from the National Science Foundation. In 2006, he was a founding member of the International Barley Genome Sequencing Consortium. In 2009, he was elected a fellow of the American Association for the Advancement of Science. His research is on genetic variation in environmental tolerance in crop plants, particularly barley, citrus, cowpea, and wheat. He has been a leader of genome resource development in these crop plants for the past 5-10 years.

Stefano Lonardi received the Laurea cum laude degree from the University of Pisa in 1994, the PhD degree from the Department of Computer Sciences, Purdue University, West Lafayette, Indiana, in Summer 2001, and the doctorate degree in electrical and information engineering from the University of Padua in 1999. He is an associate professor of computer science and engineering at the University of California, Riverside. In Summer 1999, he was an intern at Celera Genomics, Department of Informatics Research, Rockville, Maryland. His recent research interest includes computational molecular biology, data compression, and data mining. He has published more than 35 papers in major theoretical computer science and computational biology journals and has about 45 publications in referred international conferences. In 2005, he received the CAREER award from National Science Foundation. He is a member of the IEEE Computer Society.