ON THE ACCURATE CONSTRUCTION OF CONSENSUS GENETIC MAPS

[1]Yonghui Wu, [2]Timothy J. Close, and [1]Stefano Lonardi*
[1]*Department of Computer Science and Engineering, University of California, Riverside, CA 92521, USA*
[2]*Department of Botany and Plant Sciences, University of California, Riverside, CA 92521, USA*
**Email: stelo@cs.ucr.edu*

We study the problem of merging genetic maps, when the individual genetic maps are given as directed acyclic graphs. The problem is to build a *consensus map*, which includes and is consistent with all (or, the vast majority of) the markers in the individual maps. When markers in the input maps have ordering conflicts, the resulting consensus map will contain cycles. We formulate the problem of resolving cycles in a combinatorial optimization framework, which in turn is expressed as an integer linear program. A faster approximation algorithm is proposed, and an additional speed-up heuristic is developed. According to an extensive set of experimental results, our tool is consistently better than JOINMAP, both in terms of accuracy and running time.
Keywords: genetic map, consensus, genotype, population genetics

1. INTRODUCTION

Genetic linkage maps are arguably the cornerstone of a variety of biological applications including map-assisted breeding, association genetics and map-assisted gene cloning, just to name a few. Traditionally scientists have focused on building genetic maps for a single mapping population, task for which a wide variety of software tools are available and have satisfactory performance, e.g., JOINMAP[1], CARTHAGENE[2], ANTMAP[3], RECORD[4] TMAP[5] and MSTMAP[6].

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 a variety of marker types. Today it is increasingly common to find several genetic maps available for the same organism, usually for different sets of genetic markers and obtained with a variety of genotyping technologies. Notable examples are genetic linkage maps based on microsatellites in human[7] and in cattle[8], and maps based on sequence length polymorphism in mouse[9] and rat[10], just to name a few. In the case of maize, for instance, seven distinct mapping populations of *Zea mays* have been used[11]. When multiple maps are available, one could envision to construct a bigger single map (hereafter called *consensus map*) that includes all the markers.

A consensus map provides a higher density of markers and therefore a greater genome coverage than the individual maps. As the name suggests, the consensus map should be consistent with the order of the markers from the individual maps. However, this may not always be possible since the presence of errors is very likely to introduce conflicts between the individual maps. Due to the way individual genetic maps are assembled, two types of errors are observed, namely local reshuffles and global displacements. Local reshuffles refer to inaccuracies in the order of nearby markers, whereas global displacements refer to the cases where a few markers are placed at positions far from the correct ones. When addressing conflicts to build the consensus maps, one should take into account both types of errors.

1.1. Related works

Several systematic approaches have been proposed to construct consensus maps[12; 13; 1; 14; 11]. The method adopted by Beavis *et al.*[12] 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. Although this pooling strategy is commonly used, 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 dataset obtained from a double haploid population and another from a recombinant inbred lines population), then they cannot be merged and treated equivalently afterward. Second, the pooling method results in a large number of missing observations. A large amount of missing observations combined with the limited tolerance to missing data by exist-

*Corresponding author.

ing mapping algorithms inevitably deteriorates the quality of the consensus map.

An alternative approach, like the one used in the tool JOINMAP[13; 1], is to first obtain the consensus estimates of pairwise genetic distances by weighting for population structure and size. Then, the tool 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, it is well-known that distance estimates are not very accurate when based on a small sample of recombination events. Construction of genetic maps based on approximate estimates will result in inaccuracies in the ordering between markers on the consensus map. 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 took *three months* of computation to construct a consensus map from three individual maps of barley containing a total of 1,800 markers (the markers are divided into 7 linkage groups of roughly equal sizes). Despite these drawbacks, JOINMAP is still the only off-the-shelf software package available to build consensus maps.

The most recent approach to the problem relies on graph theory and was initially proposed by Yapa *et al.*[14] and later extended by Jackson *et al.*[11] Yapa *et al.*[14] use directed acyclic graphs (DAG) to represent maps from individual populations. The set of DAGs are 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.*[11] propose 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. A similar approach is to remove a minimum weight feedback vertex set from the graph. The obvious drawback of this method is that the markers corresponding to the deleted vertices will be excluded from the consensus map.

1.2. Our contribution

We follow the graph theoretical paradigm outlined in [11, 14] and represent individual genetic maps as DAGs. The individual maps are combined into a single directed graph according to their shared vertices. Any ordering conflict among the individual maps will result in cycles in the combined graph. Here, we propose to resolve the cycles by removing the smallest set of (feedback) marker occurrences. Note that we are not deleting markers but marker occurrences. A marker may occur in 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.

Trying to identify and eliminate a small number of marker occurrences from some of the maps is a better strategy than the one proposed in [11] because it more accurately reflects the type of errors that may be present in the individual maps. We formulate the optimization problem resulting from this strategy via integer linear programming (Section 3), and we propose an approximation algorithm to solve it. We also devise an heuristic to decompose the original problem, in the case the size of the instance to be solved is too large.

As soon as all cycles in the consensus map are resolved, we process the resulting graph with another novel algorithm whose objective is to simplify the DAG to help geneticists to be able to visualize and make use of the consensus map (Section 4). This step involves removing redundant edges and merging nodes on the consensus map without introducing conflicts. In the last step, a final algorithm produces a linear order of the markers which is consistent with the consensus graph (Section 5).

The last two steps of our approach, i.e., condensing the markers and linearizing the DAG, further distinguishes our approach from those in [14, 11]. The final output of our workflow is a linear order of marker bins[a] which is a format geneticists are used to work with. The output of the methods by Yapa *et al.*[14] and Jackson *et al.*[11] is however a DAG, which is often too complex and convoluted to make much sense out of it. For the same reasons, we did not compare our experimental results against those methods.

In Section 6 we carry out an extensive evaluation of our algorithms on synthetic data. We compare the perfor-

[a] A *bin* is a set of markers for which the relative orders are undetermined.

mance of our method with JOINMAP. Our approach produces consistently better results than JOINMAP, both in terms of accuracy and running time. Our method is also superior to the method of pooling together genotyping data from individual maps. We have also employed our software on the genotyping data we collected for three mapping populations (about 1,800 markers) for barley, but we had to omit those results from this manuscript due to lack of space.

2. PRELIMINARIES AND NOTATIONS

A *genetic linkage map* represents the linear order and the pairwise distance of markers on a chromosome. The distance between two adjacent markers, expressed in *centimorgans* (cM), is determined by the frequency of genetic recombination occurring in the region between them. Two markers are one centimorgan apart if one observes an average of 0.01 crossovers per meiosis in the region enclosed by the two markers. The set of markers for which no recombination is detected is called a *bin*. For markers in the same bin, their relative orders are undetermined. From this point forward, a genetic map is composed of a sequence of bins (of marker) and the distance between them.

Some notations are in order. Let Π denote a genetic linkage map, and let M_Π 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 the map, 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 of $m_j\}$ and the weight of an edge (m_i, m_j) is set to the distance between the corresponding bins. The notion of induced graph can be extended to a set of maps. Let $G_\Omega = (M_\Omega, E_\Omega)$ be 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 defined by deleting the occurrences of all markers not in R from the map Π. The subproblem $\Omega(R)$ of the original problem Ω restricted to R is defined as $\Omega(R) = \{\Pi_i(R) | \Pi_i \in \Omega\}$.

Figure 1 illustrates the notations Π, Ω, G_Π, G_Ω, M_Ω, N_Ω, $\Pi(R)$ and Ω_R for a small example.

3. RESOLVING ORDERING CONFLICTS

Let $\Omega = \{\Pi_1, \Pi_2, \ldots, \Pi_K\}$ be the set of input maps for which we want to build a consensus map. 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 have conflicting orders among the input maps, then G_Ω contains cycles. In order to resolve cycles, we propose to delete the smallest set of marker occurrences. More precisely, if we first assign weights to the individual maps to represent their quality (i.e., high weight is associated with high quality), the problem is to delete the minimum-weight set of marker occurrences so that the resulting subproblem is conflict-free. The optimization problem that emerges from this strategy is the following.

Minimum-Weight Feedback Marker Occurrence Set (MWFMOS)

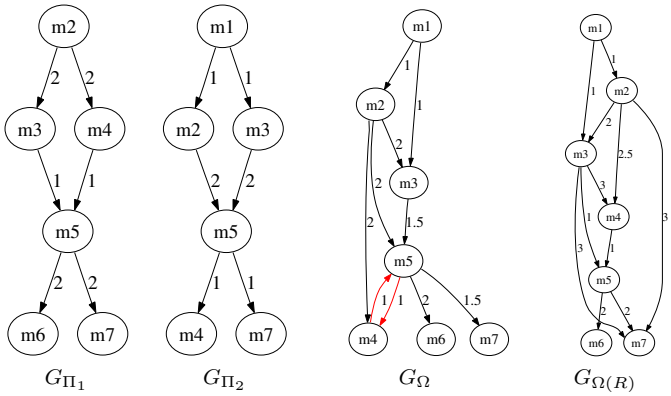
Input: Ω and w, where Ω is a set of individual maps from which one would like to build a consensus map, and w is the associated weight function on N_Ω where $w(m_i^j)$ is the weight of marker occurrence m_i^j. Without loss of generality, we assume that $w(m_i^j) > 1$ for all $m_i^j \in N_\Omega$.

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 uses a reduction from the minimum feedback edge set problem (not shown here due to space restrictions). 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 non-overlapping subproblems corresponding to the strongly connected components in G_Ω. 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 a subproblem only.

$\Omega = \{\Pi_1, \Pi_2\}$
$\Pi_1 = [(m_2)\ 2\ (m_3, m_4)\ 1\ (m_5)\ 2\ (m_6, m_7)]$
$\Pi_2 = [(m_1)\ 1\ (m_2, m_3)\ 2\ (m_5)\ 1\ (m_4, m_7)]$
$M_\Omega = \{m_1, m_2, m_3, m_4, m_5, m_6, m_7\}$
$N_\Omega = \left\{ \begin{array}{l} m_2^1, m_3^1, m_4^1, m_5^1, m_6^1, m_7^1, \\ m_1^2, m_2^2, m_3^2, m_4^2, m_5^2, m_7^2 \end{array} \right\}$
$R = \left\{ \begin{array}{l} m_2^1, m_3^1, m_4^1, m_5^1, m_6^1, m_7^1, \\ m_1^2, m_2^2, m_3^2, m_4^2, m_7^2 \end{array} \right\}$
$\Pi_1(R) = \Pi_1$
$\Pi_2(R) = [(m_1)\ 1\ (m_2, m_3)\ 3\ (m_4, m_7)]$
$\Omega(R) = \{\Pi_1(R), \Pi_2(R)\}$



G_{Π_1} G_{Π_2} G_Ω $G_{\Omega(R)}$

Fig. 1. 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.

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 $m_{i_1}^{j_1} \rightarrow m_{i_2}^{j_1}$ means 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). Note that the path starts and ends with the same marker in two different fragments. Let P be the set of such paths.

Given P, we formulate MWFMOS as an Integer Linear Program (ILP) as follows.

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

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 is to be deleted and set to 0 otherwise. The LP relaxation of the above ILP is straightforward. The number of constraints in (1) is $|P|$, which is at most $O(K!|M_\mathcal{I}|^K)$, where K is number of fragments in $\mathcal{I}$ and $|M_\mathcal{I}|$ is the total number of distinct markers in $\mathcal{I}$. The upper bound is a polynomial in the size of the input if K is fixed. The dual of the LP relaxation of (1) is the following program.

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

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).

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

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.

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

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 r^{th} row of A is on the path corresponding to the c^{th} 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

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

ϵ-optimal) solution[b]. To find such an approximate solution, we follow the method proposed by Plotkin *et al.*[15] 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 $\vec{z}$. λ is 6ϵ optimal if the following two relaxed optimality conditions are met:

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

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

A sketch of the algorithm to find a 6ϵ optimal solution is presented in Figure 2. The performance guarantee of our algorithm APPROXSOLVE is presented as Theorem 1, and the time complexity of the algorithm is presented as Theorem 2.

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 an $(1+6\epsilon)$ optimal solution.*

Proof. This Lemma corresponds as Lemma 2.1 in [15]. To be self-contained, we present the proof here.
From (5) and (6), we have: $C(\vec{z}) \geq (1-\epsilon)\vec{z}^t A\vec{y} - \epsilon\lambda\vec{z}^t\vec{w} \geq (1-\epsilon)^2\lambda\vec{z}^t\vec{w} - \epsilon\lambda\vec{z}^t\vec{w} \geq (1-3\epsilon)\lambda\vec{z}^t\vec{w}$.
Hence, $\lambda \leq (1-3\epsilon)^{-1}C(\vec{z})/\vec{z}^t\vec{w} \leq (1-3\epsilon)^{-1}\lambda^* \leq (1+6\epsilon)\lambda^*$. □

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

Proof. The theorem follows from Lemma 2.2 in [15]. 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 met 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 $\alpha \geq \alpha_0 = \frac{2\ln(2|N_{\mathcal{I}}|\epsilon^{-1})}{\lambda\epsilon}$, $\vec{z}$ as assigned by the "for" loops at line 3 and 10 in algorithm APPROXSOLVE will satisfy condition (5).

Let $I = \{i : (1-\epsilon/2)\lambda w_i \geq \vec{a_i}^t\vec{y}\}$. Let $j \in I$. $\lambda z_j w_j = \lambda e^{\alpha\vec{a_j}^t\vec{y}/w_j} \leq \lambda e^{\alpha(1-\epsilon/2)\lambda} = \lambda e^{\alpha\lambda}e^{-\alpha\epsilon\lambda/2} \leq \lambda e^{\alpha\lambda}e^{-ln(2|N_{\mathcal{I}}|\epsilon^{-1})} \leq \frac{\epsilon}{2|N_{\mathcal{I}}|}\lambda e^{\alpha\lambda} \leq \frac{\epsilon}{2|N_{\mathcal{I}}|}[\lambda\vec{z}^t\vec{w}]$. Consequently, $\lambda\vec{z}^t\vec{w} = \sum_{i\in I}\lambda z_i w_i + \sum_{i\notin I}\lambda z_i w_i \leq \sum_{i\in I}\lambda z_i w_i + \sum_{i\notin I}\frac{1}{1-\epsilon/2}z_i\vec{a_i}^t\vec{y} \leq \sum_{i\in I}\lambda z_i w_i + \frac{1}{1-\epsilon/2}\vec{z}^t A\vec{y} \leq \frac{\epsilon}{2}\lambda\vec{z}^t\vec{w} + \frac{1}{1-\epsilon/2}\vec{z}^t A\vec{y}$.
Therefore, we have $(1-\epsilon)\lambda\vec{z}^t\vec{w} \leq \vec{z}^t A\vec{y}$

Notice that α is initialized to be $2\alpha_0$ and whenever $\max_r \vec{a_r}^t\vec{y}/w_r \leq \lambda/2$, 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 $\Phi_1 = \vec{z_1}^t\vec{w}$, $\Phi_2 = \vec{z_2}^t\vec{w}$. $\Phi_1 - \Phi_2 > \lambda\epsilon^2\Phi_1/4$.*

Proof. $\Phi_2 = \vec{z_2}^t\vec{w} = \sum_i e^{\alpha\vec{a_i}^t\vec{y_2}/w_i} = \sum_i e^{\alpha\vec{a_i}^t((1-\delta)\vec{y_1}+\delta\widetilde{y})/w_i} = \sum_i e^{\alpha\vec{a_i}^t\vec{y_1}/w_i}e^{\alpha\delta\vec{a_i}^t(\widetilde{y}-\vec{y_1})/w_i}$.
Since $w_i > 1$, $\vec{y_1} = 1$ and $\widetilde{y} = 1 \implies |\vec{a_i}^t(\widetilde{y}-\vec{y_1})/w_i| < 1$. Since $\delta = \frac{\epsilon}{4\alpha} \implies |\alpha\delta\vec{a_i}^t(\widetilde{y}-\vec{y_1})/w_i| < \epsilon/4 < 1/4$.
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 $e^{\alpha\delta\vec{a_i}^t(\widetilde{y}-\vec{y_1})/w_i} < 1 + (\alpha\delta\vec{a_i}^t(\widetilde{y}-\vec{y_1})/w_i) + 2(\alpha\delta\vec{a_i}^t(\widetilde{y}-\vec{y_1})/w_i)^2$
$< 1 + (\alpha\delta\vec{a_i}^t(\widetilde{y}-\vec{y_1})/w_i) + \frac{\epsilon}{2}(\alpha\delta\vec{a_i}^t(\widetilde{y}+\vec{y_1})/w_i)$
Therefore, $\Phi_2 = \sum_i e^{\alpha\vec{a_i}^t\vec{y_1}/w_i}e^{\alpha\delta\vec{a_i}^t(\widetilde{y}-\vec{y_1})/w_i} < \sum_i e^{\alpha\vec{a_i}^t\vec{y_1}/w_i} + \alpha\delta(C(\vec{z_1}) - \vec{z_1}A\vec{y_1}) + \frac{\epsilon}{2}\alpha\delta(C(\vec{z_1}) + \vec{z_1}A\vec{y_1})$
$\Phi_2 < \Phi_1 + \alpha\delta(C(\vec{z_1}) - \vec{z_1}A\vec{y_1}) + \frac{\epsilon}{2}\alpha\delta(C(\vec{z_1}) + \vec{z_1}A\vec{y_1})$
$\Phi_1 - \Phi_2 > \alpha\delta(\vec{z_1}A\vec{y_1} - C(\vec{z_1})) - \epsilon\alpha\delta\vec{z_1}A\vec{y_1}$
Due to the fact that $(\vec{y_1}, \lambda)$ and $\vec{z_1}$ do not satisfy (6), we have $\Phi_1 - \Phi_2 > \lambda\epsilon\alpha\delta\vec{z_1}^t\vec{w}$.
According to the choice of δ, $\Phi_1 - \Phi_2 > \lambda\epsilon^2\Phi_1/4$ □

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. Notice 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 (details not

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: $\quad z_r \leftarrow e^{\alpha \vec{a_r}^t \vec{y}/w_r}/w_r$
4: **while** $(\vec{y}, \lambda, \vec{z})$ does not satisfy (6) **do**
5: $\quad \widetilde{y} \leftarrow argmin_{\vec{y}|\vec{y}=1} \vec{z}^t A\vec{y}$
6: $\quad \vec{y} \leftarrow (1-\sigma)\vec{y} + \sigma\widetilde{y}$
7: $\quad$ **if** $\max_r \vec{a_r}^t \vec{y}/w_r \leq \lambda/2$ **then**
8: $\quad\quad \lambda \leftarrow \max_r \vec{a_r}^t \vec{y}/w_r$; $\alpha \leftarrow 4\ln(2|N_{\mathcal{I}}|\epsilon^{-1})/(\lambda\epsilon)$; $\sigma \leftarrow \epsilon/(4\alpha)$;
9: $\quad$ **for** $r = 1, \ldots, |N_{\mathcal{I}}|$ **do**
10: $\quad\quad z_r \leftarrow e^{\alpha \vec{a_r}^t \vec{y}/w_r}/w_r$
11: $\lambda \leftarrow \max_r \vec{a_r}^t \vec{y}/w_r$;
12: **return** $\vec{y}, \lambda, \vec{z}$

Fig. 2. A sketch of our LP-based algorithm

shown here), 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 better one among the two.

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 the descending order according to their associated probabilities. We delete just enough marker occurrences to resolve all the conflicts. Again, the solution is further reduced to a minimal solution by removing redundancies.

3.2. A speed-up heuristic

The 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 originally proposed by Girvan and Newman[16]. 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 centrality, the more likely it is the

"true bad" marker occurrence that should be deleted. Our heuristic algorithm works by computing the centrality for every marker occurrences and then iteratively deleting the ones with the highest value. The step is repeated until the sizes of the sub-problems are all small enough to be handled by our LP-based algorithm. The pseudo-code for our algorithm is presented in Figure 3.

4. CONDENSING THE MARKERS

Having resolved the conflicts in Ω, the graph G_Ω is now a DAG. In practice however, the graph G_Ω is overly complex. For example, G_Ω contains a large number of redundant edges. A directed edge (m_i, m_j) is said to be *redundant* if there is an alternative (distinct) path from m_i to m_j in G_Ω. For reasonably large individual maps, the resulting consensus graphs obtained after the removal of redundant edges can still be very complex.

Recall that a bin represents a set of nearby markers for which the relative orders are undetermined. In this step, we aim to simplify G_Ω by condensing markers into bins. In order to clearly differentiate the bins constructed in this step from the bins in the original maps, we refer to the former ones as *super-markers*.

The rationale for combining markers into super-markers is the following. If two markers always appear paired in the same bin of the original individual maps, then there is no way to determine the relative order between them and they should be drawn as a single super-marker. Generalizing this observation, we define the notion of *co-segregating* 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 *co-segregating* if they satisfy the following two conditions (A) m_i and m_j belong to the same bin in at least one of the maps in Ω, and (B) 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 super-marker are indeed close. The second condition is intended to ensure that the relative order between the markers to be condensed into a super-marker is undetermined.

The co-segregation relation does not define an equivalent relation, because it does not satisfy the transitivity property. Furthermore, when we group markers into super-markers, we must be careful not to introduce new conflicts. In order to address these issues, we employ a greedy iterative algorithm to carry out a maximal decomposition of the markers into super-markers. In each step, we condense one pair of co-segregating markers into a super-marker. The original problem Ω is being transformed into a new problem Ω' (which has one less marker than Ω). We keep repeating this iterative process until no co-segregating markers can be found. Let Ω^f be the final set of maps and G_{Ω^f} be the corresponding induced DAG. We further remove redundant edges from G_{Ω^f}, and let the final graph obtained by this procedure be DAG_Ω. DAG_Ω is guaranteed to have the following property.

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 super-markers contained in Π^f. Consider any two super-markers 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 either m_i is before m_j or the reverse). 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 co-segregating 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 super-markers M_{Π^f}. As a result, each super-marker can have at most one immediate predecessor and one immediate successor from one individual map. Since each super-marker can appear in at most K maps, the theorem follows. □

5. LINEARIZING THE DAG

In this last step, we process DAG_Ω to produce a linear order of the bins (super-markers). The linear order must be 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. In the case 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.

Let us define $D[b_i, b_j]$ to be the distance from a bin b_i to another 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.

Now, let b_i and b_j be two bins that are not ordered in DAG_Ω. Our algorithm determines the relative order between b_i and b_j as follows. There are three cases.

```
FASTDELETE(Ω, δ)
1: S ← ∅;
2: done ← false
3: while not done do
4:    done ← true;
5:    for each connected components in G_{Ω(N_Ω − S)} do
6:       I ← the corresponding sub-problem
7:       if |N_I| > δ then
8:          done ← false;
9:          Q ← ∅;
10:         for m ∈ M_I do
11:            A ← the set of occurrences of marker m in I
12:            for m^i ∈ A do
13:               for m^j ∈ A do
14:                  p ← a shortest path from m^i to m^j if there exists one
15:                  Q ← Q + {p}
16:         calculate the node centrality for each maker occurrence in N_I based on set Q and the associated weights
17:         v ← the marker occurrence with the highest centrality
18:         S ← S + {v}
19: return S
```

Fig. 3. A sketch of our heuristic-based algorithm

- If 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 $\Delta^{(p,s)}[b_i, b_j] = D[p, s]\left(\frac{D[p,b_j]}{D[p,b_j]+D[b_j,s]} - \frac{D[p,b_i]}{D[p,b_i]+D[b_i,s]}\right)$. 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 the preferred order is b_i before b_j. Otherwise, the preferred order is b_j before b_i.
- If 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. $\Delta[b_i, b_j] = \frac{\sum_{s\in S} \Delta^s[b_i,b_j]}{|S|}$.
- If b_i and b_j have only common ancestors. $D[b_i, b_j]$ is similarly computed as in the previous case.

The algorithm we propose to linearize DAG_Ω is similar to the topological sorting algorithm. 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.

6. EXPERIMENTAL RESULTS

We implemented our algorithms in C++ and carried out extensive evaluations on both real data sets and synthetic data sets. Due to lack of space we will present only the results for synthetic data. Our software tool, called MERGEMAP, is available upon request from the authors.

6.1. Evaluation of the conflict-resolution

The purpose of this set of experiments is to assess the effectiveness and efficiency of our conflict-resolving algorithm. Each data set of this experiment consists of six individual maps, which are all noisy copies of one single *true* map. The *true* map of a data set is simply a permutation of m markers, where the parameter m ranges

from 100 to 500 (representing a spectrum of maps from medium sizes to extra large sizes). 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 we relocate γ randomly chosen markers to a random position. The η swaps are intended to mimic local reshuffles while the γ relocations are intended to mimic global displacements, the two types of errors that may 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 bin condensation and the final linearization. The consensus map was compared with the *true* map and the number of erroneous marker pairs were counted. We call 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, ten independent random data sets were generated. For each dataset, 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 Figure 4.

As Figure 4 illustrates, MERGEMAP is very accurate in detecting the problematic markers and deleting them before merging the individual maps. In most cases, the number of erroneous marker pairs is less than ten, and in a few cases the number of erroneous pairs is equal to zero. When η or γ increases, the problem becomes harder and the quality of the consensus map deteriorates. On the contrary, as m increases the number of erroneous pairs decreases[c].

The running time of MERGEMAP increases as m or η or γ increase, but our software tool is relatively efficient. For the largest dataset with $m = 500$ markers, $\eta = 30$ and $\gamma = 6$, MERGEMAP finishes within 2-3 hours. In contrast, JOINMAP takes several **weeks** to assemble maps with 300 or so markers.

6.2. Comparison with JOINMAP

The objective of this 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 datasets are 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 an attempt to model what happens in practice, where the data for individual maps only represent a *subset* of the universe of genetic markers.

The entire procedure to generate a synthetic genotyping dataset 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 centimorgan plus a random distance according to a Poisson process with a mean of 2 centimorgans. 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 DH (double haploid) 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 centimorgans, 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.

[c]The only outlier in the figure is the case $m = 300$, $\eta = 20$ and $\gamma = 6$. We examined the raw data, and found that the high mean and standard deviation is 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 pair in total. When averaged over the ten runs, the single bad marker contributed 17 to the average number of erroneous pairs.

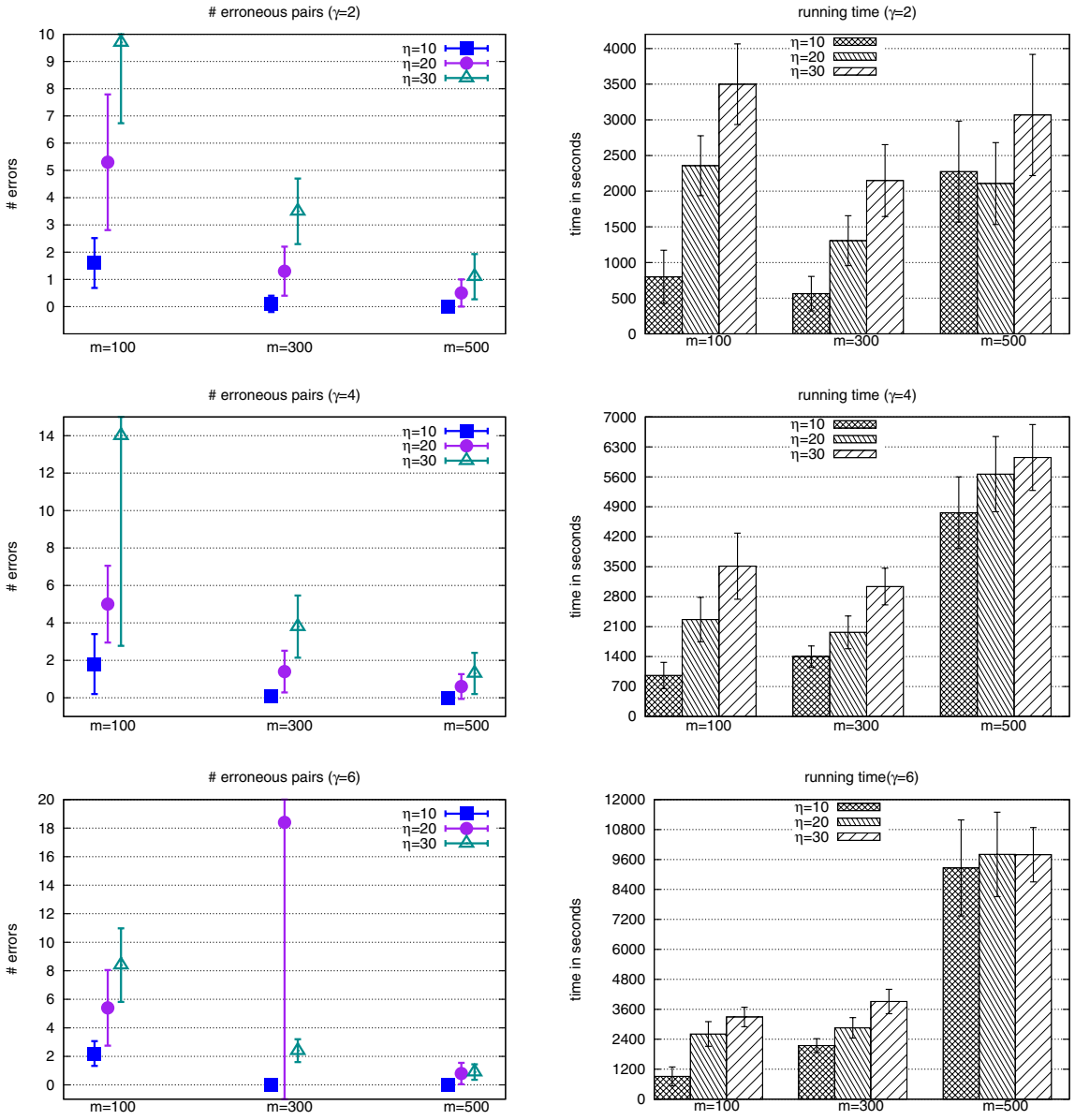


Fig. 4. The number of erroneous marker pairs obtained with MERGEMAP (LEFT) and the average running time (RIGHT) for various choices of m, η and γ. Each point in the figure is an average of the results obtained from ten independent data sets. The standard deviation for the corresponding statistic is represented as the error bar in the above figures.

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 (they are "bad" after all), 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 dataset 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 are assem-

bled by our tool MSTMAP[6]. The individual maps are then fed 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 the software tool JOINMAP. According to our knowledge, JOINMAP is the most popular tool for building consensus map in the community. However, due to the fact that JOINMAP is GUI-based (non-scriptable) and is extremely slow as soon as the number of markers exceeds 150, we were only able to collect results for a few relatively small data sets. As mentioned in the introduction, 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 datasets, a large number of missing observations will have to be introduced. Following this idea, we constructed a consensus map with MSTMAP by first combining the raw mapping data from multiple populations into a pooled dataset. We call this latter approach MSTMAP-C.

We consider two parameter sets, which we thought to be rather 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 changed $R = 2$, while the rest of the parameters were kept identical. For each choice of the parameters, ten random data sets are generated. The number of erroneous marker pairs and the running time is recorded. The results for the two parameters set are presented in Figure 5.

Figure 5-RIGHT 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 m is large. Also observe that MSTMAP-C can be faster than MSTMAP+MERGEMAP.

Figure 5-LEFT shows that (1) the consensus maps obtained by MSTMAP+MERGEMAP are significantly more accurate than the ones produced by JOINMAP and that (2) MSTMAP-C have comparable accuracy to JOINMAP. We believe that the same conclusions can be derived for larger datasets.

In order to investigate the extend of the advantages brought upon by the tool 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 Figure 4 (data not shown), whereas the running time for MSTMAP-C is always very short, within a few minutes regardless of the size of the input. Similar conclusions can be drawn for the cases where $K = 8$, $K = 10$ and $K = 12$ (data not shown due to space restrictions).

7. CONCLUSIONS

We presented a suite of novel algorithms to construct a consensus map from a set of genetic maps given as DAGs. The individual genetic maps are merged into a consensus graph on the basis of shared vertices. Cycles in the consensus graph indicate ordering conflicts among the individual maps, which are resolved according to a parsimonious approach that takes into account two types of errors that may be present in the individual maps, namely, local reshuffles and global displacement. From the set of experimental results reported here, we can conclude that our tool outperforms JOINMAP both in terms of accuracy and running time.

References

1. Jansen J, de Jong AG, van Ooijen JW. Constructing dense genetic linkage maps. *Theor Appl Genet 102* (2001), 1113–1122.
2. Schiex T, Gaspin C. CARTHAGENE: Constructing and joining maximum likelihood genetic maps. In *Proceeding of ISMB* (1997), pp. 258–267.
3. Iwata H, Ninomiya S. AntMap: constructing genetic linkage maps using an ant colony optimization algorithm. *Breeding Science 56* (2006), 371–377.
4. Os HV, Stam P, Visser RGF, Eck HJV. RECORD: a novel method for ordering loci on a genetic linkage map. *Theor Appl Genet 112* (2005), 30–40.
5. Cartwright DA, Troggio M, Velasco R, Gutin A. Genetic mapping in the presence of genotyping errors. *Genetics 174* (2007), 2521–2527.
6. Wu Y, Bhat PR, Close TJ, Lonardi S. Efficient and accurate construction of genetic linkage maps from noisy and missing genotyping. In *Proceeding of WABI* (2007), pp. 395–406.
7. Dib C, Faure S, Fizames C, *et al.* A comprehensive genetic map of the human genome based on 5,264 microsatellites. *Nature 380* (1996), 152–154.

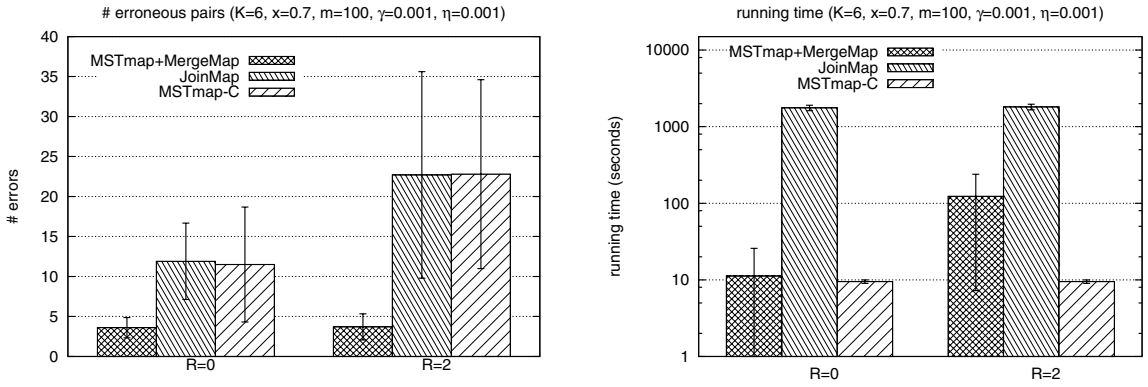


Fig. 5. Comparison between MSTMAP+MERGEMAP, JOINMAP and MSTMAP-C in terms of number of erroneous marker pairs (LEFT) and running time (RIGHT) for $R = 0$ and $R = 2$ respectively. The rest of the parameters are as shown in the title of the figures. Each bar represents an average of ten runs and the error bar indicates the standard deviation.

Table 1. Comparison between MSTMAP+MERGEMAP and MSTMAP-C for $K = 6, x = 0.7$. Each number in the table is the average of the results obtained from ten independent runs.

	# erroneous pairs for MSTMAP+MERGEMAP				# erroneous pairs for MSTMAP-C			
	$\eta = 0.001$ $\gamma = 0.001$	$\eta = 0.005$ $\gamma = 0.005$	$\eta = 0.01$ $\gamma = 0.01$	$\eta = 0.02$ $\gamma = 0.02$	$\eta = 0.001$ $\gamma = 0.001$	$\eta = 0.005$ $\gamma = 0.005$	$\eta = 0.01$ $\gamma = 0.01$	$\eta = 0.02$ $\gamma = 0.02$
$R = 0$								
$m = 100$	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

8. Ihara N, Takasuga A, Mizoshita K, *et al.* A comprehensive genetic map of the cattle genome based on 3802 microsatellites. *Genome Research 14* (2004), 1987–1998.
9. Dietrich WF, Miller JC, Steen RG, *et al.* A genetic map of the mouse with 4,006 simple sequence length polymorphisms. *Nature Genetics 7* (1994), 220 – 245.
10. Steen RG, Kwitek-Black AE, Glenn C, *et al.* A high-density integrated genetic linkage and radiation hybrid map of the laboratory rat. *Genome Research 9*, 6 (1999).
11. Jackson BN, Aluru S, Schnable PS. Consensus genetic maps: A graph theoretic approach. In *Proceeding of CSB* (2005), pp. 35–43.
12. Beavis WD, Grant D. A linkage map based on information from four f_2 populations of maize (*Zea mays L.*). *Theor Appl Genet 82*, 5 (Oct 1991), 636–644.
13. Stam P. Construction of integrated genetic linkage maps by means of a new computer package: Joinmap. *The Plant Journal 3* (1993), 739–744.
14. Yapa IV, Schneiderb D, Kleinberg J, *et al.* A graph-theoretic approach to comparing and integrating genetic, physical and sequence-based maps. *Genetics 165* (Dec 2003), 2235–2247.
15. Plotkin SA, Shmoys DB, Tardos E. Fast approximation algorithms for fractional packing and covering problems. In *Proceeding of FOCS* (1991), pp. 495–504.
16. Girvan M, Newman MEJ. Community structure in social and biological networks. *PNAS 99* (Jun 2002), 7821–7826.