Efficient Data Reduction for Large-Scale Genetic Mapping

Veronika Strnadová-Neeley
UCSB and LBL
veronika@cs.ucsb.edu

Aydın Buluç
Computational Research Division, LBL
abuluc@lbl.gov

Jarrod Chapman
Joint Genome Institute, LBL
jchapman@lbl.gov

John R. Gilbert
Computer Science Department, UCSB
gilbert@cs.ucsb.edu

Joseph Gonzalez
EECS Department, UCB
jegonzal@eecs.berkeley.edu

Leonid Oliker
Computational Research Division, LBL
loliker@lbl.gov

ABSTRACT

We present a fast and accurate algorithm for reducing large-scale genetic marker data to a smaller, less noisy, and more complete set of *bins*, representing uniquely identifiable locations on a chromosome. Our experimental results on real and synthetic data show that our algorithm runs in near-linear time, allowing for the analysis of *millions* of markers. Our algorithm reduces the problem scale while preserving accuracy, making it feasible to use existing genetic mapping tools without resorting to complex, time-intensive preprocessing methods to filter or sample the original data set. Additionally, our approach also decreases the uncertainty in genotype calls, improving the quality of the data. Preliminary results demonstrate that existing methods for marker ordering designed for the small scale settings perform with equivalent accuracy when given our reduced bin set as input.

Categories and Subject Descriptors

Applied computing [**Genomics**]: Computational Genomics

General Terms

Genetic mapping, scalable algorithms, data reduction

1. INTRODUCTION

Genetic maps are essential tools for analyzing DNA sequence data, not only providing a blueprint of the genome but also unlocking linkage patterns between *genetic markers*, chromosomal regions with more than one sequence variant. Studying these linkage patterns enables diverse applications to problems in health, agriculture, and the study of biodiversity. In the past few years, next generation sequencing technologies [16] have led to tremendous growth in the amount of genetic markers available to geneticists. However, our ability to parse, analyze, and extract knowledge from this data has not kept pace with the power to generate it. Despite advances in de-novo genome assembly, genetic maps are still critical to many non-trivial assemblies. Genetic maps are commonly used to complete the assemblies of complex and repetitive genomes, by anchoring many disconnected scaffolds to chromosomes.

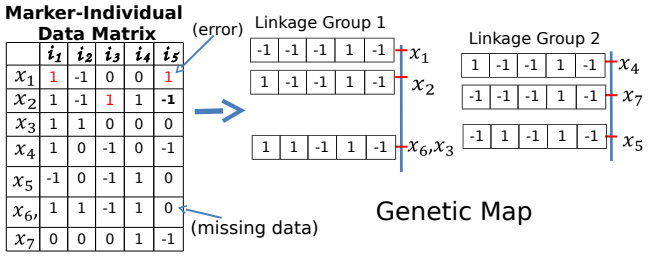

Figure 1: Producing a genetic map from a data matrix of homozygous genetic markers.

We make the following three important contributions to genetic mapping: (1) We efficiently process genetic marker data sets orders of magnitude greater than any known genetic mapping tool in nearly-linear time, reducing the data to a size that can be easily managed by traditional mapping tools. (2) We show that our method discovers the fundamental resolution of genetic marker data nearly perfectly, even for very noisy data sets with many missing values. (3) By reducing genetic marker data to its fundamental resolution, we preserve information relevant to genetic map construction while eliminating errors and uncertainty.

Genetic mapping is generally described as a three-step process [5]: (1) Cluster genetic markers into sets that represent *linkage groups*, ideally in one to one correspondence with chromosomes. (2) Order the markers within each linkage group. (3) Determine the genetic spacing between ordered markers within each linkage group. Figure 1 illustrates the outcome of genetic mapping for a data matrix of $M = 7$ *homozygous* genetic markers from a *mapping population* of $n = 5$ individuals. Homozygosity guarantees that each marker can only take on one of two values for a given individual. Each marker x_j is given as a row in the data matrix, with the number of columns equal to mapping population size. Homozygous markers are abundant for many population types used in genetic mapping (e.g. doubled-haploid (DH) and advanced recombinant inbred line (RIL) popu-

BCB'15 September 09 - 12, 2015, Atlanta, GA, USA

DOI: http://dx.doi.org/10.1145/2808719.2808732 ...$15.00.

lations) [4, 5, 22]. Each column represents the homozygous state of a genetic marker, +1 or -1, or 0 for missing data, for each individual. The end-product of the genetic mapping pipeline is a blueprint of the genome, showing which markers belong to which linkage group, and within each linkage group, genetic map locations correspond to n-dimensional vectors that indicate genotypes at these locations. Note that, as is the case for x_3 and x_6 in the figure, more than one marker can map to the same location.

Figure 1 also alludes to the computational difficulties associated with generating a high-quality, large-scale genetic map. First, large amounts of missing data make it difficult to cluster markers into linkage groups using simple similarity metrics. Second, even small genotyping error rates cause some genotypes to appear as *opposite* of their true value (i.e. +1 flips to -1, or visa-versa, in the data matrix), adding a layer of difficulty for analyzing marker linkage patterns. Third, as the amount of data (rows in the data matrix) grows into the millions, traditional polynomial-time algorithms that were designed for small-scale genetic mapping are not sufficiently fast to process the data. Errors and missing data values in large data sets have been shown to dramatically affect the quality of the genetic map produced by existing tools ([22]). Therefore, our algorithm not only improves the efficiency of the mapping process by reducing the data set size, but by reducing errors and missing values, has the potential to significantly improve the quality of the final map.

In this work, we aim to reduce genetic mapping data to its fundamental resolution, which is independent of the data set size. Differently from a map's *density*, which is limited by the number of genetic markers, the *resolution* of a map depends on the number of individuals in the mapping population. We propose an algorithm that reduces the scale of genetic marker data to its fundamental resolution, thereby reducing noise and enabling classic bioinformatic algorithms to be efficiently applied to new large-scale data sets with increased robustness. By running experiments on synthetic data, as well as real data consisting of barley and the highly complex wheat genome, we demonstrate that our approach can perform data reduction in near-linear time. Additionally, we show that our novel algorithm preserves high accuracy compared with the gold standard, and produces a reduced bin set whose ordering is almost identical to the solution obtained by MSTMap [22] given the exact bins as input.

2. PROBLEM DEFINITION

We assume that M homozygous markers are given to us as input. Thus each marker x_j is a vector with each entry $x_j(i)$ represented by values +1 or -1, or 0 for missing data. Genetic maps are built by computing probabilities of linkage between genetic markers, based on similarities between genotypes of individuals in the mapping population. In the absence of errors, differences in genotypes for a single individual at two different markers are based solely on the concept of *recombination*. Typically, one to two recombination events occur per chromosome per individual [9,11]. For two markers x_j, x_l on the same chromosome, if no recombination event occurs between them, then the vectors will be *indistinguishable* (i.e. $x_j = x_l$) in the data matrix.

This insight is the key to our argument that a fundamental resolution exists for any fixed population size, independent

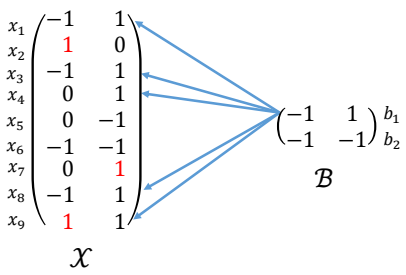

Figure 2: Input data illustration. Here, $\mathcal{B}$ is a set of 2 distinct 2-vectors which generated the 9 markers $x_1, \ldots, x_9$. The arrows indicate the markers that belong to bin_1, corresponding to bin vector b_1. The rest of the x_j's belong to bin_2. Red entries represent genotyping errors and 0's indicate missing data.

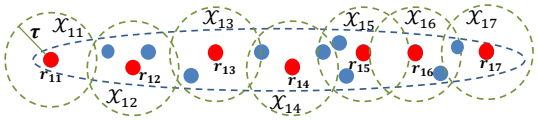

Figure 3: Visualization of our BubbleCluster algorithm. Each linkage group is represented as a cluster spanned by a set of sketch (red) points $r_{\kappa q}$, such that each marker x_j (blue) is within a threshold LOD score τ of at least one sketch point. $\mathcal{X}_{\kappa q}$ represents the sets of markers closest to sketch point $r_{\kappa q}$.

of the data set size, and leads to the effectiveness of our algorithm. The number of uniquely identifiable locations on a genetic map, referred to *bin vectors*, is limited by the number of recombination events that occur in the population, which in turn is limited by the number of chromosomes (k) in the genome and the number of individuals (n) in the mapping population. The number of recombination events is $O(kn)$, with the constant in the big-O notation typically close to 1 [1, 5].

The idea that many markers may map to the same location in a genetic map is well known in the bioinformatics community [5, 12, 15, 18, 22]. The novelty in our approach is to frame genetic mapping as a data reduction problem, and to leverage large amounts of data to discover the bin vectors that establish the fundamental resolution size.

The data reduction problem is stated as follows. The goal is to find the set $\mathcal{B}$ of distinct *bin vectors* $b_1, \ldots, b_q \in \{-1, 0, 1\}^{1\times n}$ that generated the data $\mathcal{X} \in \{-1, 0, 1\}^{M\times n}$. We assume a fixed error rate ϵ in the data. Thus, if b_k generated x_j, then $x_j(i) = b_k(i)$ with probability $1 - \epsilon$ and $x_j(i) \neq b_k(i)$ with probability ϵ. We further assume a fixed missing rate μ, giving an independent probability to the event that $x_j(i) = 0$. Finally, we assume that $M \gg n$. In addition to the bin vectors b_q, we seek to identify the set bin_q of markers generated by each b_q. Figure 2 gives a tiny example of 9×2 input data $\mathcal{X}$, generated from the 2×1 bin vectors $b_1, b_2 \in \mathcal{B}$. Here the markers x_1, x_3, x_4, x_8 and x_9 comprise the set bin_1, and the remainder of the markers belong to bin_2.

Note that the number of genetic markers can be orders of magnitude larger than the number of bins. In other words, for large data sets, multiple markers $x_j \in \mathcal{X}$ will necessarily map to the same location $b_k \in \mathcal{B}$ in the genetic map by the Pigeonhole Principle. Thus, even with high missing data rates and realistic error rates, large amounts of data allow

us to eliminate much of the uncertainty in bin vector values, as most markers x_j in the same bin will agree in genotype values for all non-missing entries $x_j(i) \neq 0$. The challenge is to reduce the M input markers to $O(kn)$ bins.

Our previous work presented the BubbleCluster algorithm [19], which demonstrated that large sets of markers can be efficiently partitioned into clusters by forming a linked chain of *sketch points* that span each cluster. Markers that achieve a LOD score of τ or greater with a particular sketch point are genetically linked with high probability. A LOD score between two homozygous markers x_j and x_k is given by the formula: $LOD(x_j, x_k) = \log_{10}\left(\frac{\left(\frac{o}{\eta_{jk}}\right)^{o}\left(\frac{\eta_{jk}-o}{\eta_{jk}}\right)^{\eta_{jk}-o}}{2^{\eta_{jk}}}\right)$ where η_{jk} is the number of entries i for which both $x_j(i) \neq 0$ and $x_k(i) \neq 0$, and o is the number of entries i for which $x_j(i) \neq 0$, $x_k(i) \neq 0$, and $x_j(i) \neq x_k(i)$. Thus η is the number of non-missing entries shared by both x_j and x_k, and o is the number of those non-missing entries for which x_j and x_k disagree in genotype. BubbleCluster can be thought as a form of density-based clustering, shown in Figure 2, which assigns markers to clusters, while also partitioning the markers within clusters into groups $\mathcal{X}_{\kappa q}$. Each $\mathcal{X}_{\kappa q}$ represents the set of markers with high probability of genetic linkage with a particular sketch point $r_{\kappa q}$, corresponding to the qth sketch point in cluster κ. We use the sets $\mathcal{X}_{\kappa q}$ as a starting point, or coarse clustering, to obtain a more refined clustering that will enable us to discover the bin vectors in our data set.

3. ALGORITHM DESCRIPTION

We now present our algorithm that follows a course-to-fine clustering approach. The algorithm first partitions markers into linkage groups, then obtains a coarse clustering within each linkage group into sets $\mathcal{X}_{\kappa q}$, and uses these sets to estimate the error rate ϵ in our data set. Then, the estimated ϵ is used to recursively bisect each set $\mathcal{X}_{\kappa q}$ until the fraction of errors within a set is close to our error estimate for each individual i. Each set of markers returned by our algorithm represents a bin bin_k, and the bin vector values $b_k(i)$ are assigned intuitively, becoming +1 if for a majority of the markers $x_j \in \text{bin}_k$, $x_j(i) = +1$, and -1 otherwise. We expand on the details of our algorithm below.

3.1 Estimating the Error Rate

Algorithm 1 describes the process used to derive bins from a set of markers $\mathcal{X}$. First, we rely on Phase I of our previously published BubbleCluster algorithm to partition the marker set $\mathcal{X}$ into clusters $\mathcal{C}$ along with respective sketch point sets $\mathcal{R}$ (Line 1). The sketch point sets obey the property that for every marker x_j in a linkage group cluster C_κ, x_j is within a threshold LOD score τ of at least one sketch point r_p in the sketch point set R_κ corresponding to C_κ. Furthermore, every sketch point r_p is within τ of at least one other sketch point in R_κ (see Figure 2).

The output sketches $R_1, \ldots, R_\kappa$ are used to further partition $\mathcal{X}$ into sets $\mathcal{X}_{\kappa q}$ (Line 5). In this step, we re-assign each marker x to the sketch point r that achieves the highest LOD score with x, breaking ties randomly. Thus, every set $X_{\kappa q}$ represents a set of markers that are all highly likely to be close together on the chromosome. In the next step of our algorithm, we use this assumption to estimate the genotyping error rate in our data and to find the bins.

To estimate the error rate ϵ, we assume that ϵ is a fixed

Algorithm 1: Algorithm FindAllBins

Inputs : $\mathcal{X} = \{x_1 \ldots x_M\}$=set of markers
Output: bins=a set of sets of markers b_k, where each b_k represents a bin,
$\mathcal{F}$=set of bin vectors corresponding to each bin set

1 $(\mathcal{R}, \mathcal{C}) = \text{BubbleCluster}(\mathcal{X}, \tau, \eta)$;
2 mles $= \emptyset$; $\mathcal{F} = \emptyset$;
3 **for** $\kappa = 1 \ldots |\mathcal{R}|$ **do**
4 **for** $q = 1 \ldots |R_\kappa|$ **do**
5 $\mathcal{X}_{\kappa q} = \{x \in \mathcal{X} | q = \max_l LOD(x, r_l \in R_\kappa \in \mathcal{R})\}$;
6 $\text{mles}_{\kappa q} = \text{median}(\text{getMLEs}(\mathcal{X}_{\kappa q}))$;
7 $\hat{\epsilon} = \left(\sum_{\kappa=1}^{|\mathcal{R}|}\sum_{q=1}^{|R_\kappa|}|\mathcal{X}_{\kappa q}|\text{mles}_{\kappa q}\right) / \sum_{\kappa=1}^{|\mathcal{R}|}\sum_{q=1}^{|R_\kappa|}|\mathcal{X}_{\kappa q}|$;
8 **if** $\hat{\epsilon} == 0$ **then**
9 $\hat{\epsilon} = 0.00001$;
10 $\alpha = \hat{\epsilon}(1-\mu)M$;
11 bins $= \emptyset$;
12 **for** $\kappa = 1 \ldots |\mathcal{R}|$ **do**
13 **for** $q = 1 \ldots |R_\kappa|$ **do**
14 bins = bins $\cup$ RecursiveBisectionBins$(\mathcal{X}_{\kappa q}, \hat{\epsilon}, \alpha)$;
15 **for** $j = 1 \ldots |\text{bins}|$ **do**
16 $\mathcal{F} = \mathcal{F} \cup \text{getBinGenotypes}(\text{bin}_j)$;
17 return $(\mathcal{F}$,bins);

constant, and that for every marker x_j, the probability that the ith individual entry $x_j(i)$ is assigned to the incorrect genotype is independently ϵ. Thus, the distribution of errors in the data set follows a Bernoulli distribution. Therefore, assuming one recombination per individual per chromosome, the maximum likelihood estimate (MLE) of the error rate for markers in the same bin, at one individual i, is $\frac{o}{\eta}$, where η is the number of markers for which the genotype is defined (not missing, $x_j(i) \neq 0$), and o represents the number of recombinant genotypes (number of markers for which the genotype is opposite that of the bin vector, $x_j(i) \neq b(i), x_j(i) \neq 0$). Having no prior information, the `getMLEs` function on Line 6 returns a vector of MLE error estimates for each individual, where each MLE is $\frac{o}{\eta}$ for that individual, simply assuming that o is number of minimally-occurring genotypes for i.

We can view the coarse-grained clustering of markers into sketch point sets $\mathcal{X}_{\kappa q}$ as a grouping containing significantly fewer bins than the original data. However, it is a challenging problem to partition each $\mathcal{X}_{\kappa q}$ into bins. We can nonetheless assure that no $\mathcal{X}_{\kappa q}$ contains markers more than $n/2$ bins by setting the LOD threshold τ in the BubbleCluster algorithm high enough and thus the *median* MLE error rate estimate will approach the true error rate for every $X_{\kappa q}$ as the amount of non-missing data in each $\mathcal{X}_{\kappa q}$ increases. After collecting the MLE's for each individual i in each set $\mathcal{X}_{\kappa q}$ (Line 6), the error estimate $\hat{\epsilon}$ is set to a weighted average of the median MLE's from each sketch point bubble $\mathcal{X}_{\kappa q}$, with weights equal to the normalized sizes of $\mathcal{X}_{\kappa q}$ (Line 7). Weights thus give $\mathcal{X}_{\kappa q}$'s with more data increased influence over the selection of $\hat{\epsilon}$.

Because the errors follow a Bernoulli distribution, we use the Beta distribution, its conjugate prior, to quantify our level of confidence in $\hat{\epsilon}$, our estimate of ϵ. Recall that a Beta distribution takes two parameters, α and β, and has mean $\frac{\alpha}{\alpha+\beta}$ and variance $\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$. Thus, as α and β increase, the variance decreases, effectively placing more confidence in the mean. In Line 10, we consider α to be the prior belief in the number of errors that occur in the data. We

Algorithm 2: Algorithm RecursiveBisectionBins

Inputs : $\mathcal{Y} = \{y_1 \dots y_w\}$=set of markers, where each y_j is a vector of dimension n with entries $y_j(i) \in \{-1, 0, 1\}$, $\hat{\epsilon}$ = estimated error rate, α = parameter of Beta distribution

Output: bins = set of sets of markers belonging to the same bin vector

```
1  bins= ∅; Z = ∅;
2  β = α/ε̂ − α; σ = sqrt( αβ / ((α+β)^2 (α+β+1)) );
3  maps = getMAPs(Y);
4  M = |Y|;
5  maxMap = max_{i=1,...,n} map(i);
6  if maxMap > ε + 2σ then
7      maxMapInd = i, s.t. map(i) = maxMap;
8      bina = ∅; binb = ∅;
9      //Bisection Step:
10     for j = 1, ... M do
11         if y_j(i) == 1 then
12             bina = bina ∪{y_j};
13         else if y_j(i) == −1 then
14             binb = binb ∪{y_j}
15         else
16             Z = Z ∪ {y_j};
17     //Zero-entry assignment step:
18     b_a =getBinGenotypes(bina);
       b_b =getBinGenotypes(binb);
19     for y_j ∈ Z do
20         if p(y_j ∈ bina) < p(y_j ∈binb) then
21             Move y_j from Z to binb
22         else if p(y_j ∈ binb) < p(y_j ∈bina) then
23             Move y_j from Z to bina
24         else
25             Move y_j from Z randomly to bina or binb
26     //Error-correction Step:
27     b_a =getBinGenotypes(bina);
       b_b =getBinGenotypes(binb);
28     for y_j ∈ bina do
29         if p(y_j ∈ bina) < p(y_j ∈binb) then
30             Move y_j from bina to binb
31     for y_j ∈ binb do
32         if p(y_j ∈ binb) < p(y_j ∈bina) then
33             Move y_j from binb to bina
34     if bina = ∅ then
35         return binb;
36     else if binb = ∅ then
37         return bina;
38     else
39         return( RecursiveBisection(bina,ε̂, α)
           ∪RecursiveBisection(binb, ε̂, α) );
40 else
41     return Y;
```

set the α parameter to be the number of markers in the data set multiplied by the estimated error rate $\hat{\epsilon}$ and by one minus the missing rate μ. Multiplying the number of markers by one minus the missing rate gives the total non-missing entries entries $x_j(i) \neq 0$ that are expected to be seen in the data set. Multiplying this value further by $\hat{\epsilon}$ gives the prior number of errors we assume are in the data set. One caveat is that α must be greater than 0 to have a well-defined Beta distribution. Thus, if the estimated error rate is exactly 0, then we set $\hat{\epsilon}$ to a small constant c_ϵ, representing an extremely low error rate (Line 9). In practice we set $c_\epsilon = 0.00001$.

Lines 11 - 14, loop through every cluster C_κ and every sketch point $r_{\kappa q}$ using Algorithm 2 to recursively split each sketch point set $\mathcal{X}_{\kappa q}$ into bins. The parameters $\hat{\epsilon}$ and α are used determine when to stop splitting and assign bin vector values, as we will explain below.

3.2 Recursive Bisection with Error Correction

Algorithm 2, details the `RecursiveBisectionBins` function, which recursively bisects an input set of markers $\mathcal{Y}$, attempting to keep together markers belonging to the same bin. Recursion stops when the a posteriori estimate of the error rate for each individual in the input set of markers is close enough to the estimated rate $\hat{\epsilon}$. The output is a set of bins that markers in $\mathcal{Y}$ map to.

Recall that the input set $\mathcal{Y}$ will be one of the sketch point sets $\mathcal{X}_{\kappa q}$ from Algorithm 1, and thus we assume there are few bins represented by the markers in $\mathcal{Y}$ relative to the number of bins represented by the entire data set $\mathcal{X}$. The estimated error rate $\hat{\epsilon}$ and the parameter α are also given, which are used to set the prior Beta distribution over ϵ.

The `getMAPs` function (Line 3) finds maximum a posterior (MAP) estimate of the error rate for each individual i, based on a prior Beta distribution. The prior Beta distribution is set to have mean $\hat{\epsilon}$ using the parameter α. Thus, β must be set to $\frac{\alpha}{\hat{\epsilon}} - \alpha$ (Line 2). The variance of the Beta distribution is then additionally computed. Given that the posterior probability distribution over ϵ is proportional to the likelihood of the data $\mathcal{Y}$ times the prior distribution over ϵ, $p(\epsilon|y_1, \dots, y_\eta) \propto p(y_1, \dots, y_\eta|\epsilon)p(\epsilon)$, if all the markers $y_j \in \mathcal{Y}$ were generated from the same bin b, then: $p(\epsilon|y_1, \dots, y_\eta) \propto (\epsilon)^o(1-\epsilon)^\eta \epsilon^{\alpha-1}(1-\epsilon)^{\beta-1} = (\epsilon)^{o+\alpha-1}(1-\epsilon)^{\eta+\beta-1}$, where o represents the number of entries $y_j(i)$ that are opposite from the true bin value $b(i)$, and η is total number of non-missing entries for individual i (the number of entries $y_j(i) \neq 0$). To compute a MAP estimate $\bar{\epsilon}$ for individual i, we set the derivative of $p(\epsilon|y_1, \dots, y_{nm})$ to 0 to get: $\text{map}(i) = \frac{\alpha+o-1}{\beta+\alpha+\eta-2} = \bar{\epsilon}$. The higher the ratio $\frac{o}{\eta}$, the less likely that the set of markers $y_1, \dots y_\eta$ have been drawn from the same bin. Additionally, more non-missing data (higher η) gives more weight to the MLE estimate of the error rate over the prior estimate, helping us leverage additional data for better estimates. In Line 5, we select the maximum MAP estimates of the error rate.

If the maximum MAP estimate of the error rate is not within two standard deviations of the mean (Line 6), then it can be reasonably assumed that all the markers in $\mathcal{Y}$ could not have come from the same bin. Since the normal distribution with mean and variance equal to that of the Beta distribution is a good approximation to the Beta distribution near the tails, a MAP larger than this threshold indicates that the most likely error estimate for at least one individual is larger than approximately 97% of the error rates that we can expect to see based on our prior distribution. In other words, the data $\mathcal{Y}$ for individual i is inconsistent with our prior assumption. Thus, we assume that $\mathcal{Y}$ contains more than one bin, and we use the values of markers $y \in \mathcal{Y}$ at individual i to split $\mathcal{Y}$ into two subsets, called bina and binb. For a given marker y, if $y(i) = 1$, we assign y to bina, if $y(i) = -1$ we assign it to binb, and if $y(i) = 0$ (missing) it is temporarily assigned to a set $\mathcal{Z}$ (Lines 7 - 16).

To assign the markers y in $\mathcal{Z}$, we temporarily assume that bina and binb represent true bins, and calculate the probability that y belongs to either bin. To do so, the `getBinGenotypes` function (Algorithm 3) first assigns a bin vector b_a to bina and a bin vector b_b to binb. The `getBinGenotypes` function loops over all individual entries i and assigns a genotype to the ith entry of the bin vector if the MAP estimate of the error rate at the ith entry is within 2 standard deviations of the mean of the Beta distribution. If so, `getBinGenotypes` assigns the entry to the maximally-occuring genotype at that individual entry. If not, then `getBinGenotypes` assigns a 0 to this entry, effectively stating that there is not enough evidence to assign this entry a value.

Then, using the assumption that errors are binomially distributed the probability that y belongs to bina becomes:

$$p(y \in \text{bina}) = \prod_{i=1}^{n} (1-\epsilon)^{b_a(i)=y(i), y(i)\neq 0} \epsilon^{b(i)\neq y(i), y(i)\neq 0} \quad (1)$$

and analogously for binb. The `for` loop on line 19 assigns each y in $\mathcal{Z}$ to the side of the partition which yields a greater probability, breaking ties randomly.

After the Zero-Assignment phase, the initial input set $\mathcal{Y}$ has been partitioned into two subsets that are more consistent with the error rate for one individual. However, only one bit of information per marker was used to assign it to one of these subsets. An error in this bit would thus cause the marker to be assigned to the wrong set, and the error would "pass down" the recursion. Therefore, after bisecting the markers we add an error correction phase (Line 26).

The Error-correction phase calculates, for each marker y, its probability of belonging to each side of the bisection. We again use the `getBinGenotype` function to assign a bin vector to each side of the partition and then calculate the probabilities according to equation 1. If for any marker $y_j \in$bina it is found that $p(y_j \in \text{bina}) < p(y_j \in \text{binb})$, y_j is moved from bina to binb (Lines 29-30), and vice-versa (Lines 32 - 33).

After the error correction phase, if either bina or binb is empty (Lines 34 and 36), the bin vector corresponding to the opposite bin is returned. Here we assume that we attempted to split $\mathcal{Y}$ based on the most informative individual, in terms of maximum MAP estimate of the error rate, but it is more likely that the set of markers represents a bin than that it is a combination of bins. Therefore, we return the bin corresponding to the entire set of markers. Otherwise, the code recursively partitions bina and binb using the same algorithm (Line 39), until each column in the input set of markers is consistent with the error rate. In this case (Line 40), we return the input set as a bin. In other words, all the markers are assumed to have come from the same bin.

3.3 Run Time Analysis

The BubbleCluster algorithm runs in $O(|\mathcal{H}| \log |\mathcal{H}| + M\bar{R})$, where M is the data set size, $\bar{R}$ the number of sketch points found and $|\mathcal{H}|$ the number of high-quality markers used in the sketch point-building phase [19]. We note that $|\mathcal{H}|$ is typically much smaller than M, and in our experiments we this was indeed the case. The number of sketch points $\bar{R}$ is bounded by kn, the number of individuals n times the number of linkage groups k, and is thus much smaller than both $|\mathcal{H}|$ and M. The running time of the error estimate in line 7 of Algorithm 1 is linear in M times the number of sketch points, $\bar{R}$. The `getBinGenotypes` function (Algorithm 3) runs in time that is linear in the input set size.

Algorithm 3: Algorithm getBinGenotypes

Inputs : $\mathcal{X} = \{x_1 \dots x_M\}$=set of markers, where each x_j is a vector with each entry $x_j(i) \in \{-1, 0, 1\}$, and each x_j has n individual entries, ϵ = estimated error rate, α = parameter of Beta distribution

Output: b = the genotypes of the bin assumed to have generated $\mathcal{X}$, with $b(i)$ = genotype of individual i

```
1 maps = getMAPs(X);
2 β = α/ε − α ;
3 σ = sqrt(αβ / ((α+β)^2 (α+β+1)));
4 mapthreshold = ε + 2σ;
5 for i = 1 ... n do
6     if maps(i) < mapthreshold then
7         b(i) = min(o, η−o) / η;
8     else
9         b(i) = 0;
```

The running time of recursive bisection binning is complex to analyze, because the depth of recursion depends on the number of times a MAP estimate for a particular individual will be greater $\hat{\epsilon} + 2\sigma$. We note, however, that if we rely on our assumption that the markers in a sketch point set $\mathcal{X}_{\kappa q}$ are very close together on the genome, then for a majority of individuals, the MAP estimate will be consistent with the error rate. Therefore, only a few splits are required before all markers in a set $\mathcal{Y}$ agree, up to $\hat{\epsilon} + 2\sigma$, in individual entries. Thus if we are successful in producing a coarse-grained clustering of the markers within each cluster into sketch point sets $\mathcal{X}_{\kappa q}$, such that each sketch point set contains at most n/λ bins, with $\lambda > 1$, then the depth of recursion is bounded by n/λ with high probability. In this case, recursive binning takes time proportional to $\sum_i \sum_j |\mathcal{X}_{\kappa q}| n/\lambda = \frac{nM}{\lambda}$, and the entire algorithm requires a runtime of $O(|\mathcal{H}| \log |\mathcal{H}| + 2M\bar{R} + \frac{nM}{\lambda})$. Since $n, \bar{R}$, and $|\mathcal{H}|$ are much smaller than M, our algorithm will run in time that is linear in the input size, *assuming we obtain a good coarse-grained clustering.* Experimental results in Section 4 validate that this is empirically correct.

4. EXPERIMENTAL RESULTS

We use several metrics to evaluate the quality of our binning algorithm using both real and simulated data. Our code is written in C++, single threaded, and available at http://gauss.cs.ucsb.edu/home/index.php/code. We ran experiments on a single core of one 12-core compute node of NERSC's Carver system, with Intel Xeon X5650 processors running at 2.67GHz, and 10GB of memory. In reporting our experimental results we will denote by f_i the ith bin vector found by our algorithm and the set of found bins $\mathcal{F}$. For experiments on simulated data, we will use the notation b_l to denote the lth golden standard bin vector.

4.1 Simulated Data Sets

To evaluate the quality of our binning algorithm, we use simulated data sets ranging in size from 12.5K to 1.6M markers, with realistic and challenging missing rates [4, 6, 13, 15, 20] μ = 0.15, 0.35, 0.50 and error rates ϵ = 0.1%, 0.5%, 1%. All data sets of size less than 1.6M are randomly chosen subsets of the 1.6M marker set for a fixed (μ, ϵ) pair.

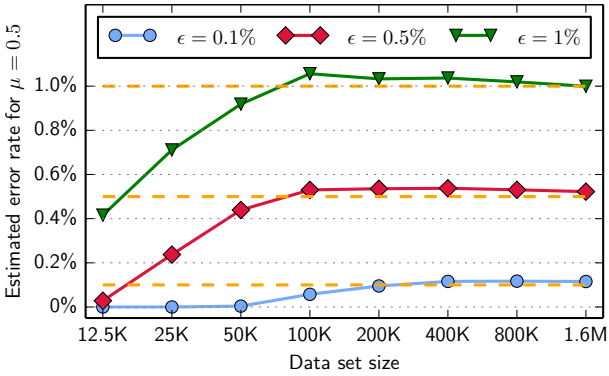

Figure 4: Estimated error rates $\hat{\epsilon}$ for missing rate $\mu = 0.5$ with increasing data set sizes. As the data set size grows, $\hat{\epsilon}$ approaches the true rate ϵ for each case tested: ϵ = 0.1%, 0.5%, and 1%.

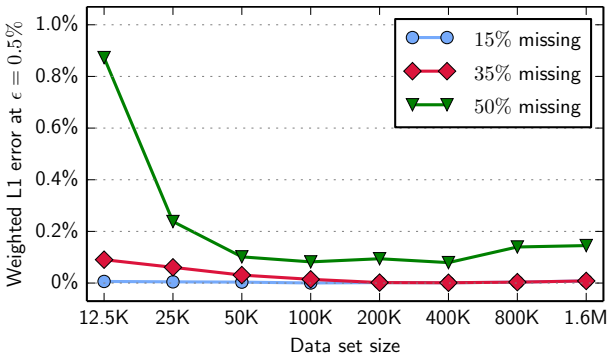

Figure 5: Weighted L_1 error for fixed error rate ϵ = 0.5% and missing rates $\mu = 0.15, 0.35, 0.50$. In all cases, average L_1 distance from a found bin vector to the golden standard was less than 1.

Experiments on real genomic data for barley and wheat are presented in Section 4.6.

To approximate realistic genetic map sizes, and for a thorough comparison between varying missing rates and error rates, we set the number of chromsomes k to 10 and the number of individuals in the mapping population n to 100 in all experiments. For each chromosome, we simulated n bin vectors in the following manner. An initial bin vector b_1 was generated by randomly assigning $b_1(i)$ to -1 or +1 with equal probability for all n individuals. Next, for $j \in \{2, \ldots, n\}$ b_j was generated by randomly choosing an individual $i \in \{1, \ldots, n\}$, and flipping the sign of the ith individual in b_{j-1}. Thus, each bin vector $b_j, j \in \{2, \ldots n-1\}$ differs in exactly one individual entry from b_{j-1} and from b_{j+1} (and b_1 and b_n differ in exactly one individual from b_2 and b_{n-1}, respectively). Note that this procedure corresponds to simulating one recombination per chromosome per individual, a common and realistic feature of true mapping populations [1,5,9,11,22]. Importantly, this also means that the fundamental resolution of our data was 1000 in all simulated experiments.

Once the golden standard bin set has been created, we simulate the markers. First, a bin vector b_q is chosen uniformly at random. Then, as long as the number of simulated markers is less than M, a marker is generated from the qth bin. A marker x_j is generated from bin vector b_q by setting the entry $x_j(i)$ is to $b_q(i)$ with probability $1-\epsilon$, and to $-1 * b_q(i)$ with probability ϵ. The entry $x_j(i)$ is set to 0 with independent probability μ. Thus, as we increase the data set sample size M, more markers are sampled from each bin at the same rate in expectation.

4.2 Error Estimation & Reduction Accuracy

To evaluate the quality of our algorithm, we quantify how well the true error rate ϵ compares with our estimate $\hat{\epsilon}$, and report three metrics to evaluate how accurately our algorithm reduces a set of M markers to a set of $|\mathcal{B}|$ bins. *Accuracy* is defined as follows: to each bin vector f_j discovered by our algorithm, we assign one golden standard bin vector b_l based on closest L_1 distance, breaking ties randomly. Then, we report accuracy as the average over all found bin vectors f_j of the number of matching individual entries where $f_j(i) = b_l(i)$, divided by the total number of non-missing entries for which $f_j(i) \neq 0$. In other words,

$$\text{accuracy} = \sum_{j=1\ldots|\mathcal{F}|} \frac{1}{\eta_j} \sum_{i=1..n} (f_j(i) == b_l(i))$$

where for each f_j, b_l is the closest golden standard bin in terms of L_1 distance and η_j is the number of non-missing entries in f_j. Thus, if accuracy is 100%, then *every* found bin vector f_j corresponds exactly to some golden standard bin vector b_l.

The second metric used to evaluate accuracy is referred to as *weighted L_1 error*. This is the average L_1 distance from a found vector f_j to its closest-matching (in terms of L_1 distance) golden standard vector b_l, weighted by the size of the set of markers assigned to bin vector f_j. Note that an L_1 distance of 2 indicates one incorrect entry in f_j, a distance of 4 indicates 2 incorrect entries, and so on. For example, the L_1 distance between $f_j = (1, 1, 1, -1)$ and $g_l = (1, 1, 1, 1)$ is 2, because $|f_j(4) - g_l(4)| = 2$ and $|f_j(i) - g_l(i)| = 0$ for $i \in \{1, 2, 3\}$. Formally, we define the weighted L_1 error as:

$$\text{L1 error} = \frac{1}{m} \sum_{j=1\ldots|\mathcal{F}|} |\text{found bin}_j| \sum_{i=1,\ldots,n} |f_j(i) - g_l(i)|$$

where m is the data set size, and $|\text{found bin}_j|$ is the number of markers assigned to bin vector f_j. Therefore, larger bins contribute more to the calculation of average L_1 error.

Finally, we report a third metric that we name *recall*, defined as the percentage of true bin vectors that were found perfectly:

$$\text{recall} = \frac{\sum_{l=1\ldots|\mathcal{B}|}(f_j = b_l)}{|\mathcal{B}|}$$

4.3 Simulated Data Experiments

Figure 4, plots the error rate $\hat{\epsilon}$ for increasing data set size and three different ϵ values, with μ fixed at 50%. We chose ϵ to reflect true genotyping errors in genetic marker data, which rarely rise above 0.5% [13]. Observe that even with 50% missing data in the input, with enough data our estimate accurately approaches the true error rate for each ϵ tested — 0.1%, 0.5%, and 1%.

Next, we report the accuracy and recall for increasing data set size and varying error rates, with μ fixed at 50%, shown in Table 1. Observe that the trend agrees with our intuition

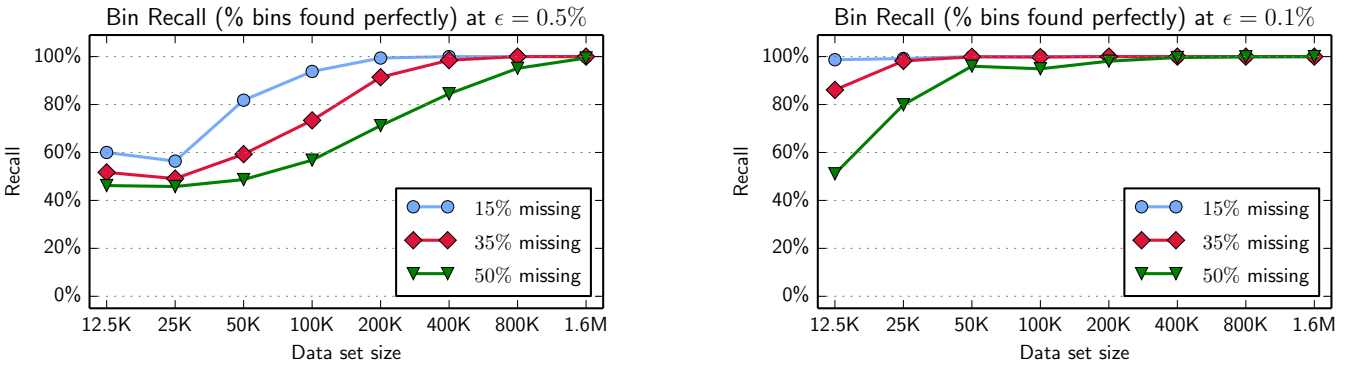

Figure 6: Bin recall for a fixed error rate (left) $\epsilon = 0.5\%$ and (right) $\epsilon = 0.1\%$ with missing rates of $\mu = 15\%, 35\%, 50\%$. As the data set grows, more bins are recovered perfectly, while lower missing rates allow our algorithm to recover the bins with less data. With the low (but realistic) error rate of $\mu = 0.1\%$, we perfectly recover all true bin vectors with less required data than when $\mu = 0.5\%$ error.

— more data is required to correctly recover the true bin vectors as the error rate grows. However, notice that the found bin vectors are extremely close to the true bin vectors; the accuracy is above 95% in all cases. The default constant c_ϵ was used to set $\hat{\epsilon}$ only for the 12.5K and 25K cases for $\epsilon = 0.1\%$. We achieve accuracy above 99% in all cases when the data set size is 100K or greater, meaning that on average, the bin vectors we find agree in 99 or more positions out of 100 with the golden standard.

In our next set of results, Figure 5, we present the weighted L_1 error for increasing data set size, with variation in the missing rate μ for a fixed ϵ at 0.5%. Observe that with a low missing rate of 15%, the errors made in bin vector assignments values are low even for small data set sizes, and remain close to 0 as the data size increases — confirming that increasing the amount of data for a low missing rate does not hurt binning accuracy. For a moderate missing rate of 35%, we see that the weighted L_1 error starts near 0.1, indicating that on average, large bins correspond to bin vectors that are within 0.1 L_1 distance of true bin vectors. Of course, the L_1 distance is always a positive integer in this case. Therefore, we can interpret this error as the effect of the extra, small-sized bins that are found in addition to the golden standard vectors. An important point demonstrated by the weighted L_1 error is that these extra bin vectors are still close to the underlying true bin vectors. In other words, few bin vector values are assigned incorrectly and those erroneous bins are themselves extremely close in L_1 distance to true bin vectors.

Data Size	Error Rate (ϵ)					
	0.1%		0.5%		1%	
	Accur	Recall	Accur	Recall	% Accur	Recall
12.5K	95.87	51.10	98.76	46.2	99.3	27.3
25K	96.13	79.90	99.81	45.8	99.67	33.1
50K	98.44	96.0	99.93	48.7	99.85	39.2
100K	99.88	94.9	99.93	56.8	99.88	47.9
200K	99.87	98.1	99.92	71.2	99.89	59.7
400K	99.87	99.6	99.94	84.5	99.93	70.7
800K	99.80	99.9	99.89	95.1	99.91	85.5
1.6M	99.76	100	99.87	99.4	99.87	95.9

Table 1: Accuracy and recall (in %) for fixed $\mu = 50\%$ and increasing error rate (ϵ). In most cases, accuracy is above 98%. Higher error rates require more data to recover all true bin vectors.

The error for the 35% missing rate quickly decreases toward 0, showing that approximately 200,000 markers are sufficient to find the 1,000 bin vectors that generated the data almost perfectly. Finally, we observe that the 50% missing rate trend line actually *increases* at some increments of data size. However, as we will discuss next, the loss in accuracy is offset by the gain in recall.

Figure 6(left) shows the bin vector recall for a fixed $\epsilon = 0.5\%$ and varying missing rates μ. Note the similar trend to the weighted L_1 error results — as the missing rate increases, more data is required to perfectly recover all the true bin vectors. However, more interestingly, even the low missing rate of 15% requires 200K markers to find all the golden standard bin vectors. With 50% missing values, it is not until we process the 1.6M marker data set that perfect recall is achieved. These results highlight the necessity for large data sets when missing data rates are high, even with relatively low error rates. An exciting implication of this result is that with enough inexpensive, low-coverage DNA sequencing, which produces much higher missing rates than more expensive, repetitive high-depth sequences [13], we can effectively recover the fundamental information for constructing the genetic map. To show the effect of the error rate on bin recall, we also present a recall plot for ϵ fixed at 0.1% in Figure 6(right). The lower error rate allows our algorithm to recover true bin vectors at much smaller scales than witnessed in Figure 6(left).

4.4 Computational Efficiency

We now examine the computational efficiency of our algorithm. Runtimes for increasing data set size, with μ fixed at 50%, are shown using $\epsilon = 0.1\%$, 0.5%, and 1% in Figure 4.4. Even up to 1.6 million markers, we observe linear running time of our algorithm. Importantly, this supports the idea that a coarse initial clustering sufficiently partitions the data into groups that represent markers close together on the genome, and that can be quickly further subdivided into bins. Our experiments across missing rates (not shown) also resulted in linear scaling for all tested cases.

4.5 Variable Bin Sizes

In addition to handling bins of approximately uniform size, our algorithm is designed to perform well even when, as

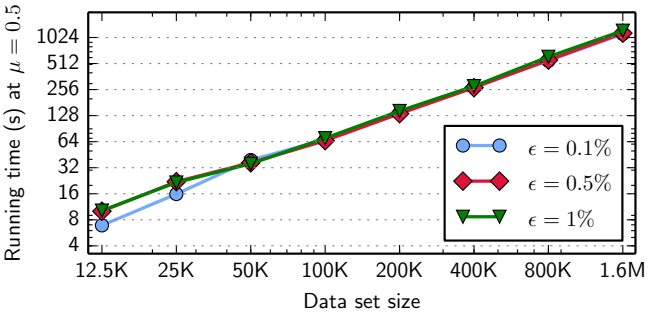

Figure 7: Running times (in seconds) for a fixed missing rate $\mu = 50\%$ with error rates $\epsilon = 0.1\%, 0.5\%, 1\%$. Both axes are on a logarithmic scale, as the data set size grows, running time grows linearly.

is sometimes the case in real data, a few large bin sizes dominate the data set. To test this claim, we generated simulated data in the same manner we described in Section 4.1, with one modification. For each simulated linkage group, instead of creating n bins of approximately equal size, we randomly select 3 bin vectors, and generate 20x more markers from these bin vectors than the others. Thus, the data is dominated by three large bins per linkage group. The number of linkage groups k remains fixed at 10 and the population size n remains fixed at 100.

The results for variable bin sizes with ϵ fixed at 0.5%, and μ fixed at 50%, are shown in Table 2. We report accuracy, recall, and running time for increasing data set size. The quality of the bin vectors found by our algorithm remains high — in all cases, over 99% of the vector entries were assigned correctly. However, we recover fewer bins than in the corresponding uniform bin size case (Figure 6) per data set size. Here again we see the advantage of using large-scale data: whereas smaller data sets do not hold enough information to identify smaller bins, our algorithm utilizes large data sets to correctly recover both small and large bins, without introducing errors to the binning accuracy. Running time remains linear with respect to data set size.

4.6 Real Data Experiments

We include results on two real data sets in Table 3. The barley data [15] contains 65,357 genetic markers from a population of 90 individuals and 7 linkage groups, with 20% missing data. The wheat data consists of 1.7 million genetic markers, from a population of 88 individuals and 21 linkage groups, with missing rate 39% [4]. We do not have true bin vectors available to test for accuracy and recall on these data sets, and thus report the estimated error rate, the running time, and number of bins found by our algorithm.

Variable Bin Size Experiments			
Data size	Accuracy (%)	Recall (%)	Running Time(s)
12.5K	99.79	33.3	39.2
25K	99.76	37.7	33.8
50K	99.87	49.2	48.2
100K	99.97	56.8	83.0
200K	99.98	71.5	150.8
400K	99.98	90.2	310.6
800K	99.98	98.8	567.1
1.6 M	99.98	100	1180.0

Table 2: Results for variable bin sizes, with a fixed $\mu = 35\%$ and $\epsilon = 0.5\%$. Runtime scales linearly, and accuracy increases, with larger data sizes. Because a few large bins dominate the input data, a large data set size is required to discover all true bin vectors.

Results for Real Data				
Data	Size	Estimated ϵ	Time(s)	Bins Found
Barley	65K	0.102%	45.9	467
Wheat	1.7M	0.064%	2728.0	1410

Table 3: Results for barley and wheat data. Our algorithm quickly reduces the data set size and estimates an error rate consistent with sequencing technologies used to produce the two data sets.

For both barley and wheat, the estimated error rate is close to 0.1%, a realistic estimate given the sequencing methods used to generate the data [4, 6, 13, 15, 20]. The time to output the bin vectors was less than a minute for barley and only 23.5 minutes for the large, complex wheat genome. Note that our wheat bin solution is within 5% (1410 vs. 1335) of previously published analysis [4]. Overall this demonstrates that our methodology enables the capability for the gene mapping community to effectively and quickly utilize multi-million marker data sets.

4.7 Upstream Analysis

Finally, we provide an initial analysis of the effect of genetic marker data reduction on the quality of the genetic map produced by taking our bin vectors as input. We use MSTMap [22], a popular genetic mapping tool, to produce a genetic map from the bin vectors of our algorithm. We use a set of 50K and 800K simulated markers with 35% missing data, 0.5% error rate, and uniform bin size. To assess the quality of this map, we compare it to the map produced by MSTMap when given the true bin vectors as input.

Table 4 displays two points of comparison between the MSTMap output on true versus found bins. First, we report Spearman's correlation coefficient, ρ, between the true bins as they are ordered when processed by MSTMap, and their order using MSTMap on our set of found bins. In other words, we use MSTMap to order and report genetic distances for both the gold standard bin set and the set found for our algorithm (inputting the bin vectors corresponding to only one linkage group at a time). Then, for each bin f_j that our algorithm found corresponding exactly to a true bin g_l, i.e. $f_j = g_l$, we compare its order in the found set to its order in the gold standard set. Recall that our algorithm found all true bins perfectly in the 800K case, but not in the 50K case. For both data sizes, our algorithm also produced a few spurious bins in each linkage group, with low L_1 distance to true bins (Figures 5 and 6). Table 4 shows that for all linkage groups, $|\rho|$ was greater than 0.999, with higher quality for the larger 800K data set. This result confirms our intuition that the spurious bin vectors produced by our algorithm do not introduce large errors in the final map product.

The effect of spurious bins is apparent in the relative map sizes of the found vs. true bins, shown in Table 4. MSTMap outputs a map size in terms of genetic distance, or *centimorgans* (cM). We compare the size of the map produced by feeding MSTMap the bins found by our algorithm, to

MSTMap Ordering: True vs. Found Bins		
Data Size	50K	800K
Spearman's $\|\rho\|$	[0.9993 – 1]	[0.9997 – 1]
Relative Map Increase (% cM)	[1.1 – 19.4]	[4.0 – 12.1]

Table 4: Comparison of genetic maps produced by MSTMap using true bins, vs. found bins using 35% missing data and 0.5% error rate, showing the range of solutions for all 10 linkage groups. Note that although our binning approach increases the map size, the mapping relative to the true bins is almost perfectly preserved.

the size of the MSTmap produced using only golden standard bins. Observe that in all cases, the map size increases, due to extra bin vectors within the map. However, the true bins remain almost perfectly ordered. This is a promising initial result — we have quickly reduced a data set with a size that is unmanageable by MSTMap to a more accurate and complete set that almost perfectly preserves the mapping quality of the underlying true bin set. Future work will explore an optimized solution to the ordering and mapping problem based on our reduced bin input.

5. RELATED WORK

Several existing theoretical works provide bounds on the data set size required to solve the related *matrix completion*, *co-clustering*, and *dimension reduction* problems. In genetic mapping research, although some notion of bins exists in several tools, we are unaware of any method that applies data reduction to the characteristically noisy and incomplete genetic marker data to efficiently and accurately discover the fundamental set of vectors making up the genetic map.

In the matrix completion problem, a matrix X is given as input with missing entries, as well as potentially noisy entries, with the goal of filling these entries correctly [3, 23]. Candes et al. show [3] that a sample of $m \geq CM^{1.2}r \log M$, where M is the data set size and r is the rank of the data matrix, is sufficient to perfectly recover the full matrix in most cases. Differently from our problem, the algorithm does not address errors in data entries. Xu et al. do address the matrix completion and noise reduction problem [23], for the specific case of co-clustering binary matrices, that is, clustering the rows and columns of an input binary matrix simultaneously. The most relevant result [23] to our work is that if the binary input matrix is block-constant, all entries can be recovered exactly with a sample of size $O(Mr^2)$, where r is the matrix rank. This interesting result cannot be applied directly to our problem as we do not assume a block-constant structure to our matrix.

Dimensionality reduction is a broad field of research with numerous applications. The goal is to reduce the input data to some lower-dimensional space which more accurately captures the structure of the data [2, 7, 17, 21]. Because we assume that all entries in our data matrix are generated by a small, finite set of bin vectors, our problem may be restated as reducing the set of markers to the n-dimensional hypercube of bin vectors. Many methods have been proposed to reduce the dimensionality of input data. We attempted to apply locally distance-preserving methods such as maximum variance unfolding [21] and locally linear embedding [17] to our data, using the LOD score as a similarity measure. However, these methods failed to map markers from the same bin to the same location in the lower-dimensional space, and they are computationally expensive. Bailey [2] addresses the problem of principal component analysis (PCA), which can be used to project input data to lower dimensions, of a noisy input matrix with missing data. However, data errors are assumed to be small changes to matrix entries, unlike the errors in our binary input data, which cause an entry to be changed to the value completely opposite its true value. Furthermore, the desired dimensionality of our data reduction is unknown a priori, and PCA requires the number of principal components k to be provided as input.

Our algorithm is similar in spirit to the unsupervised decision tree approach of Karakos et al. [10] and to the coarse-to-fine clustering approach of the BIRCH [24] and CURE [8] algorithms. The recursive construction of *Integrated Sensing and Processing Decision Trees* (ISPDTs) is used [10] to perform unsupervised classification. The initial data set is recursively split into subsets with the goal of discovering the underlying classes that generated the data with high probability, a similar problem to our binning objective. Expensive and greedy heuristics are applied at each step to either (a) maximize mutual information between the unknown class and the path from root to leaf in the ISPDT, or (b) minimize the probability of misclassification. Initial results on multispectral imaging data seem promising, but no running time analysis is provided, and it is unclear how the method performs on large-scale data sets.

BIRCH and CURE fall under the category of hierarchical clustering algorithms, which attempt to first reduce data input size into a coarse clustering, then refine the clustering within each coarse partition. However, the efficiency and accuracy of these algorithms relies on the assumption that points lie in a Euclidean d-dimensional space, and perfect data, which contains no errors or missing entries. These algorithms are also superlinear in the input data set size.

Existing genetic mapping tools include OneMap [14], MSTMap [22], and JoinMap [18]. Many of these tools provide a notion of bins as uniquely identifiable locations on the genetic map. Of these tools, MSTMap has gained popularity in recent years due to its relatively low computational complexity in comparison with other genetic mapping software. MSTMap also applies a simple heuristic algorithm to attempt to reduce the data into sets of *co-segregating* markers, or markers whose Hamming distance is 0, before beginning the map building process. The goal is of this algorithm is not to reduce the scale of the data to its fundamental resolution nor to reduce errors and fill in missing data. Thus it would be inaccurate to directly compare our bin output to MSTMap's sets of co-segregating markers. However, MSTMap requires quadratic time in the input data size, and thus is prohibitively slow on large data sets. Wu et al. also note that the quality of the map produced by MSTMap drops significantly with high error and missing rates [22]. For example, on a 25K simulated data set with 35% missing rate and 0.5% error, MSTMap runs in 2577.5 seconds to produce a map, compared with only 4.0 seconds when given golden standard bins as input. Therefore, our algorithm enables these tools to effectively handle modern-day, large-scale noisy data sets produced by next generation sequencing technologies.

6. CONCLUSION

We have introduced an algorithm for large-scale genetic marker reduction, and shown that it can quickly reduce the size of large-scale, noisy, and incomplete genetic marker data almost perfectly to a fundamental set of bin vectors that define unique locations on a genetic map. Even with challenging data sets with error rates as high as 1% (which are unlikely in real data), missing rates as large as 50%, and variable bin sizes, our algorithm correctly reproduced the vectors that correspond to the fundamental set size when given enough data. Furthermore, the reduction in data size enables traditional software tools, designed for the small-scale setting, to perform well on reduced large data sets.

Our experimental results can also serve to guide future genetic map building efforts. By simulating various missing and error rates, we have shown experimentally, for a typical genetic map population size, how much data is required to correctly recover all bin vectors, given a fundamental set size. Our results suggest that although our algorithm works well on low missing and error rates, we can utilize large amounts of data with high missing and error rates to correctly recover bins. Notably, we demonstrate the near-linear efficiency of our algorithm on both simulated and real data, up to a 1.7M wheat data set.

Future research will extend our algorithmic work to the final phase of genetic mapping: ordering the bin vectors to produce a final map. Although existing tools such as MSTMap perform reasonably well given high-quality, small-scale data, we believe that the properties of the bin vectors found by our algorithm can be leveraged to aid genetic map construction for the massive data sizes of next-generation sequencing technologies.

Acknowledgment

The work conducted by the Lawrence Berkeley National Laboratory and the U.S. Department of Energy Joint Genome Institute is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research is supported in part by NSF CISE Expeditions award CCF-1139158 and DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, SAP, Cisco, Clearstory Data, Cloudera, Ericsson, Facebook, FitWave, General Electric, Hortonworks, Huawei, Intel, Microsoft, NetApp, Oracle, Samsung, Splunk, VMware, WANdisco and Yahoo!.

7. REFERENCES

[1] A. Auton. *The Estimation of Recombination Rates from Population Genetic Data*. PhD thesis, University of Oxford, 2007.

[2] S. Bailey. Principal component analysis with noisy and/or missing data. *Publications of the Astronomical Society of the Pacific*, 124(919):1015–1023, 2012.

[3] E. J. Candès and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009.

[4] J. Chapman, M. Mascher, A. Buluç, et al. A whole-genome shotgun approach for assembling and anchoring the hexaploid bread wheat genome. *Genome Biology*, 16(26), 2015.

[5] J. Cheema and J. Dicks. Computational approaches and software tools for genetic linkage map estimation in plants. *Briefings in Bioinformatics*, 10(6):595–608, 2009.

[6] B. Collard and D. Mackill. Marker-assisted selection: an approach for precision plant breeding in the twenty-first century. *Philos Trans R Soc Lond B Biol Sci.*, 363(1491):557–572, 2008.

[7] E. Elhamifar and R. Vidal. Sparse manifold clustering and embedding. In *NIPS*, pages 55–63, 2011.

[8] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. *ACM SIGMOD Record*, 27(2):73–84, 1998.

[9] J. B. S. Haldane. The combination of linkage values and the calculation of distances between the loci of linked factors. *J Genet*, 8(29):299–309, 1919.

[10] D. Karakos, S. Khudanpur, J. Eisner, and C. E. Priebe. Unsupervised classification via decision trees: An information-theoretic perspective. In *ICASSP*, volume 5. IEEE, 2005.

[11] D. Kosambi. The estimation of map distances from recombination values. *Annals of Eugenics*, 12(1):172–175, 1943.

[12] E. S. Lander, P. Green, J. Abrahamson, A. Barlow, M. Daly, S. Lincoln, and L. Newberg. MAPMAKER: an interactive computer package for constructing primary genetic linkage maps of experimental and natural populations. *Genomics*, 1:174–181, 1987.

[13] Y. Li, C. Sidore, H. M. Kang, M. Boehnke, and G. R. Abecasis. Low-coverage sequencing: implications for design of complex trait association studies. *Genome research*, 2011.

[14] G. Margarido, A. Souza, and A. Garcia. Onemap: software for genetic mapping in outcrossing species. *Hereditas*, 144(3):78–79, 2007.

[15] M. Mascher, G. J. Muehlbauer, D. S. Rokhsar, J. Chapman, et al. Anchoring and ordering NGS contig assemblies by population sequencing (POPSEQ). *The Plant Journal*, 76(4):718–727, 2013.

[16] M. Metzker. Sequencing technologies, the next generation. *Nature Reviews Genetics*, 11:31–46, 2009.

[17] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

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

[19] V. Strnadova, A. Buluç, J. Chapman, J. Gilbert, J. Gonzalez, S. Jegelka, D. Rokshar, and L. Oliker. Efficient and accurate clustering for large scale genetic mapping. In *BIBM*, 2014.

[20] N. Tinker. Spaghetti: Simulation software to test genetic mapping programs. *Journal of Heredity*, 101(2):257–259, 2010.

[21] K. Q. Weinberger and L. K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70(1):77–90, 2006.

[22] Y. Wu, P. Bhat, T. Close, and S. Lonardi. Efficient and accurate construction of genetic linkage maps from the minimum spanning tree of a graph. *PLoS Genet.*, 4(10), 2008.

[23] J. Xu, R. Wu, K. Zhu, B. Hajek, R. Srikant, and L. Ying. Exact block-constant rating matrix recovery from a few noisy observations. *arXiv preprint arXiv:1310.0512*, 2013.

[24] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *ACM SIGMOD Record*. ACM, 1996.