Efficient and Accurate Clustering for Large-Scale Genetic Mapping

Veronika Strnadova * Aydın Buluç† Jarrod Chapman‡ John R. Gilbert§
Joseph Gonzalez¶ Stefanie Jegelka‖ Daniel Rokhsar** Leonid Oliker††

Abstract

High-throughput "next generation" genome sequencing technologies have out-paced Moore's law, producing a flood of inexpensive genetic information that is invaluable to research, ranging from the development of new and improved crops to understanding the genetic variation that underlies cancer. However, this flood of new information presents a fundamental new challenge to genetic mapping, the process of assembling genetic data, which is a core operation in genomics research. The current generation of genetic mapping tools were designed for the small data setting, and are now limited by the the prohibitively slow clustering algorithms they employ in the genetic marker-clustering stage of automatic genetic map construction. In this work, we present a new approach to genetic mapping based on a fast new clustering algorithm. Our theoretical and empirical analysis shows that the algorithm can correctly recover linkage groups. Using real-world and synthetic data, we demonstrate that our approach is able to quickly process orders of magnitude more genetic markers than existing tools and that by exploiting domain knowledge, it is able to out-perform more generic approaches based on spectral clustering. Finally, we demonstrate that by scaling to the available sequence data we are able to improve the quality of genetic marker clusters, leading to a higher quality ultra-high-density genetic map that can be used to improve genome assemblies and map quantitative traits.

Keywords: genetic mapping, clustering, single linkage

1 Motivation and Background

Genetic maps are essential for organizing DNA sequence information along chromosomes, and they enable diverse applications of genetics to problems in health, agriculture, and the study of biodiversity [5]. Early genetic maps were constructed using only dozens or a few hundred *genetic markers* (chromosomal regions with two or more sequence variants in a population), and computation of a map from an underlying dataset was data limited and therefore computationally inexpensive. With the advent of inexpensive high-throughput "next generation" sequencing [20], however, the situation is now reversed, and it is becoming a simple matter to generate data corresponding to millions of genetic markers across a genome. This flood of data creates a new challenge: to produce accurate high-density genetic maps in a computationally efficient manner.

A genetic map is a linear ordering of genetic markers that is consistent with observed patterns of inheritance in a population. An essential concept is the *linkage group* which collects together markers that are found on a single chromosome. Genetic maps are therefore organized into multiple linkage groups, with the number of groups equal to the number of chromosomes in the species. Within a linkage group, there is a natural measure of proximity which arises from the linear structure of chromosomes and the mechanics of their transmission from generation to generation. Given a pair of markers in the same linkage group, we can estimate their proximity on the chromosome by comparing their sequence across a *mapping population* of related individuals. This estimate is made based on the LOD score, a logarithm of odds that two markers are genetically linked, based on the similarities and differences across each individual's genotype. The fundamental problem of genetic map construction is to take as input the sequences of n related individuals at m genetic markers, possibly with errors and often missing data (unknown genotypes of particular markers for particular individuals), and to organize these markers into linear chains that represent the structure of chromosomes.

The first step of genetic mapping involves clustering markers into linkage groups. This is traditionally performed by first computing pairwise similarities between all pairs of markers. Markers are then grouped using various standard clustering algorithms, often applying a biologically meaningful cutoff to the similarity matrix. Computing the similarity score between two markers can be a simple attribute comparison or a computationally intensive procedure, such as estimating the recombination rate of two genetic markers via nonlinear regression [23, 31]. While clustering algorithms that work directly on similarities typically start with the similarity matrix as input, the cost of constructing this matrix is often overlooked. Even when the similarity cal-

*UC Santa Barbara. Email: veronika@cs.ucsb.edu
†Lawrence Berkeley Nat. Lab. Email: abuluc@lbl.gov
‡Lawrence Berkeley Nat. Lab. Email: jchapman@lbl.gov
§UC Santa Barbara. Email: gilbert@cs.ucsb.edu
¶UC Berkeley. Email: jegonzal@eecs.berkeley.edu
‖UC Berkeley. Email: stefje@eecs.berkeley.edu
**LBNL and UC Berkeley. Email: dsrokhsar@lbl.gov
††Lawrence Berkeley Nat. Lab. Email: loliker@lbl.gov

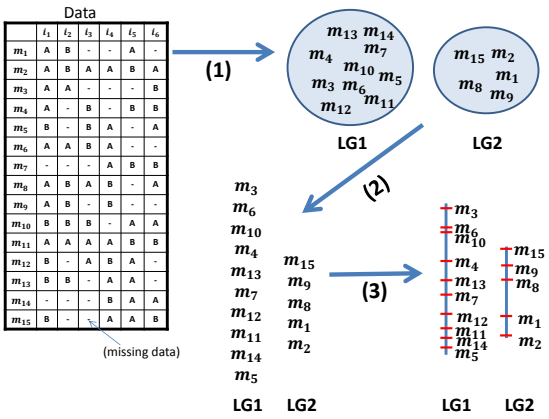


Figure 1: Genetic map construction. Input is given in the form of a marker (m) - individual (i) matrix. The markers are clustered into linkage groups (**LG**'s) (1), ordered within each linkage group (2), and finally spaced according to their genetic distance (3).

culation is fast, it requires $O(m^2)$ time to calculate all pairwise similarity scores for m genetic markers, which quickly becomes prohibitive for large numbers of data points. Our work addresses this important clustering computation for large-scale gene mapping problems by taking advantage of our prior knowledge that the linkage groups (i.e., chromosomes) have an intrinsically linear substructure. Our primary contributions are:

1. A new, fast clustering algorithm, BubbleCluster, that exploits the structure of genetic markers (Section 3.1).
2. An analysis of the BubbleCluster algorithm demonstrating accurate cluster estimation (Section 3.2).
3. An empirical evaluation on synthetic and real-world data demonstrating the scalability and accuracy of the BubbleCluster algorithm (Section 4).

2 Problem Definition

Computational tools for genetic mapping follow three phases: (1) Grouping markers into linkage groups (typically chromosomes), (2) Ordering markers within chromosomes and (3) Map distance estimation (Fig. 1). Linkage group construction is typically performed using single linkage clustering on the similarity graph where the similarity function is the LOD score, a logarithm of odds that two markers are genetically linked. The LOD score between two data points is based on the *recombination fraction* between the two markers, the probability that an odd number of crossovers occur during meiosis along the region between them. Given the population size, a critical LOD score (*linklod* [26]) is estimated and serves as the cut-off threshold for constructing clusters from the single linkage dendrogram.

Current software tools, such as the popular MSTMap [31] and JoinMap [26], fail to scale up to even tens of thousands of markers, let alone the millions of markers measured simultaneoulsy by new technologies. Our initial benchmarks revealed that the bottleneck is the pairwise distance calculation step in the linkage group construction phase. This is unexpected, given that the exact solution of the second phase (map ordering) is shown to be a variant of the NP-complete symmetric traveling salesman problem [24]. There are several reasons for this phenomenon. First, heuristics and approximations, such as using a minimum-spanning tree [31], work well in the ordering phase, due to the underlying simple geometry of the problem. Second, after breaking the data into linkage groups, we operate on smaller-size data. Finally, but perhaps most importantly, the number of bins are smaller than the number of actual markers, especially for large numbers of markers. A bin is a minimal genetic interval, which might contain several markers, that is genetically resolved from its adjacent set of bins. The markers within a bin are not resolvable due to the limited size of the population and the limited number of recombination events in a given population. Ordering is performed on the bins themselves as opposed to markers, significantly reducing the runtime.

In this paper, we therefore focus on the bottleneck of finding ordered, connected clusters (linkage groups) $C_1, ..., C_k$, given m markers measured across n individuals in the mapping population. The entries of a marker feature vector x_i are individual genotypes at marker sites. For illustration, all genotypes are assumed to be homozygous, and hence the n entries of a marker feature vector can take only two values A,B, and '$-$' for missing values. Since changes in population type only affect the LOD computation, our algorithm generalizes to more complex populations.

The LOD score for two markers x_i, x_j is

$$LOD(x_i, x_j) = \log_{10}\left(\frac{(1-\theta)^{NR}\,\theta^{R}}{0.5^{NR+R}}\right), \tag{2.1}$$

where $\theta = \frac{R}{NR+R}$ is the recombination fraction, R is the number of recombinant individuals, and NR is the number of non-recombinant individuals. For phase-known populations, R is the number of matches and NR is the number of mismatches in the observed (not $-$) portions of the feature vectors x_i and x_j.

Finally, we point out that the fixed order of genetic markers along chromosomes is a key property of the data. Exploiting this linear, one-dimensional structure enables a specialized procedure for finding linkage groups that is faster than generic clustering algorithms.

In practice, due to genotyping errors and missing data, markers do not reside exactly on a line, but approximately enough to build an ordered backbone sketch of the line along which a linkage group lies.

3 Methods and Techniques

In this section, we detail our BubbleCluster algorithm and prove its correctness under mild assumptions. To stay as general as possible, we will use a divergence measure d when describing our algorithm. In practice, we use the inverse of the LOD score for our distance. This is a valid divergence measure since the LOD score is a symmetric, monotonically decreasing function of genetic distance [10, 13].

3.1 The BubbleCluster algorithm Algorithm 1 clusters in three phases:

1. Perform an initial clustering $\mathcal{C}$ using all high-quality markers (lines 1–21);
2. Assign low-quality markers to their most likely cluster $C \in \mathcal{C}$ (lines 22–30);
3. If any of the initial clusters are too small, attempt to merge them with a large cluster (lines 31–34).

This algorithm is a coarse-to-fine approach in two respects: first, it relies on a good clustering of reliable high-quality data points in Phase 1 as a skeleton to assign more noisy low-quality points in Phase 2. Second, as detailed below, the clustering in Phase 1 itself establishes a clustering and ordering among representative sketches of the high-quality data that again serve as landmarks for the remaining points. Such a hierarchical approach is related to theoretically well-grounded clustering techniques like core sets [4, 8] or nearest neighbor clustering [29], but, as opposed to the cited algorithms, BubbleCluster is deterministic. Apart from robustness, sketches offer a big gain in scalability.

In Phase 3, we then draw on the biological laws that govern true linkage group sizes in living organisms to determine whether any cluster is too small to consider a true linkage group. We attempt to merge all such clusters with the larger clusters from Phases 1 and 2.

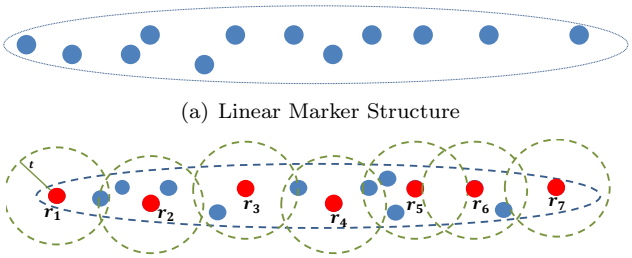
Phase 1: Initial Clustering. The first phase of our algorithm is the most important, exploiting the structure of genetic linkage groups in order to quickly cluster high-quality genetic markers. The algorithm has two parameters, a distance cutoff threshold t and a nonmissing data threshold n.

The missing data threshold is used to distinguish high-quality markers from lower-quality markers with limited information content. In Phase 1, we only process

Algorithm 1: BubbleCluster Algorithm

Inputs: $\mathcal{X} = \{x_1 \dots x_M\}$,
t = distance threshold,
n = nonmissing threshold
c = difference threshold

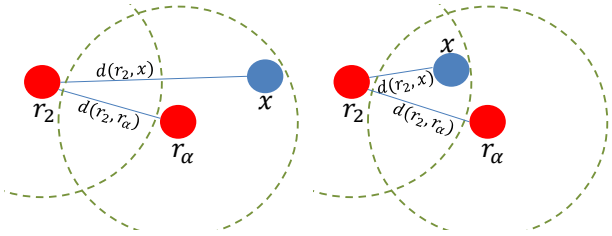
1 $\mathcal{C} \leftarrow \emptyset$; `// List of cluster members sets`
2 $\mathcal{R} \leftarrow \emptyset$; `// List of representatives sets`
3 sort $\mathcal{X}$ by increasing missing data;
4 $\mathcal{H} = \{x_i \in \mathcal{X} |$ nonmissing$(x_i) > n\}$;
5 **for** *point* $x \in \mathcal{H}$ *in sequence* **do**
6 **if** $\mathcal{R} = \emptyset$ **then**
7 define new cluster C_α: $C_\alpha \leftarrow \{x\}$;
8 define new rep. set R_α: $R_\alpha \leftarrow \{x\}$;
9 $\mathcal{C} \leftarrow \mathcal{C} \cup C_\alpha, \mathcal{R} \leftarrow \mathcal{R} \cup R_\alpha$;
10 **else**
11 $r_{\min} = \operatorname{argmin}_r \ d(x,r)$ s.t. $r \in R_\alpha \in \mathcal{R}$;
12 $r_{\min_2} = \operatorname{argmin}_{r \notin R_\alpha} \ d(x,r)$;
13 **if** $d(x, r_{\min}) < t$ **then**
14 assign x to cluster C_α:
$C_\alpha \leftarrow C_\alpha \cup \{x\}$;
15 **if** `isBoundaryPoint` (x, R_α) **then**
16 add x to the correct end of R_α;
17 **if** $d(x, r_{\min_2}) < t$ **then**
18 `MergeReppts` (R_α, R_β);
19 `MergeClusters` (C_α, C_β);
20 **else**
21 set up new cluster:
$\mathcal{C} \leftarrow \mathcal{C} \cup \{x\}, \mathcal{R} \leftarrow \mathcal{R} \cup \{x\}$;
22 **for** $y \in \mathcal{X} \setminus \mathcal{H}$ **do**
23 $r_{\min} = \operatorname{argmin}_r \ d(y,r)$ s.t. $r \in R_\alpha \in \mathcal{R}$;
24 $r_{\min_2} = \operatorname{argmin}_{r \notin R_\alpha} \ d(y,r)$;
25 $d_{\min_2} = d(y, r_{\min_2})$;
26 $d_{\min} = d(y, r_{\min})$;
27 **if** $(d_{\min_2} - d_{\min}) > c \cdot d_{\min_2} \cdot d_{\min}$ **then**
28 $C_\alpha \leftarrow C_\alpha \cup \{y\}$;
29 **else**
30 set up new cluster:
$\mathcal{C} \leftarrow \mathcal{C} \cup \{y\}, \mathcal{R} \leftarrow \mathcal{R} \cup \{y\}$;
31 **for** *all small clusters* C_s *in* $\mathcal{C}$ **do**
32 pick a $c_j \in C_s$;
33 **if** $d(c_j, r_k) < t$ *for any* r_k *in any* $R_\alpha \in \mathcal{R}$ **then**
34 `MergeClusters` (C_s, C_α);



(a) Linear Marker Structure

(b) Representative Points

Figure 2: **(a)** The *linear* structure of a genetic marker cluster C_i. **(b)** The red points are the *representative points* for cluster C_i. Each marker in the cluster is within the threshold distance t of at least one representative point r_i.



(a) Boundary Point (b) Not a Boundary Point

Figure 3: The isBoundaryPoint function determines whether a marker lies at the boundary of the representative points. Either **(a)** x lies on the outside of boundary point r_α; isBoundaryPoint(x, R_α) returns true or **(b)** x lies on the inside of r_α; isBoundaryPoint(x, R_α) returns false.

high-quality markers that have at least n non-zero entries. The algorithm proceeds by establishing clusters on the fly: each incoming point is either close to, and hence assigned, to an existing cluster, or it creates a new cluster. Two clusters are merged if their distance is smaller than the cutoff threshold.

To avoid storing all data points and comparing distances between a new point and all previous points, we only keep a representative sketch R_α for each cluster C_α. To create and maintain sketches, we exploit the special structure of the data: genetic markers have a fixed order within each linkage group, making the structure of each cluster appear linear, as illustrated in Figure 2(a). The resulting sketch is therefore an ordered list of representative points as illustrated in Figure 2(b) where for every point in C_α, there is a point $r_\alpha(x)$ in R_α with $d(x, r_\alpha(x)) \leq t$.

For each incoming point x, we find the closest sketch point $r_{\min}$. If $d(x, r_{\min}) < t$, then x is assigned to the same cluster as $r_{\min}$. Otherwise, it sprouts a new cluster (line 21). If x is added to an existing cluster, we need to check whether it is well represented by the current sketch, or whether we need to augment R_α. Here, we use the linearity assumption. If x is outside of the boundaries specified by R_α, we add x as a new (boundary) sketch point. This is determined by comparing x to the closest current boundary point r_α in R_α, and the point $r_2 \in R_\alpha$ immediately next to r_α in the ordered list R_α, as illustrated in Figures 3(a) and 3(b): if $d(x, r_2) \geq d(x, r_\alpha)$, then x extends the boundary.

When x becomes a new representative point, it extends C_α in the linear dimension along which we assume the representative points to lie. It succeeds r_α and becomes a new end of R_α.

Finally, we determine whether x connects two clusters (lines 17–19) by finding the nearest sketch point $r_{\min_2}$ that is *not* in the cluster to which x was assigned. If $d(x, r_{\min_2}) < t$, then x forms a bridge $r_{\min}, x, r_{\min_2}$ between the two clusters. When merging clusters, we must also merge their sketches R_α and R_β. To merge R_α and R_β, we compute the four pairwise distances between the two boundary points of R_α and the two boundary points of R_β, in order to find the two closest boundary points $r_\alpha \in R_\alpha$ and $r_\beta \in R_\beta$. We then append R_β to the end of R_α in the order which maintains the closest distance between r_α and r_β.

Phase 2: Low quality marker assignment. At the completion of Phase 1, we have an initial clustering $\mathcal{C}$ of all the high-quality data points $x \in \mathcal{H}$, along with their ordered sketches. In Phase 2 (lines 22–30), we rely on the sketches to assign the remaining low quality markers $y \in \mathcal{X} \setminus \mathcal{H}$ to one of the existing clusters. We use a simple heuristic: for each low-quality marker, we find the difference between $d_{\min_2} = d(y, r_{\min_2})$ and $d_{\min} = d(y, r_{\min})$, where $r_{\min}$ and $r_{\min_2}$ are defined as in Section 3.1. If this difference is greater than a threshold $c \cdot d_{\min} \cdot d_{\min_2}$, then we add y to the cluster C_α containing $r_{\min}$. Otherwise, we simply create a new, temporary "singleton" cluster containing only the point y. Because we use $d = 1/LOD$ in practice, this choice of difference threshold corresponds to the situation when the odds that c_j belongs to cluster C_α is 10^c greater than the odds that c_j belongs to any other cluster. Intuitively, we assign y to C_α only if we believe that it is much more likely that y belongs to C_α than to any other cluster. Moreover, we only assign points to existing clusters for which we have high confidence in our assignment.

Phase 3: Merging small clusters with large clusters At this stage, we rely on further assumptions about the underlying structure of our clusters, based on our knowledge of true linkage groups. We know that each marker comes from exactly one linkage group, and that these groups tend to be relatively large. Though chromosomes vary in size, it is extremely unlikely that

the smallest of k chromosomes of a particular species will contain less than a quarter of the genetic markers of the next-smallest chromosome. In this phase, therefore, we find all such small clusters, and attempt to merge them with existing clusters. We pick a random point x within each small cluster C_s and compare its distance to all the sketch points r in large clusters. If this point is found to lie within the threshold distance of a sketch point $r_\alpha \in R_\alpha$, then we merge the small cluster C_s with the large cluster C_α.

Running time: The BubbleCluster runs in time $O(m \log(m) + mr)$ for m markers and r representative points, with the former term due to sorting markers. In practice, the number of sketch points scales logarithmically with the number of markers, and was always smaller than 100 per linkage group in our experiments.

3.2 Correctness of BubbleCluster For the analysis, we make the following assumptions on the true underlying linkage groups $C_1^*, \ldots C_K^*$ that are roughly reasonable for real data.

A1. Separation: there exists a $\lambda_{\text{sep}} > 0$ such that for any C_α^* and any two points $x \in C_\alpha^*$, $y \notin C_\alpha^*$, it holds that $d(x, y) > \lambda_{\text{sep}}$.

A2. Connectedness: there exist constants $0 < \lambda_{\text{conn}} + 2\epsilon < \lambda_{\text{sep}}$, $\epsilon \geq 0$, such that for every C_α^* and each $x_1, x_2 \in C_\alpha^*$, there is a *path* of points $y_1, \ldots y_m \in C_\alpha^* \cap \mathcal{H}$ with $d(x_1, y_1) < \lambda_{\text{conn}}$, $d(y_m, x_2) < \lambda_{\text{conn}}$ and $d(y_j, y_{j+1}) < \lambda_{\text{conn}}$ for all $1 \leq j \leq m$.

A3. The divergence measure d roughly reflects the underlying arrangement: for any true ordering $r_1, r_2, r_3 \in C_\alpha^*$ for any three points with $\epsilon \leq d(r_1, r_2) \leq 2\lambda_{\text{conn}} + 4\epsilon$ and $\epsilon < d(r_2, r_3) \leq 2\lambda_{\text{conn}} + 4\epsilon$, it holds that $d(r_1, r_2) < d(r_1, r_3)$ and $d(r_2, r_3) < d(r_1, r_3)$.

A4. All sketch points $r_1, r_2 \in R_\alpha$ are ϵ-separated: $d(r_1, r_2) \geq \epsilon$.

The ϵ in A3 and A4 adds noise robustness. Assumption A4 is simple to satisfy in the algorithm if we add the condition that a point x becomes a boundary point only if $d(x, r_\alpha) \geq \epsilon$.

LEMMA 3.1. *If $\lambda_{conn} + 2\epsilon < t < \lambda_{sep}$ and if Assumptions A1-A4 hold, then the algorithm identifies the correct clusters for all points in $\mathcal{H}$ within one pass over the sorted data. In addition, the sketch of each cluster is ordered correctly.*

Proof. Let $C'_1, \ldots C'_J$ be the clusters returned by the algorithm, with an ordered sketch $r_1^\alpha, \ldots, r_L^\alpha \in C'_\alpha$ for each cluster. We also observe that if A2 holds, then, by construction of the algorithm, two adjacent sketch points within a cluster have distance at most $\lambda_{\text{conn}} + 2\epsilon$.

First, we argue that for all C'_α, $C'_\alpha \subseteq C_\beta^*$ for some β after Phase 1. This is an invariant that holds throughout Phase 1: When a cluster is created, it consists of one point and therefore certainly is contained in a single true cluster. If a new point x gets added to C'_α, that point is within a distance of $t < \lambda_{\text{sep}}$ of $r_{\min} \in C_\beta^*$, and hence, by A1, x and $r_{\min}$ must be in the same true cluster. Two clusters are merged only if there is a path $(r_{\min}, x, r_{\min_2})$ between them with hop distances less than $\lambda_{\text{conn}} + \epsilon$. By A2, these clusters must therefore belong to the same true cluster.

Second, we see that if $C'_\alpha \subseteq C_\gamma^*$ and $C'_\beta \subseteq C_\gamma^*$, then $\alpha = \beta$, i.e., no true cluster is split: If C_γ^* was split, then, by A2, there would be at least two clusters $C'_\alpha, C'_\beta \subseteq C_\gamma^*$ and points $r_\alpha \in R_\alpha$, $r_\beta \in R_\beta$ with $d(r_\alpha, r_\beta) \leq \lambda_{\text{conn}} + 2\epsilon < t$, and hence these clusters will be merged.

Third, the ordering is recovered. If we ensure that the sketch points are at least ϵ apart, then A3 implies that the merge of two correctly ordered sketches preserves the correct ordering. A new sketch point x is added to R_α if it is farther away than ϵ from any sketch point and if it is identified as a boundary point. By A3, it is then correct to place x at the beginning/end of the list R_α, next its closest boundary point.

3.3 Related Work A preclustering of the data aims to establish which markers fall on the same chromosomes (linkage groups). A common choice of linkage metric is the LOD score, which measures the log-likelihood that two markers are linked on the same chromosome. While there are an overwhelming number of clustering methods [1], from flat to hierarchical methods, bottom-up to top-down methods, and including spectral approaches [28], single linkage and k-means [18], we are here interested in a clustering that exploits the known biological structure in the data, that is, the the linear ordering of markers along a chromosome.

Several computational tools exist for the construction of genetic linkage maps, as explained in the survey by Cheema and Dicks [6]. Since then, OneMap [19] and Lep-MAP [22] have also been proposed. All these tools, without exception, perform all-pairs comparisons among markers, which makes them unsuitable for millions of markers. Structural clustering methods that have been applied to gene mapping include connected components [31, 12, 14] and single linkage clustering [15]. Our algorithm is related to the concept of single linkage, but differently from single linkage, we construct and merge clusters on the fly, requiring only one pass

through the data after sorting. Our simple and efficient algorithm is correct due to the linear organization of chromosomes.

Our aim to retain both correctness and efficiency requires a compressed representation of the data and clusters that preserves enough information to obtain accurate solutions. General compressed representations for clustering problems have been addressed by core sets [4, 8], and hierarchical re-clustering ideas for streaming and distributed clustering [21, 25]. Our approach is closely related. As opposed to general sampling techniques, we extract a problem-specific representative core set deterministically within one pass, and exploit the specific structure of the marker data.

The way our algorithm maintains an ordering of the dataset is similar in spirit to the OPTICS algorithm [2]. However, different from OPTICS, our algorithm is not density based and uses several representative points to provide an accurate coverage of the underlying cluster. In that sense, our algorithm is closest to the CURE algorithm [9], which also maintains representative points. The specific insight we draw from the genetic mapping problem enables our algorithm to maintain a better performance bound than CURE's $O(n^2 \log(n))$ bound, and also prove correctness with mild assumptions.

4 Empirical Evaluation

We compare our algorithm to two popular genetic mapping tools: JoinMap and MSTMap. Furthermore, we compare our algorithm to a general spectral clustering method called the power iteration clustering (PIC) algorithm [16]. PIC could in theory reduce the running time close to linear by decomposing the similarity matrix times vector multiplication phase to two raw matrix times vector multiplication steps, hence avoiding the quadratic similarity matrix construction step. This "path folding" idea has been applied to accelerate clustering of big data sets in the field of text mining [17].

Most of the experiments were run on a quad-core server with AMD Opteron 8378 Processors running at 2.4GHz. Because JoinMap requires the Windows OS, experiments with JoinMap were performed on a Windows desktop with Intel 2.93GHz Core 2 Duo processors. Even though both machines are multicore, all experiments are single threaded and use a single core.

4.1 Data We evaluate BubbleCluster on both real and synthetic datasets. The first dataset, barley, consists of 65,357 genetic markers from a population of 90 individuals and 20% missing data. This species of barley has 7 true linkage groups. The second, larger switchgrass dataset of 548,281 genetic markers comes from a population size of 500 individual columns (with some duplicate coverage) in the data matrix and 65% missing data, with 18 true linkage groups. Due to its size, previous clustering efforts on this data focused only on the 113,326 highest-quality markers. We cluster both the 113K subset of markers and the complete 548K dataset in our experiments to demonstrate the scalability of our algorithm.

For scaling studies, we rely primarily on synthetic data. We generate synthetic data with SPAGHETTI, software which was created to simulate genetic marker data with real-life complications [27]. SPAGHETTI allows for a wide range of parameter settings, and we only list the most important ones here.

To simulate the very real-life complications (especially with respect to LOD score calculations) caused by missing data, we created datasets for a range of missing data rates, from 0 to 65%. In addition to the missing rate, we varied the number of markers from 12.5K to 400K, doubling the size at each increment. The population size was fixed at 300, the sequencing error rate at 0.1%, and the number of linkage groups at 10 in all experiments.

4.2 Evaluation Metric We use the overall F-score to evaluate the quality of each clustering. The F-score ranges from 0 to 1, and evaluates a test cluster C_i with respect to a "golden standard" cluster G_j in terms of precision and recall [30]. Formally, if $G_j \in \mathcal{G}$ is one golden standard cluster, then the F-score for a test cluster C_i with respect to G_j is defined as:

$$F(G_j, C_i) = \frac{2 p_{ij} r_{ij}}{r_{ij} + p_{ij}}$$

where r_{ij} and p_{ij} are recall and precision, respectively, and defined as $r_{ij} = \frac{|C_i \cap G_j|}{|G_j|}$ and $p_{ij} = \frac{|C_i \cap G_j|}{|C_i|}$. The *overall* F-score is a normalized, weighted sum of the F-scores for each golden standard cluster $G_j \in \mathcal{G}$. An overall F-score comparing a test clustering $\mathcal{C}$ with a golden standard clustering $\mathcal{G}$ is given by:

$$F(\mathcal{G}, \mathcal{C}) = \frac{1}{n} \sum_{j=1}^{k} |G_j| \max_{i=1 \ldots l} F(G_j, C_i)$$

where k is the number of true, or golden standard, clusters, l is the number of test clusters, and n is the total number of datapoints, i.e. $n = \sum_j |G_j|$. An F-score of 1, therefore, would indicate that a test clustering exactly matches a golden standard clustering.

4.3 Results Our results for real datasets are summarized in Table 1. We report the runtime as well as the overall F-score achieved by BubbleCluster for each dataset. The golden standard clusters used to calculate the F-score were found by single-linkage clustering as

Dataset	Markers	F-score	Time
Barley	64K	0.99927	15 s
Switchgrass	113K	0.974539	534 s (8.9 min)
Switchgrass	548K	0.989461	6779 s (1.9 hrs)

Table 1: Clustering performance on Barley and Switchgrass from the Joint Genome Institute using BubbleCluster. MSTmap and JoinMAP are unable to effectively handle data sets at this scale.

described in Section 4.1. If the assumptions stated in Section 3.2 hold, then single linkage clustering will provably find the correct clusters, given a divergence threshold smaller than λ_{sep}. A divergence threshold of $t = 1/8$ with nonmissing data threshold $n = 66$ was used to cluster the 65K barley dataset; for switchgrass, we used thresholds $t = 1/20$ and $n = 132$. The c parameter was set to 5 in all experiments. We motivate our choices of t and n in the Supplementary Text. We do not report F-score for the popular mapping tools MSTMap and JoinMap, because the size of these datasets was prohibitively large for these tools due to memory limitations. In fact, no mapping tool we know of has been successful in clustering genetic marker datasets at this scale.

Table 1 demonstrates that we achieve very accurate clusters in $O(m \log(m))$ time for m markers, a significant improvement over the $O(m^2)$ algorithms used by other genetic marker clustering tools. We emphasize that these datasets come from real-world sequence data, where missing data entries are not distributed in any predictable manner, such as having a uniform or Poisson shape, among the markers. Nonetheless, our algorithm handles missing data effectively and efficiently for all these datasets, recovering the true linkage groups with both precision and recall above 97%.

Table 2 underscores our ability to outperform existing genetic clustering methods as well as more general clustering methods applied to synthetic genetic marker data. Here, we show that even on relatively small datasets, where JoinMap and MSTMap complete the clustering stage in a fairly reasonable amount of time, we achieve nearly identical F-scores, but within a fraction of the time. In fact, the results are slightly biased in favor of JoinMap. Although this tool will automatically construct the single-linkage dendrogram from an input data matrix, it is up to the user to select which dendrogram edges to cut in order to produce the final clusters. We used our prior knowledge of the synthetic data clusters to select those edges which would produce the best clustering result. The time reported is only the time that it took JoinMap to output the dendrogram.

In the third row of Table 2, we give the best F-score achieved by the spectral-clustering-based PIC

Clustering	12.5K Markers		25K Markers	
	F-Score	Time	F-Score	Time
JoinMAP	0.99964	14 min	0.99982	46 min
MSTMap	0.99964	4.5 min	0.99982	20 min
PIC	0.47024	11 sec +(16 min)	0.60782	44 sec +(74 min)
Bubble	**0.99944**	**6 sec**	**0.99972**	**15 sec**

Table 2: Performance comparison of clustering algorithms using synthetic Spaghetti for 12.5K and 25K markers with 35% missing data and 0.1% error rate. All experiments run on Neumann, except for JoinMap that ran on the Windows machine. The number in parenthesis for the PIC algorithm is the preprocessing time for pairwise calculations.

algorithm. We took extensive measures to improve both the accuracy and efficiency of PIC, but could improve neither the F-scores nor the runtimes beyond those reported in Table 2. For example, we chose the best of several PIC runs with different counts of pseudo-eigenvectors, finding that k-means clustering in the 2-dimensional space spanned by two pseudo-eigenvectors performed best. Additionally, in the final stage of PIC we used k-means++[3], an improvement over Lloyd's k-means algorithm, in order to improve cluster quality. We include the running time of PIC given the similarity matrix as input, but we note that the algorithm necessitates the construction of this matrix, the time for which is included in parentheses and which contributes the $O(m^2)$ term in its time complexity. As stated in Section 4, a decomposition of the similarity matrix into a sparse matrix-sparse-matrix-vector product would improve running time, but even then, the F-score indicates that PIC falls short in terms of clustering quality.

The inability of PIC to produce competitive results in terms of clustering quality motivates the need for a more application-specific approach in this domain. Although spectral clustering methods enjoy popularity in the data mining community, we point out the difficulty in applying these methods to a problem with erroneous and/or missing data, a similarity function that cannot be expressed as an inner product, and an underlying structure that can only be exploited if the application-specific problem is well understood.

Our ability to scale, while simultaneously making effective use of more data availability, is demonstrated in Figure 4 and Table 3, respectively. Here, we increase the size of our synthetic dataset from 12.5K to 400K genetic markers. We report the clustering quality for both 35% and 65% missing data in terms of the errors we make; the entries in Table 3 are (1 -F-score) for each (dataset size, missing data rate) pair. The running times corresponding to Table 3 are plotted in Figure 4 on a log-log scale. We make two points

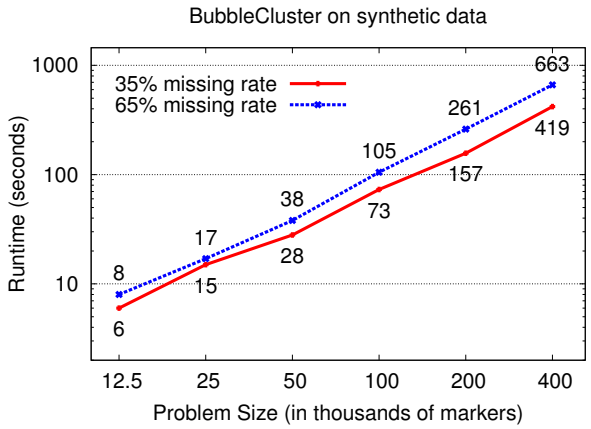


Figure 4: Runtimes showing how the BubbleCluster Algorithm performs on increasing dataset sizes. Both axes are on a logarithmic scale.

about these empirical results: (1) as Table 3 shows, the error we make in clustering decreases linearly, in almost exact correlation with the size of the data, and (2) the running time increases with $O(m \log m)$, promising reliable performance up to almost half a million markers, even with an enormous amount of missing data. Comparing Table 1 with these results, we believe the behavior of our algorithm in these experiments is predictive of its performance in real world.

5 Significance and Impact

Current approaches to genetic mapping were designed for a small data setting, and use algorithms that scale quadratically with the number of markers. These algorithms do not scale to the level needed to construct genetic maps in a modern setting, when markers are abundant. The value of the ultra-high-density genetic maps (millions of markers) that can be readily constructed with our algorithm is especially evident when only short-range (i.e., gene-scale) sequence assemblies are available. In this case, the power of genetic mapping and sequence assembly can be combined to represent genes in order along chromosomes. Our method exploits the underlying linear structure of chromosomes to avoid the expensive (quadratic) marker-by-marker pairwise calculation. The resulting linkage groups (i.e., marker clusters) are highly concordant with computationally expensive quadratic calculations, but our improved scaling allows far denser maps to be constructed with minimal computation.

After the formation of linkage groups, the next step in constructing a high quality genetic map is inferring the detailed ordering of markers along chromosomes, and estimating their genetic separation. Since

Synthetic Data Cluster Quality: (1− F-score)		
Dataset size	35% missing data:	65% missing data:
12.5K	5.6×10^{-4}	11.0×10^{-4}
25K	2.8×10^{-4}	4.0×10^{-4}
50K	1.5×10^{-4}	2.5×10^{-4}
100K	0.7×10^{-4}	1.3×10^{-4}
200K	0.38×10^{-4}	0.7×10^{-4}
400K	0.18×10^{-4}	0.3×10^{-4}

Table 3: Clustering performance on synthetic data of varying dataset sizes. The lower the (1− F-score) value, the better the algorithm recovers true clusters.

our method takes into account the linear structure of chromosomes from the start, the result is an approximate marker ordering that is an excellent starting point for detailed marker order by simulated annealing or other similar methods that explore short-range perturbations of our approximate ordering. In fact, the representative point order found by our algorithm for the barley dataset described in Section4.1, was highly correlated with the true marker order in barley linkage groups: for 6 out of 7 groups, the Spearman Rank Correlation Coefficient ρ was above 0.9. When we impose a stricter ϵ separation between representative points as described in Section 3.2, we find that 2 of the linkage groups' representative points are in a perfect order ($\rho = 1$). We consider this a preliminary result, but it is an interesting avenue for future research to investigate the conditions under which we can quickly and precisely identify the true marker order in all clusters, given a missing data rate, a dataset and population size, and assumptions about the linearity of these clusters.

An important application of this method is in the efficient construction of ultra-dense genetic maps for large and complex genomes that are filled with repetitive sequence that frustrate genome assembly but do not limit the number of genetic markers. The most economically important of these genomes are various grasses, including crops grown for food (e.g., barley and wheat, whose genome sizes are two- to seven-fold larger than the human genome) or as biofuel feedstocks (e.g, switchgrass and miscanthus, polyploids that contain multiple, subtly different copies of a basic genome). Ordering genes and genetic markers along chromosomes enables the identification of markers associated with key traits (e.g., grain or biomass yield, drought and pest tolerance) and the development of approaches such as marker-assisted selection [7] and genomic-selection [11] for genetic improvement of these important crops.

Acknowledgements

We thank Nicholas Tinker for providing us with his SPAGHETTI simulation software. 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!. Stefanie Jegelka was supported in part by the Office of Naval Research under contract/grant number N00014-11-1-0688.

References

[1] B. Andreopoulos, A. An, X. Wang, and M. Schroeder. A roadmap of clustering algorithms: finding a match for a biomedical application. *Briefings in Bioinformatics*, 10(3):297–314, 2009.

[2] M. Ankerst, M. M. Breunig, H. Kriegel, and J. Sander. OPTICS: ordering points to identify the clustering structure. *ACM SIGMOD Record*, 28(2):49–60, 1999.

[3] D. Arthur and S. Vassilvitskii. k-means++ the advantages of careful seeding. In *ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[4] M. Badoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *STOC*, 2002.

[5] T. Brown. *Introduction to genetics: a molecular approach*. Garland Science, 2012.

[6] 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.

[7] BCY Collard and DJ 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.

[8] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *STOC*, 2011.

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

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

[11] BJ Hayes, HA Lewin, and ME Goddard. The future of livestock breeding: genomic selection for efficiency, reduced emissions intensity, and adaptation. *Trends in Genetics*, 2012.

[12] B.N. Jackson, P.S. Schnable, and S. Aluru. Consensus genetic maps as median orders from inconsistent sources. *IEEE Trans on Comp. Biology and Bioinformatics*, 5(2), 2008.

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

[14] A. Kozik and R. Michelmore. MadMapper and CheckMatrix – python scripts to infer orders of genetic markers and for visualization and validation of genetic maps and haplotypes. In *Proceedings of the Plant and Animal Genome XIV Conference, San Diego*, 2006.

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

[16] F. Lin and W. W. Cohen. Power iteration clustering. In *Proc. of ICML*, volume 10, 2010.

[17] F. Lin and W. W. Cohen. A very fast method for clustering big text datasets. In *Proc. of ECAI*, pages 303–308, 2010.

[18] S.P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Information Theory*, 28(2):129–137, 1982.

[19] GRA Margarido, AP Souza, and AAF Garcia. Onemap: software for genetic mapping in outcrossing species. *Hereditas*, 144(3):78–79, 2007.

[20] ML Metzker. Sequencing technologies—the next generation. *Nature Reviews Genetics*, 11(1):31–46, 2009.

[21] A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE TKDE*, 15(3):515–528, 2003.

[22] P. Rastas, L. Paulin, I. Hanski, R. Lehtonen, and P. Auvinen. Lep-MAP: fast and accurate linkage map construction for large SNP datasets. *Bioinformatics*, page advance access, 2013.

[23] M. V Rockman and L. Kruglyak. Recombinational landscape and population genomics of caenorhabditis elegans. *PLoS genetics*, 5(3):e1000419, 2009.

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

[25] M. Shindler, A. Wong, and A. Meyerson. Fast and accurate k-means for large datasets. In *NIPS*, 2011.

[26] 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.

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

[28] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[29] U. von Luxburg, S. Bubeck, S. Jegelka, and M. Kaufmann. Consistent minimization of clustering objective functions. In *NIPS*, 2007.

[30] S. Wagner and D. Wagner. *Comparing Clusterings-An Overview*. Universität Karlsruhe, Fakultät für Informatik, 2007.

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

6 Appendix

Choosing thresholds for BubbleCluster A full exploration of the entire biologically relevant parameter space (including, for instance, number and type of markers, sampled population size, missing data rate, error rates, and number and size of underlying linkage groups), and how the interplay between these parameters affect both the efficiency and accuracy of our algorithm, is beyond the scope of this paper. However, we can motivate our choices of the divergence and non-missing data thresholds used in the empirical evaluation of the BubbleCluster algorithm based on a few assumptions about the missing data rate in genetic marker datasets. Here, we explain how we were able to set thresholds that we believed would help BubbleCluster find high-quality clusters quickly and efficiently.

If the clusters we are attempting to recover are indeed well-separated, then there exists a λ_{sep} such that $d(x_i, x_j) > \lambda_{\text{sep}}$ for any two markers $x_i \in C_i^*$ and $x_j \in C_j^*$, $C_i^* \neq C_j^*$. Since we choose $d = 1/LOD$ in practice, then it must be true that

$$\frac{1}{LOD(x_i, x_j)} > \lambda_{\text{sep}} \Rightarrow LOD(x_i, x_j) < \frac{1}{\lambda_{\text{sep}}}$$

However, λ_{sep} may be very small – in our assumptions we make clear that no LOD score between a marker $x_j \in C_j^*$ and a marker $x_i \in C_i^*$ is above $1/\lambda_{\text{sep}}$ in order for clusters to be "well-separated." If λ is too small, then setting $t < \lambda_{\text{sep}}$ will cause our algorithm to sprout too many new clusters and representative points, increasing computation time significantly.

What we use in practice, therefore, is a threshold distance t which will make it very unlikely that $d(x_i, x_j) < t$ when x_i and x_j are in different clusters. For a given t, let p be the probability that $LOD(x_i, x_j) > 1/t$ even though x_i and x_j are in different clusters. In other words, when x_j is a representative point, p is the probability that setting the threshold divergence to t will cause us to mistakenly assign a random marker x_i to the same cluster as the point x_j, because x_i now falls within the threshold "bubble" surrounding x_j. A schematic depicting this situation is shown in Figure 5.

Let n_{comp} be the total number of LOD score computations that we make in Phase 1 of our algorithm, which is dependent upon the number of representative points we have per cluster. The probability that none of these LOD scores is above the threshold, i.e. the probability that we make no errors in assigning two markers to the same linkage group, is then:

$$P(\text{no mistakes}) = (1-p)^{n_{\text{comp}}}$$

The LOD score, as explained in Section 2, is a logarithm

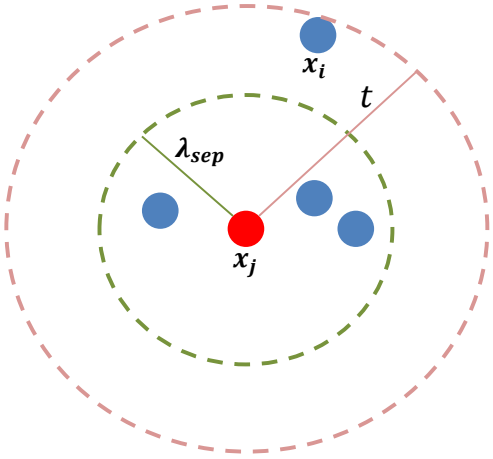


Figure 5: A "mistake" occurs in BubbleCluster if $t > \lambda_{\text{sep}}$, and we encounter two markers x_i and x_j which do not belong to the same cluster but do lie within a threshold divergence t of each-other.

of odds, telling us how much more likely it is that two markers are genetically linked than that they are not. In other words, for markers x_i and x_j it is defined as:

$$LOD(x_i, x_j) = \log_{10} \frac{P(\text{linkage})}{P(\text{no linkage})}$$

where $P(\text{linkage})$ is determined by calculating R and NR, the number of recombinants and non-recombinants in the non-missing data entries common to x_i and x_j. $P(\text{no linkage})$ is $(1/2)^{R+NR}$, the probability that a random pair of markers will match in $R+NR$ positions purely by chance. Thus, by the definition of the LOD score,

$$p = P\left(\frac{P(\text{linkage})}{P(\text{no linkage})} < 10^{\left(\frac{1}{t}\right)}\right) \leq \frac{1}{10^{\left(\frac{1}{t}\right)}}$$

$$\Rightarrow P(\text{no mistakes}) \geq \left(1 - \frac{1}{10^{\left(\frac{1}{t}\right)}}\right)^{n_{\text{comp}}}$$

If we want to ensure that $P(\text{no mistakes}) > 1 - e$, then we require that

$$\left(1 - \frac{1}{10^{\left(\frac{1}{t}\right)}}\right)^{n_{\text{comp}}} > 1 - e$$

$$\Rightarrow t < \frac{1}{\log_{10}\left(\frac{1}{1-(1-e)^{\frac{1}{n_{\text{comp}}}}}\right)}$$

An upper bound for n_{comp} is simply $\binom{m}{2}$ for m markers, which would imply that we compute all the pairwise LOD scores for all the markers in our dataset, a grossly pessimistic assumption. In order to guarantee that

P(no mistakes) is high, we must use this upper bound to set the divergence threshold t.

We must also keep in mind, however, that the amount of missing data limits our choice of t, because the maximum achievable LOD score for a particular marker is dependent upon the number of non-missing entries in the data matrix for that marker. If a marker x_i has n_{nm} non-missing entries, the most similar it can be to another marker x_j is to match x_j in genotype perfectly in all n_{nm} non-missing data entries, giving $NR = n$. As NR approaches n_{nm}, R approaches 0, and the LOD achievable by x_i approaches its upper bound:

$$\lim_{NR \to n_{\text{nm}}} LOD(x_i, x_j)$$

$$= \lim_{NR \to n_{\text{nm}}} \log_{10} \left(\frac{\left(1 - \frac{R}{R+NR}\right)^{NR} \left(\frac{R}{R+NR}\right)^{R}}{0.5^{NR+R}} \right)$$

$$= n_{\text{nm}} \log_{10} 2$$

For a threshold divergence of t in Phase 1 of our algorithm, x_i must achieve a LOD score $LOD(x_i, x) > \frac{1}{t}$ with at least one other data point x in order to join any existing cluster. Therefore, the number of non-missing entries n_{nm} in x_i must satisfy:

$$n_{\text{nm}} > \frac{1}{t \log_{10} 2}$$

Clearly, if n_{nm} is less than this constant for a given t, then x_i will never achieve a LOD score with another point in the data set which would allow it to join an existing cluster, and we should exclude this point from the high-quality datapoint set in Phase 1 of our algorithm. However, we can be more aggressive in excluding non-informative data points.

Given a mean missing data rate μ, we expect that the number of non-missing entries a data point x_i, with n_{nm} nonmissing entries, will share with any other point x will be:

$$E\left[\text{shared nonmissing entries}(x_i)\right] = (1 - \mu) n_{\text{nm}}$$

Therefore if n_{nm} satisfies:

$$n_{\text{nm}} > \frac{1}{(1 - \mu) t \log_{10} 2}$$

then we expect x_i to fall within the threshold divergence t of at least one other data point x_j. The assumption that points within a cluster are well-connected makes it probable that if $d(x_i, x_j) < t$, then $d(x_i, x) < t$ for all points x within a small divergence measure of x_j, raising our expectation that the divergence of x_i and at least one other point will be less than t. Thus, we want to include x_i in the high-quality set in Phase 1 of BubbleCluster.

Our selection of t and the cutoff for non-missing entries n is thus a balancing act, where we try to ensure that t guarantees a high P(no mistakes), but is not so small that a large fraction of our data would be excluded from Phase 1, which we rely on to build an initial sketch of each cluster. The threshold settings stated in our paper were set according to the line of reasoning presented herein, choosing lower divergence thresholds for higher missing data rates in order to maximize the precision in cluster assignments without penalizing the efficiency of our algorithm.

The LOD score as a similarity measure Here we provide a simple counterexample to show that the reciprocal of the LOD score is not a valid distance metric. Suppose that we have three genetic markers x_1, x_2, and x_3, where:

$$x_1 = \begin{pmatrix} - & - & - & A & B & - & B & B \end{pmatrix}$$

$$x_2 = \begin{pmatrix} A & A & A & - & B & - & B & B \end{pmatrix}$$

$$x_3 = \begin{pmatrix} A & A & A & A & - & - & - & - \end{pmatrix}$$

Then:

$$LOD(x_1, x_2) = \log_{10}(8) \Rightarrow 1/LOD(x_1, x_2) = 1/\log_{10}(8)$$

$$LOD(x_2, x_3) = \log_{10}(8) \Rightarrow 1/LOD(x_2, x_3) = 1/\log_{10}(8)$$

$$LOD(x_1, x_3) = \log_{10}(2) \Rightarrow 1/LOD(x_1, x_3) = 1/\log_{10}(2)$$

$$\Rightarrow 1/LOD(x_1, x_3) > 1/LOD(x_1, x_2) + 1/LOD(x_2, x_3)$$

Thus, for the divergence measure $1/LOD$, the triangle inequality does not hold.