Summer 2019

K-haplotypes

Yudi Zhang

k-HAPLOTYPES

YUDI ZHANG, KARIN DORMAN

Clustering next generation sequencing reads is an important first step for many amplicon datasets Knight et al. [18]. What's more, errors generated during gene sequencing, such as Illumina sequencing, can be addressed by clustering sequences into OTUs ([3, 23]).

Methods to group similar observations in datasets are widely developed in statistics. The k-means algorithms (Lloyd's [20]; MacQueen [21]; Hartigan's algorithm [12]; Hartigan and Wong [13]) and centroid-based hierarchical clustering Guha, Rastogi, and Shim [10] are widely used for clustering, but they can only be used for numerical observations. For categorical datasets, these methods are not suitable. Many methods have been proposed for clustering categorical data ([7–9, 11, 14, 16]). The method proposed by Guha, Rastogi, and Shim [11] uses the the number of common neighbors (a pair of points are neighbors if the value given by the distance/similarity function is greater than a user defined threshold) shared by two points as clustering criterion, where all pairwise neighbors are involved, being computationally costly. Similarly, the hierarchical clustering proposed by Gowda and Diday [9] is hard to implement for large datasets due to the need to compute all pairwise distances. The k-modes algorithm proposed by Huang [16] is similar to k-means (MacQueen [21]). Rather than finding the mean of each cluster, its centers are defined as the modes. The objective function is the sum of Hamming distances between the observations and their assigned cluster centers. The proposed optimization algorithm assigns each observation to its closest center then update the center to be the mode at each coordinate and ends when no observation is transferred after a full pass through of all the observations.

Date: March 2019.

While in the past few years, many scholars have proposed methods to cluster and denoise sequences ([1, 2, 5]). DADA2 [2] is a divisive algorithm for denoising amplicon data. It first dereplicates the input reads as a list of unique sequences and their abundances, then performs an iterative partitioning algorithm by testing whether each unique sequence is explained by the current model, and prioritizing for new clusters those unique sequences with smallest p-values.

Few of the existing methods took advantage of quality scores which are associated with the probability of error in the reads [4]. Quality scores are most often used during data pre-processing, *e.g.* to remove reads with high error probabilities, but then they are often ignored. However, quality scores important information that may improve clustering results. Qcluster proposed by Comin, Leoni, and Schimd [4] extends some D-type similarity measure statistics proposed by Wan et al. [25], Reinert et al. [24] that take quality scores into consideration. They incorporate quality scores to define the probability of a word w at position i of a read is correct, and use the sum of probabilities of all the occurrences of w to redefine the D-type similarity statistics between reads. Qcluster then performs k-means to cluster the reads based on the similarity statistics.

This article describes a practical, unsupervised hard clustering method for massive amplicon sequence datasets which take the quality scores into consideration. We adapted the k-means algorithms (Lloyd's [20]; MacQueen [21]; Hartigan's algorithm [12]) to maximize the discrete indicator variables that assign each observation to one of K clusters. Particularly, we compared the performance of our algorithms in terms of speed and accuracy on simulated and real datasets. We also compare our method to DADA2 [2], another model-based amplicon read clustering method.

1. Methodology

1.1. **Model.** Consider an amplicon sequence dataset comprising n single-end reads $\mathcal{R} = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_n\}$, along with the quality score sequences $\mathcal{Q} = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n\}$, both of length m. The quality scores are discretized error probabilities. Usually $p_{ij} = 10^{-\left(\frac{q_{ij}-33}{10}\right)}$ is the approximate probability of an error at position j of read i, and all probabilities for read i are stored in vector $\boldsymbol{p}_i$.

We consider the hard-clustering problem, where every read originates from a single cluster. Let $\boldsymbol{C} = (C_1, C_2, \ldots, C_n), C_i \in \{1, 2, \ldots, K\}$ be unobserved integers that indicate which of the K source haplotypes generated each read. If we assume the quality scores are correct assessments of the error probability, then the ith read $\boldsymbol{r}_i$ is sampled from distribution

$$f(\boldsymbol{r}_i) = \sum_{k=1}^{K} \mathbb{1}\left\{C_i = k\right\} \phi_k(\boldsymbol{r}_i; \boldsymbol{h}_k, \boldsymbol{p}_i),$$

where $\phi_k(\boldsymbol{r}_i; \boldsymbol{h}_k, \boldsymbol{p}_i)$ is the probability of generating the ith observation from the kth cluster, given the kth haplotype sequence $\boldsymbol{h}_k = (h_{k1}, h_{k2}, \ldots, h_{km})$ and the computed error probabilities $\boldsymbol{p}_i$. The log likelihood is

$$l(\boldsymbol{C}, \mathcal{H}, \boldsymbol{\theta} \mid \mathcal{R}) = \sum_{i=1}^{n} \ln \left[\sum_{k=1}^{K} \mathbb{1}\left\{C_i = k\right\} \phi_k(\boldsymbol{r}_i; \boldsymbol{h}_k, \boldsymbol{p}_i) \right]. \tag{1}$$

If we further assume that sites are independent and assume all substitutions are equally likely when there has been an error, the likelihood of the ith read $\boldsymbol{r}_i$ given $C_i = k$ reduces to

$$\phi_k(\boldsymbol{r}_i; \boldsymbol{h}_k) = \prod_{j=1}^{m} (1 - p_{ij})^{\mathbb{1}\left\{r_{ij} = h_{kj}\right\}} \left(\frac{p_{ij}}{3}\right)^{\mathbb{1}\left\{r_{ij} \neq h_{kj}\right\}}. \tag{2}$$

1.2. **Algorithms.** The proximity of the required optimization to the k-means problem [26] suggests algorithms designed for that purpose may work well. Three k-means algorithms (Lloyd's [20]; MacQueen [21]; Hartigan and Wong [13]) can be generalized to fit our model.

1.2.1. *Lloyd's Algorithm.* Lloyd's algorithm [20] prescribes alternating updates of the haplotypes $\mathcal{H}$ with updates of the cluster assignments $\boldsymbol{C}$.

(1) At iteration t, we maximize the likelihood over the cluster indicators as

$$C_i^{(t)} = \arg\max_k \phi_k(\boldsymbol{r}_i; \boldsymbol{h}_k),$$

an $O(Kmn)$ operation. When there is a tie, we prioritize the cluster k with a lower rank, i.e. smaller k.

(2) For site j in cluster k, we find the most likely haplotype nucleotide as

$$h_{kj}^{(t)} = \arg\max_{\mathrm{N}\in\{\mathrm{A,C,G,T}\}} \sum_{i=1}^{n} \mathbb{1}\left\{C_i^{(t)} = k\right\}\left[\mathbb{1}\left\{r_{ij} = \mathrm{N}\right\}\ln(1-p_{ij}) + \mathbb{1}\left\{r_{ij} \neq \mathrm{N}\right\}\ln\left(\frac{p_{ij}}{3}\right)\right],$$

an $O(4Kmn)$ operation. We define an order of nucleotides $\{A < C < G < T\}$, such that the haplotype is updated to the lower rank nucleotide if there is a tie.

(3) Repeat from Step 1 until there is no change in either step.

We propose some adjustments to the algorithm to increase computational efficiency. We precalculate $p_{ij} = 10^{-\left(\frac{q_{ij}-33}{10}\right)}$ first. Then to reduce the complexity of Step 1 to less than $O((m+K)n)$ and Step 2 to less than $O(n+4Km)$, we first allocate $O(2nm)$ space to store $d_{ij} := \ln(1-p_{ij})$ and $e_{ij} := \ln\left(\frac{p_{ij}}{3}\right)$ for $1 \le i \le n$ and $1 \le j \le m$, saving repeated calculations of the log. The cost is a one-time $O(jn)$ calculation. The benefit depends on the cost of the implementation of ln(). For Step 2, allocate $e_{kj\mathrm{N}}$ in $O(4Km)$ space for each $1 \le k \le K$, $1 \le j \le m$ and $\mathrm{N} \in \{\mathrm{A,C,G,T}\}$ and initialize it to

$$e_{kj\mathrm{N}} = \sum_{i=1}^{n} \mathbb{1}\left\{C_i^{(t)} = k\right\}\left[\mathbb{1}\left\{r_{ij} = N\right\}\ln(1-p_{ij}) + \mathbb{1}\left\{r_{ij} \neq N\right\}\ln\left(\frac{p_{ij}}{3}\right)\right],$$

at a one-time $O(4Kmn)$ cost. When $C_i^{(t)}$ changes from cluster l to k in Step 1, then the updates are

$$\begin{aligned} e_{kj\mathrm{N}} &= e_{kj\mathrm{N}} + \mathbb{1}\{r_{ij} = N\}\ln(1-p_{ij}) + \mathbb{1}\{r_{ij} \neq N\}\ln\left(\frac{p_{ij}}{3}\right) \\ e_{lj\mathrm{N}} &= e_{lj\mathrm{N}} - \mathbb{1}\{r_{ij} = N\}\ln(1-p_{ij}) - \mathbb{1}\{r_{ij} \neq N\}\ln\left(\frac{p_{ij}}{3}\right), \end{aligned}$$

at less than $O(n)$ cost. Step 2 now requires less than $O(n+4Km)$ calculations. For Step 1, allocate $O(nK)$ space to store $v_{ik} := \ln \phi_k(\boldsymbol{r}_i; \boldsymbol{h}_k)$ with a one-time setup cost of $O(Kmn)$. If $\boldsymbol{h}_k$ is updated in Step 2, then set

$$v_{ik}^{(t)} = \ln \phi_k(\boldsymbol{r}_i; \boldsymbol{h}_k^{(t)}),$$

for all $1 \leq i \leq n$ at $O(mn)$. Otherwise, $v_{ik}^{(t)} = v_{ik}^{(t-1)}$. (Note: It is not necessary to store $v_{ik}^{(t)}$ and $v_{ik}^{(t+1)}$.) The cost of Step 1 is now reduced to $O((m+K)n)$. Moreover, set $v'_{ik} = v_{ik}^{(t-1)}$. If $h_{kj}^{(t)}$ is updated in Step 2, update

$$\begin{aligned} v'_{ik} &= v'_{ik} + \left\{\mathbb{1}\left\{r_{ij} = h_{kj}^{(t)}\right\}\left[\ln(1-p_{ij}) - \ln\left(\frac{p_{ij}}{3}\right)\right]\right. \\ &\quad \left. + \mathbb{1}\left\{r_{ij} \neq h_{kj}^{(t)}, r_{ij} = h_{kj}^{(t-1)}\right\}\left[\ln\left(\frac{p_{ij}}{3}\right) - \ln(1-p_{ij})\right]\right\} \end{aligned}$$

for all $1 \leq i \leq n$. Once $j = m$, then $v_{ik}^{(t)} = v'_{ik}$. The operations of updating v_{ik} are further reduced to $O(m_h n)$, where $m_h \leq m$ are the number of sites where $\boldsymbol{h}_k$ is updated.

Lloyd's Algorithm: Initialization

Data: Amplicon sequence dataset $\mathcal{R}$ of size $n \times m$; Haplotype sequences $\mathcal{H}^{(0)}$ of size $K \times m$.
Result: Reads assignment C_i; Updated haplotypes $\mathcal{H}$.

`/* initialize log probabilities to zero */`
for $k = 1, \ldots, K$ **do**
  for $j = 1, \ldots, m$ **do**
    for $\mathrm{N} \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ **do**
      $e_{kj\mathrm{N}} = 0$;
  for $i = 1, \ldots, n$ **do**
    $v_{ik} = 0$;

`/* compute log read probabilities for each cluster */`
for $i = 1, \ldots, n$ **do**
  for $k = 1, \ldots, K$ **do**
    $v_{ik} = \ln \phi_k(\boldsymbol{r}_i; \boldsymbol{h}_k^{(0)})$;
  $C_i = \arg\max_{k=1,\ldots,K} v_{ik}$;

`/* update log probability of nucleotide */`
for $k = 1, \ldots, K$ **do**
  for $j = 1, \ldots, m$ **do**
    for $\mathrm{N} \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ **do**
      for $i = 1, \ldots, n$ **do**
        if $\mathrm{N} = r_{ij}, C_i = k$ **then**
          $e_{kj\mathrm{N}} = e_{kj\mathrm{N}} + \ln(1 - p_{ij})$;
        else
          $e_{kj\mathrm{N}} = e_{kj\mathrm{N}} + \ln\left(\frac{p_{ij}}{3}\right)$;
    $h_{kj} = \arg\max_{\mathrm{N} \in \{\mathrm{A},\mathrm{C},\mathrm{G},\mathrm{T}\}} e_{kj\mathrm{N}}$;

Lloyd's Algorithm: Iteration $t = 2, \ldots$

for $i = 1, \ldots, n$ **do**
  for $k = 1, \ldots, K$ **do**
    if $h_{kj}^{(t)} \neq h_{kj}^{(t-1)}$ **then**
      $v_{ik} = v_{ik} + \mathbb{1}\left\{r_{ij} = h_{kj}^{(t)}\right\}(\ln(1-p_{ij}) - \ln\left(\frac{p_{ij}}{3}\right)) +$
        $\mathbb{1}\left\{r_{ij} \neq h_{kj}^{(t)}, r_{ij} = h_{kj}^{(t-1)}\right\}(\ln\left(\frac{p_{ij}}{3}\right) - \ln(1-p_{ij}))$;
  $C_i^{(t)} = \arg\max_k v_{ik}$;
  if $C_i^{(t)} \neq C_i^{(t-1)}$ **then**
    Let $l = C_i^{(t-1)}$, $k = C_i^{(t)}$;
    for $j = 1, \ldots, m$ **do**
      for $\mathrm{N} \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ **do**
        $e_{kj\mathrm{N}} = e_{kj\mathrm{N}} + \mathbb{1}\{r_{ij} = N\}\ln(1-p_{ij}) + \mathbb{1}\{r_{ij} \neq N\}\ln\left(\frac{p_{ij}}{3}\right)$;
        $e_{lj\mathrm{N}} = e_{lj\mathrm{N}} - \mathbb{1}\{r_{ij} = N\}\ln(1-p_{ij}) - \mathbb{1}\{r_{ij} \neq N\}\ln\left(\frac{p_{ij}}{3}\right)$;
for $k = 1, \ldots, K$ **do**
  for $j = 1, \ldots, m$ **do**
    for $\mathrm{N} \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ **do**
      for $i = 1, \ldots, n$ **do**
        if $\mathrm{N} = r_{ij}, C_i = k$ **then**
          $e_{kj\mathrm{N}} = e_{kj\mathrm{N}} + \ln(1-p_{ij})$;
        else
          $e_{kj\mathrm{N}} = e_{kj\mathrm{N}} + \ln\left(\frac{p_{ij}}{3}\right)$;
    $h_{kj}^{(t)} = \arg\max_{\mathrm{N}\in\{\mathrm{A},\mathrm{C},\mathrm{G},\mathrm{T}\}} e_{kj\mathrm{N}}$;

1.2.2. *MacQueen's Algorithm.* MacQueen [21] suggested a slight modification on Lloyd's algorithm: update the affected haplotypes in $\mathcal{H}$ after every change to $\boldsymbol{C}$. Hence, the computation cost differs mainly during iteration.

In the initialization stage, we use the same method as in Lloyd's algorithm, except that at the end we need to update v_{ik}. In the iteration stage, we loop through all the observations $\boldsymbol{r}_i$. If read $\boldsymbol{r}_i$'s assignment $C_i^{(t)}$ changes from cluster l to k, we update $e_{kj\mathrm{N}}$ and $e_{lj\mathrm{N}}$ as what we did for Lloyd's algorithm at a $O(4m)$ cost. Then if the haplotypes h_{kj} and h_{lj}

changed, we update v_{ik} and v_{ij} using Step 3 at $O(nm)$ cost. Hence the entire cost for the iteration step of MacQueen is $O(2(4+n)mn)$. It is much faster than Lloyd's since while we are looping i, we update centroids as well.

Macqueen's Algorithm: Initialization

Data: Amplicon sequence dataset $\mathcal{R}$ of size $n \times m$; Haplotype sequences $\mathcal{H}^{(0)}$ of size $K \times m$.
Result: Reads assignment C_i; Updated haplotypes $\mathcal{H}$.

```
/* initialize log probabilities to zero                                    */
```
for $k = 1, \ldots, K$ **do**
  for $j = 1, \ldots, m$ **do**
    for $\mathrm{N} \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ **do**
      $e_{kj\mathrm{N}} = 0$;

```
/* compute log read probabilities for each cluster                         */
```
for $i = 1, \ldots, n$ **do**
  for $k = 1, \ldots, K$ **do**
    $v_{ik} = \ln \phi_k(\boldsymbol{r}_i; \boldsymbol{h}_k^{(0)})$;
  $C_i = \arg\max_{k=1,\ldots,K} v_{ik}$;
  `/* update log probability of each cluster, position, haplotype nucleotide */`
  for $j = 1, \ldots, m$ **do**
    for $\mathrm{N} \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ **do**
      $e_{C_i j\mathrm{N}} = e_{C_i j\mathrm{N}} + \mathbb{1}\{r_{ij} = N\} \ln(1 - p_{ij}) + \mathbb{1}\{r_{ij} \neq N\} \ln\left(\frac{p_{ij}}{3}\right)$;

```
/* compute most likely haplotype nucleotides and update v_ik               */
```
for $k = 1, \ldots, K$ **do**
  for $j = 1, \ldots, m$ **do**
    $h_{kj}^{(1)} = \arg\max_{\mathrm{N} \in \{\mathrm{A},\mathrm{C},\mathrm{G},\mathrm{T}\}} e_{kj\mathrm{N}}$;
    if $h_{kj}^{(1)} \neq h_{kj}^{(0)}$ **then**
      for $i = 1, \ldots, n$ **do**
        $v_{ik} = v_{ik} + \mathbb{1}\left\{r_{ij} = h_{kj}^{(1)}\right\} \left(\ln(1 - p_{ij}) - \ln\left(\frac{p_{ij}}{3}\right)\right) +$
        $\mathbb{1}\left\{r_{ij} \neq h_{kj}^{(1)}, r_{ij} = h_{kj}^{(0)}\right\} \left(\ln\left(\frac{p_{ij}}{3}\right) - \ln(1 - p_{ij})\right)$;

Macqueen's Algorithm: Iteration $t = 2, \ldots$

for $i = 1, \ldots, n$ **do**

  $C_i^{(t)} = \arg\max_k v_{ik}$;

  if $C_i^{(t)} \neq C_i^{(t-1)}$ **then**

    Let $l = C_i^{(t-1)}$, $k = C_i^{(t)}$;

    for $j = 1, \ldots, m$ **do**

      for $\mathrm{N} \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ **do**

        $e_{kj\mathrm{N}} = e_{kj\mathrm{N}} + \mathbb{1}\{r_{ij} = N\} \ln(1 - p_{ij}) + \mathbb{1}\{r_{ij} \neq N\} \ln\left(\frac{p_{ij}}{3}\right)$;

        $e_{lj\mathrm{N}} = e_{lj\mathrm{N}} - \mathbb{1}\{r_{ij} = N\} \ln(1 - p_{ij}) - \mathbb{1}\{r_{ij} \neq N\} \ln\left(\frac{p_{ij}}{3}\right)$;

      for $p \in \{k, l\}$ **do**

        $h_{pj}^{(t)} = \arg\max_{\mathrm{N} \in \{\mathrm{A},\mathrm{C},\mathrm{G},\mathrm{T}\}} e_{pj\mathrm{N}}$;

        if $h_{pj}^{(t)} \neq h_{pj}^{(t-1)}$ **then**

          for $i1 = 1, \ldots, n$ **do**

            $v_{i1p} = v_{i1p} + \mathbb{1}\left\{r_{i1j} = h_{pj}^{(t)}\right\} \left(\ln(1 - p_{i1j}) - \ln\left(\frac{p_{i1j}}{3}\right)\right) +$

            $\mathbb{1}\left\{r_{i1j} \neq h_{pj}^{(t)}, r_{i1j} = h_{pj}^{(t-1)}\right\} \left(\ln\left(\frac{p_{i1j}}{3}\right) - \ln(1 - p_{i1j})\right)$;

1.2.3. *Hartigan and Wong Algorithm.* Hartigan's algorithm [12] improves on Lloyd's and MacQueen's algorithms by only assigning the ith read $\boldsymbol{r}_i$ to a new cluster if the objective function is guaranteed to improve. In particular, it takes into account any change in $\mathcal{H}$ that will occur as a result of the move and makes sure that, even with this change, the value of the objective function will still improve. Lloyd's [20] and MacQueen's [21] algorithms do not consider any change that will occur to $\mathcal{H}$. Hartigan and Wong [13] additionally define a "live" set to decide when to terminate the algorithm and to avoid unnecessary calculations. A cluster k is no longer a "live" set only after a full cycle of the observations is made and none of them moved to or from k.

In our context, the initialization stage is the same as Lloyd's, except we additionally initialize live set $live[k] = n + 1$ at $O(K)$ cost. If $i \leq \text{live}[C_i]$, then $\boldsymbol{r}_i$ is in a live cluster, and the following iteration phase will consider moving $\boldsymbol{r}_i$ to any clusters. Otherwise, the next iteration will only loop through the clusters k such that $\{k : \text{live}[k] \geq i, k = C_i\}$. Additionally, if we decide to move $\boldsymbol{r}_i$ from cluster k to l, we update $\text{live}[k] = \text{live}[l] = n + 1 + i$. Then, after a full cycle of the observations, we deduct n from live$[k]$. The live set concept reduce the computational time since sometimes the loop over K is less than $O(K)$ cost. During the iteration step, for a particular read $\boldsymbol{r}_i$ we loop through every live cluster k. Then, we first decide the new haplotype $\boldsymbol{h}_k$ after moving $\boldsymbol{r}_i$ out of its current assigned cluster C_i or to another cluster k by computing e_{kjn} at $O(4m)$ cost. To check which cluster could improve the objective function (1) the most, we only need to compute the differences of the log probabilities brought by the changed position j of the affected haplotypes $\boldsymbol{h}_k$. Set $\text{diff}'_k = \text{diff}_k^{(t-1)}$. If h_{kj} is updated, we first initialize the differences as:

$$\begin{aligned}
\text{diff}'_k &= \text{diff}'_k + \mathbb{1}\left\{r_{ij} = h_{kj}^{(t)}\right\} \ln(1 - p_{ij}) + \mathbb{1}\left\{r_{ij} \neq h_{kj}^{(t)}\right\} \ln\left(\frac{p_{ij}}{3}\right), k \neq c_i^{t-1} \\
\text{diff}'_{c_i^{t-1}} &= \text{diff}'_{c_i^{t-1}} - \mathbb{1}\left\{r_{ij} = h_{c_i^{t-1}j}^{(t-1)}\right\} \ln(1 - p_{ij}) - \mathbb{1}\left\{r_{ij} \neq h_{c_i^{t-1}j}^{(t-1)}\right\} \ln\left(\frac{p_{ij}}{3}\right)
\end{aligned}$$

at $O(m)$ cost. Then for the other observations with in cluster k, update the difference as:

$$\begin{aligned}
\text{diff}'_k = \text{diff}'_k + \Bigg\{ & \mathbb{1}\left\{r_{ij} = h_{kj}^{(t)}\right\} \left[\ln(1 - p_{ij}) - \ln\left(\frac{p_{ij}}{3}\right)\right] \\
& + \mathbb{1}\left\{r_{ij} \neq h_{kj}^{(t)}, r_{ij} = h_{kj}^{(t-1)}\right\} \left[\ln\left(\frac{p_{ij}}{3}\right) - \ln(1 - p_{ij})\right] \Bigg\}
\end{aligned}$$

at a cost much less than $O(mn)$. Additionally, we keep track of updated positions j of the haplotypes and once we decide to transfer of read $\boldsymbol{r}_i$ from its last assigned cluster k to a new cluster l, we only update the changed position for those two haplotypes, which can reduce the cost to much less than $O(m)$.

HW's Algorithm: Initialization

Data: Amplicon sequence dataset $\mathcal{R}$ of size $n \times m$; Haplotype sequences $\mathcal{H}^{(0)}$ of size $K \times m$; Error probablity $\mathcal{P}$ of size $n \times m$.

Result: Reads assignment $C_i^{(0)}$; Updated haplotypes $\mathcal{H}$.

`/* initialize conditional likelihoods` $e_{kj\mathrm{N}}$ `to 0 */`

for $i = 1, \ldots, n$ **do**

  `/* find the closest haplotype for each observation */`

  for $k = 1, \ldots, K$ **do**

    $v_{ik} = \ln \phi_k(\boldsymbol{r}_i; \boldsymbol{h}_k^{(0)})$;

  $C_i^{(0)} = \arg\max_{k=1,\ldots,K} v_{ik}$;

  `/* update log probability of each cluster, position, haplotype nucleotide */`

  for $j = 1, \ldots, m$ **do**

    for $\mathrm{N} \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ **do**

      $e_{C_i j\mathrm{N}} = e_{C_i j\mathrm{N}} + \mathbb{1}\{r_{ij} = N\} \ln(1 - p_{ij}) + \mathbb{1}\{r_{ij} \neq N\} \ln\left(\frac{p_{ij}}{3}\right)$;

`/* Update the haplotypes and` v_{ik} `*/`

for $k = 1, \ldots, K$ **do**

  for $j = 1, \ldots, m$ **do**

    $h_{kj}^{(1)} = \arg\max_{\mathrm{N} \in \{\mathrm{A},\mathrm{C},\mathrm{G},\mathrm{T}\}} e_{kj\mathrm{N}}$;

  live$[k] = n + 1$;

HW's Algorithm: Optimal transfer stage: repeat until there is no live set

for $i = 1, \ldots, n$ **do**
  if $i \leq \text{live}[C_i^{(t-1)}]$ **then**
    for $k = 1, \ldots, K$ **do**
      $\text{dif}_k = 0$;
      for $j = 1, \ldots, m$ **do**
        for $\text{N} \in \{\text{A}, \text{C}, \text{G}, \text{T}\}$ **do**
          if $k \notin \{C_i^{(t-1)}\}$ *and* $k \in \mathcal{K}$ **then**
            $e_{kj\text{N}} = e_{kj\text{N}} + \mathbb{1}\{r_{ij} = N\} \ln(1 - p_{ij}) + \mathbb{1}\{r_{ij} \neq N\} \ln\left(\frac{p_{ij}}{3}\right)$;
          else
            $e_{kj\text{N}} = e_{kj\text{N}} - \mathbb{1}\{r_{ij} = N\} \ln(1 - p_{ij}) - \mathbb{1}\{r_{ij} \neq N\} \ln\left(\frac{p_{ij}}{3}\right)$;
        $h_{kj}^{(t)} = \arg\max_{\text{N} \in \{\text{A,C,G,T}\}} e_{kj\text{N}}$;
        if $k \neq C_i^{(t-1)}$ **then**
          $\text{dif}_k = \text{dif}_k + \mathbb{1}\left\{r_{ij} = h_{kj}^{(t)}\right\} \ln(1 - p_{ij}) + \mathbb{1}\left\{r_{ij} \neq h_{kj}^{(t)}\right\} \ln\left(\frac{p_{ij}}{3}\right)$;
        else
          $\text{dif}_k = \text{dif}_k - \mathbb{1}\left\{r_{ij} = h_{kj}^{(t-1)}\right\} \ln(1 - p_{ij}) - \mathbb{1}\left\{r_{ij} \neq h_{kj}^{(t-1)}\right\} \ln\left(\frac{p_{ij}}{3}\right)$;
        if $h_{kj}^{(t)} \neq h_{kj}^{(t-1)}$ **then**
          for $i1 = 1, \ldots i1 \neq i \ldots, n$ *&* $C_{i1}^{(t-1)} = k$ **do**
            $\text{dif}_{C_{i1}^{(t-1)}} = \text{dif}_{C_{i1}}^{(t-1)} + \mathbb{1}\left\{r_{ij} = h_{kj}^{(t)}\right\} \left(\ln(1 - p_{i1j}) - \ln\left(\frac{p_{i1j}}{3}\right)\right) +$
            $\mathbb{1}\left\{r_{ij} = h_{kj}^{(t-1)}\right\} \left(\ln\left(\frac{p_{i1j}}{3}\right) - \ln(1 - p_{i1j})\right)$;
      $\text{dif}_{C_i^{(t-1)}} = -\text{dif}_{C_i^{(t-1)}}, C_i = \arg\max_{k=1,\ldots,K} \text{dif}_k$;
    if $C_i \neq C_i^{(t-1)}$ **then**
      $\text{live}[k] = n + 1 + i, h_{kj}^{(t-1)} = h_{kj}; k \in \{C_i, C_i^{(t-1)}\}$;
      $e_{kj\text{N}} = e_{kj\text{N}}^{(t-1)}; k \notin \{C_i, C_i^{(t-1)}\}$;
  else
    Set $\mathcal{K} = \{k : \text{live}[k] \geq i, C_i^{(t-1)}\}$ and repeat the above steps;
$\text{live}[k] = \text{live}[k] - n$ for all k;

2. Analysis

2.1. **Simulation.** There are several factors that we wish to consider in a simulation study. Changing the number of clusters K, the data dimension n and p, the similarity among the observations and cluster sizes can affect the difficulty of clustering. Specifically, increasing K increases the complexity as we need to learn more information from the same np nucleotides. The number of observations n is somewhat superfluous. For a fixed K, increasing n will make the problem easier as we have more information. Hence, one can run all simulations with the same $E[\frac{n}{K}]$, which controls the amount of information we have to estimate $\mathcal{H}$. Increasing the fraction f of altered positions between haplotypes leads to relatively easier clustering, since it increases the separation of the clusters. Reducing p increases the similarity among observations, which has a similar effect to varying f. Increasing the imbalance of cluster sizes decreases the information we can use for the small clusters since they contain less observations, thus clustering becomes more difficult.

Therefore, to vary the difficulty of clustering, we run simulations based on changing K, the imbalance, and the degree of cluster overlap. The unequal-sized clusters are simulated as follows: generate the cluster proportions $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)$ from the exchangeable Dirichlet distribution Dir $\sim (\alpha_1, \ldots, \alpha_K)$. Given $\boldsymbol{\pi}$ and the number of observations n, cluster sizes $(|\mathcal{C}_1|, |\mathcal{C}_2|, \ldots, |\mathcal{C}_K|)$ follow a Multinomial$(n, \boldsymbol{\pi})$ distribution. We change the Dirichlet parameters to control the cluster size imbalance. The observations are simulated based on a continuous time Markov chain model. Each coordinate of a single ancestor are independently simulated with uniform category probabilities. Then, every coordinate of the K haplotypes is independently sampled from a continuous time Markov chain initialized with the ancestor, except we require the distance between the true haplotypes to be at least 1. We chose the expected number of mutations (f) along time to be small, to enhance the difficulty of clustering, so most haplotypes differ by at most two nucleotides. Reads within clusters are simulated by taking the real quality scores. We simulate datasets based on

two real amplicon datasets, one contains high quality scores (data is from Callahan et al. [2]), the other one contains relative low quality scores (data is from(McKenna et al. [22]). Quality scores $\mathcal{Q}$ are used to determine the read nucleotides according to a categorical distribution, assuming equal probability of each error and assuming no error matches the cluster haplotype.

With fixed sample size $n = 10,000$ (so the information decreases as K increases) and fixed Dirichlet parameters $\boldsymbol{\alpha} = 0.5$ (so we have unequal-sized clusters), we simulated 20 replicates for each simulation condition shown as below:

Simulation Condition			
p	K	f	Quality
251	$\{2\}$	0.000005	Low Quality
251	$\{5, 8\}$	0.00005	Low Quality
290	$\{2\}$	0.000005	High Quality
290	$\{5, 8\}$	0.00005	High Quality

TABLE 1. Simulation conditions: p is the number of coordinates; K is the true number of clusters; f indicates the degree of overlapping, smaller f corresponding to greater cluster overlap; Quality is the degree of the quality scores of the datasets.

2.2. **Analysis Methodology.** We want to compare the algorithms in terms of the accuracy and the time to reach an optimum of the objective function (1). Since our algorithms are sensitive to the starting haplotypes, we need to run multiple initializations, specifically, 10,000 initializations for each of the algorithm per simulated dataset. And use the obtained statistics to compare the performance and speed. The initialization method we used is random initialization, meaning we randomly chose K out of n observations as the starting haplotypes, and we set seed for this random initialization to make sure each algorithm has the same initial haplotypes. The following analysis compares pairs of algorithms, *i.e.* MacQueen with Lloyd, MacQueen with Hartigan and Wong, Hartigan and Wong with Lloyd.

Since the algorithms may not reach the same optimum when the clustering problem is difficult, we assess how often the algorithms reach a target log likelihood. For example, we define the target log likelihood to be either as the maximum log likelihood value reached by all of the algorithms if the maximum log was achieved frequently or to avoid the max log likelihood only appears less than twice, we define the target as the log likelihood value that was at or above the 95th percentile achieved across all initializations and algorithms.

2.2.1. *Accuracy Comparison.* To compare any two algorithms, we count the number of times the first algorithm achieves the target but the second does not, denoted as n_{10} and similarly n_{01} represents the number of times the second algorithm gets the target but the first does not. Thus, if there is no difference between two algorithms, we have $N_{10} \sim \text{Bin}(n_{10} + n_{01}, 0.5)$ given $N_{10} + N_{01} = n_{10} + n_{01}$. Further, the p-value for the one-sided test of $H_0 : p_{10} \geq 0.5$ is $\Pr(N_{10} \leq n_{10})$. If $n_{10} >> n_{01}$, it is very likely to obtain extreme p-values 1, thus we instead compute $\Pr(N_{01} \leq n_{01}) = 1 - \Pr(N_{10} \leq n_{10})$. Then we report the minumum of $\Pr(N_{10} \leq n_{10})$ and $\Pr(N_{01} \leq n_{01})$. Since we replicated the simulation, we use Holm's method [15] to adjust the p-values, and control the family-wise error rate at 0.05.

We also constructed the confidence interval for the rate ratio $r = \frac{n_{10}+1}{n_{01}+1}$:

$$e^{\{\log(r) \pm [Z_{0.975} \times SE(\log(r))]\}}$$

We can as well compare our estimated haplotypes, which are the haplotypes obtained when the algorithm reached the max value of our objective function (1) across all initializations, with the true haplotypes of the simulation datasets. Then we compare algorithms by how many true haplotypes it was able to estimate. Further, we compare our algorithms with DADA2 [2] in terms of the number of true sequences detected.

2.2.2. *Time Comparison.* The actual time algorithm takes to achieve the optimum is important because the timing of each initialization may vary among algorithms. Assuming one algorithm gets more accurate clustering results compared to the other one for each simulation. In particular, the accuracy advantage of the more accurate algorithm can be made up by a speed advantage of the less accurate algorithm since given a same amount of running time the less accurate one might reach the same optimum given by the accurate algorithm. We construct confidence interval for the ratio of two mean waiting time to get the optimum. Record the wait times for the paired algorithms to get the target optimum, denoted as T_1 and T_2. We use Fieller's theorem [6] to construct the confidence interval for the ratio of the mean wait time $r = \frac{\overline{T}_1}{\overline{T}_2}$:

$$\frac{1}{(1-g)}\left[\frac{\overline{T}_1}{\overline{T}_2} \mp \frac{t_\alpha S}{T_2}\sqrt{\frac{1}{n_{T_1}}(1-g)+\frac{r^2}{n_{T_2}}}\right]$$

where $g = \frac{t^2_{0.975}S^2}{n_{T_2}\overline{T_2}}$ and $s^2 = \frac{(n_1-1)s_1^2+(n_2-1)s_2^2}{(n_1-1)+(n_2-1)}$

3. Results

3.1. **Implementation Time Comparison for Fast and Slow Version of Lloyd's algorithm.** We tested two versions of Lloyd's algorithm in C, a traditional Lloyd's algorithm and an efficient implementation supplemented with auxiliary variables. We tested the algorithms on two different sets of datasets (amplicon dataset with quality scores and a category dataset) with varying choices of number of clusters K. The efficient version is usually $30\% - 40\%$ faster than the original algorithm. The following two Tables 2 and 3 show the specific time cost on these two algorithms. In the rest of this presentation, we use the efficient Lloyd's algorithm.

Algorithm	Random Initialization	
	$K = 3$	$K = 12$
Lloyds	0.770	0.991
Lloyds (efficient)	0.336	0.613

TABLE 2. Average running time (s) for one initialization on amplicon data

Algorithm	Random Initialization	
	$K = 3$	$K = 5$
Lloyd	1.344	2.508
Lloyd (efficient)	0.852	1.390

TABLE 3. Average running time (ms) for one initialization on category data

3.2. **Accuracy Comparison of All Three Algorithms.**

3.2.1. *True Sequence Detected by Algorithms.* We report the number of true unique sequences each algorithm found (including DADA2 [2]) along with the maxima of the objective functions (1). Table 4 and Table 5 show five representative results for simulation data with low and high quality scores respectively. Comparing the two tables, low quality scores lower the accuracy of the algorithms, especially for DADA2 [2]. DADA2 is not able to detect true haplotypes when the quality scores are low. In contrast, using the haplotypes found at the optimum by each algorithm, all of our algorithms are able to find some part of the truth, although none of them find all of the true haplotypes they still find a good proportion of the true haplotypes. Even on the simulation datasets with high quality scores, our algorithms, for example Hartigan and Wong and MacQueen, sometimes found more haplotypes than DADA2 [2].

As K increases, clustering performance declines. From Table 5, all of our algorithms were able to find both of the true haplotypes when $K = 2$, however when $K = 5$, HW identifies more true haplotypes and achieves higher maxima most of the time. On two

simulated datasets with low quality scores and $K = 8$, Lloyd's algorithm discovered one more true haplotype than either of the other two algorithms, however, the values of the objective function (1) given by Lloyd's are smaller.

Algorithm	K		
	2	5	8
Lloyd	{1, 1, 2, 1, 1}	{3, 4, 4, 4, 4}	{6, 6, 6, 7, 7}
MacQueen	{1, 1, 2, 1, 1}	{3, 4, 4, 4, 4}	{5, 6, 6, 6, 6}
HW	{1, 1, 2, 1, 1}	{3, 4, 4, 4, 4}	{5, 7, 6, 7, 6}
DADA2	{1, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}

TABLE 4. Number of captured true haplotypes on simulation data with low quality scores: in each brace {} the 5 results are from 5 replicates, each number represents the number of true haplotypes detected by the algorithms each time.

Algorithm	K		
	2	5	8
Lloyd	{2, 2, 2, 2, 2}	{4, 5, 4, 4, 5}	{6, 6, 6, 7, 6}
MacQueen	{2, 2, 2, 2, 2}	{4, 5, 4, 4, 5}	{6, 6, 6, 7, 8}
HW	{2, 2, 2, 2, 2}	{4, 5, 4, 5, 5}	{8, 6, 6, 7, 8}
DADA2	{2, 2, 2, 2, 1}	{4, 5, 4, 4, 3}	{6, 6, 6, 7, 7}

TABLE 5. Number of captured true haplotypes simulation data with high quality scores: in each brace {} the 5 results are from 5 replicates, each number represents the number of true haplotypes detected by the algorithms each time.

3.2.2. *Adjusted Rand Index.* On the simulation data we can make use of the fact that we know the true assignment of each observation. Therefore, we can compute the adjusted rand index (ARI) (Hubert and Arabie, 1985) [17], which is a measure used to compare two different partitions. Low ARI reflects agreement in the clusters. We report the the median, among the replicates, of mean adjusted Rand index computed over those initializations reaching the max log likelihood for each algorithm. The range of the adjusted range index is from -1 to 1, higher ARI indicates better agreement in clusters. On the simulation data

shown in Table 6, all of the algorithms have a good clustering resultsn with HW achieving a slightly better performance. However, on the simulation dataset with low quality scores (Table 7), when $K = 2$, the average ARI are low for all of the three algorithms although HW reached a better ARI. This might be due to the imbalance of the two clusters: most of the datasets contain one cluster with abundance 95% smaller than the other one.

Algorithm	K		
	2	5	8
Lloyd	0.862	0.948	0.951
MacQueen	0.862	0.954	0.946
HW	0.877	0.961	0.961

TABLE 6. ARI on the simulation data with high quality scores

Algorithm	K		
	2	5	8
Lloyd	0.554	0.856	0.741
MacQueen	0.555	0.835	0.739
HW	0.641	0.858	0.743

TABLE 7. ARI on simulation data with low quality scores

3.3. **Empirical Log-likelihood Density.** To compare the algorithms, we compare the log-likelihood values that pairs of algorithms achieved across all 10,000 initializations and count the number of pairs where the value given by one algorithm is smaller than another one. Comparing the proportions reveals which algorithms achieve better maxima on average. A more detailed comparison can be obtained by comparing the empirical density of the log likelihoods achieved by our algorithms across the initializations. The algorithm with the heavier right is better.

Representative densities are plotted in Figure 1 and Figure 2. MacQueen and Lloyd's algorithms are quite similar especially on the datasets with high quality scores. However, in some of the plots the log likelihoods given by Hartigan and Wong are quite different

in the right tails of those density plots. We compute the KL divergence [19] of paired densities for all of cases (since KL divergence is not symmetric, we gave the minimum of the two divergences). KL divergences for comparing MacQueen and Lloyd 's are extremely low, normally around e^{-11}. However, KL divergences between Hartigan and Wong (HW) and the other two algorithms are sometimes large. Details are shown in Table 9 and 10. Although HW does not produce different log likelihoods for all replicates, there are cases where there is a clear difference.

Additionally, in Table 8 we report the mean proportion of the HW reaches bigger log likelihood than MacQueen and the mean proportion of MacQueen reaches bigger log likelihood than Lloyd's algorithms over all the replicates. Specifically, the proportion is calculated after filtering the pairs that both of the algorithms gave the same results (due to the numerical error, differences of two log-likelihood values smaller than 1 is considered as the same). More than 90% of time Hartigan and Wong reached a better results and it is consistent across all the replicates. Thus, combining the differences shown by KL divergences, HW is sometimes superior than those two algorithms.

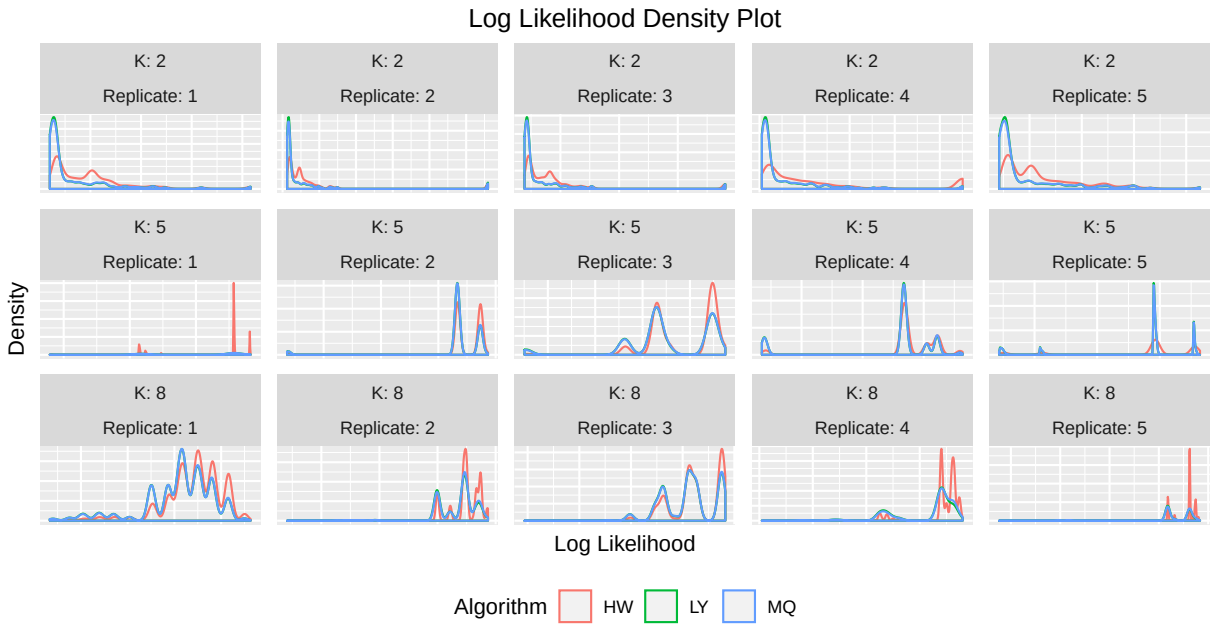


FIGURE 1. Comparison of the values of the objective function reached by the three algorithms on the simulation dataset with high quality scores. HW represents Hartigan and Wong, MQ represents MacQueen, LY represents Lloyd. There are 5 replicates in total.

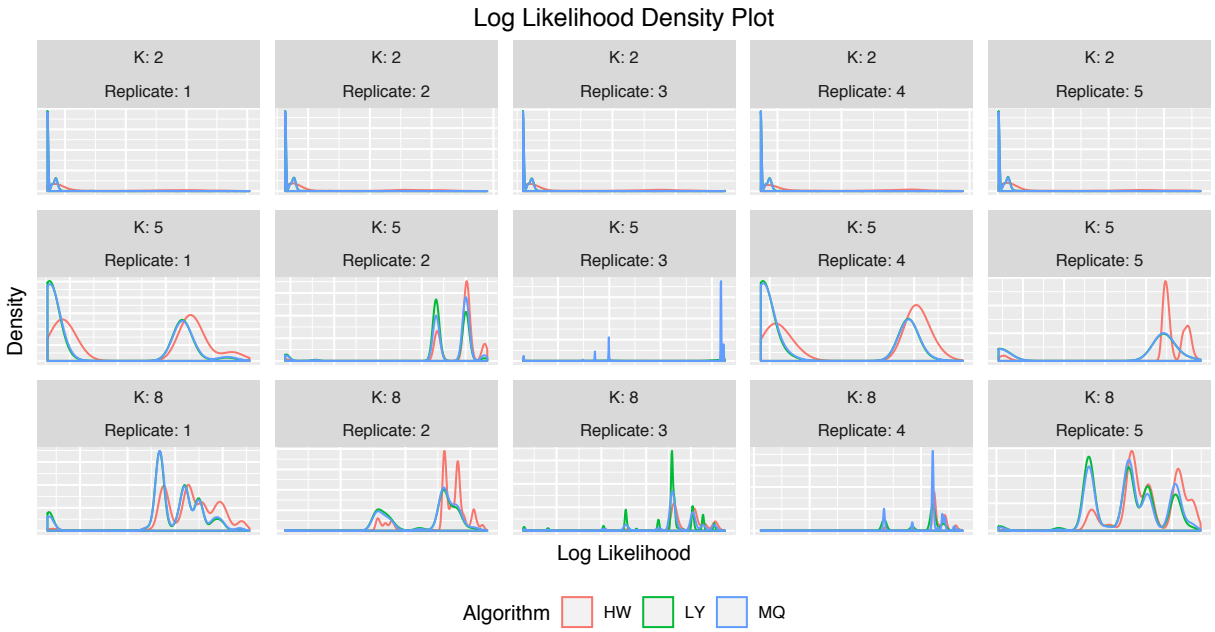


FIGURE 2. Comparison of the values of the objective function by the three algorithms on the simulation dataset with low quality scores. HW represents Hartigan and Wong, MQ represents MacQueen, LY represents Lloyd. There are 5 replicates in total.

Comparison	K		
	2	5	8
LY & MQ	0.848	0.772	0.748
MQ & HW	0.958	0.960	0.960

TABLE 8. Mean proportion of the second algorithm reaches bigger log likelihood than the first algorithm. K is the number of clusters.

Comparison	KL Divergence (K, replicate)	
	(5, 1)	(8, 4)
LY & HW	1.01	1.30
MQ & HW	1.00	1.26

TABLE 9. KL Divergence between HW and two other algorithms on the simulation data with high quality scores. K is the number of clusters, replicate is the index of replicate in Figure 1.

Comparison	KL Divergence (K, replicate)		
	{(2, 1), (2, 4)}	{(5, 2), (5, 5)}	(8, 2)
LY & HW	{2.00, 2.00}	{0.8, 0.8}	0.75
MQ & HW	{2.00, 2.00}	{0.8, 0.8}	0.75

TABLE 10. KL Divergence between HW and two other algorithms on the simulation data with low quality scores. K is the number of clusters, replicate is the index of replicate in Figure 2.

3.3.1. *P-value.* Briefly remind the method of comparing the accuracy of two algorithms. For a pair of algorithms, we count the number of times the first algorithm gets the target but the second does not, denoted as n_{10} and n_{01} vice versa. Thus, If there is no difference between the algorithms, then the count N_{10} follows $\text{Bin}\,(n_{10} + n_{01}, 0.5)$ given $N_{10} + N_{01} = n_{10} + n_{01}$. Setting the family-wise error rate at 0.05 and if the adjusted p-values are bigger than 0.05, meaning the null hypothesis of equal accuracy cannot be rejected. Small probability $\Pr\,(N_{10} \leq n_{10})$ indicates the second algorithm listed in Table 11 and Table 12 is better than the. A null hypothesis of superior performance of the first algorithm is strongly rejected in nearly all cases except for the situation at $K = 2$, Lloyd's and MacQueen have similar clustering performance. In particular, the simulation datasets with high quality scores shown in 11, there is a trend that when K increases, the HW algorithm produces better, more accurate results. Although, there is no similar trend in the low quality score simulation (Table 12), the HW algorithm is still superior to the other two algorithms.

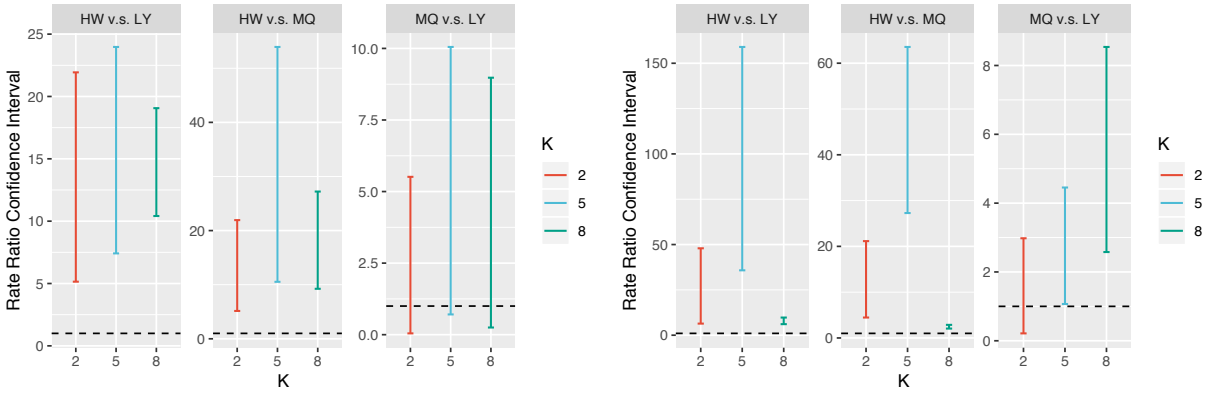
Comparison	K		
	2	5	8
LY & HW	−20.46	−80.75	−117.74
MQ & HW	−21.34	−82.89	−115.60
LY & MQ	−0.22	−2.16	−2.61

TABLE 11. Log transformed mean (20 replicates) adjusted p-values on the simulation data with high quality scores

Comparison	K		
	2	5	8
LY & HW	−11.22	−251.96	−211.94
MQ & HW	−10.49	−249.91	−65.00
LY & MQ	−0.53	−2.10	−4.98

TABLE 12. Log transformed mean (20 replicates) adjusted p-values on the simulation data with low quality scores

3.3.2. *Rate Ratio Confidence Interval for the Optimal Counts.* We estimate the ratio $r = \frac{n_{10}+1}{n_{01}+1}$ and plot the distribution across simulations in Figure 3a and Figure 3b, the dotted horizontal line in the plots indicate no difference between the paired algorithms. Once above the dotted line, a higher rate ratio $r = \frac{n_{10}+1}{n_{01}+1}$ indicates the first algorithm is superior to the second one. These two figures display the results for the replicate where the HW algorithm was least favored against the MacQueen and Lloyd's algorithms. It is obvious that the HW algorithm is is significantly better than the other two algorithms, however when K = 2 and sometimes when $K \in \{5, 8\}$ there is no significant difference between MacQueen and Lloyd's algorithms.



(A) simulation data with high quality scores (B) simulation data with low quality scores

FIGURE 3. Rate ratio 95% confidence interval for comparing the accuracy of paired algorithms. HW represents Hartigan and Wong, MQ represents MacQueen, LY represents Lloyd. The dotted horizontal lines is 1. Across all the replicates, these two figures are the least favorable situations for the algorithm listed at the first place.

3.4. **Time Comparison.** Among all three algorithms, MacQueen is the fastest and Hartigan and Wong is the most slowest if we simply looking at the average running time for each initialization shown in Table 13. However, considering the wait time to get the optima of our objection function, the situation might be different. Figure 4 and 5show the time ratio $r = \frac{\overline{T}_1}{\overline{T}_2}$ confidence interval. If the box plots cross the dotted horizontal line, then it indicates no difference between the paired algorithms, if those box plots are lower than the dotted horizontal line, then it means the algorithm listed at the first place is slower than the second since when we did the comparison the denominator of the ratio $r = \frac{\overline{T}_1}{\overline{T}_2}$ is the first algorithm. MacQueen is significantly faster than Lloyd's algorithm on the low quality score data, except when $K = 2$ and one replicate when $K = 5$. As K increases, MacQueen tends to be much faster on the low quality datasets Figure 5. For the HW and Lloyd, on the low quality score datasets displayed in Figure 5, Hartigan and Wong is almost always better, but on high quality score datasets in Figure 4 they have the roughly same performance. Macqueen is still faster most of the time when compared with Hartigan

and Wong on the datasets with high quality scores and on the datasets with low quality scores with $K = 8$, although the difference between them is not that big compared with MacQueen and Lloyd. Also, when $K \in \{2, 5\}$ they have the roughly same speed on the datasets with low quality scores .

Algorithm	Average Running Time (s)			Variance of Running Time		
	$K = 2$	$K = 5$	$K = 8$	$K = 2$	$K = 5$	$K = 8$
Lloyd	0.11	0.14	0.26	0.0001	0.0004	0.0003
MacQueen	0.03	0.05	0.06	0.0008	0.0008	0.003
HW	0.15	0.64	0.68	0.003	0.06	0.03

TABLE 13. Average running time (s) and the variance of running time for one initialization of the algorithms.

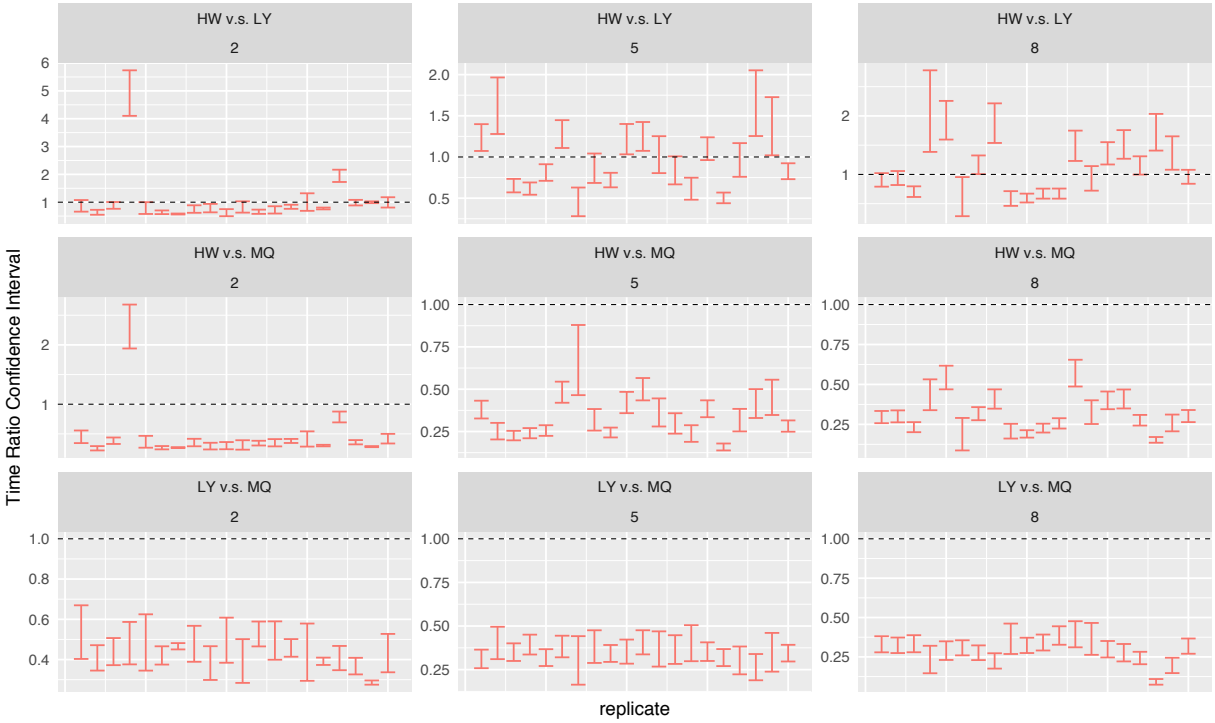


FIGURE 4. Time ratio 95% confidence interval for comparing the speed of paired algorithms for all the replicates on datasets with high quality scores. HW represents Hartigan and Wong, MQ represents MacQueen, LY represents Lloyd. 2, 5, 8 represent the number of real clusters. There are twenty replicates in total.

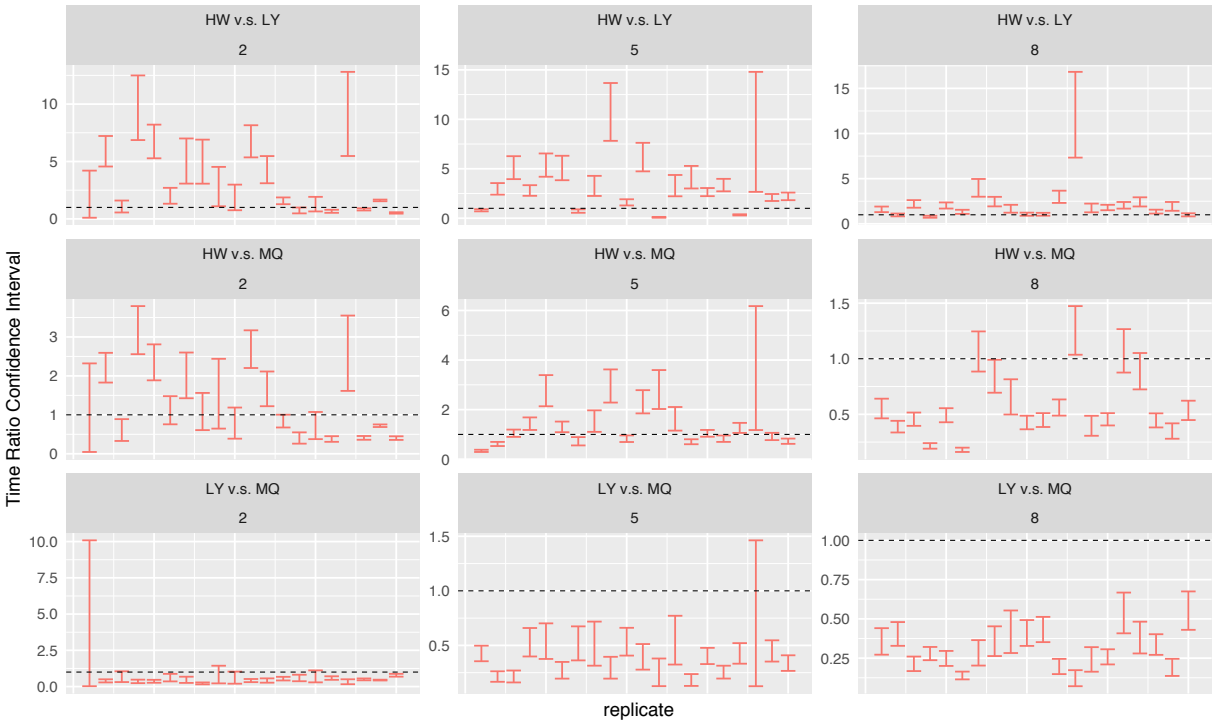


FIGURE 5. Time ratio 95% confidence interval for comparing the speed of paired algorithms for all the replicates on datasets with low quality scores. HW represents Hartigan and Wong, MQ represents MacQueen, LY represents Lloyd. 2, 5, 8 represent the number of real clusters. There are twenty replicates in total.

4. Discussion

Although Hartigan and Wong is sometimes slower than MacQueen when comparing the waited time to achieve the target, it is the most accurate per initialization among the three algorithms and it is more likely to achieve a higher objective function (1) value given an initialization. To speed up the Hartigan and Wong's algorithm, it is possible to only check if transferring reads to their second closet clusters would improve the objective function or not and save the time of computing the cost of moving to others clusters. More efficient coding and polish the algorithm may further improve the speed of the Hartigan and Wong's algorithm.

We also showed that our algorithms give better results compared to DADA2 [2], the differences of the number of true haplotypes found are especially big on the datasets with low quality scores. Although our results are based on 10,000 initializations which took a longer time than DADA2, we can adopt other initialization methods rather than random initialization to give our algorithms a set of better starting haplotypes and further enhance the speed of our algorithms.

What's more, although our algorithms are intended for clustering the amplicon sequence datasets with quality scores, they can be also used to cluster the categorical dataset without that quality information as long as we treat $p_{ij} = 0$, then we are maximizing the similarity among the clusters which is identical to the k-modes algorithm. We can compare the speed of k-modes algorithm (Huang [16]) with our algorithms, if ours are proven to be faster, we can rearrange our code to fit both data structures. Additionally, our algorithms can be used to cluster the data that contains variables with confidence or with the probabilities of the certainties of the data. These probabilities can be considered as p_{ij} in our context.

References

[1] A. Amir et al. "Deblur Rapidly Resolves Single-Nucleotide Community Sequence Patterns". In: *mSystems* 2.2 (2017). Ed. by Jack A. Gilbert.

[2] B. J. Callahan et al. "DADA2: High-resolution sample inference from Illumina amplicon data". In: *Nature Methods* 13 (2016), pp. 581–583.

[3] J. G. Caporaso et al. "A perspective on 16S rRNA operational taxonomic unit clustering using sequence similarity". In: *Nature Methods* 7 (2010), p. 335.

[4] M. Comin, A. Leoni, and M. Schimd. "QCluster: Extending Alignment-Free Measures with Quality Values for Reads Clustering". In: *Algorithms in Bioinformatics*. Springer Berlin Heidelberg, 2014, pp. 1–13.

[5] R. C. Edgar. "Search and clustering orders of magnitude faster than BLAST". In: *Bioinformatics* 26 (2010), pp. 2460–2461.

[6] E. C. Fieller. "Some Problems in Interval Estimation". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 16 (1954), pp. 175–185.

[7] V. Ganti, J. Gehrke, and R. Ramakrishnan. "CACTUS&Mdash;Clustering Categorical Data Using Summaries". In: (1999), pp. 73–83.

[8] D. Gibson, J. Kleinberg, and P. Raghavan. "Clustering categorical data: an approach based on dynamical systems". In: *The VLDB Journal* 8.3 (2000), pp. 222–236.

[9] K. C. Gowda and E. Diday. "Symbolic Clustering Using a New Dissimilarity Measure". In: *Pattern Recogn.* 24.6 (1991), pp. 567–578.

[10] S. Guha, R. Rastogi, and K. Shim. "CURE: An Efficient Clustering Algorithm for Large Databases". In: *SIGMOD Rec.* 27 (1998), pp. 73–84.

[11] S. Guha, R. Rastogi, and K. Shim. "ROCK: a robust clustering algorithm for categorical attributes". In: *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*. Vol. 25. 1999, pp. 512–521.

[12] J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, 1975.

[13] J. A. Hartigan and M. A. Wong. "Algorithm AS136. a K-means clustering algorithm." In: *Applied Statistics* 28 (1979), pp. 100–108.

[14] Z. He, X. Xu, and S. Deng. "Squeezer: An efficient algorithm for clustering categorical data". In: *Journal of Computer Science and Technology* 17.5 (2002), pp. 611–624.

[15] S. Holm. "A Simple Sequentially Rejective Multiple Test Procedure". In: *Scandinavian Journal of Statistics* 6 (1979), pp. 65–70.

[16] Z. Huang. "A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining". In: *Proceedings of the SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery* 28 (1997), pp. 1–8.

[17] L. Hubert and P. Arabie. "Comparing partitions". In: *Journal of Classification* (1985), pp. 193–218.

[18] R. Knight et al. "Best practices for analysing microbiomes". In: *Nature Reviews Microbiology* 16 (2018), pp. 410–422.

[19] S. Kullback and R. A. Leibler. In: *Ann. Math. Statist.* 22.1 (Mar. 1951), pp. 79–86.

[20] S. Lloyd. "Least squares quantization in PCM". In: *Information Theory, IEEE Transactions on* 28.2 (1982), pp. 129–137.

[21] J. MacQueen. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*. Ed. by L. M. Le Cam and J. Neyman. Vol. 1. Berkeley, CA: University of California Press, 1967, pp. 281–297.

[22] A. McKenna et al. "Whole-organism lineage tracing by combinatorial and cumulative genome editing". In: *Science* 353.6298 (2016).

[23] N. Nguyen et al. "A perspective on 16S rRNA operational taxonomic unit clustering using sequence similarity". In: *Npj Biofilms And Microbiomes* 2 (2016), 16004 EP.

[24] G. Reinert et al. "Alignment-free sequence comparison (I): statistics and power". In: *Journal of computational biology : a journal of computational molecular cell biology* 16 (2009), pp. 1615–1634.

[25] L. Wan et al. "Alignment-free sequence comparison (II): theoretical power of comparison". In: *Journal of computational biology : a journal of computational molecular cell biology* 17 (2010), pp. 1467–1490.

[26] R. Xu and D. C. Wunsch II. "Partitional Clustering". In: *Clustering*. Hoboken, New Jersey: Wiley-Blackwell, 2008. Chap. 4, pp. 63–110.