Supplementary Materials

1.1 Amplicon denoising running time with real data

We asked if existing amplicon denoising packages (Callahan, et al., 2016; Quince, et al., 2011; Rosen, et al., 2012) could be applied to sequencing data of complex barcode pools using the Data Set 2 (below). The fastest reported method, which is also the only package designed for Illumina sequencing data, is DADA2 (Callahan, et al., 2016). Using standard settings, we found that DADA2 could not complete the clustering process in 2 weeks using a desktop equipped with 3.5 GHz 6-core intel xeon E5 and 64 GB memory. In contrast, Bartender takes less than 10 minutes.

1.2 Algorithm overview, seed selection, and binning

A schematic overview of Bartender is shown in Figure S1. Bartender processes barcodes with different lengths independently. First, total reads are tabulated to form a list of unique reads and corresponding frequencies. Using this list of unique reads, a sequence of overlapping 5-8 base pair seeds is selected. These groups of unique reads are considered as initial clusters, and are partitioned into bins using the first seed. Clusters within each bin are merged using a statistical test described in the main text. These new clusters then serve as the input for the next round of binning, and this is repeated until all seeds have been processed. Lastly, three output data files are generated: the consensus cluster sequence (the barcode) and its counts, the quality of each cluster, and a map between each unique read and the cluster it belongs to. Unique Molecular Identifiers (UMIs) for each read that are stored with the input of a list of unique reads, which is associated with its frequency summarized from the raw file. It outputs a list of clusters to three files: 1) the most important information, such as cluster center, quality and size, 2) the position weight matrix of each cluster (for checking cluster quality), and 3) The mapping between a unique reads and a cluster. The identifier to link these three files is the unique cluster id.

Algorithm 1 describes the main procedure of the clustering. It takes an input of a list of unique reads, which is associated with its frequency summarized from the raw file. It outputs a list of clusters to three files: 1) the most important information, such as cluster center, quality and size, 2) the position weight matrix of each cluster (for checking cluster quality), and 3) The mapping between a unique reads and a cluster. The identifier to link these three files is the unique cluster id.

Data: F : Barcode frequency table
Result: C : A list of clusters

Initialize C with F by setting each unique barcode as a cluster itself; Identify a list of seeds S using the position weight matrix of F; for s in S do
size.threshold ← trimmed mean of the cluster size in C ; l ← []; shatter C into buckets B using s ; for b in B do
lb ← ClusterSingleBucket(b, size.threshold) ; append lb to l ;
end
C ← l ;
end

Algorithm 1. Main procedure of clustering algorithm

Figure S2 illustrates the seed selection and binning procedure.

There are two major computational advantages of binning by seeds. First, unnecessary pairwise comparisons between distant clusters are dramatically reduced because only clusters within the same bin will be compared. Second, each bin can be processed independently, making it easy to parallelize the algorithm on multiple-core computers. Intuitively, a longer seed generates more bins and correspondingly decreases the number of clusters in each bin. The seed length is a critical parameter that balances the speed and accuracy of the clustering algorithm. That is, longer seeds increase speed by reducing pairwise comparisons, but are more likely to leave spurious barcodes ungrouped, thereby increasing false positives. By default, seed length is set to be five, and Bartender iterates through the seed position list using a sliding window with a size of the seed length and slide one nucleotide at a time.
1.3 Detailed description of greedy clustering

We first split clusters into high- and low-frequency bins based on the mean cluster size, which is derived from the empirical distribution of the sizes of all clusters. Since error reads are expected to greatly outnumber reads with true barcode sequences, the mean cluster size is expected to partition the majority of error-containing sequences (and maybe a minority of true barcodes) into the low-frequency group. Each cluster in the low-frequency group is compared to clusters in the high-frequency group using the statistical test described in the main text, and three scenarios may occur: if no similar high-frequency cluster is found, this low-frequency cluster is added to the high-frequency cluster group, as it is likely to be a true barcode cluster; if only one similar high-frequency cluster is found, the low-frequency cluster is merged to this cluster; if >1 similar high-frequency clusters are found, the low-frequency cluster will be merged with the high-frequency cluster with the closest sequence distance. If multiple high-frequency clusters are equally close to this low-frequency cluster, then it will be merged into the cluster with highest frequency. Following this step, pairwise comparisons between all high-frequency clusters are performed using the same statistical test described in the main text and clusters are merged if they pass.

Algorithm 2 illustrates the clustering process in each bin. This module takes a subset of clusters that belong in each bin and outputs a list of updated clusters. This module is called on each bin in each clustering round.

1.4 Bartender extractor

Bartender extractor is a simple command-line tool designed to extract barcodes from raw reads. It assumes that each read has at most one barcode, flanked by some fixed sequences which are used to locate the barcode. A barcode template could be one large random region or consist of multiple small random regions separated by spacers. Because indels may occur during barcode generation, users can specify a length range for each random region. While extracting, this tool keeps track of the total number of base errors identified in the flanking (constant) regions and estimates the sequence error using the percentage of these errors. The extractor allows at most one mismatch in each flanking region, so, in order to obtain relatively accurate error estimation, the length of a flanking sequence should not be set to be too long (we recommend 5 bases). If sequence error is not uniform across all positions in a barcode amplicon and error biases in the flanking regions exist, this sequence error estimate might be slightly incorrect. However, we recommend using this error rate as the baseline for Bartender clustering. The Bartender clustering command provides a second estimate of the error rate based on the random regions. If the error rate estimated by the clustering component is significantly different from that estimated by the extractor, we recommend rerunning Bartender clustering with more conservative parameter values (i.e. with smaller d and/or z).

The extractor transfers the specified barcode template into regular expression and extracts the matched subsequence from each read in the raw file (FASTQ or FASTA format). A user defined quality threshold...
Bartender: fast and accurate barcode clustering

can also be used to filter out low-quality barcodes (only applicable to FASTQ format). When the average quality is below this threshold, these reads will be ignored. The matched sequence (flanking regions removed) and the corresponding line number in raw (FASTQ or FATSA) file are outputted to a file. This file can be used directly for Bartender clustering. Alternatively, this file can be used in combination with user-generated UMI file, to replace the line number with the UMI prior to clustering. If UMIs are included, Bartender will by default remove duplicates during clustering.

1.5 Simulated and real datasets

All simulated datasets consist of 100,000 true barcodes, whose cluster sizes follow an exponential distribution with mean of 100. Each simulated dataset contains ~10M total reads. Table S1 shows the four simulated data sets used in this study.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Data Set 1</th>
<th>Data Set 2</th>
<th>Data Set 3</th>
<th>Data Set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of barcodes</td>
<td>100,000</td>
<td>100,000</td>
<td>100,000</td>
<td>100,000</td>
</tr>
<tr>
<td>Barcode length</td>
<td>26</td>
<td>38</td>
<td>38</td>
<td>64</td>
</tr>
<tr>
<td>Random nucleotides</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>Barcode size distribution</td>
<td>Exponential distribution with λ=0.01 (mean = 100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of reads</td>
<td>~10,000,000</td>
<td>~10,000,000</td>
<td>~10,000,000</td>
<td>~10,000,000</td>
</tr>
<tr>
<td>Sequence error</td>
<td>2%</td>
<td>2%</td>
<td>0.33%</td>
<td>2%</td>
</tr>
<tr>
<td>Relevant sections</td>
<td>Bartender is flexible and fast</td>
<td>Bartender is flexible and fast</td>
<td>Bartender errors and sequencing depth</td>
<td>Bartender is flexible and fast</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table S1. Simulated data set parameters

The 26mer barcode contains 20 random nucleotides evenly separated by three 2-bp spacers, and the 38mer barcodes are identical with the addition of two 6-bp fixed flanking at the two ends. These designs are used to mimic the barcode structure from our real data (Levy, et al., 2015). The 64mer barcode is generated by concatenating two 38mer barcodes but removing the 12-bp fixed region in the middle that consists of the last 6-bp flanking region of first barcode and the first 6-bp flanking region of the second barcode. Two different error rates (2% or 0.33%) have been used in our simulations. Errors are introduced to true barcode reads by replacing a correct nucleotide with any of the other three nucleotides with the specified error probabilities. For the four datasets, 59.5%, 46.5%, 88.2% and 27.4% of the reads can be exactly matched with true barcodes in dataset 1, 2, 3 and 4, respectively. The features the Data Set 2 barcodes are shown in Fig. S3.

We used our published data as the real data in this study (Levy, et al., 2015). It is a deeply sequenced high-complexity barcode library that contains ~0.5M barcodes (20 random bases, 26 total bases) with ~136M total reads and ~3M unique reads.

1.6 Clustering parameters and comparison to Starcode, SEED and cd-hit

For the 26mer and 38mer barcode libraries (datasets 1, 2 and 3), clustering was performed by Bartender using the parameters d=2, l=5, s=1, and z=5 (default), by Starcode using a maximum Levenshtein distance of 2 and 4 threads (starcode –d 2 –t 4), by SEED allowing 2 mismatches (SEED --mismatch 2; --short for the 26mer), and by cd-hit with a seed size of 9 and clustering sequences with 94% similarity (cd-hit-est –c 0.94 –n -9 –M 0 –r 0). For the 64mer barcode library (dataset 4), clustering was performed by Bartender using the parameters d = 3, l = 5, s = 1 and z = 5, by Starcode using a maximum Levenshtein distance of 3 and 4 threads (starcode –d 3 –t 4), by SEED allowing 3 mismatches (SEED --mismatch 3), and by cd-hit with a seed size of 9 and clustering sequences with 94% similarity (cd-hit-est –c 0.94 –n -9 –M 0 –r 0).

Real data (Levy, et al., 2015) was clustered by Bartender using the parameters d = 2, l = 5, s = 1 and z = 5, and by Starcode using starcode –d 2 –t 4.

1.7 Distribution of Bartender errors on simulated data

Fig. S4. Summary of 38mer simulated barcode data. (a) Histogram of the true barcode counts. The x-axis is log base 10 of the count for each barcode. (b) Position weight matrix of the raw sequence data. (c) Scatter plot between true size and the number of exact matches after sequencing. The number of lost barcode reads is proportional to the barcode size.

Fig. S5. Histogram of the counts of false negatives (a) and false positives(b).
An analysis of Bartender clustering errors on simulated data set 3 (Table S1) finds that most false positives and false negatives have low counts.

1.8 Bartender running time on real data

Running time for simulated data is shown in Fig. 1 in the main text. Fig. S4 shows the detailed decomposition of Bartender running time on the real data. The time for I/O process includes the actual wall-clock time for loading the raw reads and dumping results to the disk. Since this dataset has about 136 million reads, the I/O process takes a majority of the total running time. The clustering results using seed lengths from 4 to 8 are quite close. The time for clustering decreases when the seed length increases. The efficiency gains by increasing seed length diminish once the seed length reaches 5, which indicates a seed length of 5 might be a good choice on real datasets (see more discussion is in the main text). In addition, the clustering process takes less time using 12 threads compared to 4 threads, showing Bartender can scale.

Fig. S6. Running time on real data. Two parameters are examined here: the number of threads (t) and the seed length (l).

1.9 Coefficient of variation plot for real data

An analysis errors introduced by sampling, sequencing, and clustering (see Fig. 3 of main text) was also applied to the real data set. The clustering results of the large whole dataset (~130M reads) is considered as the “ground truth” for comparison with clustering results from its subsets. Fig. S2 shows the results. The three lines almost overlap with each other, suggesting that Bartender does not introduce extra error on top of sampling error. This is consistent with clustering of simulated data with a low error rate of 0.33%, which is close to the error rate in this dataset.

1.10 Multiple time point mode

For time-course bar-seq data, Bartender first clusters data at each time point separately, then orders these clustering results by time point for further processing.

For a time point t, Bartender first joins clusters with the exact same center (sequence) to t-1. Then, any unjoined cluster in t is used to query every cluster in t-1 to check if their centers that are within one mismatch. If no match is found and the size of cluster in time t is reasonably large (larger than a user-defined threshold), it will be recorded as a true barcode for time point t. If only one match is found in t-1, these barcode clusters are joined through time. If more than one match is found in t-1, the cluster in time t is joined with the largest cluster found in time t-1. This process is performed iteratively for all time points. Since the size of each barcode cluster has been recorded for all time points, with “zeros” for clusters absent at any time point, the final result contains a list of clusters and their counts at each time point (lineages). The overall cluster quality and position weight matrix across all time points are also included in the final report.

Algorithm 3 describes the core logic of merging strategy in the multiple time mode. It combines the cluster result backward through time.

Fig. S8. A flowchart of cluster merging across multiple time points. The merging strategy starts from a list of clustering results of each time point. Clusters from the latest available time point are merged to identical clusters from the previous time point. Clusters without an identical match merged with the most appropriate (see the main text) cluster within one mismatch. Other unmatched clusters will be kept only if its size is larger than the user specified size threshold.
**Bartender: fast and accurate barcode clustering**

**Algorithm 3. Merging procedure in the multiple time point mode**

Algorithm 4 explains how the previous time point helps remove false positive barcodes and corrects miscalled centers in current time point.

**Algorithm 4. Detailed procedure for merging clustering results between two adjacent time points**

```
1.11 Simulated data set for multiple time points experiment
```

Since the goal of this simulation is to evaluate the merging strategy used in the “multiple time point” mode, no spacer or flanking sequences were included in the barcodes (these constant regions have little impact on the effectiveness of merging strategy). The simulation mimics a batch evolution experiment using a two-stage schema.

---

**Table S2. Multiple time point simulation parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seed length</td>
<td>5</td>
</tr>
<tr>
<td>Overlap between adjacent seeds</td>
<td>4</td>
</tr>
<tr>
<td>Hamming distance</td>
<td>3</td>
</tr>
<tr>
<td>Z-value</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table S3. Bartender parameters in multiple time points simulation**

The first stage simulates growth in a constant population size (parameters described in Table S1 and see below). Each barcode grows at a certain rate each generation and survival is simulated by Poisson sampling. The second step simulates sequencing by a second round of Poisson sampling.

**Cell fitness distribution:** Most of these cells were set to have an identical fitness which we define as 0. However, 5000 (5%) cells were set to have a higher fitness that is drawn from a truncated exponential distribution ($\lambda = 0.00083$ and bounds of 0.05, 0.15). We assume the fitness of each cell lineage is unchanging over the course of the competition (no adaptive mutations). Over time, higher fitness lineages will expand and drive lower fitness lineages to low frequencies and eventually to extinction.

**Growth and bottlenecks:** Growth is performed in discrete generations where the population of cells in the $t + 1$ generation $N_{t+1}$ follows Poisson distribution with $\lambda = \sum_{i=0}^{\infty} n_i (1 + g_i)\Lambda$, assuming that the whole population has already saturated the whole medium from the initial generation.
**Sequencing**: The number of sequenced cells containing a specific barcode follows a Poisson distribution with $\lambda = N_1$. Errors are introduced at this stage.

Bartender parameters used for this dataset are shown in Table S3. The merging strategy is used to combine the clustering results across different time points and is run with $z$-value 5 and frequency cutoff 1.

### 1.12 References


