A comparison of QCluster and isONclust for clustering synthetic sequences

Alberto Schiabel ✉, **1236598**

Algorithms for Bioinformatics • February 10, 2021
Supervisor: **Prof. Matteo Comin**
Department of Information Engineering, University of Padova

Abstract

In this report, we examine and compare two open-source software tools for the task of clustering short-read sequences according to their gene family of origin: QCluster [CLS15] and isONclust [SM19]. We perform our analysis on six synthetic datasets simulated using 100 sequences selected from the human cDNA assembly `GRCh38.p13`. These datasets have 10000, 20000, and 50000 synthetic sequences, with two different fixed read depths (100bp and 700bp) and uniform simulated read errors. The purpose of our experiment is to evaluate how well QCluster and isONclust are able to cluster the simulated sequences with respect to the original 100 human cDNA sequences. We repeat our experiments using several configurations offered by the two software tools. We assess the quality of the clustering results using multiple external validation metrics, like *V-Measure* and *Adjusted Mutual Information*, and we empirically determine the interactions between the configuration parameters of the software using plot visualizations. Moreover, we compare the clustering results with the outcome of a random baseline clustering method. Finally, we provide some suggestions about how to improve QCluster and isONclust.

This document has been written in partial fulfillment of the requirements for the *Algorithms for Bioinformatics* class, supervised by Professor Matteo Comin. The source code for our the experiments documented in this report is available at https://github.com/jkomyno/bioalgo-QCluster-vs-isONclust.

Keywords Clustering, QCluster, isONclust

1 Introduction

Clustering genomic sequences is an interesting challenge because this task offers a variety of applications, including reducing data redundancy in databases [LJG01], identifying patterns of gene expressions, error correction [QiHM09], and explaining phylogenetic relationships. Traditionally, sequence clustering methods have relied on alignment algorithms [NW70, SW81, Got82] to gather supplementary information to guide the creation of clusters [AGM+90]. Alignment methods, however, do not scale well with the size and number of sequences, as they can easily exceed computational resources. Moreover, the combinatorics of genomic rearrangements and duplications can make it impossible to align entire genomes. In the recent years, the large amount of sequencing data produced by Next-Generation Sequencing[1] first and Long-Read Sequencing[2] created even more challenges, as the huge volume of data increased the computational requirements for genome comparisons in bioinformatics [MSP+08]. The research has thus shifted to alignment-free approaches to overcome the limitations of previous methods [VA03]. A popular alignment-free approach is based on counting the occurrence of fixed length segments, known as k-mers.
In this report, we focus our attention on the comparison of QCluster and isONclust, two clustering methods based on k-mer statistics for clustering read sequences, also leveraging the read quality values produced by sequencing instruments.

[1] **Next-Generation Sequencing** (NGS) is a sequencing method in which an entire genome is sequenced from fragmented DNA, producing short sequencing reads (typically shorter than 300 base pairs) at high speed and low cost.

[2] **Long-Read Sequencing** (LRS) is a more recent DNA sequencing technique that can read DNA sequences of much longer DNA fragments at a time, generally in a range between 10.000 and 100.000 base pairs, but it suffers from much higher error rates, approximately set at 15% of the reads.

QCluster is a software developed by Matteo Comin *et al.* [CLS15] at the University of Padova to collapse redundant reads in a single cluster to improve the run time and memory requirements of NGS and long-read sequencing while also enhancing the quality of postprocessing steps, like assembly and error correction. **isONclust** is a software developed by Kristoffer Sahlin *et al.* [SM19] at Pennsylvania State University whose purpose is to overcome the lack of scalable and accurate algorithms for clustering long reads according to their gene family of origin. isONclust uses a novel greedy clustering technique to scale well even for larger datasets while aiming to keep the quality of the clustering results sufficiently high. Both software tools use alignment-free methods in principle, even though isONclust falls back to using the Smith-Waterman [SW81] local alignment algorithm in particular cases.

QCluster and isONclust offer several configuration parameters. We only investigate a subset of them, for two main reasons:

1. The two software tools are so different that they share only a configuration parameter, the k-mer length.
2. By tweaking only a handful of software options, we avoid a combinatorial explosion of parameters.

We perform our experiments on six datasets simulated from the human cDNA assembly `GRCh38.p13`. We select 100 sequences and we use them as a seed to generate 10000, 20000, and 50000 simulated sequences with uniform error distributions, using 100 and 700 as fixed simulated read lengths. We then group the simulated reads together into clusters according to their origin, for every simulated dataset and for every combination of the selected parameters of QCluster and isONclust. Moreover, we collect some metrics to assess the quality of the clustering results, using a randomized clustering as a baseline.

We developed some modules written in `Python3` to perform our experiments:

- `python/isonclust`: the module responsible for executing isONclust on all datasets with multiple configurations, saving the results;
- `python/parameter_grid`: a data structure to lazily iterate over parameter configurations;
- `python/preprocess`: the module used to select and save 100 sequences from the human cDNA assembly;
- `python/cluster`: the module responsible for executing QCluster on all datasets with multiple configurations, saving the results;
- `python/quality`: the module used to assess quality metrics and statistics on the clustering results, saving the insights;
- `python/random_cluster`: the module responsible for performing a random clustering to be used as a baseline for our experiments;
- `python/simulate`: the module used for creating the six simulated datasets with fixed read lengths and uniform simulated read error distributions.

Outline

This document is structured as follows. Section 2 presents QCluster and isONclust in more detail. Section 3 recalls the definition of cluster analysis and introduces the metrics used to compare clustering results. Section 4 describes the originating cDNA dataset used for the experiments, its preprocessing phase, and the simulation phase that generates six synthetic datasets. Section 5 describes the methods used for the clustering experiments and quality assessments. Section 6 discusses the obtained results and the fundamental patterns that emerged from the experiments. Section 7 draws the conclusions, introduces some possible future work, and gives some opinionated suggestions to the contributors to the software used for this work. Appendixes A, B, and C contain the tables with the number of clusters, statistics, and metrics for the clustering results of QCluster and isONclust. Appendix D provides a brief description of the main file formats we encountered during our experiments. Appendix E lists the code instructions to reproduce our experiments using scripts that interact without `Python` code.

2 Clustering software tools

We now recall the definition of **cluster analysis**.

▶ **Definition 1** (Cluster analysis)**.** Cluster analysis *is the task of partitioning a set of elements in such a way that elements in the same partition (called* cluster*) are more* similar *to each other than to those in other partitions. An entire collection of clusters is commonly referred to as* clustering.

Cluster analysis is not tied to one specific algorithm *per se*: several algorithms exist, and their efficiency and measure of similarity can vary significantly.

Both QCluster and isONclust perform *strict partitioning clustering*, i.e., each element is assigned to one and only one partition. Moreover, both tools rely on different combinations of *a priori* probabilities and try to leverage read quality values to better adjust their clustering results.
QCluster uses the K-Means algorithm under the hood. The centroid-based K-Means optimization problem is known to be NP-hard, so the well-known Lloyd's approximation algorithm is used instead. isONclust, on the other hand, uses a greedy algorithm based on the concept of *minimizers*, and it falls back to exact local sequence alignments to deal with exceptional cases.

2.1 QCluster

QCluster is a software developed at the University of Padova for clustering sequence reads using alignment-free measures and leveraging sequencing quality values. It is developed on top of `afcluster`[SL13], which uses the well-known K-Means[Llo82] algorithm for computing the clustering according to multiple possible distance measures. Before [CLS15], each k-mer's contribution in every alignment-free statistics was considered irrespective of its quality value. QCluster, however, models sequencing as the process of sequencing k-mers from a reference, assigning a probability to them that can be used as a weight in alignment-free statistics. It assumes that, given the quality values, the error probability of a base is independent from its position within a read and from the quality values of the other bases.

The most important new similarity measure introduced by [CLS15] (and implemented in QCluster) is D_2^{*q}, which is a weighted and corrected for bias extension of the D_2 distrance that measures the correlation between the number of occurrences of all k-mers appearing in two sequences X and Y from and alphabet σ (in our case, $\Sigma = \{A,\ C,\ G,\ T\}$, as we are evaluating cDNA sequences). For the sake of clarity, let us introduce some definitions.

▶ **Definition 2** (Base calling)**.** Base calling *is the process of inferring the order of nucleotides in a sequence. Given a read sequence r, we refer to $Q_r(i)$ as the probability of base call error at position $1 \leq i \leq |r|$ of the read r. These probabilities are usually in a* phred*-scaled form, i.e., $Q_r(i) = -10log_{10}Prob\{$the base i of read r is incorrect$\}$. As an example, $Q_r(i) = 30$ means that there is 0.1% possibility that base i or read r is wrong.*

▶ **Definition 3** (Read-correctness Probability)**.** *Assuming that quality scores derived from* base calling *occur independently from each other, the probability of an entire read r being correct is defined as:*

$$P_r(\text{the read } r \text{ is correct}) = \prod_{j=1}^{n}(1 - 10^{-\frac{Q_r(j)}{10}}) \tag{1}$$

where n is the length of the sequence r.

▶ **Definition 4** (Word-correctness Probability)**.** *Assuming that quality scores derived from* base calling *occur independently from each other, the probability of a word w of length k (also called k-mer) occurring at position i or r begin correct is defined as:*

$$P_{w,i}(\textit{the word } w \textit{ at position } i \textit{ of read } r \textit{ is correct}) = \prod_{j=1}^{k}(1 - 10^{-\frac{Q_r(i+j)}{10}}) \tag{2}$$

▶ **Definition 5** (X_w^q). *Given a read x, we define X_w^q as the sum of probabilities of every occurrence of w within x:*

$$X_w^q = \sum_{i \in \{i \mid w \textit{ occurs in } x \textit{ at position } i\}} P_{w,i} \tag{3}$$

▶ Note. Using X_w^q, every occurrence is not counted as 1, but it is rather weighted with a probability depending on the reliability of the read x.

▶ **Definition 6** (Centralized k-mer count). *Let x be a read of length n. Let p_w be the i.i.d. probability that a word w in x is correct. Let $\mathbb{E}(P_w)$ be the expected probability that the k-mer w is correct given the quality scores. The centralized k-mer counts are defined as:*

$$\widetilde{X}_w^q = X_w^q - (n-k+1)p_w\mathbb{E}(P_w) \tag{4}$$

Since $\mathbb{E}(P_w)$ is impractical to compute exactly, it is approximated using either Average Word Probability (**AWP**) *or* Average Quality Probability (**AQP**) *prior probability estimators [CLS15].*

▶ **Definition 7** (D_2^{*q} alignment-free measure). *Given two reads x and y, and the word length k, the D_2^{*q} alignment-free measure is defined as:*

$$D_2^{*q} = \sum_{w \in \Sigma^k} \frac{\widetilde{X}_w^q \widetilde{Y}_w^q}{(n-k+1)p_w\mathbb{E}(P_w)} \tag{5}$$

Moreover, QCluster uses a technique called **quality value redistribution**, which is easier to understand via an example. Let us suppose that base A has a 70% quality score: if we ignore *indel* errors, it means that there is 30% of chance that the base is correct. Under the i.i.d. model, it implies that bases C, G, T have 10% chance of being the actual base (rather than A); we thus redistributed the "missing quality" among the other bases. QCluster allows for using a similar redistribution operation for k-mers quality scores, without considering the edge case where two or more bases are wrong at the same time (due to the computational cost and the relative insignificant possible improvement).

Since QCluster uses the centroid-based K-Means algorithm, it tries to partition the reads such that the reads on the same cluster have minimum distance between them and maximum distance with elements of different clusters. The number of clusters c must be explicitly given by the user. The software offers multiple possible distance measures, but we restrict ourselves to D_2^{*q}, and the well-known L_2 (Euclidean norm) and χ^2 metrics. For distances that do not provide convergence guarantees, namely D_2^{*q}, QCluster uses an early-stopping criterion where the algorithm is interrupted if there is no improvement for a certain number of iterations. Since the K-Means algorithm's initialization is not deterministic, QCluster only reports the best clustering solution over multiple runs.

QCluster supports only `FASTQ` files as input, and by default it does not produce any files as output, as it prints the clustering results on the standard output. We decided to capture the content written to the standard output and to redirect it to a `TSV` file named `inferred_clusters.tsv`, where the first column contains the read ID, and the second column contains the corresponding cluster ID assigned by QCluster.

QCluster is written in GNU C++ and is freely available on Github[3].

[3] https://github.com/CominLab/QCluster

2.2 isONclust

Before presenting isONclust in more detail, we need to present some other definitions.

▶ **Definition 8** (Average base error rate). *Given a read sequence r, we refer to $\epsilon_r = \sum_{i=1}^{|r|} Q_r(r)/|r|$ as the average base error rate.*

▶ **Definition 9** (Minimizer). *Given a read sequence r and two integers k, w such that $1 \leq k \leq w \leq |r|$, the **minimizer at position i** is the is the smallest subsequence of r in lexicographical order of length k that starts at a position in the $[i, i + w)$ interval. We refer to the integer w as* window size*.*

▶ **Definition 10** (Set of minimizers). *We refer to $M(r)$ as the set of ordered pairs which contain all the minimizers of a read r and their start positions on r. All subsequences of the set $M(r)$ are called* minimizers *of r.*

▶ **Definition 11** (Shared probability). *It is the fraction of a read r's sequence that is shared with a* representative *of a cluster.*

isONclust is a software for clustering long-reads according to their gene family of origin. It uses a greedy algorithm that claims to scale better than other clustering algorithms, QCluster included. The behavior of the algorithm is as follows:

(i) It sorts the reads so that longer and better-quality sequences appear earlier. To do so, a score $s(r)$ of a read r is computed, which represents the expected number of error-free k-mers in r. This score is computed in a linear scan via a running product over a sliding window of k quality scores. Larger values of w give sparser sampling of *minimizers*.

(ii) It processes the reads one-by-one according to the sorted order. At any point of the processing, it maintains a clustering of the processed reads and, for each cluster, it keeps one read as the *representative* of a cluster. Once a representative is fixed, it is never recomputed. Representatives are chosen such that they have the largest expected number of error-free k-mers of any future reads in the cluster they are assigned to. Also, it maintains a hash table $H(x)$ that returns the representatives that have a k-mer x as a *minimizer*.

The processing step is as follows:

(a) It counts the minimizers within every representative computed so far, querying H. Any representative that shares at least a minimizer with the read r is called *candidate*.

(b) It tries to assign a read to the candidate representative's cluster c. The candidate representatives are sorted such that the ones that share most of the minimizers are processed first. For each representative, an estimate of the *shared probability* is computed, and if it is greater than 0.7, the read is assigned on that representative's cluster, otherwise the next candidate is processed. If the current representative's shared minimizers are less than 70% of the top-sharing representative's, or less than 5 units, the algorithm falls back to step (c).

(c) If the previous step does not succeed, align the read r with the representative with the highest amount of shared minimizers, using the slow but exact Smith-Waterman alignment algorithm [SW81], and assign the read to a representative's cluster similarly to (b).

The scoring parameters that isONclust uses for the Smith-Waterman alignment of a read r to a representative c of a cluster are: **match**: 2, **mismatch**: -2, **gap extension**: -1. The gap opening penalties vary according to the combined error rate of the read r and the representative c, denoted by $\epsilon = \epsilon_r + \epsilon_c$. The **gap opening** penalty is defined as:

$$\text{gap opening penalty} = \begin{cases} 2 & \text{if } \epsilon > 0.1 \\ 3 & \text{if } 0.04 < \epsilon \leq 0.1 \\ 4 & \text{if } 0.01 < \epsilon \leq 0.04 \\ 5 & \text{if } \epsilon \leq 0.01 \end{cases}$$

isONclust applies homopolymer compression to reads before processing them, which guarantees both a reduction of indel errors during alignment and a removal of repetitive minimizers.
A priori probabilities are used to estimate the fraction f of a read r's sequence that would be aligned to a representative c. $p(\epsilon_r, \epsilon_c)$ is the probability known a priori that a minimizer in a read r doesn't match with another read c, knowing that both r and c are generated from the same transcript with error rates ϵ_r and ϵ_c, respectively. There is no known closed-form expression to determine the a priori probabilities, thus they are computed via simulation and saved in a lookup table[4].

Compared to QCluster, isONclust has much less configurable parameters: only the window size w and the k-mer size k. In particular, it does not let the user specify the number of clusters. However, it allows clustering batches of sequences in parallel. Moreover, it supports both `FASTA` and `FASTQ` file formats as input, and it produces a `TSV` file `final_clusters.tsv` as output. The output has two columns like QCluster, but it has a slightly different structure: the first column is the cluster ID, while the second one is the complete header of the corresponding clustered read.

isONclust is written in Python 3 and is freely available on Github[5].

3 Cluster Validation Metrics

Clustering validation, which evaluates the quality of clustering results, is one of the most relevant issues essential to the success of clustering applications [Sar90, MB02]. Assessing the quality of a clustering result is usually a problem as difficult as cluster analysis itself. The two main approaches of clustering validation are *external evaluation* and *internal evaluation*. The main difference between these two evaluation categories is whether or not external information is used for clustering validation.

Internal evaluation methods measure the goodness of a clustering structure without requiring external information [TSK05]. Examples of these measures are the Silhouette coefficient [Rou87] and the Dunn index [Dun74]. **External evaluation** methods, on the other hand, compare the results of a cluster analysis to a *ground truth* result, and measure the extent to which cluster labels match the externally supplied ground truth [TSK05], up to a permutation.

3.1 Selected Clustering Scores

Since we only have access to the final outcome of QCluster and isONclust and not to the internally computed distances between the elements of the clusters, we cannot use internal evaluation methods for our experiments. Instead, we adopt the external evaluation scores presented in the following paragraphs. The scores we deemed appropriate for this task are:

- **Homogeneity**;
- **Completeness**;
- **V-Measure**;
- **Adjusted Mutual Information (AMI)**;
- **Adjusted Rand Index (ARI)**;
- **Purity**.

The definition and meaning of these scores in explained in the following paragraphs.

[4] https://github.com/ksahlin/isONclust/blob/master/modules/p_minimizers_shared.py
[5] https://github.com/ksahlin/isONclust

Assume a data set comprising N data points, and two partitions of these points:

- a set of class labels, $C = \{c_i | i = 1, \dots, n\}$;
- a set of clusters, $K = \{k_i | 1, \dots, m\}$

Let us call a_{ij} the number of data-points belonging to the class $c_i \in C$ and cluster $k_j \in K$.

The maximum reduction in entropy clustering C can provide is defined as:

$$H(C) = -\sum_{c=1}^{|C|} \frac{\sum_{k=1}^{|K|} a_{ck}}{n} \cdot \log(\frac{\sum_{k=1}^{|K|} a_{ck}}{n})$$

Analogously, we explicitly write $H(K)$ as:

$$H(K) = -\sum_{k=1}^{|K|} \frac{\sum_{c=1}^{|C|} a_{ck}}{n} \cdot \log(\frac{\sum_{c=1}^{|C|} a_{ck}}{n})$$

$H(C|K)$ is the conditional entropy of the class distribution given the proposed clustering:

$$H(C|K) = -\sum_{k=1}^{|K|} \sum_{c=1}^{|C|} \frac{a_{ck}}{N} \cdot \log(\frac{a_{ck}}{\sum_{c=1}^{|C|} a_{ck}})$$

$H(K|C)$ is the conditional entropy of the proposed cluster distribution given the ground truth labels C:

$$H(K|C) = -\sum_{c=1}^{|C|} \sum_{k=1}^{|K|} \frac{a_{ck}}{N} \cdot \log(\frac{a_{ck}}{\sum_{k=1}^{|K|} a_{ck}})$$

Homogeneity

A clustering satisfies Homogeneity if all of its clusters contain only data points which are members of a single class label [HR07]. It is bounded between 0 and 1, with low values indicating a low Homogeneity, and with 1 being the best value. Homogeneity is defined as:

$$h = \begin{cases} 1 & \text{if } H(C|K) = 0 \\ \frac{H(C|K)}{H(C)} & \text{else} \end{cases}$$

The most homogeneous clustering is the one where each cluster k has data-points belonging to the same class label. The *trivial Homogeneity* is the extreme case where the number of clusters $|K|$ is equal to the number of data points N, and each point is exactly in one cluster. In this case, the Homogeneity is 1 and the Completeness is 0. Homogeneity is the analogous to the *Precision* score for binary classification problems.

Completeness

A clustering satisfies Completeness if the data points which are members of a given class are also elements of the same cluster [HR07]. Like Homogeneity, the Completeness measure is bounded between 0 and 1, with 1 being the best score. Completeness is defined as:

$$c = \begin{cases} 1 & \text{if } H(K|C) = 0 \\ \frac{H(K|C)}{H(K)} & \text{else} \end{cases}$$

The most complete clustering is the one where all data points belonging to the same class label c are grouped into the same cluster. The *trivial Completeness* is the extreme case where the data

points are grouped into a single cluster. In this case, the Completeness is 1 and the Homogeneity is 0. Completeness is the analogous to the *Recall* score for binary classification problems.

V-Measure

V-Measure is an entropy-based measure which explicitly measures how successfully the criteria of homogeneity and completeness have been satisfied [HR07]. It is computed as the harmonic mean between homogeneity and completeness scores. Again, V-measure is bounded between 0 and 1, with 1 as the best score.

$$v = \frac{(1+\beta) * h * c}{\beta * h + c}$$

The parameter β can be adjusted to favor either the Homogeneity or the Completeness of the clustering algorithm. If $\beta > 1$ then Completeness is weighted more strongly in the evaluation, whereas if $\beta < 1$, Homogeneity is preferred. For our experiments, we fixed $\beta = 1$.

▶ Note. Homogeneity and Completeness of a clustering are sometimes opposite: increasing the homogeneity of a clustering often results in decreasing its Completeness.

▶ Note. Homogeneity, Completeness, and V-Measure scores are completely independent of the number of classes $|C|$, the number of clusters $|K|$, the size of the dataset and the particular clustering method used.

Homogeneity is the analogous to the F-score for binary classification problems.

(Adjusted) Mutual Information

Mutual Information (MI) is computed between two variables and it measures the reduction in uncertainty for a variable given a known value for the other variable. Mutual Information measures the dependence between two random variables and is thus a symmetric measure. Mutual Information can also be thought of as the Kullback-Leibler divergence between the joint probability distribution and the product of the marginal probabilities of each variable. MI is always non-negative, with 0 indicating that the two variables are independent, and thus do not share any mutual information. Normalized Mutual Information (NMI) is a normalized of the MI score that scales the results between 0 and 1, with 1 indicating the perfect correlation.

Adjusted Mutual Information (AMI) is Mutual Information accounted for chance. The pure MI score, in fact, is generally higher for two clusterings with a large number of clusters, regardless of whether there is actually more information shared. The Adjusted Mutual Information score is bounded between 0 and 1, with 1 as the best score. The interested reader can refer to [VEB10] for more details.

(Adjusted) Rand Index

The previously introduced metrics, except for AMI, are not normalized with respect to random labeling: this means that depending on the number of samples, clusters and ground truth classes, a completely random labeling will not always yield the same values for Homogeneity, Completeness and hence V-Measure. In particular, random labeling will not yield zero scores, especially when the number of clusters is large.

The Rand Index computes a similarity measure between two clusterings by considering all pairs of samples and counting pairs that are assigned to the same or different clusters in the predicted and true clustering [Ran71].

We can call TP the number of pairs of elements that share the same class label and are grouped in the same cluster, and TN the number of pairs of elements that do not have class labels in common and are grouped in different clusters. The Rand Index score is then computed as:

$$RI = \frac{TP + TN}{\binom{n}{2}}$$

The Rand Index measure is bounded between 0 and 1, with 1 as the best score.
A problem of the Rand Index is that it does not guarantee to return a low score when the clustering result is created at random. Moreover, false positives and false negatives are equally weighted. The Adjusted Rand Index solves these issues: it is ensured to have a value close to 0 for random labeling independently of the number of clusters and samples, and 1 when the clustering results are identical to the ground truth (up to a permutation). It can also take negative values. Other details can be found in [HA85].

Adjusted Mutual Information (AMI) and Adjusted Rand Index (ARI) scores are similar, but they are not equivalent. In fact, ARI is better suited for evaluating clustering on equal sized clusters, whereas AMI is more appropriate when the clustering is unbalanced and there exist small clusters. We decided to include both scores to observe how these metrics vary in particular when using IsONclust, that produces unbalanced clusters with many small groups.

Purity

The Purity metric is a measure of the extent to which a cluster contains only elements from the same single class label. It is the percentage of the total number of points that were clustered correctly according to the ground truth. It is bounded between 0 and 1, with 1 being the best score. The trivial case for Purity occurs when each data point is partitioned in a distinct singleton cluster. The Purity score is computed as:

$$Purity = \sum_{k \in K} \frac{|k|}{N} \max_{c \in C} \frac{|k \cap c|}{|k|}$$

▶ Note. Purity penalizes the noise in a cluster, but it does not reward grouping items from the same class together: the higher the number of clusters, the more likely a high purity is returned.

▶ Note. The Purity metric may give misleading results when the ground truth clusters are not balanced. We do not worry about this issue in this report, as the datasets built in Section 4 are perfectly balanced.

4 Dataset

We used the Homo Sapiens (human) cDNA[6] assembly `GRCh38.p13` to generate the synthetic datasets for our experiments. This is one of the datasets used by Sahlin in [SM19]. The `GRCh38.p13` assembly, which we refer to as *originating dataset*, contains 251298 distinct cDNA sequences in a `FASTA` file. We named this file `input.fasta` and placed it in the `data/original` folder, but since it is particularly heavy (400 MB), we did not add it to our project repository. The `GRCh38.p13` assembly, can be downloaded from the `ENSEMBL` repository:

- Visit `ENSEMBL` BioMart[7].
- On the website, choose the database "Ensembl Genes" and the dataset "Human genes (GRCh38.p13)".
- Click on the "Attributes" entry in the menu and check the "Sequences" box.

[6] A cDNA library represents a sampling of the transcribed genes.
[7] https://www.ensembl.org/biomart/martview

- Expand the "SEQUENCES" tab and check the "cDNA sequences" box.
- Click on the "Results" tab on the menu.
- In the "Export all results to" select box, choose "Compressed web file (notify by email)", and write an email address in the "Email notification to" box.
- Press the "Go" button, and in less than an hour, an email with a download link should be delivered to the email address specified in the previous step.

▶ Note. One could also try to download the originating dataset directly, without resorting to email services, but in our experience that would not work because of `ENSEMBL` server issues.

To keep the data volume easily tractable for the limited hardware and time constraints at disposal, we selected only 100 distinct random sequences from the originating dataset as a basis for the synthetic datasets on which our experiments were based. We will refer to these 100 distinct sequences as *originating sequences*.

4.1 Preprocessing

Rather than choosing the 100 originating sequences completely at random, we decided to guide the decision according to the following requirements:

(a) We should select sufficiently large sequences, i.e. the shortest sequence should be at least 1000bp long;
(b) The differences in size among the selected sequences should be negligible;
(c) We should have a sufficiently large pool of sequences (e.g. at least 1000) that satisfy requirements (a) and (b) to sample 100 sequences from with uniform probability.

Sequence Partitioning

To satisfy requirement (b), we performed a greedy sequence partitioning based on the standard deviation of the sequence lengths.

We read the `data/original/input.fasta` file containing the original sequences and saved the (*key*, *length*) pairs, where *key* is the sequence identifier and *length* is the length of the sequence. We accessed the file in a streaming fashion to avoid loading it in memory all at once. Time: $\mathcal{O}(n)$, where n is the number of cDNA sequences read (251298). We used the `BioPython`[CAC+09] library to manipulate `FASTA` files.

We then sorted the list of $(key, length)$ pairs such that the shortest sequences were placed at the beginning of the list, which we will call S. Time: $\mathcal{O}(n \cdot \log n)$.

Next, we calculated the standard deviation σ of the gaps between two consecutive sequences' lengths, and we greedily partitioned the sequences according to σ. A new partition is created only when the length gap between any two consecutive sequences sorted by length is larger than a certain threshold. The first and smallest sequence s_1 is always assigned to the first partition P_1. All other distinct sequences $s_2, \ldots, s_n \in S$ induce the creation of a new partition when the condition

$$\frac{length(s_j) - length(s_i)}{\sigma} > \alpha$$

holds, where $i + 1 = j$ and α is a threshold on the standard deviation to determine experimentally. This partitioning step requires time $\mathcal{O}(n)$. We have determined that $\alpha = 0.0005$ is a threshold value that creates partitions that are suitable enough for our experiments.

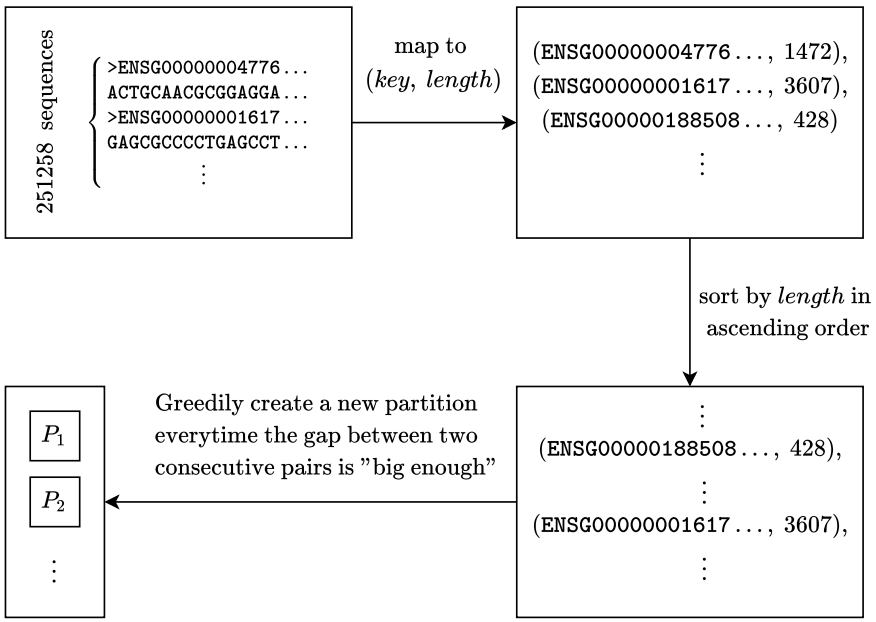


Figure 1 Summary scheme of sequence partitioning. The 251298 originating sequences in the `input.fasta` file are mapped to the (*key*, *length*) pairs, sorted by *length*, and partitioned using the greedy *standard deviation* criterion.

Partition Pruning

Recalling the (a), (c) requirements, we want to find a partition P with a large cardinality ($|P| \geq 1000$) where the length gap between the smallest and the longest sequence is minimum. Table 1 shows the four biggest partitions, their cardinalities, and the upper and lower bounds of the length of the sequences contained in those partitions. Partition P_1 has the highest cardinality and the worst gap among the sequences' lengths. P_4 has the best (i.e. shortest) gap among the sequences' lengths, but it is too small ($|P_4| \not\geq 1000$). Thus, the partition P_3 is selected, as it contains 1107 sequences to sample from with lengths varying in the $[5481, 5572]$ base pairs range.

Uniform Random Sampling

We finally sampled 100 sequences without repetition from partition P_3, with uniform probability, and saved them to a file named `preprocessed.fasta` in the `data/preprocess` folder.

The overall preprocessing phase should just take some seconds.

	$\lvert P_i \rvert$	$\text{length}_{\text{min}}$	$\text{length}_{\text{max}}$	Gap
P_1	237799	41	4748	4707
P_2	1254	4750	4951	201
$\boldsymbol{P_3}$	**1107**	**5481**	**5572**	**91**
P_4	927	5165	5204	39

Table 1 Statistics about the 4 biggest sequence partitions found in the preprocessing phase. Partition P_3 contains the subset of originating sequences used to create the synthetic datasets we used for our experiments.

4.2 Dataset Simulation

We used the SimLoRD [SKR16] software for generating synthetic long reads starting from the 100 originating sequences selected in the preprocessing step. SimLoRD accepts different options for determining the read lengths. We provided the size of the simulated data to generate via the `-n` option: 10000, 20000, and 50000. In accordance with the supervisor Professor, we specified two fixed read lengths (via the `-fl` option): 100 and 700 base pairs.

Moreover, we used SimLoRD default configuration for generating imperfect sequences with the following error probabilities:

- $prob_{\text{ins}} = 11\%$;
- $prob_{\text{del}} = 4\%$;
- $prob_{\text{sub}} = 1\%$;

where $prob_{\text{ins}}$, $prob_{\text{del}}$, $prob_{\text{sub}}$ are, respectively, the probability for insertions, the probability for deletions, and the probability for substitutions for reads with one pass. Introducing errors in the synthetic reads is fundamental for real case scenarios and establish how QCluster and isONclust might react to imperfect reads.

SimLoRD creates two files:

1. A `FASTQ` file (which we called `simulated.fastq`) that contains the actual simulated long reads with the read error qualities;
2. A `SAM` file (which we called `simulated.sam`) that contains the alignment reference for the corresponding `simulated.fastq` file.

We thus generated six datasets with 10000, 20000, and 50000 simulated sequences of length 100 and 700. We named these datasets after the corresponding SimLoRD option names. For instance, the simulated dataset with 10000 simulated sequences of length 700 is called `n-10000-fl-700`. The synthetic datasets are saved in the `data/simulated/` folder. Table 2 summarizes the features of the six simulated datasets.

	Read length = 100	**Read length = 700**
Cardinality (N)	**Dataset name**	
10000	`n-10000-fl-100`	`n-10000-fl-700`
20000	`n-20000-fl-100`	`n-20000-fl-700`
50000	`n-50000-fl-100`	`n-50000-fl-700`

Table 2 Summary of the six simulated datasets whose names vary with respect to the cardinality (number of distinct sequences) and the length of said sequences.

SimLoRD is written in Python 3 and is publicly available on BitBucket[8].

Unsuccessful Simulation Attempts

At first, we tried to use NanoSim[9] to simulate the synthetic datasets, under initial Sahlin's suggestion. After facing multiple installation issues first, inconsistent documentation, and troubles with the `FASTQ` output, we decided to abandon our efforts on NanoSim and resorted to using SimLoRD.

[8] https://bitbucket.org/genomeinformatics/simlord/src/master
[9] https://github.com/bcgsc/NanoSim

About Datasets Size

Initially, we wanted to extract 1000 originating sequences to generate the simulated datasets with 100.000, 500.000, and 1.000.000 sequences. This type of setup would have allowed us to reproduce some of the experiments of [SM19] more faithfully. However, we quickly realized that the time and memory needed to perform multiple cluster analysis with different parameter combinations of QCluster and isONclust are excessive with respect to the limited hardware and time at our disposal. We also restricted ourselves to consider only short-reads for similar reasons.

5 Method

We selected a subset of parameters for both QCluster and isONclust and we ran the two software tools on the simulated datasets. We used the `subprocess` module of `Python3` to spawn additional OS processes to run the executables of QCluster and isONclust. The only parameter that QCluster and isONclust share in common is the k-mer length, specified by the `-k` and `--k` command-line options, respectively.

Once we fixed the range of parameters that the clustering tools should accept in input, we spawned such processes lazily iterating over the possible parameter combinations in a deterministic order, using a structure called *parameter grid*. This parameter grid is akin to the structure used in grid search tasks for hyperparameter validation: given a dictionary of software parameters with the relative considered option range as values, it iteratively invokes the clustering processes with all parameter combinations among the specified ranges.
We even generated an additional random clustering as a baseline to compare the performance of QCluster and isONclust. The output of the clustering processes is captured and redirected to `TSV` files that describe the clustering result: one column indicates the cluster ID assigned by the software, and the other column identifies the sequence assigned to that cluster.

The `TSV` files containing the clustering results are later processed by another `Python3` module to evaluate the external cluster metrics listed in Section 3, along with other statistical evaluations. In the quality assessment phase, we also take into account the possible small clusters created by isONclust's greedy algorithm (that automatically determines the appropriate number of clusters). We define **singletons** as cluster with a single sequence, and **trivial clusters** as those partitions with at most 5 clusters. The result of the quality assessment is saved into 3 different `CSV` files for each parameter combination and for each dataset (each file has only the header and a single row of values): one file that considers every cluster determined by an algorithm, one that excludes the *singletons*, and another one that excludes every *trivial* clusters from the evaluation.

Finally, we imported the `CSV` files in a `Python` notebook; we aggregated the `CSV` files using the `Pandas`

n	fl	Homogeneity	Completeness	V-Measure	ARI	AMI	Purity
10000	100	**0.1217**	**0.1216**	**0.1217**	-0.0001	-0.0011	**0.0417**
	700	0.1198	0.1197	0.1197	-0.0004	-0.0033	0.0411
20000	100	0.06	0.06	0.06	0	-0.0008	0.0319
	700	0.0602	0.0602	0.0602	-0.0001	-0.0006	0.0306
50000	100	0.023	0.023	0.023	**0.0001**	**0.0007**	0.0228
	700	0.022	0.022	0.022	0	-0.0003	0.0227

Table 3 Metrics of the baseline random clustering result. n is the number of simulated sequences in the dataset. fl is the fixed length of all these n sequences. As expected, the cluster quality values of the Adjusted Rand Index and Adjusted Mutual Information scores are close to 0. Highest values are in bold.

library [pdt20], and we visually inspected how the change of a parameter would influence the other parameters in our experiments. We performed what we called an *orthogonal visual analysis*: we fixed some parameters, and we observed the interactions between two variables while varying a third variable used to determine the colour encoding. We used the `Seaborn` library [Wtsdt20] to generate visual insights from the quality results, and `IPython` notebooks [PG07] to code it and save snapshots of the obtained plots.

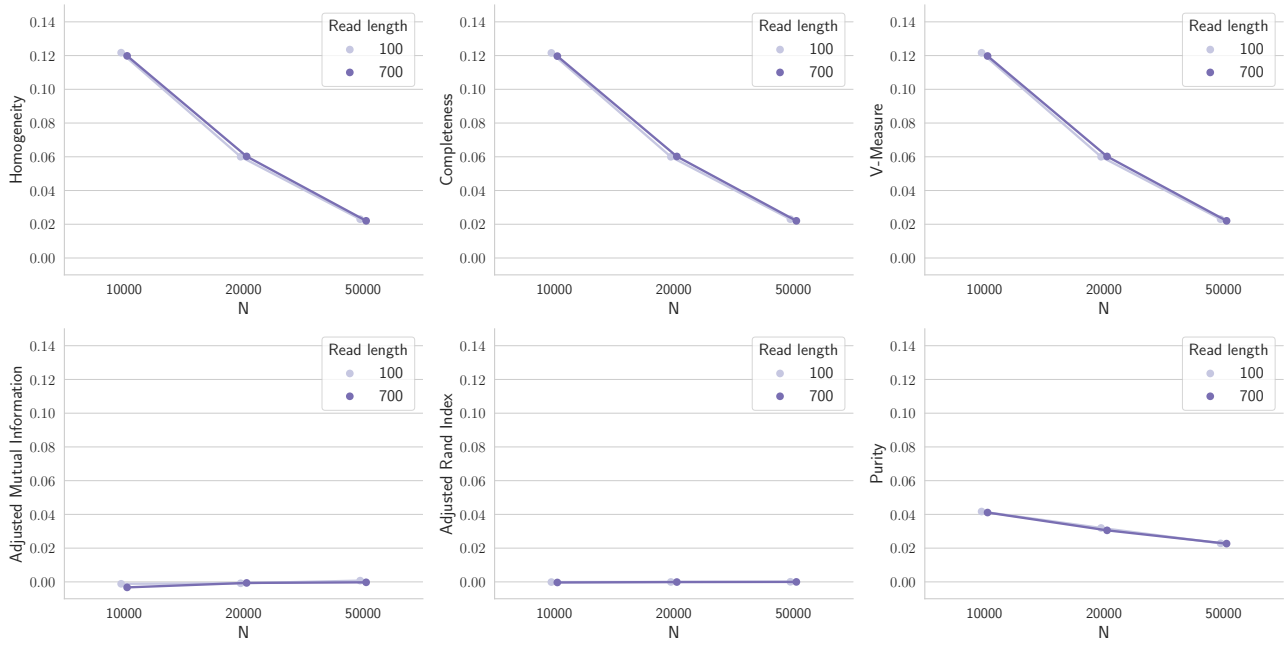


Figure 2 Clustering quality of the baseline random clustering. We can notice how the AMI and ARI scores are 0, as expected, and that Purity is very low as well.

We have run our experiments on a laptop running Ubuntu 20.04 on a 64-bit architecture, mounting a quad-core Intel(R) Core(TM) i7-8550U @1.80GHz CPU with 16GB of RAM. These specifications are summarized in Table 4.

5.1 Reproducibility

In the following two paragraphs, we document our concern about reproducing the experiments we performed.

Docker

Docker is a cross-platform software tool for packaging an application with all parts it needs, such as build artifacts and third-party dependency, regardless of the operating system and the third-party libraries installed in the system in use. We decided to use Docker to ensure the complete cross-platform reproducibility of our experiments after facing installation or compilation issues in two different computers at our disposal. Since multiple third-party dependencies that either ISONCLUST, SimLoRD, or our code use are native (i.e., written in `Fortran`, `C`, or `C++` and embedded in a `Python` module), there is a high probability to have build errors due to how different operating systems, system architectures or even `Python` versions resolve native modules. Moreover, some libraries require the presence of other system libraries that are not commonly installed, such as `libbz2-dev` or `zlib1g-dev`.
Another benefit of using Docker is that researchers who want to reproduce our experiments will not have to clutter their computers with dozens of new modules and libraries, or with incompatible `Python`

versions.
For further information on how to download and install Docker, please refer to the official website[10]. Instructions on how to execute the docker images we prepared can be found in Appendix E.

Random Seed

A *random seed* is a number used to initialize a pseudorandom number generator *rng*. If *rng* is re-initialized multiple times with the same fixed random seed, it will produce the same sequence of numbers.

QCluster optionally accepts a fixed random seed as a configuration parameter, to ensure reproducing the same clustering results. isONclust, on the other hand, does not give such opportunity to the user. We decided to avoid explicitly setting a random seed both because it would have created a bias in QCluster results with respect to isONclust's results, and also under Professor M. Comin's suggestion.

5.2 Baseline Clustering

We performed a randomized clustering of the simulated reads as a baseline model to corroborate the hypothesis that both QCluster and isONclust produce meaningful results, i.e. that the quality metrics of their clustering results is substantially better than a uniformely random partitioning of the simulated sequences.

We started by reading the `simulated.fastq` file for each simulated dataset, keeping track of the unique sequence read ID in a list. Time: $\mathcal{O}(n)$, where $n \in 10000,\ 20000,\ 50000$ is the size of the simulated dataset taken as input. We then shuffled the list with uniform probability and partitioned it into $k = 100$ different clusters. Time: $\mathcal{O}(n)$.

The baseline clustering results are written in the `inferred_clusters.csv` file in the `data/random_cluster/n-*-fl-*/` folder, where `n-*-fl-*` is the name of each simulated dataset. The metrics of the baseline clustering are reported in Table 3.

OS	Ubuntu 20.04
Arch	x64
CPU	Intel(R) Core(TM) i7-8550U @1.80GHz
RAM	16GB

Table 4 Summary of the hardware specifications of the laptop used for the experiments.

5.3 QCluster Clustering

For QCluster, we chose integer k-mer lengths in the $[4,\ 9]$ range (option `-k`) and three possible distance metrics for the K-Means clustering, specified with the command `-d` parameter:

- D_2^{*q} (`-d a`);
- L_2 (`-d e`);
- χ^2 (`-d c`).

We also left unchanged two notable default options of QCluster: we used **AWP** prior probability estimators the D_2^{*q} distance and we performed a single run of the K-Means algorithm. Most importantly, since the number of *originating sequences* (i.e. the ground truth for the examined clustering task) is 100, we explicitly specified the number of clusters with the `-c 100` option. Moreover, after we realized

[10] https://www.docker.com/get-started

that QCluster is order of magnitudes slower than isONclust, we decided to keep the number of runs of the K-Means algorithm to 1 (option `-t 1`), which is the default option. This decision prevented QCluster from selecting the best clustering results across many runs, but helped us in keeping the time for the experiments under control.

For each simulated dataset we analyzed with QCluster, we saved a `TSV` file called `inferred_clusters.tsv` with the inferred clusters in a folder whose name is a composition of the parameters used. E.g., the clustering results with L_2 distance metric and k-mer length equal to 7 (`qCluster -d e -k 7`) for the `n-10000-fl-700` dataset are saved in the `n-10000-fl-700/d-e-k-7/` folder inside `data/qCluster/`.

Even though we set the k-mer lengths to very small values, we were not able to obtain clustering results using QCluster with $k = 9$ and, in some cases, even with $k = 8$ and $k = 7$. In fact, one major drawback of QCluster is its extensive use of memory.

5.4 isONclust Clustering

For isONclust, we expanded the k-mer lengths to the $[4,\ 11]$ range (option `--k`), and we considered two window sizes mentioned in [SM19]:

- 20 (`--w 20`);
- 50 (`--w 50`).

We also leveraged multithreading with the `--t` option and instructed isONclust to process the `FASTQ` simulated sequences with the `--fastq` option.

Similarly to what we did in Section 5.3, we saved a `TSV` file named `final_clusters.tsv` with the inferred clusters in a folder whose name is a composition of the parameters used, for each processed dataset. E.g., the clustering results with a window of size 50 and k-mer length equal to 7 (`isONclust --w 50 --k 7`) for the `n-10000-fl-700` dataset is saved in the `n-10000-fl-700/w-50-k-7/` folder inside `data/isONclust/`.

5.5 Clustering Quality Assessment

For each clustering result computed over the six synthetic datasets, we collected the external evaluation metrics that we presented in Section 3.1.
We also gathered some basic statistics about the clustering results:

(a) *# clusters*: number of clusters obtained (fixed for QCluster, variable for isONclust);
(b) *Min size*: minimum number of sequences in a cluster;
(c) *Max size*: maximum number of sequences in a cluster;
(d) *Average size*: average number of sequences in a cluster;
(e) *Standard dev*: standard deviation of the number of sequences in a cluster.

Moreover, we computed the same metrics and statistics for the random baseline. The logic of this process is defined in the `Python3` module `python/quality`.

Given a dataset, we import the `simulated.sam` alignment file, which constitutes the *ground truth*. This file is read using the `pysam` library, skipping the header lines (the ones that start with the `@` symbol). For each sequence in the alignment file:

(i) We collect the read ID, e.g. `m0/100/CCS`;
(ii) We collect the originating chromosome string, e.g. `ENSG00000070061|ENSG00000070061.16|ENST00000674938|ENST00000674938.1`. This chromosome is a string that uniquely identifies the cDNA sequence that was used to simulate the given dataset.

(iii) We then populate a map `classes` that associates each read ID to originating chromosome and another map `chromosome_to_read_ids_map` that associates each chromosome to set of read IDs.

We then read the `clusters` map and the number k of distinct cluster labels assigned by the specific clustering tool. This procedure reads the `TSV` file produced by either QCluster or isONclust. `clusters` maps a read ID (e.g. `m0/100/CCS`) to a cluster ID, i.e. an integer number in $\{0, \dots, k-1\}$. We also compute a map that associates a cluster ID to the set of read IDs grouped in the same cluster.

Afterwards, we compute the external evaluation metrics and statistics, discriminating between *singleton*, *trivial*, and *standard* clusters. Singletons are clusters with cardinality equal to 1. Trivial clusters are clusters whose cardinality is $\leq$ `threshold`, a parameter of the `quality` module. In our case, `threshold` is 5.

Finally, we generate three CSV files:

(i) One file with the metrics and stats computed on the clusters as they have been originally computed;
(ii) One with the same information but without considering singletons;
(iii) Another one with the same information but without considering trivial clusters.

We opted for this distinction to offer more insights about the clustering results, especially for isONclust, which tends to generate many small clusters.

▶ Note. *Singletons* are a subset of the group of *trivial clusters*.

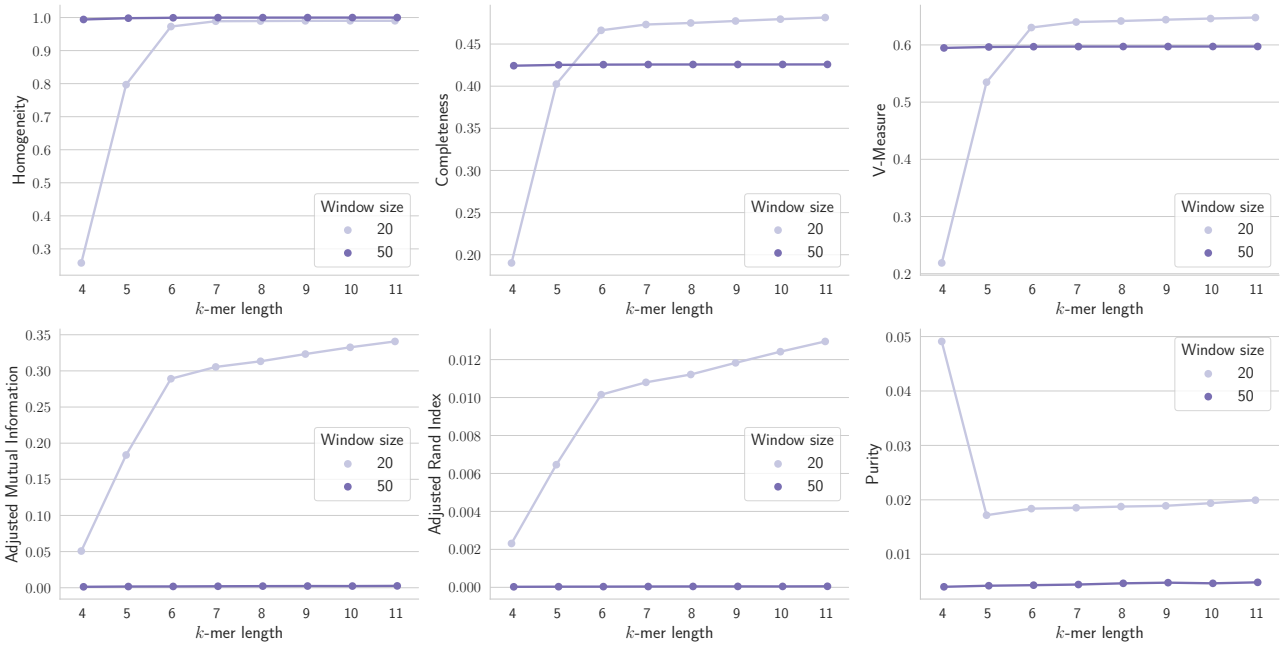


Figure 3 isONclust: Clustering quality metrics vs k-mer length distinguished by window size. All clusters on dataset `n-50000-fl-100` are considered. For $w = 50$, the clustering is even worse than the random clustering.

6 Results

Due to the greedy nature of isONclust, and the fact that it does not accept the desired number of clusters as input, the clustering results have a very diversified structure both in terms of number of partitions founds and their cardinality. The tables in Appendix A present a summary of the number of

clusters identified by QCluster and isONclust across all datasets, varying the *distance* metric and the *window* size, respectively. These tables also distinguish between trivial, non-trivial, and non-singleton clusters. As expected by the nature of the K-Means algorithm used under the hood, QCluster always finds the number of clusters it is told to determine: 100. We notice that in more than 95% of the cases, clusters found for read length equal to 700 are non-trivial. The situation is diametrically different when the read length is equal to 100, especially for smaller datasets and as the k-mer grows, where 25% to about 50% of the clusters of the `n-10000-fl-100` datasets are singletons. This is probably due to a combination of a bad K-Means centroid initialization, the fact that Lloyd's algorithm is an approximation algorithm without strong guarantees, and that finding 100 dense clusters with just 10000 sequences at disposal is difficult. Moreover, K-Means is known for being badly affected by outliers, which might emerge due to the errors simulated by SimLoRD. This drawback of QCluster might be mitigated by running the underlying clustering algorithm multiple times with the `-t` options and saving just the result of the bad run. However, the extremely long running times of QCluster did not allow us to repeat the K-Means algorithm multiple times.

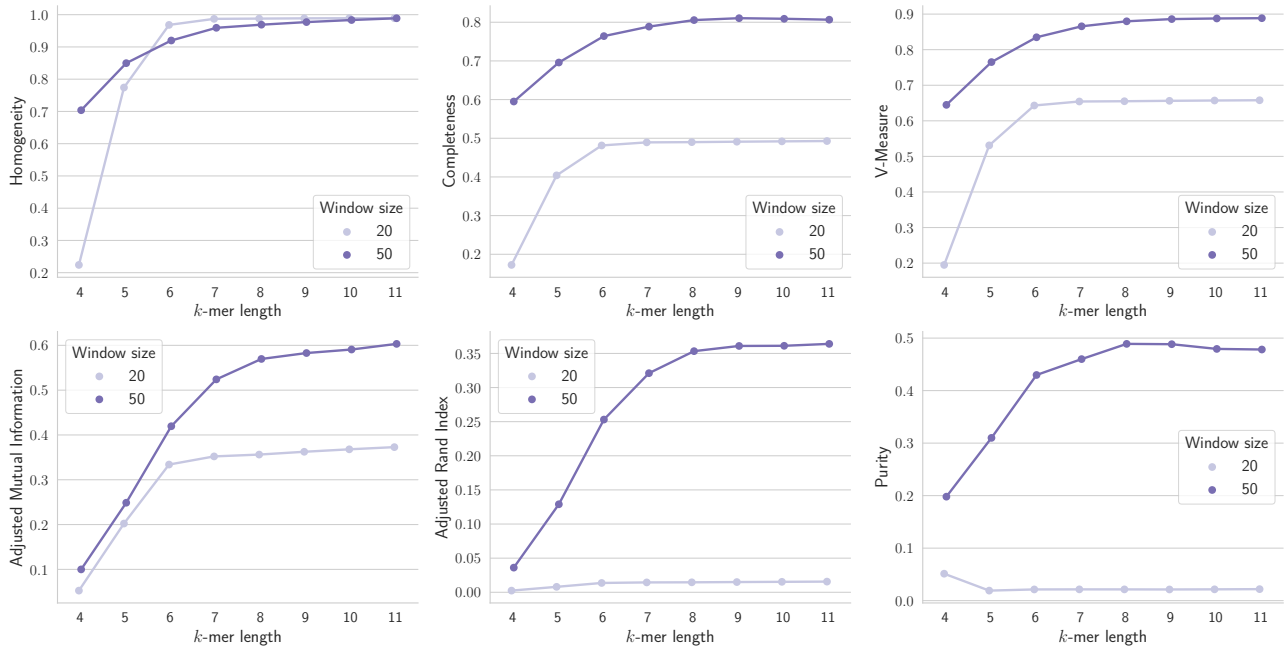


Figure 4 isONclust: Clustering quality metrics vs k-mer length distinguished by window size. Only non-singleton clusters on dataset `n-50000-fl-100` are considered. Clusterings with $w = 50$ are substantially better with respect to every metric.

On the other hand, it can be noted that isONclust determines many thousands of clusters, most of which are trivial. As an extreme case, for the dataset `n-50000-fl-100`, it determines 49550 singletons. For read length equal to 100, the least number of clusters found is 607, due to the simplest configuration: windows size $w = 20$, k-mer length equal to 4, and dataset `n-10000-fl-100`. If we do not consider the increasing pattern between $k = 4$ and $k = 5$, the number of clusters found, is stable with respect to the window size and the size of the dataset, without influence from k when $k \geq 6$. With a bigger read length, the number of singletons is drastically reduced and is often 0. In this case, even though there are still hundres of trivial clusters, the majority of the clusters found are non-trivial, and are always ≤ 1800. Still, isONclust is still off the mark when it comes to determine the optimal number of clusters, 100.

The tables in Appendix B show the statistics obtained from the clustering results in terms of average

size, standard deviation of the size, minimum and maximum sizes. Again, the statistics are aggregated to show the influence of QCluster's *distance* metric and isONclust's *window* size in the clustering results. For QCluster, the D_2^{*q} is the distance metric with the highest standard deviation in cluster cardinality, as the smallest cluster found with $k = 7$ is a singleton and the biggest contains 19242 sequences when clustering on the `n-20000-fl-100` dataset. The biggest cluster cardinality is instead found with same distance metric on the `n-50000-fl-100` dataset: 22825 sequences in a cluster. The average size, however, is constant as expected: $\frac{n}{100}$, i.e. 100 for datasets with $n = 10000$ sequences, 200 for $n = 20000$, and 500 for $n = 50000$. Finally, there seems to be an increasing correlation between the k-mer length and the maximum cardinality obtained for a cluster.

Appendix C lists the values of the clustering metrics mentioned in Section 3.1 organized by multiple pivot tables. For QCluster, the metric results are aggregated by the k-mer length and the three chosen distance metrics (D_2^{*q}, L_2, χ^2), and a single metric is considered for each table, varying on the number of sequences N and the read lengths (100 or 700). We only show the metrics for non-trivial clusters, as they represent the absolute majority of clusters found by QCluster. No row for $k = 9$ is shown, because QCluster exhausted the memory before even attempting run its clustering algorithm for all datasets. The same is true for $k = 8$ except for the two smallest datasets, `n-10000-fl-100` and `n-10000-fl-700`. Moreover, QCluster was not able to compute the clustering results for `n-50000-100` using the L_2 and χ^2 distances.
For isONclust, analogously, the metric results are aggregate by the k-mer length and the two chosen window sizes (20 and 50), varying on N and the read lengths, considering a single external evaluation metric per table. isONclust succeeds in considering all k-mer lengths in the $[4, 11]$ range. However, some values are missing for read length equal to 100 from $k \leq 5$, in correspondance to window size $w = 50$. In those cases, isONclust throws an index-out-of-bounds error. From our understanding of the tool's open-source code[11], it is merely a bug in the function that computes the k-mer minimizers, where the accessed array is the original read length (of size 100 in this case) on indexes dependent on k and w. We notified the author of the software about this issue.

Figure 7 shows the histograms of the clustering quality metrics of QCluster on all datasets, varying on the k-mer length. Only non-trivial clusters are considered.

We can safely say that isONclust is not able to compute any meaningful clustering when the read length is 100 and the window size w is 50, as shown by the AMI, ARI, and Purity scores constantly at 0 in Figure 3. Only the `n-50000-fl-100` dataset is considered in the picture, but the plot for `n-10000-fl-100` and `n-20000-fl-200` is very similar. These are the only cases in which the clustering results are comparable to the randomized baseline clustering. The basically perfect Homogeneity score is due to the huge number of singletons found, since Homogeneity penalizes over-clustering (clustering together reads from originating sequences), which does not happen when a cluster contain a single sequence. Excluding singleton clusters from the analysis reveals a much different quality result on the same dataset (`n-50000-fl-100`): as Figure 4 shows, with $w = 50$, isONclust obtains very high V-Measure, a decent purity and a significant AMI score. We notice that the ARI score is still low, but since the clustering results in this case are strongly unbalanced, the AMI score is more relevant. Again, the results are similar for `n-10000-fl-100` and `n-20000-fl-200`, even though dataset `n-10000-fl-100` has more spikes and much better quality (V-Measure and Purity approaching 1 for $k \leq 7$, AMI and ARI approaching 0.8).

QCluster, on the other hand, presents less variability between the distance metrics considered, and for

[11] Link of the buggy isONclust function that throws index-out-of-bounds errors: https://github.com/ksahlin/isONclust/blob/master/modules/cluster.pyL17

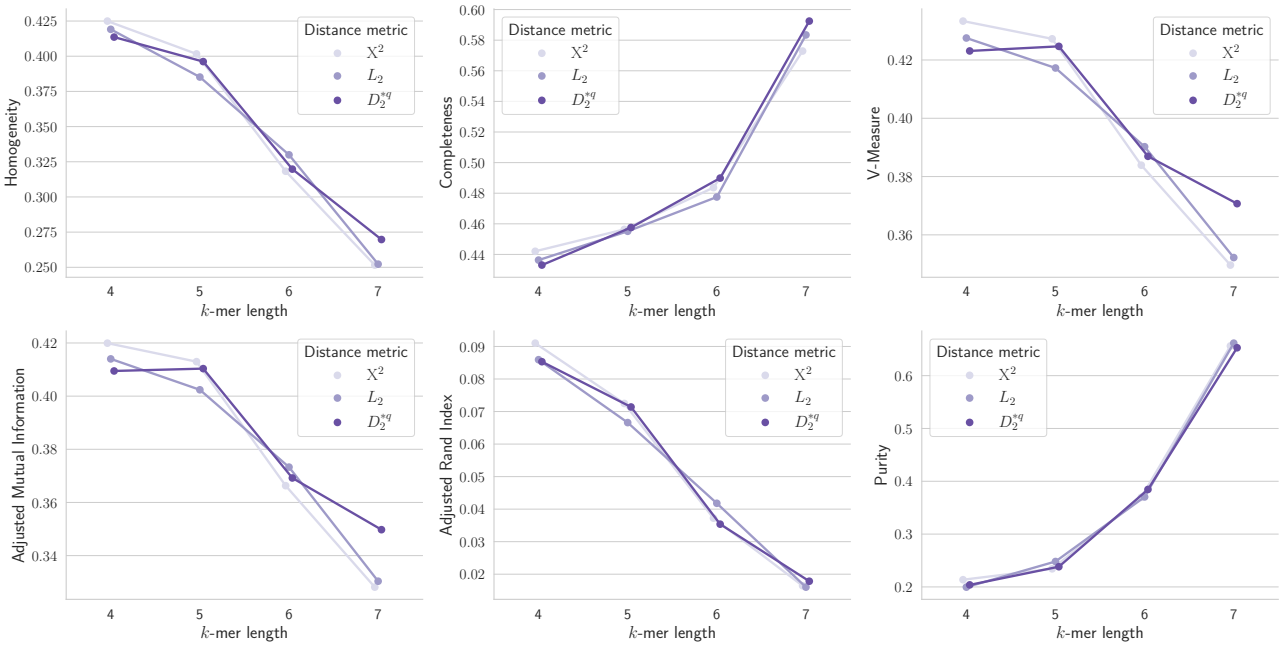


Figure 5 QCluster: Clustering quality metrics vs k-mer length distinguished by distance metric. Non-trivial clusters on dataset `n-50000-fl-700` are considered.

read length equal to 700, it also presents a much stronger, almost linear correlation between the value of k and the clustering quality, without much variation across datasets except for the absolute values, as shown in Figure 5. Even though the Completeness and Purity increase as k grows substantially, the Homogeneity plummets. The AMI and ARI scores decrease as well, although at a slower rate.

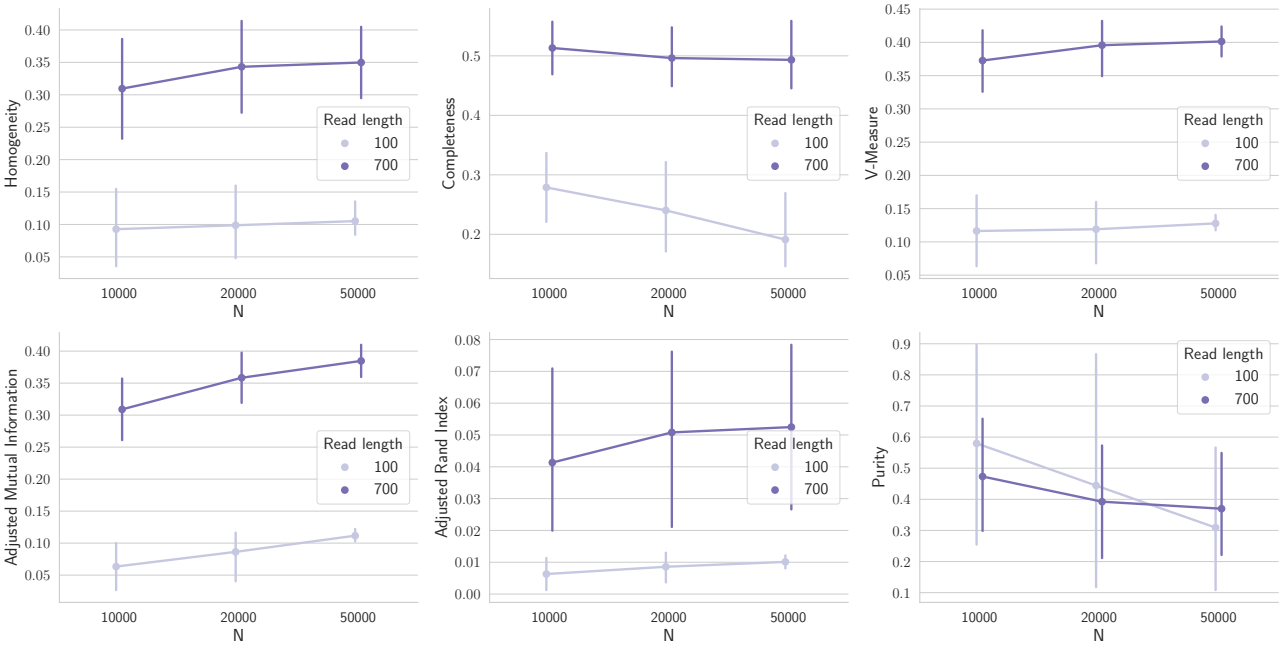
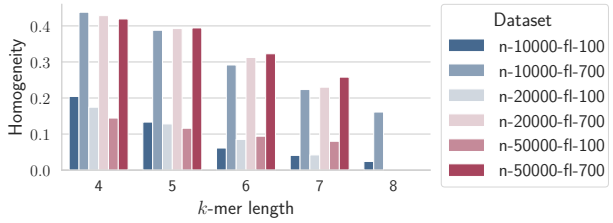
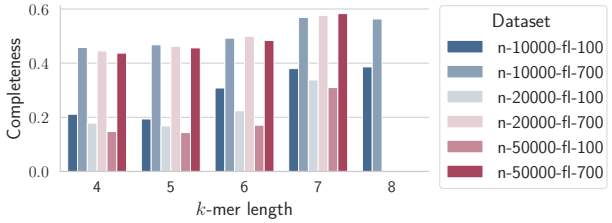


Figure 6 QCluster: Clustering quality metrics vs dataset size distinguished by read length of the dataset. Non-trivial clusters on dataset `n-50000-fl-700` with distance D_2^{*q} are considered.

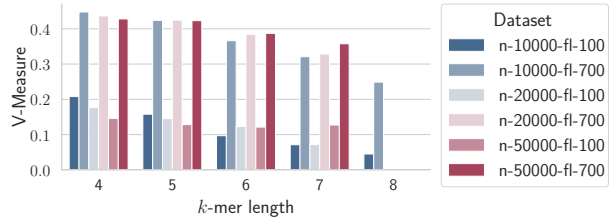
This pattern does not seem to be exclusively related to the impossibility to compute the clustering in some cases, because for Homogeneity, AMI, and ARI, the decreasing tendency starts between $k = 5$ and $k = 6$. We can gain more insights by differentiating by read length. We notice that clustering on datasets with read length equal to 700 is generally much better, as shown in Figure 6. However, by looking at the histograms of the quality metrics in Figure 7, the overall tendency seems the same: Homogeneity, AMI and ARI decrease as k grows, whereas Completeness and Purity increase. Purity reaches its best results with $k = 7$ and $k = 8$. This means that the clustering results that do not fail because of memory errors tend to have a few clusters that contain sequences originally simulated by the same class, and many clusters whose sequences are unrelated. The memory limitations of QCluster hinder our ability to formulate stronger hypotheses, but we conjecture that long enough reads (e.g. 1500bp or 3000 base pairs) would be able to counter-balance the negative effect that the k-mer length has on overall quality of the clustering.



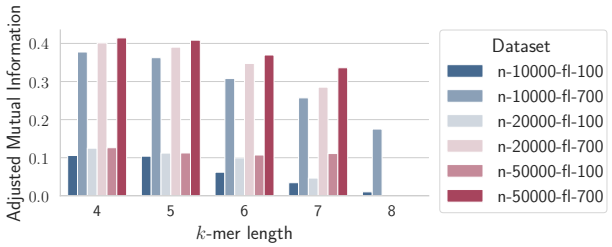
(a) Homogeneity vs k-mer length.



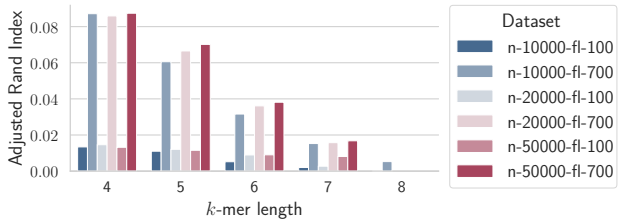
(b) Completeness vs k-mer length.



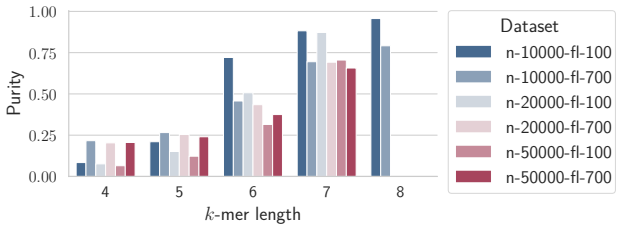
(c) V-Measure vs k-mer length.



(d) Adjusted Mutual Information (AMI) vs k-mer length.

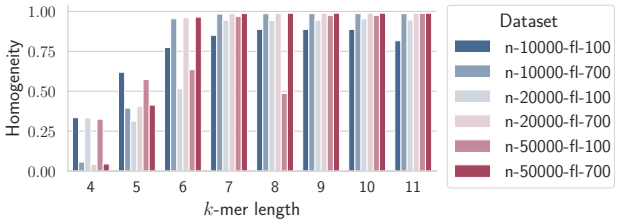


(e) Adjusted Rand Index (ARI) vs k-mer length.

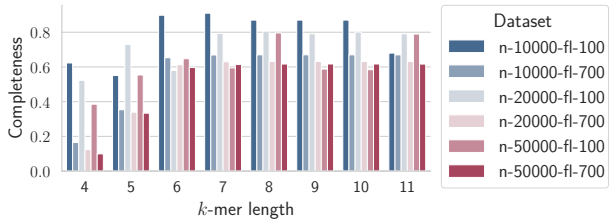


(f) Purity vs k-mer length.

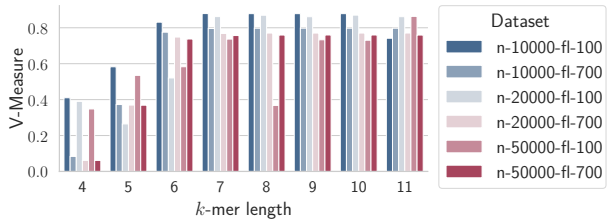
Figure 7 QCluster: Clustering quality metrics vs k-mer length. Only non-trivial clusters are considered. We can see that Completeness is generally higher than Homogeneity, and that higher values of k-mer length suggest lower Homogeneity, ARI and a slightly decrease of AMI. Purity, on the other hand, increases significantly reaching almost perfect quality for the smallest datasets with $k = 8$. The very low values for both ARI and AMI, however, indicate that when the k-mer length is relatively big and the dataset is relatively small, the clustering result is basically random.



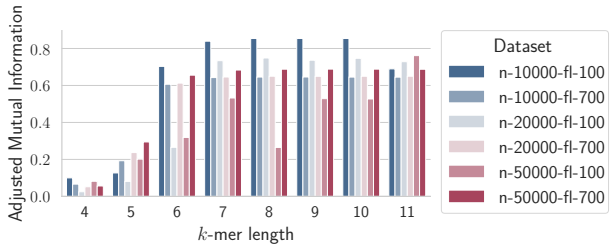
(a) Homogeneity vs k-mer length.



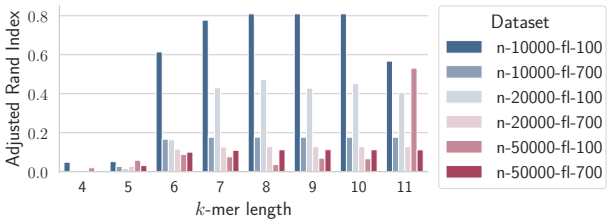
(b) Completeness vs k-mer length.



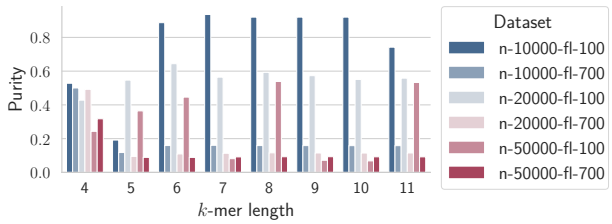
(c) V-Measure vs k-mer length.



(d) Adjusted Mutual Information (AMI) vs k-mer length.



(e) Adjusted Rand Index (ARI) vs k-mer length.

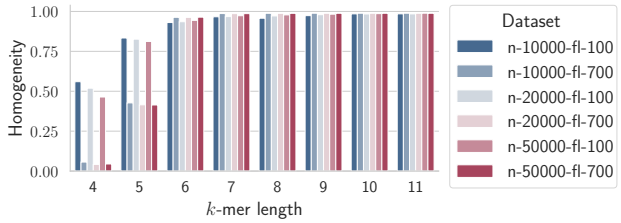


(f) Purity vs k-mer length.

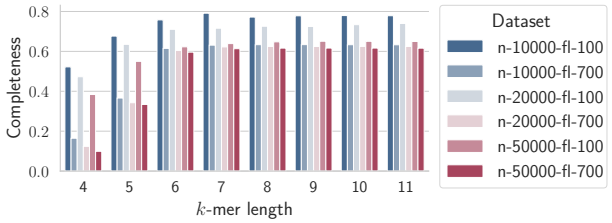
Figure 8 isONclust: Clustering quality metrics vs k-mer length. Only non-trivial clusters are considered. We can see that Homogeneity is much higher and stable than Completeness, reaching almost perfect quality for $k \geq 6$, and that higher values of k-mer length suggest an increase of almost every metric. Purity, on the other hand, does not have a clear distribution. The very low values for both ARI and AMI, for $k = 4, 5$ indicate the clustering result is basically random, but for $k \geq 6$, again, we witness a significant improvement of the quality. Moreover, we notice very high spikes for ARI and Purity for $k \geq 6$, but only for the datasets with original read length equal to 100, with the smallest dataset obtaining the best quality values.

7 Conclusions

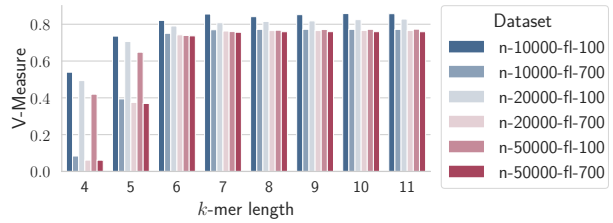
The experiments reported for this project have given empirical evidence that QCluster and isONclust generally perform meaningful clustering results, i.e. they have better quality than the randomized clustering. The only cases in which the clustering results tend to be random occur when using isONclust with window size close to the read length, i.e. $w = 50$ and read length equal to 100. We do not believe this is a substantial limitation of isONclust, as it is a tool mainly designed to cluster long reads. For what concerns QCluster's distance metrics, there is no clear winner. D_2^{*q}, L_2, and χ^2 give very similar results, with D_2^{*q} and χ^2 giving slightly better clustering results in the average case. D_2^{*q}, however, consumes less memory: it is the only distance metric with which QCluster is able to perform a clustering task on the `n-50000-fl-100` dataset with k-mer length equal to 7. isONclust, on the other hand, is extremely sensible to the window size w, as remarked in [SM19] by Sahlin. The author



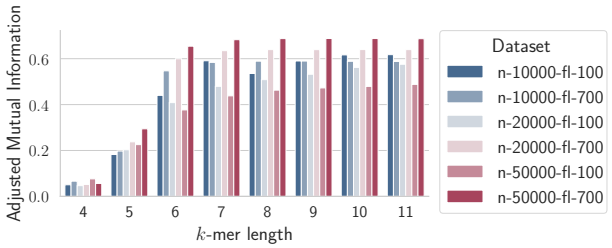
(a) Homogeneity vs k-mer length.



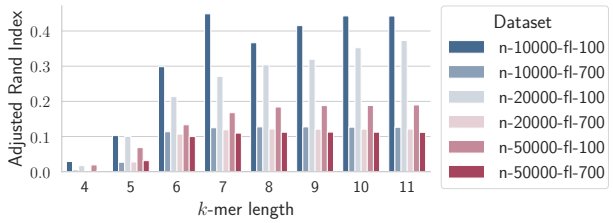
(b) Completeness vs k-mer length.



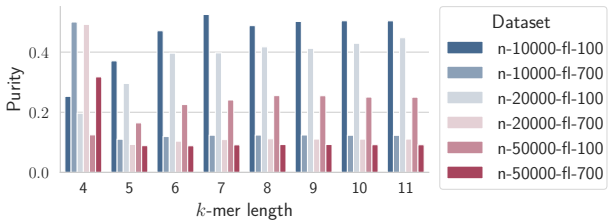
(c) V-Measure vs k-mer length.



(d) Adjusted Mutual Information (AMI) vs k-mer length.



(e) Adjusted Rand Index (ARI) vs k-mer length.

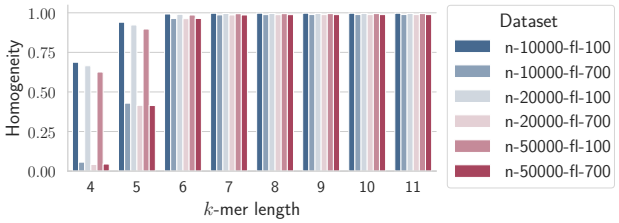


(f) Purity vs k-mer length.

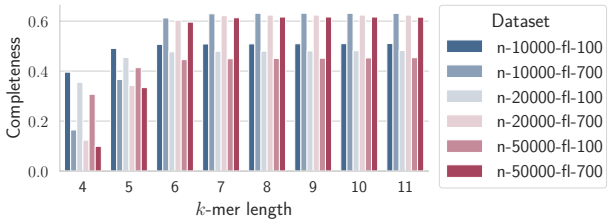
Figure 9 isONclust: Clustering quality metrics vs k-mer length. Only non-singleton clusters are considered. The quality metrics tend to follow the same patterns as the case with non-trivial clusters only. Both AMI and ARI, however, are lower, which suggests that the high quality of singletons may give a significant bias for the overall quality metrics of isONclust.

states that the proposed value of the window size (20 and 50) were merely determined by chance. With $w = 50$, isONclust only computes trivial clusterings for most of the datasets with read length equal to 100 with $k \geq 5$. For datasets with read length equal to 700, there is a higher ration of non-trivial clusters, and for $k \geq 5$ there is no significant difference in clustering quality between the two window sizes if we restrict ourselves to only non-trivial clusters. If we considers singletons and trivial clusters, however, the clustering quality is much higher, and with better results on datasets with bigger read length.

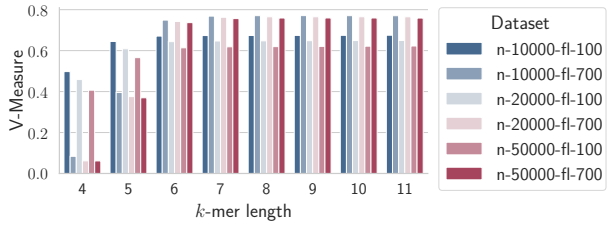
Our experiments confirmed that isONclust tends to have more fragmented clusters as the class size increases, as noted in [SM19]. The number of clusters determined by isONclust's greedy algorithm is much bigger than the ground truth number of clusters, even reaching 490 times more clusters than QCluster. The number of clusters found is limited by the size of our dataset of course, so we expect that for bigger datasets it would reach even more than the 50000 clusters found in some of our experiments. We did not have the possibility to test isONclust on long reads as Sahlin did in [SM19],



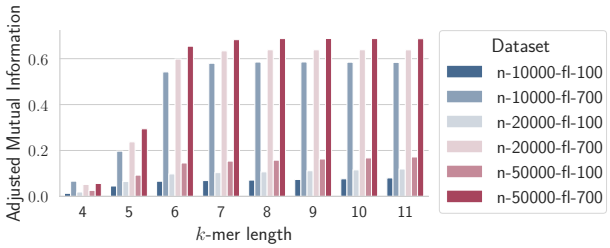
(a) Homogeneity vs k-mer length.



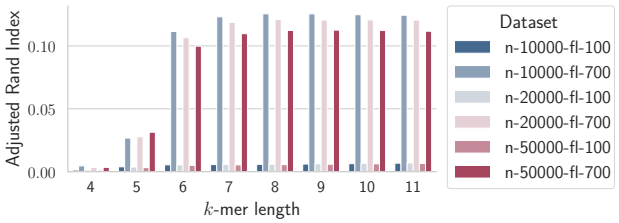
(b) Completeness vs k-mer length.



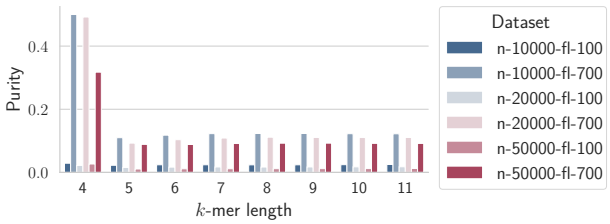
(c) V-Measure vs k-mer length.



(d) Adjusted Mutual Information (AMI) vs k-mer length.



(e) Adjusted Rand Index (ARI) vs k-mer length.



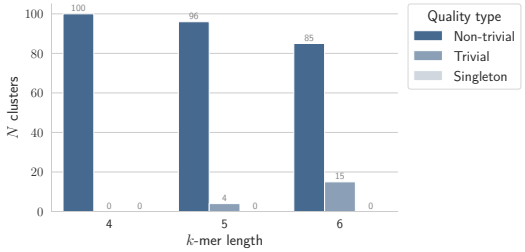
(f) Purity vs k-mer length.

Figure 10 isONclust: Clustering quality metrics vs k-mer length. All clusters are considered. We can see that Homogeneity is much higher and stable than Completeness, reaching almost perfect quality for $k \geq 6$, similarly to the experiment that considers all clusters. Higher values of k-mer length suggest an increase of almost every metric except Purity, which is significantly low. The very low values for both ARI and AMI, for $k = 4, 5$ indicate the clustering result is basically random, but for $k \geq 6$, again, we witness a significant improvement of the quality. The high Purity spikes that appeared in non-trivial clusters disappear completely, whereas the ARI and AMI have peaks for all the datasets with original read length equal to 700.

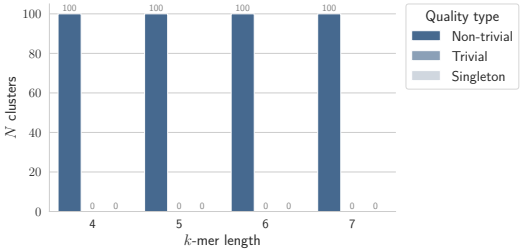
but we can state that isONclust is not suitable for clustering long reads because there are many cases in which more than 95% of the clusters are singletons, i.e. in which the clustering task fails. In the same paper, Sahlin *et al.* stress the fact that QCluster is impractical due to the enormous memory requirements. For k-mer length equal to 15 and a simulated dataset with 100000 sequences, QCluster causes memory segmentation faults after occupying more than 264GB of memory. Again, we were able to reproduce similar issues with smaller values of k and smaller datasets. Professor Comin had alerted us about the fast growing memory usage when incrementing k: this is the primary reason why we submitted a Pull Request to the isONclust repository[12] to support k-mer lengths smaller than 10, so to have some overlapping values of k to compare QCluster and isONclust. The frequent memory

[12] https://github.com/ksahlin/isONclust/pull/13

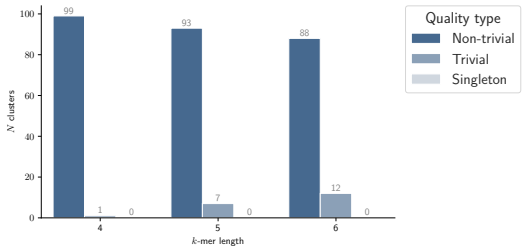
errors of QCluster limit our ability to formulate hypotheses, but we have indication that the bigger the k-mer length, the worse the clustering result is. After $k \geq 6$, fewer clusters correctly partition the sequences, and more clusters are composed of mixed sequences coming from different genes. We suppose that higher read lengths and multiple runs of the K-Means algorithm could mitigate this problem.



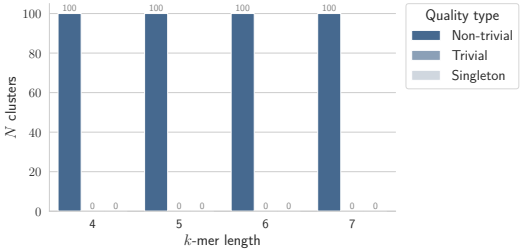
(a) Number of clusters found with respect to k-mer length when using L_2 as a distance metric with 100 as original read length.



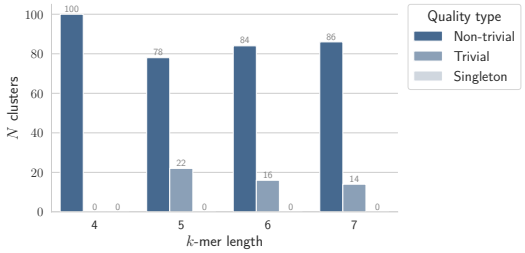
(b) Number of clusters found with respect to k-mer length when using L_2 as a distance metric with 700 as original read length.



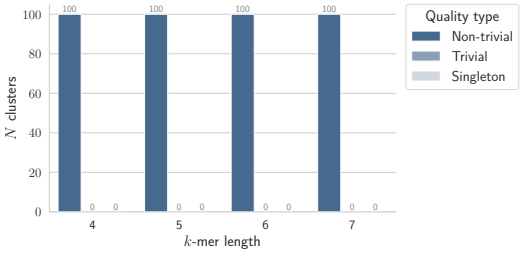
(c) Number of clusters found with respect to k-mer length when using χ^2 as a distance metric with 100 as original read length.



(d) Number of clusters found with respect to k-mer length when using χ^2 as a distance metric with 700 as original read length.

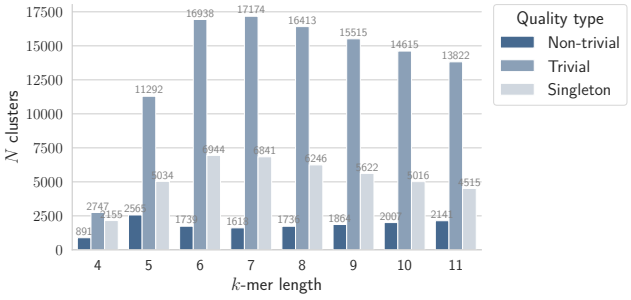


(e) Number of clusters found with respect to k-mer length when using D_2^{*q} as a distance metric with 100 as original read length.

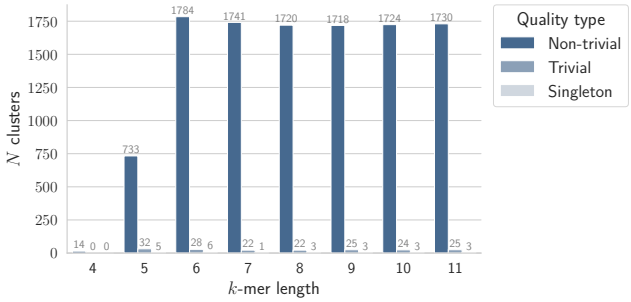


(f) Number of clusters found with respect to k-mer length when using D_2^{*q} as a distance metric with 700 as original read length.

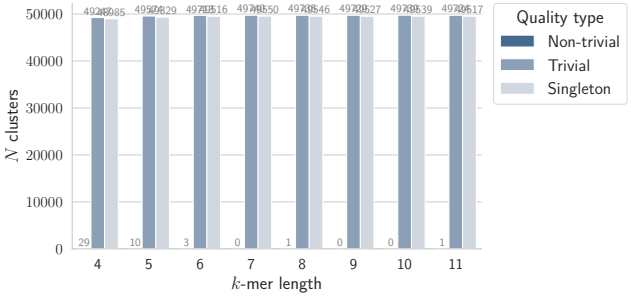
Figure 11 QCluster: Number of clusters found varying the distance metric and the k-mer length when using datasets with 50000 reads. We can observe that QCluster consistently finds many more non-trivial clusters rather than trivial clusters and that no cluster found is a singleton. With original read length equal to 700, all clusters found are non-trivial. With the L_2 and χ^2 distance metrics, the number of trivial clusters seems to increase as the k-mer length grows. D_2^{*q} presents what seems an opposite trend starting from $k = 5$.



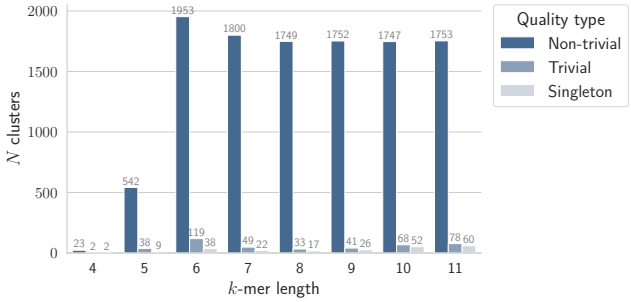
(a) Number of clusters found with respect to k-mer length when using window size equal to 20 with 100 as original read length.



(b) Number of clusters found with respect to k-mer length when using window size equal to 20 with 700 as original read length.



(c) Number of clusters found with respect to k-mer length when using window size equal to 50 with 100 as original read length.



(d) Number of clusters found with respect to k-mer length when using window size equal to 50 with 700 as original read length.

Figure 12 isONclust: Number of clusters found varying the window size and the k-mer length when using datasets with 50000 reads. We notice that generally, the number of clusters found are orders of magnitude bigger than the number of originating sequences (100), i.e. the original clusters. The most degenerate case arises with window size equal to 50 and original read lengths equal to 100, where almost every simulated read is partitioned into a singleton cluster. On datasets whose read length is 700, isONclust consistently finds less clusters (less than 2000), with very few trivial or singleton clusters. When combining the smallest simulated read length with the smallest window size, the number of clusters is somewhat in the middle of the two extremes, but with few non-trivial clusters and still with many more clusters than QCluster. If we don't consider $k \leq 5$, the k-mer length does not seem to influence the number of clusters found.

We did not manage to accurately record the absolute time and memory requirements of isONclust and QCluster, but all the clustering analysis tasks we ran on isONclust took about 4 hours in total without consuming more than 8GB of memory, whereas the experiments on QCluster took about 54 hours, with many runs failing because of memory errors. We observed peaks of 28GB of memory consumption (physical + virtual RAM).

We performed extensive comparative experiments on six simulated datasets, considering $18 \times 6 = 108$ parameter combinations for QCluster and $16 \times 6 = 96$ parameter combinations for isONclust. We created and examined over 100 visualization plots. We elaborated the `CSV` files created during the quality assessment phase to produce dozens of tables, of which we reported more than 30 between the main document and the appendixes. We invite the reader to refer to the `IPython` notebook `analysis.ipynb` in the project repository for reviewing additional plots and tables.

We published the code used for these experiments in a publicly available Github repository[13]. For instructions on how to reproduce our experiments, please first download the human cDNA assembly as stated in Section 4 and then follow the instructions in Appendix E.

7.1 Future Work

We identified four possible areas of improvement for this comparative report.

1. Increase the number of sequences used to create the simulated datasets.
2. Increase the sequence-length variance in the sequences used to generate the simulated datasets. In this work, we decided to keep the sequence-length variance stable with respect to a standard-deviation threshold to keep the obtained clusters as similar in cardinality as possible, thus making it easier to compare the clustering results. However, changing this part of the setup might provide better insights into the behavioral differences between isONclust and QCluster.
3. Include real datasets for which the ground-truth is not known. Doing so would remove the bias we introduced in QCluster's evaluation when we explicitly provided the actual number of clusters as one of its parameters. However, it would require a more complex preliminary alignment step as noted in [SM19].
4. Add other clustering tools for long-read sequences to the comparison, and test how these other tools can be ranked with respect to isONclust and QCluster. [SM19], for instance, mentions also the software tools linclust [SS18] and PacBio's Isoseq3-cluster.
5. We could dynamically adapt the threshold t for which we consider a clustering result trivial according to the size of the dataset.

7.2 Personal Suggestions to the Software Maintainers

To help future students or other newcomers to the bioinformatics research field, we would like to give some suggestions to the maintainers of the main tools we have used for this project.

QCluster

- Rewrite the software with portability in mind. Currently, it is impossible to build and run QCluster on a Windows platform without using WSL (Windows Subsystem for Linux).
- Provide parallel and possibly distributed support. As of January 2021, QCluster is much slower than isONclust, and it has a huge memory consumption. In fact, the memory consumption is so high that even with relative small values of the `"-k"` parameter (which regulates the k-mer length), it often fails with a segmentation fault error. We suspect that these bad memory management is caused by the original `afcluster` implementation QCluster is based on. Our suggestion is to rewrite QCluster from scratch using the parallel K-Means++[AV07] implementation that the Apache Spark[ZXW+16] framework provides, potentially in a distributed context. Apache Spark officially supports the `Java 8+`, `Scala`, and `Python3` languages. While `Python` would most likely offer the worst run time performance, it is probably the best option because of third-party `Python` libraries' mature ecosystem for manipulating `FASTQ` files, like the `BioPython` library we used.
- Add some documentation about the code structure to motivate open-source developers to contribute to the code.

[13] https://github.com/jkomyno/bioalgo-QCluster-vs-isONclust

isONclust

- Give the possibility to provide the number of desired clusters as input parameter. Doing so would be probably quite impacting on the current greedy algorithm implemented in isONclust, but it is our opinion that it would be more helpful in practice. Moreover, it would have been advantageous to better compare isONclust to QCluster, whose clustering result is guided by the number of desired clusters that the user imposes.
- Only a restricted number of k-mer lengths is supported. We did not expect this when we started performing our experiments. Luckily, thanks to Sahlin's inputs, we have been able to extend the k-mer support from the initial $[10, 31]$ length range to $[4, 31]$. This has been essential for our comparison with the QCluster tool since with $k \geq 7$, QCluster tends to throw memory errors.
- Lock the third-party dependency versions in a `Python requirements.txt` file. Currently, the documentation simply asks to update third-party libraries whose version is greater than or equal to a specified version. This can be a source of install errors due to sudden third-party library updates that either break the previous API or build system. In fact, we have struggled to install the `"parasail"` dependency with a version greater than `1.1.10`. The latest version at the time of installation, `v1.2.2`, had in fact broken the build system compatibility with `v1.2.1`. After we have signaled the problem with a Github issue[14], a new compatible version was released by the parasail's author that allowed us to use isONclust.
- Introduce unit and integration testing to easily spot bugs like the one we encountered.

NanoSim

- Properly support the `FASTQ` format. As of January 2021, the reads simulated by NanoSim are not fully compliant with the `FASTQ` format, as determined using the `fqtools`[Dro16] tool suite.
- Provide better documentation for beginners. In particular, we would have appreciated it if there was some documentation on how to retrieve the required `FASTA`, `SAM`, and `TSV` files for a sample gene assembly.
- Provide faster technical support. Even though we understand this would not be an easy task, the delays of the NanoSim team's reply to the Github issue we opened forced us to switch to a different simulator, SimLoRD.
- Correct the installation tutorial. We have even proposed a *Pull Request*[15] on NanoSim's Github repository to fix the problem partially, but we have had no feedback as of January 2021.

SimLoRD

- Lock the third-party dependency versions in a `Python requirements.txt` file. Currently, the library's `README.md` file asks to *pip install* some dependencies. Although we did not have any issue using SimLoRD, locking the dependencies is advisable, as already noted in the suggestions for isONclust.

Acknowledgment

We hereby would like to thank Professor **Matteo Comin**[16] from the Department of Information Engineering of the University of Padova for his patient assistance and guidance for this project.
We would also like to thank Assistant Professor **Kristoffer Sahlin**[17] from the Department of Mathematics at Stockholm University for his helpful support about our questions and doubts on the isONclust software.

[14] https://github.com/jeffdaily/parasail-python/issues/57
[15] https://github.com/bcgsc/NanoSim/pull/103
[16] https://scholar.google.com/citations?user=HXG7EpYAAAAJ&hl=en
[17] https://scholar.google.com/citations?user=IAGyYNEAAAAJ&hl=en

References

AGM+90 Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

AV07 David Arthur and Sergei Vassilvitskii. K-means++: the advantages of careful seeding. In *In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.

CAC+09 P. J. A. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, and M. J. L. de Hoon. Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, March 2009.

CLS15 Matteo Comin, Andrea Leoni, and Michele Schimd. Clustering of reads with alignment-free measures and quality values. *Algorithms for Molecular Biology*, 10(1), January 2015.

Dro16 Alastair P. Droop. fqtools: an efficient software suite for modern FASTQ file manipulation. *Bioinformatics*, 32(12):1883–1884, February 2016.

Dun74 J. C. Dunn†. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, January 1974.

Got82 Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, December 1982.

HA85 Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, December 1985.

HR07 Julia Bell Hirschberg and Andrew Rosenberg. V-measure: A conditional entropy-based external cluster evaluation. 2007.

LJG01 W. Li, L. Jaroszewski, and A. Godzik. Clustering of highly homologous sequences to reduce the size of large protein databases. *Bioinformatics*, 17(3):282–283, March 2001.

Llo82 S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

MB02 U. Maulik and S. Bandyopadhyay. Performance evaluation of some clustering algorithms and validity indices. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(12):1650–1654, December 2002.

MSP+08 Duccio Medini, Davide Serruto, Julian Parkhill, David A. Relman, Claudio Donati, Richard Moxon, Stanley Falkow, and Rino Rappuoli. Microbiology in the post-genomic era. *Nature Reviews Microbiology*, 6(6):419–430, May 2008.

NW70 Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

pdt20 The pandas development team. pandas-dev/pandas: Pandas, February 2020.

PG07 Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.

QiHM09 W. Qu, S. i. Hashimoto, and S. Morishita. Efficient frequency-based de novo short-read clustering for error trimming in next-generation sequencing. *Genome Research*, 19(7):1309–1315, May 2009.

Ran71 William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

Rou87 Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65, 1987.

Sar90 Warren S. Sarle. Algorithms for clustering data. *Technometrics*, 32(2):227–229, 1990.

SKR16 Bianca K. Stöcker, Johannes Köster, and Sven Rahmann. SimLoRD: Simulation of long read data. *Bioinformatics*, 32(17):2704–2706, May 2016.

SL13 Alexander Solovyov and W Ian Lipkin. Centroid based clustering of high throughput sequencing reads based on n-mer counts. *BMC Bioinformatics*, 14(1), September 2013.

SM19 Kristoffer Sahlin and Paul Medvedev. De novo clustering of long-read transcriptome data using a greedy, quality-value based algorithm. In *Lecture Notes in Computer Science*, pages 227–242. Springer International Publishing, 2019.

SS18 Martin Steinegger and Johannes Söding. Clustering huge protein sequence sets in linear time. *Nature Communications*, 9(1), June 2018.

SW81 T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, March 1981.

TSK05 Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison Wesley, us ed edition, May 2005.

VA03 S. Vinga and J. Almeida. Alignment-free sequence comparison–a review. *Bioinformatics*, 19(4):513–523, March 2003.

VEB10 Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(95):2837–2854, 2010.

Wtsdt20 Michael Waskom and the seaborn development team. mwaskom/seaborn, September 2020.

ZXW+16 Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark. *Communications of the ACM*, 59(11):56–65, October 2016.

A Tables: Number of Clusters

		Read length=100			Read length=700		
k	**Distance**	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		# clusters			# clusters		
4	D_2^{*q}	98	98	**100**	**100**	99	**100**
	L_2	**100**	99	**100**	99	**100**	**100**
	X^2	98	**100**	99	99	**100**	**100**
5	D_2^{*q}	44	62	78	99	**100**	**100**
	L_2	46	67	96	98	**100**	**100**
	X^2	34	65	93	99	**100**	**100**
6	D_2^{*q}	3	21	84	97	**100**	**100**
	L_2	16	27	85	96	99	**100**
	X^2	6	34	88	98	**100**	**100**
7	D_2^{*q}	7	15	86	96	99	**100**
	L_2	6	23	0	96	**100**	**100**
	X^2	8	25	0	93	**100**	**100**
8	D_2^{*q}	4	0	0	92	0	0
	L_2	5	0	0	92	0	0
	X^2	7	0	0	95	0	0

Table 5 QCluster: Number of non-trivial clusters. Highest results are in bold.

		Read length=100			Read length=700		
k	**Distance**	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		# clusters			# clusters		
4	D_2^{*q}	2	2	0	0	**1**	0
	L_2	0	1	0	1	0	0
	X^2	2	0	1	1	0	0
5	D_2^{*q}	56	38	**22**	1	0	0
	L_2	54	33	4	2	0	0
	X^2	66	35	7	1	0	0
6	D_2^{*q}	**97**	79	16	3	0	0
	L_2	84	73	15	4	**1**	0
	X^2	94	66	12	2	0	0
7	D_2^{*q}	93	**85**	14	4	**1**	0
	L_2	94	77	0	4	0	0
	X^2	92	75	0	7	0	0
8	D_2^{*q}	96	0	0	**8**	0	0
	L_2	95	0	0	**8**	0	0
	X^2	93	0	0	5	0	0

Table 6 QCluster: Number of trivial clusters. Highest results are in bold.

		Read length=100			Read length=700		
k	Distance	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		# clusters			# clusters		
4	D_2^{*q}	1	0	0	0	0	0
	L_2	0	0	0	0	0	0
	X^2	1	0	0	0	0	0
5	D_2^{*q}	11	5	0	0	0	0
	L_2	13	3	0	0	0	0
	X^2	20	1	0	0	0	0
6	D_2^{*q}	25	15	0	0	0	0
	L_2	27	11	0	0	0	0
	X^2	38	8	0	0	0	0
7	D_2^{*q}	27	13	0	0	0	0
	L_2	29	10	0	0	0	0
	X^2	33	**18**	0	**1**	0	0
8	D_2^{*q}	42	0	0	0	0	0
	L_2	35	0	0	0	0	0
	X^2	**47**	0	0	**1**	0	0

Table 7 QCluster: Number of singleton clusters. Highest results are in bold.

		Read length=100			Read length=700		
k	Window	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		# clusters			# clusters		
4	20	**310**	527	891	7	11	14
	50	3	7	29	19	15	23
5	20	175	**564**	**2565**	369	591	733
	50	0	1	10	285	417	542
6	20	10	102	1739	883	**1462**	**1784**
	50	0	2	3	800	1497	1953
7	20	5	85	1618	883	1429	1741
	50	0	0	0	866	1475	1800
8	20	4	77	1736	886	1434	1720
	50	0	0	1	879	1436	1749
9	20	4	83	1864	**894**	1443	1718
	50	0	0	0	880	1443	1752
10	20	4	96	2007	893	1446	1724
	50	0	0	0	879	1437	1747
11	20	5	103	2141	887	1435	1730
	50	0	0	1	882	1442	1753

Table 8 isONclust: Number of non-trivial clusters. Highest results are in bold.

		Read length=100			Read length=700		
k	**Window**	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		# clusters			# clusters		
4	20	975	1548	2747	0	0	0
	50	9873	19772	49247	1	1	2
5	20	5478	8238	11292	240	119	32
	50	9947	19891	49574	182	95	38
6	20	7390	11781	16938	687	245	28
	50	9978	19929	49713	**922**	**405**	**119**
7	20	7564	11958	17174	665	220	22
	50	9986	19939	**49740**	725	248	49
8	20	7501	11706	16413	655	216	22
	50	9986	19953	49735	678	232	33
9	20	7408	11375	15515	647	212	25
	50	**9987**	**19955**	49729	681	227	41
10	20	7315	11096	14615	652	208	24
	50	9986	**19955**	49739	688	238	68
11	20	7188	10808	13822	663	220	25
	50	9986	19954	49724	696	255	78

Table 9 isONclust: Number of trivial clusters. Highest results are in bold.

		Read length=100			Read length=700		
k	**Window**	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		# clusters			# clusters		
4	20	607	1048	2155	0	0	0
	50	9796	19666	48985	0	0	2
5	20	3465	4459	5034	37	18	5
	50	9900	19809	49329	35	14	9
6	20	5462	6972	6944	74	40	6
	50	9957	19875	49516	**137**	**68**	38
7	20	5685	7121	6841	77	29	1
	50	9972	19883	**49550**	78	44	22
8	20	5567	6758	6246	78	27	3
	50	9972	19908	49546	82	38	17
9	20	5420	6344	5622	78	27	3
	50	**9974**	**19912**	49527	86	40	26
10	20	5276	5959	5016	74	28	3
	50	9972	**19912**	49539	86	50	52
11	20	5091	5613	4515	75	29	3
	50	9972	19910	49517	90	63	**60**

Table 10 isONclust: Number of singleton clusters. Highest results are in bold.

B Tables: Clustering statistics

k	Distance	Average size	Standard dev	Min size	Max size
4	D_2^{*q}	100	54.0293	1	270
	L_2	100	43.9898	**9**	249
	X^2	100	60.8411	1	307
5	D_2^{*q}	100	218.6498	1	1345
	L_2	100	210.2220	1	1293
	X^2	100	246.6189	1	1493
6	D_2^{*q}	100	948.8892	1	9540
	L_2	100	471.1660	1	3651
	X^2	100	669.4340	1	4867
7	D_2^{*q}	100	679.1839	1	5687
	L_2	100	961.3436	1	9665
	X^2	100	952.4572	1	9576
8	D_2^{*q}	100	922.3633	1	9266
	L_2	100	**968.0617**	1	**9732**
	X^2	100	964.7635	1	9699

Table 11 QCluster: Cluster statistics for dataset `n-10000-fl-100`. The number of clusters obtained is omitted as with QCluster it is always 100. Highest results are in bold.

k	Distance	Average size	Standard dev	Min size	Max size
4	D_2^{*q}	100	59.8286	**10**	245
	L_2	100	65.3122	5	310
	X^2	100	62.0055	4	257
5	D_2^{*q}	100	152.3263	5	792
	L_2	100	135.5101	4	608
	X^2	100	164.7027	3	834
6	D_2^{*q}	100	343.9199	4	2251
	L_2	100	360.2570	3	2463
	X^2	100	301.2096	4	1763
7	D_2^{*q}	100	569.0243	4	4325
	L_2	100	547.0517	5	4022
	X^2	100	567.6344	1	4194
8	D_2^{*q}	100	587.3750	4	4863
	L_2	100	817.5952	4	8225
	X^2	100	**832.8355**	1	**8380**

Table 12 QCluster: Cluster statistics for dataset `n-10000-fl-700`. The number of clusters obtained is omitted as with QCluster it is always 100. Highest results are in bold.

k	Distance	Average size	Standard dev	Min size	Max size
4	D_2^{*q}	200	89.2886	5	428
	L_2	200	85.1845	2	448
	X^2	200	79.2929	**8**	353
5	D_2^{*q}	200	361.5623	1	1932
	L_2	200	341.1279	1	1655
	X^2	200	320.1719	1	1624
6	D_2^{*q}	200	1103.6785	1	8433
	L_2	200	884.4835	1	6943
	X^2	200	913.0564	1	6287
7	D_2^{*q}	200	**1913.9370**	1	**19242**
	L_2	200	1877.8720	1	18881
	X^2	200	1287.4149	1	9450

Table 13 QCluster: Cluster statistics for dataset `n-20000-fl-100`. The number of clusters obtained is omitted as with QCluster it is always 100. Highest results are in bold.

k	Distance	Average size	Standard dev	Min size	Max size
4	D_2^{*q}	200	112.8575	4	551
	L_2	200	120.8816	14	604
	X^2	200	105.9409	**15**	422
5	D_2^{*q}	200	250.4778	14	1356
	L_2	200	267.8684	10	1173
	X^2	200	291.9678	10	1371
6	D_2^{*q}	200	641.8192	13	3639
	L_2	200	624.9294	5	3975
	X^2	200	582.1593	13	3652
7	D_2^{*q}	200	1095.2992	4	8379
	L_2	200	**1159.9754**	7	**8863**
	X^2	200	1100.4152	7	8439

Table 14 QCluster: Cluster statistics for dataset `n-20000-fl-700`. The number of clusters obtained is omitted as with QCluster it is always 100. Highest results are in bold.

k	Distance	Average size	Standard dev	Min size	Max size
4	D_2^{*q}	500	202.8350	**51**	965
	L_2	500	198.4981	19	989
	X^2	500	218.1591	5	883
5	D_2^{*q}	500	884.4347	2	4944
	L_2	500	589.9571	4	2684
	X^2	500	662.0350	3	3455
6	D_2^{*q}	500	1695.0348	2	13512
	L_2	500	1600.6332	2	10752
	X^2	500	1674.8195	2	12811
7	D_2^{*q}	500	3190.8458	2	**22825**
	L_2	500	1159.9754	7	8863
	X^2	500	**1100.4152**	7	8439

Table 15 QCluster: Cluster statistics for dataset `n-50000-fl-100`. The number of clusters obtained is omitted as with QCluster it is always 100. Highest results are in bold.

k	Distance	Average size	Standard dev	Min size	Max size
4	D_2^{*q}	500	309.0766	39	1215
	L_2	500	288.4667	39	1382
	X^2	500	287.9389	39	1108
5	D_2^{*q}	500	610.8591	**42**	2912
	L_2	500	675.1176	35	2941
	X^2	500	579.7439	39	2549
6	D_2^{*q}	500	1452.0380	26	7979
	L_2	500	1265.0766	36	8134
	X^2	500	1377.8241	39	7679
7	D_2^{*q}	500	2611.6191	**42**	19858
	L_2	500	**2700.2961**	33	**20355**
	X^2	500	2663.2347	24	20309

Table 16 QCluster: Cluster statistics for dataset `n-50000-fl-700`. The number of clusters obtained is omitted as with QCluster it is always 100. Highest results are in bold.

k	Window	N clusters	Average size	Standard dev	Min size	Max size
4	20	1285	**7.7821**	**19.3090**	1	**305**
	50	9876	1.0126	0.1773	1	8
5	20	5653	1.7690	1.4498	1	18
	50	9947	1.0053	0.0819	1	4
6	20	7400	1.3514	0.6834	1	7
	50	9978	1.0022	0.0490	1	3
7	20	7569	1.3212	0.6334	1	7
	50	9986	1.0014	0.0374	1	2
8	20	7505	1.3324	0.6377	1	7
	50	9986	1.0014	0.0374	1	2
9	20	7412	1.3492	0.6526	1	7
	50	**9987**	1.0013	0.0361	1	2
10	20	7319	1.3663	0.6718	1	7
	50	9986	1.0014	0.0374	1	2
11	20	7193	1.3902	0.6991	1	7
	50	9986	1.0014	0.0374	1	2

Table 17 ISONCLUST: Cluster statistics for dataset `n-10000-fl-100`. Highest results are in bold.

k	Window	N clusters	Average size	Standard dev	Min size	Max size
4	20	7	**1428.5714**	**1500.8944**	**60**	**4545**
	50	20	500.0000	1041.9107	3	3878
5	20	609	16.4204	33.5948	1	524
	50	467	21.4133	47.6561	1	586
6	20	1570	6.3694	3.4723	1	26
	50	**1722**	5.8072	3.7119	1	29
7	20	1548	6.4599	3.4807	1	26
	50	1591	6.2854	3.4172	1	25
8	20	1541	6.4893	3.4638	1	26
	50	1557	6.4226	3.4922	1	26
9	20	1541	6.4893	3.4687	1	26
	50	1561	6.4061	3.5011	1	26
10	20	1545	6.4725	3.4385	1	22
	50	1567	6.3816	3.4552	1	21
11	20	1550	6.4516	3.4503	1	22
	50	1578	6.3371	3.4305	1	21

Table 18 isONclust: Cluster statistics for dataset `n-10000-fl-700`. Highest results are in bold.

k	Window	N clusters	Average size	Standard dev	Min size	Max size
4	20	2075	**9.6386**	**28.6733**	1	**573**
	50	19779	1.0112	0.2155	1	17
5	20	8802	2.2722	2.3052	1	54
	50	19892	1.0054	0.0976	1	7
6	20	11883	1.6831	1.0372	1	13
	50	19931	1.0035	0.0772	1	6
7	20	12043	1.6607	0.9988	1	9
	50	19939	1.0031	0.0621	1	5
8	20	11783	1.6974	1.0176	1	9
	50	19953	1.0024	0.0505	1	3
9	20	11458	1.7455	1.0544	1	9
	50	**19955**	1.0023	0.0495	1	3
10	20	11192	1.7870	1.0788	1	10
	50	**19955**	1.0023	0.0495	1	3
11	20	10911	1.8330	1.1086	1	10
	50	19954	1.0023	0.0500	1	3

Table 19 isONclust: Cluster statistics for dataset `n-20000-fl-100`. Highest results are in bold.

k	Window	N clusters	Average size	Standard dev	Min size	Max size
4	20	11	**1818.1818**	**3603.9764**	**30**	**12668**
	50	16	1250.0000	1623.3701	3	5460
5	20	710	28.1690	61.6790	1	1089
	50	512	39.0625	69.4301	1	553
6	20	1707	11.7165	5.9338	1	41
	50	**1902**	10.5152	6.3093	1	59
7	20	1649	12.1286	5.9733	1	44
	50	1723	11.6077	5.9077	1	45
8	20	1650	12.1212	5.9957	1	49
	50	1668	11.9904	6.0426	1	46
9	20	1655	12.0846	5.9224	1	49
	50	1670	11.9760	5.9769	1	44
10	20	1654	12.0919	5.9076	1	49
	50	1675	11.9403	6.0299	1	44
11	20	1655	12.0846	5.9777	1	47
	50	1697	11.7855	6.0676	1	43

Table 20 ISONCLUST: Cluster statistics for dataset `n-20000-fl-700`. Highest results are in bold.

k	Window	N clusters	Average size	Standard dev	Min size	Max size
4	20	3638	**13.7438**	**68.4371**	**1**	**2235**
	50	49276	1.0147	0.3842	1	52
5	20	13857	3.6083	4.2006	1	81
	50	49584	1.0084	0.1605	1	19
6	20	18677	2.6771	1.9516	1	22
	50	49716	1.0057	0.1064	1	9
7	20	18792	2.6607	1.8763	1	15
	50	**49740**	1.0052	0.0922	1	4
8	20	18149	2.7550	1.9138	1	15
	50	49736	1.0053	0.0958	1	6
9	20	17379	2.8770	1.9792	1	19
	50	49729	1.0054	0.0940	1	5
10	20	16622	3.0081	2.0413	1	19
	50	49739	1.0052	0.0903	1	5
11	20	15963	3.1322	2.0988	1	19
	50	49725	1.0055	0.0959	1	7

Table 21 ISONCLUST: Cluster statistics for dataset `n-50000-fl-100`. Highest results are in bold.

k	Window	N clusters	Average size	Standard dev	Min size	Max size
4	20	14	**3571.4286**	**4402.0188**	**88**	**15682**
	50	25	2000.0000	3317.7055	1	14478
5	20	765	65.3595	110.6593	1	1227
	50	580	86.2069	192.6688	1	2886
6	20	1812	27.5938	11.7290	1	110
	50	**2072**	24.1313	13.0646	1	103
7	20	1763	28.3607	11.7466	1	104
	50	1849	27.0416	12.0481	1	101
8	20	1742	28.7026	11.9835	1	105
	50	1782	28.0584	12.1821	1	100
9	20	1743	28.6862	12.0719	1	105
	50	1793	27.8862	12.3472	1	100
10	20	1748	28.6041	12.0108	1	105
	50	1815	27.5482	12.5436	1	100
11	20	1755	28.4900	11.9013	1	102
	50	1831	27.3075	12.4934	1	102

Table 22 isONclust: Cluster statistics for dataset `n-50000-fl-700`. Highest results are in bold.

C Tables: Clustering Metrics

		Read length=100			Read length=700		
k	Distance	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		Homogeneity			Homogeneity		
4	D_2^{*q}	0.2031	0.1718	**0.1447**	**0.4401**	0.4282	0.4136
	L_2	**0.2073**	0.1751	0.1441	0.4367	0.4281	0.4191
	χ^2	0.2019	**0.1759**	0.1441	0.4363	**0.4290**	**0.4250**
5	D_2^{*q}	0.1245	0.1180	0.1062	0.3888	0.3997	0.3962
	L_2	0.1249	0.1279	0.1217	0.3993	0.3901	0.3852
	χ^2	0.1128	0.1236	0.1178	0.3745	0.3880	0.4016
6	D_2^{*q}	0.0047	0.0659	0.0870	0.2868	0.3045	0.3198
	L_2	0.0690	0.0765	0.0964	0.2807	0.3154	0.3299
	χ^2	0.0518	0.0824	0.0954	0.3045	0.3169	0.3183
7	D_2^{*q}	0.0490	0.0099	0.0789	0.2198	0.2400	0.2697
	L_2	0.0051	0.0167	-	0.2319	0.2161	0.2522
	χ^2	0.0092	0.0657	-	0.2138	0.2337	0.2516
8	D_2^{*q}	0.0071	-	-	0.2065	-	-
	L_2	0.0037	-	-	0.1333	-	-
	χ^2	0.0056	-	-	0.1359	-	-

Table 23 QCluster: Homogeneity metric values of non-trivial clusters. Best results are in bold.

		Read length=100			Read length=700		
k	Distance	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		Completeness			Completeness		
4	D_2^{*q}	0.2106	0.1765	0.1478	0.4582	0.4449	0.4330
	L_2	0.2122	0.1793	0.1471	0.4586	0.4467	0.4363
	χ^2	0.2109	0.1796	0.1478	0.4567	0.4435	0.4421
5	D_2^{*q}	0.1828	0.1599	0.1432	0.4672	0.4609	0.4576
	L_2	0.1798	0.1694	0.1432	0.4688	0.4589	0.4551
	χ^2	0.1797	0.1603	0.1424	0.4666	0.4667	0.4564
6	D_2^{*q}	0.1718	0.2302	0.1606	0.4935	0.5022	0.4899
	L_2	0.1907	0.1909	0.1720	0.4924	0.5039	0.4775
	χ^2	0.2938	0.2031	0.1761	0.4908	0.4923	0.4837
7	D_2^{*q}	0.2781	0.2852	**0.3085**	**0.5751**	0.5770	**0.5925**
	L_2	0.2831	0.2752	-	0.5659	**0.5805**	0.5834
	χ^2	0.2829	**0.2955**	-	0.5668	0.5706	0.5729
8	D_2^{*q}	0.1371	-	-	0.5725	-	-
	L_2	**0.3342**	-	-	0.5446	-	-
	χ^2	0.2878	-	-	0.5742	-	-

Table 24 QCluster: Completeness metric values of non-trivial clusters. Best results are in bold.

		Read length=100			**Read length=700**		
k	**Distance**	***N*=10000**	***N*=20000**	***N*=50000**	***N*=10000**	***N*=20000**	***N*=50000**
		V-Measure			**V-Measure**		
4	D_2^{*q}	0.2068	0.1741	**0.1462**	**0.4490**	0.4364	0.4231
	L_2	**0.2097**	0.1772	0.1456	0.4474	**0.4372**	0.4275
	χ^2	0.2063	**0.1778**	0.1459	0.4462	0.4361	**0.4333**
5	D_2^{*q}	0.1481	0.1358	0.1220	0.4244	0.4281	0.4247
	L_2	0.1474	0.1457	0.1315	0.4312	0.4217	0.4172
	χ^2	0.1386	0.1396	0.1290	0.4155	0.4237	0.4273
6	D_2^{*q}	0.0092	0.1025	0.1129	0.3628	0.3792	0.3869
	L_2	0.1014	0.1092	0.1236	0.3575	0.3879	0.3902
	χ^2	0.0881	0.1172	0.1238	0.3758	0.3856	0.3839
7	D_2^{*q}	0.0833	0.0191	0.1256	0.3181	0.3390	0.3707
	L_2	0.0101	0.0314	-	0.3290	0.3150	0.3522
	χ^2	0.0177	0.1075	-	0.3105	0.3316	0.3496
8	D_2^{*q}	0.0135	-	-	0.3036	-	-
	L_2	0.0073	-	-	0.2142	-	-
	χ^2	0.0110	-	-	0.2198	-	-

Table 25 QCluster: V-Measure metric values of non-trivial clusters. Best results are in bold.

		Read length=100			**Read length=700**		
k	**Distance**	***N*=10000**	***N*=20000**	***N*=50000**	***N*=10000**	***N*=20000**	***N*=50000**
		Adjusted Mutual Information			**Adjusted Mutual Information**		
4	D_2^{*q}	0.1056	0.1232	**0.1263**	**0.3780**	0.4013	0.4095
	L_2	0.1054	0.1256	0.1258	0.3780	**0.4021**	0.4140
	χ^2	**0.1062**	**0.1260**	0.1263	0.3764	0.4005	**0.4199**
5	D_2^{*q}	0.1032	0.1073	0.1089	0.3624	0.3938	0.4104
	L_2	0.0995	0.1162	0.1147	0.3692	0.3872	0.4024
	χ^2	0.1028	0.1098	0.1129	0.3560	0.3897	0.4130
6	D_2^{*q}	0.0042	0.0931	0.0995	0.3042	0.3418	0.3692
	L_2	0.0842	0.0964	0.1106	0.2991	0.3512	0.3733
	χ^2	0.0813	0.1008	0.1101	0.3198	0.3492	0.3663
7	D_2^{*q}	0.0739	0.0112	0.1105	0.2556	0.2964	0.3498
	L_2	0.0032	0.0180	-	0.2642	0.2709	0.3304
	χ^2	0.0080	0.0959	-	0.2500	0.2883	0.3281
8	D_2^{*q}	0.0076	-	-	0.2431	-	-
	L_2	0.0022	-	-	0.1391	-	-
	χ^2	0.0033	-	-	0.1407	-	-

Table 26 QCluster: Adjusted Mutual Information metric values of non-trivial clusters. Best results are in bold.

		Read length=100			Read length=700		
k	Distance	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		Adjusted Rand Index			Adjusted Rand Index		
4	D_2^{*q}	**0.0138**	0.0146	0.0131	**0.0885**	0.0858	0.0853
	L_2	0.0133	**0.0147**	**0.0133**	0.0875	**0.0865**	0.0859
	χ^2	0.0131	0.0146	0.0132	0.0856	0.0855	**0.0910**
5	D_2^{*q}	0.0116	0.0113	0.0113	0.0601	0.0666	0.0714
	L_2	0.0102	0.0127	0.0119	0.0650	0.0657	0.0666
	χ^2	0.0112	0.0121	0.0114	0.0566	0.0673	0.0725
6	D_2^{*q}	0.0000	0.0084	0.0080	0.0291	0.0341	0.0354
	L_2	0.0080	0.0087	0.0097	0.0280	0.0361	0.0418
	χ^2	0.0074	0.0098	0.0093	0.0377	0.0381	0.0372
7	D_2^{*q}	0.0059	0.0000	0.0081	0.0151	0.0167	0.0178
	L_2	0.0000	0.0001	-	0.0156	0.0147	0.0160
	χ^2	0.0000	0.0080	-	0.0150	0.0158	0.0165
8	D_2^{*q}	0.0001	-	-	0.0138	-	-
	L_2	0.0000	-	-	0.0009	-	-
	χ^2	0.0000	-	-	0.0010	-	-

Table 27 QCluster: Adjusted Rand Index metric values of non-trivial clusters. Best results are in bold.

		Read length=100			Read length=700		
k	Distance	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		Purity			Purity		
4	D_2^{*q}	0.0869	0.0773	0.0651	0.2168	0.2016	0.2039
	L_2	0.0795	0.0768	0.0642	0.2175	0.2084	0.1996
	χ^2	0.0856	0.0742	0.0649	0.2154	0.1999	0.2136
5	D_2^{*q}	0.2117	0.1587	0.1514	0.2727	0.2399	0.2382
	L_2	0.1919	0.1535	0.1011	0.2559	0.2488	0.2481
	χ^2	0.2366	0.1436	0.1156	0.2688	0.2730	0.2343
6	D_2^{*q}	0.9768	0.5867	0.3154	0.4592	0.4440	0.3845
	L_2	0.4849	0.4530	0.3109	0.4792	0.4438	0.3704
	χ^2	0.7467	0.4983	0.3196	0.4346	0.4164	0.3696
7	D_2^{*q}	0.7392	**0.9751**	**0.7060**	0.7060	0.6843	0.6528
	L_2	0.9873	0.9553	-	0.6794	**0.7107**	**0.6615**
	χ^2	0.9756	0.7179	-	0.7037	0.6814	0.6562
8	D_2^{*q}	0.9454	-	-	0.7166	-	-
	L_2	**0.9928**	-	-	0.8252	-	-
	χ^2	0.9866	-	-	**0.8393**	-	-

Table 28 QCluster: Purity metric values of non-trivial clusters. Best results are in bold.

		Read length=100			Read length=700		
k	**Window**	***N*=10000**	***N*=20000**	***N*=50000**	***N*=10000**	***N*=20000**	***N*=50000**
		Homogeneity			**Homogeneity**		
4	20	0.2811	0.2542	0.1996	0.0590	0.0383	0.0489
	50	0.3872	0.4118	0.4504	0.0536	0.0451	0.0399
5	20	0.6189	0.6259	0.6678	0.4529	0.4699	0.4908
	50	-	0.0000	0.4783	0.3361	0.3421	0.3352
6	20	0.7736	0.8892	0.9355	0.9816	0.9836	0.9851
	50	-	0.1425	0.3339	0.9260	0.9387	0.9432
7	20	0.8503	0.9445	0.9691	0.9856	0.9881	0.9885
	50	-	-	-	0.9808	0.9838	0.9840
8	20	**0.8874**	0.9438	0.9713	0.9858	**0.9883**	0.9885
	50	-	-	0.0000	0.9847	0.9873	0.9879
9	20	**0.8874**	0.9451	0.9736	0.9859	0.9883	0.9886
	50	-	-	-	0.9858	0.9882	0.9885
10	20	**0.8874**	**0.9533**	0.9757	**0.9860**	0.9883	**0.9886**
	50	-	-	-	0.9857	0.9883	0.9886
11	20	0.8160	0.9464	0.9757	0.9859	0.9882	0.9886
	50	-	-	**1.0000**	0.9857	0.9882	0.9886

Table 29 isONclust: Homogeneity metric values of non-trivial clusters. Best results are in bold.

		Read length=100			Read length=700		
k	**Window**	***N*=10000**	***N*=20000**	***N*=50000**	***N*=10000**	***N*=20000**	***N*=50000**
		Completeness			**Completeness**		
4	20	0.2458	0.2065	0.1585	0.1883	0.1467	0.1146
	50	**1.0000**	0.8392	0.6122	0.1406	0.1009	0.0835
5	20	0.5506	0.4584	0.3978	0.3930	0.3830	0.3816
	50	-	**1.0000**	0.7095	0.3134	0.2954	0.2857
6	20	0.8980	0.7866	0.5709	0.6675	0.6274	0.6124
	50	-	0.3742	0.7237	0.6390	0.5979	0.5815
7	20	0.9097	0.7931	0.5945	0.6700	**0.6321**	0.6164
	50	-	-	-	0.6678	0.6268	0.6113
8	20	0.8700	0.8064	0.5914	0.6697	0.6320	0.6175
	50	-	-	**1.0000**	0.6698	0.6313	0.6160
9	20	0.8700	0.7916	0.5878	0.6691	0.6313	**0.6177**
	50	-	-	-	**0.6706**	0.6314	0.6163
10	20	0.8700	0.8008	0.5838	0.6690	0.6311	0.6174
	50	-	-	-	0.6704	0.6318	0.6164
11	20	0.6798	0.7910	0.5792	0.6694	0.6318	0.6170
	50	-	-	**1.0000**	0.6698	0.6314	0.6160

Table 30 isONclust: Completeness metric values of non-trivial clusters. Best results are in bold.

		Read length=100			Read length=700		
k	Window	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		V-Measure			V-Measure		
4	20	0.2622	0.2279	0.1767	0.0898	0.0607	0.0685
	50	0.5583	0.5525	0.5190	0.0776	0.0623	0.0540
5	20	0.5827	0.5292	0.4986	0.4208	0.4220	0.4293
	50	-	0.0000	0.5714	0.3244	0.3170	0.3084
6	20	0.8312	0.8348	0.7091	0.7946	0.7661	0.7553
	50	-	0.2064	0.4570	0.7562	0.7305	0.7194
7	20	**0.8790**	0.8622	0.7370	0.7977	**0.7710**	0.7593
	50	-	-	-	0.7946	0.7657	0.7541
8	20	0.8786	0.8697	0.7352	0.7976	0.7710	0.7601
	50	-	-	0.0000	0.7973	0.7702	0.7588
9	20	0.8786	0.8615	0.7330	0.7972	0.7705	**0.7603**
	50	-	-	-	**0.7982**	0.7705	0.7593
10	20	0.8786	**0.8704**	0.7305	0.7972	0.7703	0.7601
	50	-	-	-	0.7980	0.7708	0.7594
11	20	0.7417	0.8618	0.7269	0.7974	0.7708	0.7598
	50	-	-	**1.0000**	0.7976	0.7705	0.7590

Table 31 isONclust: V-Measure metric values of non-trivial clusters. Best results are in bold.

		Read length=100			Read length=700		
k	Window	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		Adjusted Mutual Information			Adjusted Mutual Information		
4	20	0.0255	0.0372	0.0509	0.0804	0.0523	0.0647
	50	0.1745	0.0129	0.1126	0.0523	0.0518	0.0480
5	20	0.1268	0.1618	0.2087	0.2351	0.2825	0.3519
	50	-	0.0000	0.1944	0.1505	0.1914	0.2370
6	20	0.7032	0.6553	0.4859	0.6398	0.6418	0.6811
	50	-	-0.1245	0.1519	0.5736	0.5838	0.6303
7	20	0.8395	0.7341	0.5323	0.6459	**0.6509**	0.6875
	50	-	-	-	0.6395	0.6406	0.6793
8	20	**0.8540**	**0.7486**	0.5297	0.6457	0.6507	0.6891
	50	-	-	0.0000	0.6453	0.6494	0.6868
9	20	**0.8540**	0.7366	0.5286	0.6449	0.6496	**0.6895**
	50	-	-	-	**0.6466**	0.6496	0.6874
10	20	**0.8540**	0.7469	0.5268	0.6446	0.6493	0.6890
	50	-	-	-	0.6460	0.6502	0.6875
11	20	0.6897	0.7285	0.5228	0.6453	0.6504	0.6884
	50	-	-	**1.0000**	0.6448	0.6496	0.6869

Table 32 isONclust: Adjusted Mutual Information metric values of non-trivial clusters. Best results are in bold.

		Read length=100			Read length=700		
k	**Window**	***N*=10000**	***N*=20000**	***N*=50000**	***N*=10000**	***N*=20000**	***N*=50000**
		Adjusted Rand Index			**Adjusted Rand Index**		
4	20	0.0027	0.0028	0.0025	0.0065	0.0023	0.0040
	50	0.0950	0.0039	0.0377	0.0027	0.0044	0.0029
5	20	0.0517	0.0332	0.0154	0.0365	0.0341	0.0424
	50	-	0.0000	0.1026	0.0181	0.0221	0.0205
6	20	0.6146	0.4044	0.0659	0.1759	0.1253	0.1090
	50	-	-0.0772	0.1114	0.1577	0.1068	0.0920
7	20	0.7778	0.4312	0.0762	0.1776	**0.1292**	0.1121
	50	-	-	-	0.1761	0.1241	0.1081
8	20	**0.8099**	**0.4723**	0.0732	0.1769	0.1291	0.1133
	50	-	-	0.0000	0.1778	0.1288	0.1119
9	20	**0.8099**	0.4286	0.0697	0.1758	0.1280	**0.1137**
	50	-	-	-	**0.1783**	0.1282	0.1121
10	20	**0.8099**	0.4520	0.0662	0.1755	0.1277	0.1132
	50	-	-	-	0.1780	0.1287	0.1121
11	20	0.5675	0.4058	0.0625	0.1763	0.1287	0.1127
	50	-	-	**1.0000**	0.1768	0.1283	0.1115

Table 33 isONclust: Adjusted Rand Index metric values of non-trivial clusters. Best results are in bold.

		Read length=100			Read length=700		
k	**Window**	***N*=10000**	***N*=20000**	***N*=50000**	***N*=10000**	***N*=20000**	***N*=50000**
		Purity			**Purity**		
4	20	0.0562	0.0435	0.0534	**0.5452**	**0.6503**	**0.3385**
	50	**1.0000**	0.8125	0.4330	0.4550	0.3340	0.2965
5	20	0.1916	0.0936	0.0337	0.1262	0.1028	0.0894
	50	-	**1.0000**	0.6951	0.1091	0.0861	0.0884
6	20	0.8871	0.5381	0.0738	0.1603	0.1131	0.0917
	50	-	0.7500	0.8182	0.1578	0.1056	0.0858
7	20	0.9355	0.5643	0.0812	0.1599	0.1151	0.0926
	50	-	-	-	0.1611	0.1114	0.0907
8	20	0.9200	0.5927	0.0770	0.1583	0.1161	0.0926
	50	-	-	**1.0000**	0.1595	0.1156	0.0927
9	20	0.9200	0.5730	0.0717	0.1573	0.1150	0.0933
	50	-	-	-	0.1600	0.1148	0.0923
10	20	0.9200	0.5507	0.0680	0.1567	0.1143	0.0927
	50	-	-	-	0.1599	0.1147	0.0916
11	20	0.7419	0.5584	0.0652	0.1577	0.1148	0.0925
	50	-	-	**1.0000**	0.1589	0.1152	0.0914

Table 34 isONclust: Purity metric values of non-trivial clusters. Best results are in bold.

		Read length=100			Read length=700		
k	Window	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		Homogeneity			Homogeneity		
4	20	0.33909	0.29802	0.22408	0.05896	0.03829	0.04887
	50	0.78065	0.74063	0.70381	0.05381	0.04519	0.03994
5	20	0.81951	0.80349	0.77415	0.48878	0.47899	0.49165
	50	0.84781	0.84926	0.84970	0.36539	0.35100	0.33639
6	20	0.96694	0.97120	0.96830	0.98463	0.98404	0.98508
	50	0.89401	0.89989	0.91997	0.94185	0.94169	0.94330
7	20	0.98733	0.98783	0.98703	0.98866	0.98846	0.98846
	50	0.94575	0.94742	0.95931	0.98476	0.98415	0.98403
8	20	0.98886	0.98829	0.98787	0.98880	**0.98865**	0.98854
	50	0.92634	0.95552	0.96897	0.98806	0.98765	0.98793
9	20	0.98898	0.98913	0.98868	0.98879	0.98863	0.98857
	50	0.95842	0.96993	0.97703	**0.98883**	0.98851	0.98849
10	20	0.98869	**0.98960**	0.98905	0.98882	0.98863	0.98859
	50	0.98088	0.97907	0.98343	0.98882	0.98859	0.98858
11	20	**0.98899**	0.98960	**0.98913**	0.98881	0.98863	0.98859
	50	0.98088	0.97992	0.98892	0.98881	0.98858	**0.98859**

Table 35 isONclust: Homogeneity metric values of non-singleton clusters. Best results are in bold.

		Read length=100			Read length=700		
k	Window	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		Completeness			Completeness		
4	20	0.27452	0.22931	0.17248	0.18826	0.14666	0.11456
	50	0.76997	0.71636	0.59503	0.14097	0.10108	0.08350
5	20	0.49751	0.45141	0.40406	0.40478	0.38551	0.38172
	50	0.85588	0.81914	0.69588	0.32739	0.29961	0.28614
6	20	0.58735	0.52870	0.48140	0.62986	0.62016	0.61201
	50	0.92836	0.89253	0.76413	0.60014	0.58837	0.58020
7	20	0.60161	0.53728	0.48937	0.63360	0.62509	0.61600
	50	**0.98124**	0.89498	0.78869	0.62888	0.61956	0.61077
8	20	0.60034	0.53634	0.48998	0.63397	**0.62513**	0.61709
	50	0.94372	0.91553	0.80557	0.63316	0.62425	0.61570
9	20	0.59819	0.53589	0.49110	**0.63398**	0.62466	**0.61724**
	50	0.95842	0.91378	**0.81048**	0.63359	0.62458	0.61606
10	20	0.59658	0.53484	0.49200	0.63352	0.62473	0.61699
	50	0.96248	0.93290	0.80896	0.63311	0.62491	0.61614
11	20	0.59486	0.53424	0.49280	0.63346	0.62494	0.61658
	50	0.96248	**0.94548**	0.80660	0.63263	0.62452	0.61568

Table 36 isONclust: Completeness metric values of non-singleton clusters. Best results are in bold.

		Read length=100			Read length=700		
k	**Window**	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		V-Measure			**V-Measure**		
4	20	0.30341	0.25919	0.19492	0.08980	0.06072	0.06851
	50	0.77527	0.72829	0.64486	0.07789	0.06246	0.05403
5	20	0.61915	0.57806	0.53098	0.44283	0.42719	0.42977
	50	0.85183	0.83393	0.76514	0.34535	0.32327	0.30923
6	20	0.73079	0.68468	0.64309	0.76826	0.76083	0.75497
	50	0.91086	0.89619	0.83484	0.73313	0.72423	0.71848
7	20	0.74765	0.69600	0.65433	0.77227	0.76586	0.75900
	50	0.96317	0.92046	0.86567	0.76757	0.76041	0.75372
8	20	0.74711	0.69533	0.65506	0.77259	**0.76594**	0.75985
	50	0.93495	0.93509	0.87975	0.77176	0.76499	0.75861
9	20	0.74548	0.69516	0.65623	**0.77260**	0.76559	**0.75998**
	50	0.95842	0.94102	0.88600	0.77232	0.76549	0.75906
10	20	0.74414	0.69439	0.65712	0.77226	0.76564	0.75979
	50	**0.97159**	0.95543	0.88770	0.77196	0.76576	0.75914
11	20	0.74288	0.69388	0.65785	0.77221	0.76580	0.75948
	50	**0.97159**	**0.96239**	**0.88850**	0.77160	0.76547	0.75879

Table 37 isONclust: V-Measure metric values of non-singleton clusters. Best results are in bold.

		Read length=100			Read length=700		
k	**Window**	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		Adjusted Mutual Information			**Adjusted Mutual Information**		
4	20	0.02716	0.03930	0.05260	0.08036	0.05227	0.06466
	50	0.07389	0.05416	0.09994	0.05229	0.05179	0.04799
5	20	0.13734	0.16388	0.20227	0.24009	0.28274	0.35172
	50	0.22764	0.24224	0.24873	0.15504	0.19265	0.23699
6	20	0.30746	0.30211	0.33395	0.58089	0.63073	0.68060
	50	0.57291	0.51506	0.41949	0.51327	0.56859	0.62840
7	20	0.34115	0.32371	0.35212	0.58934	0.64027	0.68696
	50	0.84071	0.63531	0.52406	0.57846	0.62951	0.67859
8	20	0.34145	0.32486	0.35622	0.59021	**0.64033**	0.68861
	50	0.72961	0.69243	0.56962	0.58829	0.63861	0.68641
9	20	0.34118	0.32867	0.36245	**0.59023**	0.63943	**0.68885**
	50	0.83871	0.73577	0.58281	0.58919	0.63923	0.68698
10	20	0.34115	0.32961	0.36795	0.58919	0.63958	0.68847
	50	**0.89186**	0.79443	0.59068	0.58805	0.63982	0.68709
11	20	0.34320	0.33189	0.37283	0.58902	0.63996	0.68784
	50	0.89186	**0.81958**	**0.60326**	0.58686	0.63901	0.68639

Table 38 isONclust: Adjusted Mutual Information metric values of non-singleton clusters. Best results are in bold.

		Read length=100			Read length=700		
k	Window	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		Adjusted Rand Index			Adjusted Rand Index		
4	20	0.00270	0.00278	0.00243	0.00650	0.00229	0.00404
	50	0.05524	0.03150	0.03612	0.00274	0.00436	0.00286
5	20	0.01755	0.01201	0.00796	0.03556	0.03370	0.04227
	50	0.18780	0.18888	0.12897	0.01813	0.02189	0.02050
6	20	0.04958	0.02478	0.01364	0.12435	0.11755	0.10873
	50	0.54742	0.40190	0.25308	0.10285	0.09643	0.09101
7	20	0.05704	0.02691	0.01445	0.12734	0.12167	0.11178
	50	**0.84071**	0.51558	0.32106	0.12250	0.11650	0.10772
8	20	0.05591	0.02656	0.01461	**0.12761**	**0.12173**	0.11295
	50	0.67729	0.58131	0.35333	0.12713	0.12117	0.11165
9	20	0.05455	0.02657	0.01498	0.12756	0.12089	**0.11330**
	50	0.77612	0.61309	0.36095	0.12721	0.12096	0.11191
10	20	0.05398	0.02626	0.01531	0.12686	0.12086	0.11290
	50	0.83188	0.67853	0.36119	0.12643	0.12130	0.11188
11	20	0.05351	0.02616	0.01563	0.12697	0.12131	0.11238
	50	0.83188	**0.72021**	**0.36399**	0.12566	0.12090	0.11123

Table 39 isONclust: Adjusted Rand Index metric values of non-singleton clusters. Best results are in bold.

		Read length=100			Read length=700		
k	Window	N=10000	N=20000	N=50000	N=10000	N=20000	N=50000
		Purity			Purity		
4	20	0.04982	0.04000	0.05131	**0.54520**	**0.65035**	**0.33846**
	50	0.45588	0.35329	0.19803	0.45490	0.33390	0.29651
5	20	0.05157	0.03134	0.01910	0.11713	0.10084	0.08927
	50	0.69000	0.56021	0.30999	0.10346	0.08481	0.08826
6	20	0.08131	0.04099	0.02134	0.12412	0.10867	0.09151
	50	0.86047	0.75200	0.42975	0.11416	0.09904	0.08528
7	20	0.08575	0.04247	0.02148	0.12486	0.11086	0.09246
	50	**0.96429**	0.75214	0.46000	0.12246	0.10714	0.09050
8	20	0.08301	0.04131	0.02144	0.12407	0.11195	0.09245
	50	0.89286	0.79348	**0.48899**	0.12422	0.11131	0.09253
9	20	0.08079	0.04079	0.02129	0.12397	0.11100	0.09311
	50	0.92308	0.78409	0.48837	0.12457	0.11067	0.09215
10	20	0.07959	0.04045	0.02154	0.12301	0.11050	0.09255
	50	0.92857	0.81818	0.47939	0.12407	0.11063	0.09144
11	20	0.07924	0.03983	0.02192	0.12343	0.11066	0.09231
	50	0.92857	**0.85556**	0.47826	0.12311	0.11100	0.09127

Table 40 isONclust: Purity metric values of non-singleton clusters. Best results are in bold.

```
>ENSG00000004776|ENSG00000004776.13|ENST00000004982|ENST00000004982.6
ACTGCAACGCGGAGGAGCAGGATGGAGATCCCTGTGCCTGTGCAGCCGTCTTGGCTGCGC
CGCGCCTCGGCCCCGTTGCCCGGACTTTCGGCGCCCGGACGCCTCTTTGACCAGCGCTTC
GGCGAGGGGCTGCTGGAGGCCGAGCTGGCTGCGCTCTGCCCCACCACGCTCGCCCCCTAC
```

Listing 1 Excerpt of the first four lines of the FASTA file that encodes the human genome assembly (GRCh38.p13) used for our experiments.

```
@m0/70/CCS Read=0;...;chromosome=ENSG00000244754|ENSG00000244754.9|...
GCAAAACAAGCTATCGATCAGGGAAGATCTCCAGTTATAATCTAGATAACACTAATATACAAGCTTGGGA
+
3333333333333333333333333333333333333333335555555555555555555555555555
```

Listing 2 Excerpt of the first four lines of a FASTQ file simulated with SimLoRD with 70bp-long sequences. The header has been truncated for typographic reasons.

D File formats

The downloaded dataset, the simulated datasets, and the clustering software's results are represented in different file formats: FASTA and FASTQ for representing long-read sequences, SAM for describing sequence alignments, TSV and CSV for describing the cluster results. We will now give a brief description and example of each file format's feature relevant to this work.

FASTA

The FASTA format is a text-based format used for representing nucleotide sequences. Nucleic acid bases are encoding using a single letter: **A** (adenosine), **C** (cytidine), **G** (guanine), and **T** (thymidine). Sequences in FASTA files begin with a single-line header distinguished by a "`>`" symbol at the beginning of the line. The header contains a sequence identification code and is optionally followed by a textual description of the sequence. A FASTA file may comprise more than a single sequence.

FASTQ

FASTQ is a format similar to FASTA that also indicates the quality scores related to the sequencing process. The sequence identifier and the quality scores are both encoded in ASCII. A FASTQ file uses four lines per sequence.

1. Line 1 begins with a "`@`" symbol followed by a sequence identifier and an optional description, similarly to a FASTA file header.
2. Line 2 contains a nucleotide sequence encoded using the same letter as the FASTA format.
3. Line 3 starts with a "`+`" symbol and is optionally followed by a description of the sequence.
4. Line 4 encodes the quality values for the sequence in line 2. It must contain the same number of symbols as letters in the nucleotide sequence.

```
@SQ SN:ENSG00000255036|ENSG00000255036.6|ENST00000529787|ENST00000529787.2  LN:5586
@SQ SN:ENSG00000169247|ENSG00000169247.14|ENST00000502274|ENST00000502274.2 LN:5474
m0/100/CCS Read=0;length=100bp;...;chromosome=ENSG00000244754|ENSG00000244754.9|...
m1/100/CCS Read=1;length=100bp;...;chromosome=ENSG00000169855|ENSG00000169855.2|...
```

Listing 3 Excerpt of four lines of a SAM file simulated with SimLoRD with 100bp-long sequences. The lines have been truncated for typographic reasons.

```
m0/100/CCS  3
m1/100/CCS  0
m2/100/CCS  51
m3/100/CCS  0
```

Listing 4 Example of the first four rows of a `inferred_clusters.tsv` file generated by QCluster. The first column represents a simulated read identifier; the second column indicates the cluster ID. For instance, we can see that the reads `m1/100/CCS` and `m3/100/CCS` are grouped in the same cluster.

```
0 m22455/700/CCS_Read=22455;...;chromosome=ENSG00000085982|ENSG00000085982.13...
0 m27791/700/CCS_Read=27791;...;chromosome=ENSG00000085982|ENSG00000085982.13...
0 m41495/700/CCS_Read=41495;...;chromosome=ENSG00000274311|ENSG00000274311.4...
0 m48098/700/CCS_Read=48098;...;chromosome=ENSG00000101040|ENSG00000101040.19...
```

Listing 5 Example of the first 4 rows of a `final_clusters.tsv` file generated by isONclust. The first column indicates the cluster ID of a read, whereas the second column represents the simulated read header. The read header has been truncated for typographic reasons. isONclust's output is thus slightly more informative than QCluster's and presented in a different order.

SAM

The Sequence Alignment/Map (SAM) is a file format to save alignment information of short reads mapped against reference sequences. It usually starts with a header section followed by alignment information as tab-separated lines for each read. Header lines start with the "`@`" symbol.

CSV

The comma-separated values (CSV) format is a textual representation of spreadsheets, i.e. tabular data in plain text. Each line of the file is a data record. An optional header line appears as the first line of the file; this header contains names corresponding to the fields in the file. Each line contains the same number of fields throughout the file, separated by a comma ("`,`") symbol. A comma must not follow the last field of a record.
We used CSV files to save the quality metrics of the examined clustering results.

TSV

The tab-separated values (TSV) format is analogous to CSV, but the tab character is used instead of he comma for separating record fields.
The TSV format is used in the output of both isONclust and QCluster.

```
homogeneity,completeness,v_measure,...,adjusted_rand_index,k
0.9840325904960752,0.6106160508656244,0.7536031182627078,...,0.10763226661511985,1849
```

Listing 6 Compact example of the CSV generated by the Python script that computes the metrics of the clustering results. It contains the quality metrics and the number of clusters determined by the clustering algorithm.

E Scripts Execution

The following paragraphs describe how to reproduce our experiments. We assume that the reader has downloaded our project repository, has installed Docker, and has downloaded the human genome as described in Section 4.

Processessing Sequences

The steps to preprocess the human cDNA sequences as described in Section 4.1 are:

- Create the `clustering/preprocess` Docker image with the command:

```
docker build -t clustering/preprocess \
    -f Dockerfile.preprocess ./python/preprocess
```

- Run the `clustering/preprocess` Docker image:

```
./scripts/preprocess.sh
```

  The `Python` module in `python/preprocess` will be run under the hood. The file `preprocessed.fasta` in `data/preprocess` will be overwritten.

Simulating Datasets

The steps to create the simulated datasets as described in Section 4.2 are:

- Create the `clustering/simlord` Docker image with the command:

```
docker build -t clustering/simlord -f Dockerfile.simlord .empty
```

- Run the `clustering/simlord` Docker image:

```
./scripts/simulate.sh
```

  The `Python` module in `python/simulate` will be run under the hood. The six dataset folders in `data/simulated` will be overwritten.

Perform Baseline Clustering

The steps to perform baseline clustering on all datasets as described in Section 5.2 are:

- Create the `clustering/random_cluster` Docker image with the command:

```
docker build -t clustering/random_cluster \
    -f Dockerfile.random_cluster ./python/random_cluster
```

- Run the `clustering/random_cluster` Docker image:

```
./scripts/random_cluster.sh
```

  The `Python` module in `python/random_cluster` will be run under the hood. The `data/random_cluster` folder will be overwritten.

Perform Clustering with QCluster

The steps to perform clustering with QCluster on all datasets as described in Section 5.3 are:

- Create the `clustering/qcluster` Docker image with the command:

```
docker build -t clustering/qcluster -f Dockerfile.qcluster .empty
```

- Run the `clustering/qcluster` Docker image:

```
./scripts/qCluster.sh
```

  The `Python` module in `python/qCluster` will be run under the hood. The `data/qCluster` folder will be overwritten.

Perform Clustering with isONclust

The steps to perform clustering with isONclust on all datasets as described in Section 5.4 are:

- Create the `clustering/isonclust` Docker image with the command:

```
docker build -t clustering/isonclust \
    -f Dockerfile.isonclust ./third-party
```

- Run the `clustering/qcluster` Docker image:

```
./scripts/isONclust.sh
```

  The `Python` module in `python/isonclust` will be run under the hood. The `data/isONclust` folder will be overwritten.

Compute Clustering Metrics and Statistics

The steps to compute the clustering metrics and statistics on all clustering results as described in Section 5.5 are:

- Create the `clustering/quality` Docker image with the command:

```
docker build -t clustering/quality \
    -f Dockerfile.quality ./python/quality
```

- Run the `clustering/quality` Docker image:

```
./scripts/quality.sh
```

  The `Python` module in `python/quality` will be run under the hood. The `data/quality` folder will be overwritten.

Visualize Clustering Quality

To visualize the clustering quality plots generated for this report, please refer to the `analysis.ipynb` `Python` notebook.