Journal of Bioinformatics and Computational Biology

Transformation of reads sets into feature vectors for unsupervised compression of sequence data

Tao Tang
Advanced Analytics Institute, Faculty of Engineering and IT, University of Technology Sydney
Broadway, NSW 2007, Australia
13183963@student.uts.edu.au

Jinyan Li *
Advanced Analytics Institute, Faculty of Engineering and IT, University of Technology Sydney
Broadway, NSW 2007, Australia
jinyan.li@uts.edu.au

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

FASTQ data sets of short reads are usually generated in tens or hundreds for a biomedical study. However, current compression of these data sets is carried out one-by-one without consideration of the inter-similarity between the data sets which can be otherwise exploited to enhance compression performance of de novo compression. We show that clustering these data sets into similar sub-groups for a group-by-group compression can greatly improve the compression performance. Our novel idea is to detect the lexicographically smallest k-mer (k-minimizer) for every read in each data set, and uses these k-mers as features and their frequencies in every data set as feature values to transform these huge data sets each into a characteristic feature vector. Unsupervised clustering algorithms are then applied to these vectors to find similar data sets and merge them. As the amount of common k-mers of similar feature values between two data sets implies an excessive proportion of overlapping reads shared between the two data sets, merging similar data sets creates immense sequence redundancy to boost the compression performance. Experiments confirm that our clustering approach can gain up to 12% improvement over several state-of-the-art algorithms in compressing reads databases consisting of 17 to 100 data sets (48.57 - 197.97 GB).

Keywords: Sequence reads; k-minimizer; Compression; Clustering.

1. Introduction

The growth rate of genomic short reads data from high-throughput Next Generation Sequencing (NGS) platforms [21,26,22] has outpaced Moore's law, resulting in petabytes of whole genome sequencing or RNA-sequencing data stored in major

*corresponding author

databases. For example, NCBI Sequence Read Archive maintained over 14 petabytes NGS data as of Aug 2020. The raw sequencing data are normally stored in the FASTQ format [8] as a set of reads, where each read consists of an identifier (header), a DNA string of fixed length and a list of quality scores indicating the confidence of correct base-calling for each nucleotide base in the string. For instance, one study [5] sequenced 769 individuals of Dutch ancestry and produced 4574 NGS reads data sets, each of which contains around 50 million reads, having a total file size of 27.74TB in the FASTQ format. A simplified FASTQ format is the FASTA format whereby only the DNA strings are stored excluding the quality scores. The vast quantity of reads data has proposed great challenges to data analysis, storage and transmission; efficient data compression techniques are required to address all of these issues.

Intensive research has focused on reference-free (de-novo) compression of a single reads set. The key idea is to utilize the redundancy information between the reads in the same set to form groups of reads and de-novo assemble them into consensus sequences (contigs), and then encode the reads with regard to these consensus sequences. Mince [23] buckets similar reads together and reorder the reads in each bucket, it also handles paired-end reads by concatenating the left and right ends of the pair together in accordance with a user-provided library type. HARC [7] searches the maximum overlaps of the reads in the set to locate their approximate positions in a genome and then creates contigs from reads in the neighbourhood of these positions in the genome. SPRING [6] is an advanced version from the same team of authors, improving HARC with options such as compressing reads with arbitrarily long read lengths. FaStore [25] and Minicom [19] both group reads based on the redundancy information contained in sub-string patterns called k-minimizers [24]; the reads in these groups are then de-novo assembled or condensed into contigs for referential encoding. A more recently published method PgRC [17] divides the whole set of reads into two groups: high-quality (without 'N' base) and low-quality (with 'N' bases), then generates an approximate shortest common superstring over each group via merging reads with suffix-prefix overlapping to the existing string until no reads are left, which is referred as pseudo genome construction [15]. PgRC 1.2 [16] is a multi-thread version of PgRC with improvements in compression speed. Earlier reference-free algorithms such as ReCoil [28], SCALCE [12], Quip [14], Fqzcomp [4], ORCOM [11] and Assembltrie [10] also achieved high compression ratio on single FASTQ file.

Algorithms for compressing large collections (databases) of FASTQ files are still lacking. Although the one-by-one compression approach is straightforwardly applicable, optimizing de-novo compression algorithms for the compressing multiple FASTQ files to achieve better performance is challenging. Suppose there are n number of reads data sets to compress. An easy question is whether we can concatenate the n data sets together as a single large data set for compression. A difficulty is the extremely high usage of main memory, with no guarantee of compression performance improvement. A challenging question is how to convert a large reads data

set into a characteristic feature vector and how to cluster these n vectors to find small groups of similar data sets to merge for compression.

Intuitively in theory, merging similar data sets for compression is potentially useful for performance improvement. Denote a reads set as Y, the assembled consensus sequence(s) from Y as $\overline{Y}$. The key idea of reference-free compression is merging these reads into consensus sequence(s) and record each read with regard to the consensus sequence(s). Denote the information of reads set Y after compression as $I(Y), I(Y) = I(\overline{Y}) + I(Y|\overline{Y})$. When Y is split into two sets Y_1 and Y_2. Then $I(Y_1) + I(Y_2) = I(\overline{Y_1}) + I(Y_1|\overline{Y_1}) + I(\overline{Y_2}) + I(Y_2|\overline{Y_2})$. Under the ideal condition (optimal solution for consensus sequences construction and no correction for mismatching bases), the consensus sequence(s) $\overline{Y_1}, \overline{Y_2} \subseteq \overline{Y}, I(\overline{Y}) = I(\overline{Y_1}) + I(\overline{Y_2}) - I(\overline{Y_1} \cap \overline{Y_2}) \leq I(\overline{Y_1}) + I(\overline{Y_2})$. $I(Y|\overline{Y})$ can be considered as the information of $|Y|$ records, where each record contains the position of one read in $\overline{Y}$ (assume no mismatching and all reads have the same length), with an encoding method such as Lempel-Ziv parsing [29], the average record length of each compression can be considered as constant, hence $I(Y|\overline{Y}) \approx I(Y_1|\overline{Y_1}) + I(Y_2|\overline{Y_2})$, $I(Y) \approx I(Y_1) + I(Y_2) - I(\overline{Y_1} \cap \overline{Y_2}) \leq I(Y_1) + I(Y_2)$. Compressing two sets together will reduce the compressed size in comparison with compressing them one-by-one separately, and the amount of reduced size should have a positive relationship with the size of the intersection set of the consensus sequences generated from the two sets.

Multiple methods have been proposed to search the matched substring between sequence data [18]. For reference-free compression algorithms, the focus is on utilizing the similarity consensus and sequence redundancy information among the reads, The main idea is to search the overlapping sub-strings in the pairs of reads through a 'seed-extension' strategy, in which a short common subsequence between two reads are used as the 'seed', then the 'seed' is extended, although there are some other ideas of taking suffix-prefix overlapping (e.g. PgRC), or some without the explicit extending process (e.g. minicom). Another effective search strategy is based on the concept of minimizers [24]. A minimizer of a string is the lexicographically smallest k-mer of the string, which is widely used in processing of reads data [20]. It is a useful concept to identify sequence similarity — it has been previously proved [24] that two reads are likely to have a large overlap if they share a k-minimizer.

Hence for a high-performance compression of multiple reads sets, we should increase the occurrence of k-minimizer redundancy and should combine only those reads sets of more similar reads together.

With these foundations, we propose to detect the k-minimizer substring for every read in each data set, and use these k-minimizers as features and their frequencies in the data set as feature values to transform each data set into a feature vector. Unsupervised clustering algorithms are then applied to these vectors to find similar data sets and merge them. Table 1 is an example to illustrate the high similarity between two sequences (reads) when they share the same minimizer.

A real example (Table 2) of the feature vector is presented below (only the first 10 elements of the feature vector displayed). The vector is transformed from a huge

Two sequences	G	T	A	C	T	G	A	T	T	G	C	A	C	T	G	A
3-mers and 3-minimizers (in bold)	G	T	A						T	G	C					
		T	A	C						G	C	A				
			A	**C**	**T**						C	A	C			
				C	T	G						**A**	**C**	**T**		
					T	G	A						C	T	G	
						A	G	T						T	G	A

Table 1: Two sequences that share the same minimizer

IDs of 8-minimizers	0	1	2	3	4	5	6	7	8	9
Normalized frequency	0.56	0.43	0.61	0.79	0.57	0.15	0.28	0.34	0.74	0.42

Table 2: First 10 elements of the feature vector transformed from a reads set

reads set named ERR532390, which contains 10831122 reads of length 100 having a total file size 4.27GB. Each feature value in the vector is the percentage of reads in this data set containing the same specific minimizer. For instance, feature values 0.56 and 0.43 suggest that there are 0.56% of the reads in ERR532390 containing minimizer AAAAAAAA (feature ID 0) and 0.43% of the reads containing minimizer AAAAAAAC (feature ID 1), where k=8.

If the amount of common k-mers with similar feature values between two data sets is big, then the two data sets should be merged to hold an excessive proportion of reads containing the same minimizer sub-strings to create immense sequence redundancy for boosting compression performance. Driven by this principle, we propose a novel clustering method (named MRC — **M**ultiple **R**eads Sets **C**lustering) for compressing reads sets databases. MRC constructs feature vectors based on the minimizer frequencies of each reads set, then applies K-means clustering recursively to divide the reads sets into subgroups with higher inter-similarity than the original reads sets. Then the compression of these clustered reads sets is carried out group-by-group instead of one-by-one separately. We evaluate MRC on various benchmark databases to demonstrate that the method improves the compression ratio as compared to the original algorithms in all cases.

2. Method

Let $R = \{R_1, R_2...R_n\}$ be a collection (database) of reads sets, $R_i = \{r_{i1}, r_{i2}..r_{ij}\}$, where r_{ij} denotes a single read sequence, $r_{ij}[l] \in$\{A,C,G,T,N\} for $l = 1, 2, \ldots, L$ where L is the length of the reads.

The proposed algorithm MRC consists of three stages: transformation of a reads set into a feature vector, clustering of the feature vectors, and group-by-group compression.

(1) **Transformation of a reads set into a feature vector**: A feature space with n feature vectors is constructed, each of them is transformed from a reads set R_i using the minimizers as features and the minimizers' frequency as feature values.
(2) **Clustering of the feature vectors**: A variant of K-means clustering, that is, a K-means clustering with a limit on cluster size is applied to the feature space to divide the original set R into subgroups.
(3) **Group-by-group compression**: The reads sets in each subgroup are compressed together by state-of-the-art compression algorithms with additional information recorded for lossless decompression.

2.1. *Transformation of a reads set into a feature vector*

For a reads set R_i, the reads are classified into two types: the reads without 'N' and the others, during transformation, only reads without 'N' is considered. The set of reads without 'N' from R_i is denoted as $\underline{R_i} = \{\underline{r_{i1}}, \underline{r_{i2}}..\}, \underline{r_{ij}}[l] \in \{A, C, G, T\}$. Via an encoding function, each k-mer (subsequence s with length k) in $\underline{r_{ij}}$ can be mapped to a $2 \times k$ bit integers, such as $f(s) = \sum_{a=1}^{a=k} \phi(s[a] \cdot 4^{a-1}), \phi(A) = 0, \phi(C) = 1, \phi(G) = 2, \phi(T) = 3$.

In de novo compression, the overlap search between reads normally focuses on finding one suitable overlapping for merging and encoding (to reduce computation time) instead of searching all possible overlaps among reads. To fit the procedure of de novo compression, we extract one representative k-mer from each read.

The lexicographically smallest k-mer (minimizer) of each read is selected as the representative k-mer for the read, and count the frequency of each unique k-mer in the reads set to estimate the similarity between reads sets. Here we use a normalized frequency as feature value. For a reads set R_i, the corresponding feature vector $V_i = \{v_{i1}, v_{i2}..v_{i4^k}\}, v_{ij} = \frac{\text{frequency of minimizer with value}_j \text{ in } \underline{R_i}}{|\underline{R_i}|}$, the value of minimizer s is computed using $f(s) = \sum_{a=1}^{a=k} \phi(s[a]) \cdot 4^{k-a}, \phi(A) = 0, \phi(C) = 1, \phi(G) = 2, \phi(T) = 3$. In order to reduce storage usage, only the frequency of minimizers that appears at least one time are recorded in implementation. Table 3 is an example of the minimizers and corresponding feature vector of a set of 10 reads with k=2.

To reduce processing time and avoid potential problems such as curse of dimensionality [3] or overfitting [27], m features are selected from the original 4^k: the variance of each feature $\{v_{1j}, v_{2j}...v_{nj}\}$ is computed, then m features with highest variance are selected. Finally, a set of m-dimensional n vectors are generated, denoted by $\underline{V} = \{\underline{V}_1, \underline{V}_2...\underline{V}_n\}$.

Sequence	Minimizer(s)	Mapped value $f(s)$
G T C G C C G A	CC	5
A T T G T C G G	AT	3
G A C G A A T C	AA	0
C T G T T G C A	CA	4
A G G T C C G C	AG	2
G T C A A G C A	AA	0
G G T C T C T A	CT	7
C G A G T T C G	AG	2
T G T T G T G C	GC	9
A C C T A T C G	AC	1

Example of minimizers in a reads set with k=2 and the corresponding values $f(s)$.

Minimizer	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC
Frequency	2	1	2	1	1	1	0	1	0	1

Frequency of each minimizer in above reads set.

Feature	AA	CA	GA	TA	AC	CC	GC	TC	AG	CG	GG	TG	AT	CT	GT	TT
Value	0.2	0.1	0.2	0	0.1	0.1	0	0.2	0	0.1	0	0	0	0	0	0

Normalized feature vector of the above reads set.

Table 3: Example of the minimizers and corresponding feature vector of a set of 10 reads with k=2

2.2. *Clustering of the feature vectors*

Clustering is the process of grouping a set of points (feature vectors) into a number of subgroups such that similar points are placed in the same subgroup. Here our clustering is to ensure higher similarity in each subgroup that are to be compressed together to achieve better performance of compression.

A critical part of this stage is the size of a cluster: As mentioned before, the reads sets in each cluster will be compressed as a single file by state-of-the-art algorithms, however, the complexity of most algorithms is larger than $O(|R_i|)$, which causes sharply increasing compression time with respect to the number of reads. Hence the total size of each cluster should be limited to control compression time. We apply a variant of K-means clustering to ensure the efficiency and practicability for the method.

Variant of K-means Clustering

K-means is a method that divides n points into K clusters in which each point belongs to the cluster with the nearest centroid [13]. The main difference between our clustering process and the standard K-means is as follows:

(1) The K-means clustering is applied on reads sets recursively with K set as 2 until the size of all clusters is less than or equal 3.
(2) When a cluster with size 1 is generated, one object from another cluster will be tried to added to this size 1 cluster, the criteria is shown in Algorithm 1.

The clustering process is shown as Algorithm 1, the standard version of K-means is shown in supplementary A1.

Algorithm 1 Optimized K-means clustering

Input: vector set $\underline{V} = \{\underline{V}_1, \underline{V}_2 ... \underline{V}_n\}$, threshold λ, cluster set C;
$C \leftarrow \emptyset$;

procedure CLUSTERING($\underline{V}$)
    if $|\underline{V}| \leq 3$ **then**
        Add $\underline{V}$ to C;
    else
        $c_1, c_2 \leftarrow$ result of K-means on $\underline{V}$ with K=2
        if $|c_1| + |c_2| \geq 4$ AND ($|c_1| == 1$ OR $|c_2| == 1$) **then**
            $a \leftarrow$ id of cluster with size 1, $b \leftarrow$ id of the other cluster;
            $c_b[min] \leftarrow$ the vector in c_b with minimum distance to
        $c_a[0]$;
            if distance($c_a[0], c_b[min]$) $\leq \lambda \cdot$ minimum distance
        between $c_b[min]$ and other vectors in c_b **then**
                add $c_b[min]$ to c_a;
                remove $c_b[min]$ from c_b;
            Clustering(c_1);
            Clustering(c_2);
Clustering($\underline{V}$);
return C;

2.3. *Group-by-group compression*

For each cluster $C_i \in C$, C_{ij} represents the jth vector in C_i, which corresponds to one specific reads set in R.

The corresponding reads sets of each C_i is merged together in order, denote the array of reads number of reads sets in each cluster by $L = L[1...l]$, the $\sum_{a=1}^{i-1} L[a]$th to $\sum_{a=1}^{i-1} L[a+1]$th reads in the merged set are the reads of the ith reads set in cluster. The merged set is then compressed under the order preserving mode, the name and read number of each reads set are recorded. The decompression process is the reverse process of the compression. First, the compressed file of each merged set is decompressed, then each reads set is restored using the recorded read number.

Cluster	Reads set	Feature vector
A	1	0.299 0.104 0.007 0.035 0.049 0.032 0.009 0.025 0.023 0.108
	2	0.300 0.105 0.007 0.035 0.051 0.031 0.009 0.025 0.024 0.109
	3	0.297 0.105 0.007 0.034 0.048 0.031 0.007 0.025 0.023 0.109
B	1	0.148 0.067 0.021 0.063 0.070 0.051 0.021 0.046 0.044 0.102
	2	0.140 0.063 0.022 0.072 0.069 0.051 0.021 0.047 0.045 0.101

Table 4: Example of the first 10 features of feature vectors from two clusters in the database of 21 reads sets

3. Result and Performance Analysis

Compression experiments were conducted at a computing cluster running Red Hat Enterprise Linux 6.7 (64 bit) with 2 Intel Xeon E5-2695 processors(2.3GHz,14 Cores), 128 GB of RAM. Four state-of-the-art compression algorithms, FaStore [25], SPRING [6], minicom [19] and PgRC1.2 [16], were tested on four collections of reads sets to understand the performance improvement achieved by our clustering approach in comparison with compressing the set separately via the same compression algorithm. PgRC1.2 and minicom focus on compression of the sequence in reads data, FaStore and SPRING retain all data in FASTQ files. All algorithms were executed with the order preserving mode, other parameters follow the default setting (e.g. 24 threads for minicom, 8 threads for PgRC1.2).

Details about the collections of reads data sets are shown in Table 5. The original data were downloaded from http://ftp.sra.ebi.ac.uk/vol1/fastq/. The URLs of each file are listed in Supplementary Data.

3.1. *Compression performance gains*

To measure the performance improvement brought by MRC, we define performance *gain* as:

$$gain = \left(1 - \frac{\text{Compressed size with MRC}}{\text{Compressed size of one-by-one compression}}\right) \times 100\%$$

A higher value of *gain* indicates more compression improvement.

Table 6 shows the compression results by minicom and PgRC1.2 using the 'one-by-one' approach or the 'MRC' approach. The best compression algorithm PgRC1.2 can compress the 48.57 gigabytes of database-17 into 1.01 gigabytes, achieving 48.09 folds of compression ratio, when the straightforward one-by-one approach is taken. The compression ratio is improved to 54.86 folds when the clustering approach MRC is used. In fact, PgRC1.2 achieves an average performance gain of 7.69% on the four

Table 5: Four collections of reads data sets

Collections	number of reads sets	total size (in GB)	length of reads
database-17	17	48.57	101
database-21	21	135.68	100
database-50	50	71.04	50
database-100	100	197.97	36

More information of each file can be found in Supplementary Data

Table 6: Compressed file size (in byte) comparison between the straightforward *one-by-one* approach and our MRC approach (with $k = 7$ for the setting of k-minimizer)

Collection	minicom			PgRC1.2		
	one-by-one	with MRC	*gain*	one-by-one	with MRC	*gain*
database-17	1184071680	1053224960	11.05%	1087246442	950620160	12.57%
database-21	6074501120	5800316800	4.47%	6263079421	5722808320	8.63%
database-50	3714529280	3548344320	4.41%	3006844741	2880102400	4.22%
database-100	9593702400	9485285120	1.13%	8828937571	8355092480	5.37%

Collection	SPRING			FaStore		
	straightforward	with MRC	*gain*	straightforward	with MRC	*gain*
database-17	8628849779	8484075520	1.68%	9666709566	9437614080	2.37%
database-21	23029063680	22549555200	2.08%	25241707218	24566077440	2.68%
database-50	13646202880	13519677440	0.93%	15492111069	15372031320	0.78%
database-100	28880547840	28573671680	1.06%	23964221072	23478440608	2.03%

databases when MRC is used. Specifically, the gains are respectively 8.63%, 4.22%, 5.37%, and 12.57%, revealing a positive correlation with the average file size in each database (database-17: 2.86GB, database-21: 6.46GB, database-50: 1.42GB, database-100: 1.97GB). For minicom, the improvement is 11.05%, 4.51%, 4.47%, 1.13% for the four databases and 5.29% on average.

The improvements on SPRING and FaStore are lower than the other two algorithms, which varies from 0.78% to 2.68% on different databases, the main reason is that SPRING and FaStore also compress the identifier and quality score (i.e., compressing the FASTQ format files), as the compression performance of quality scores is mainly dependent on the rate of smoothed quality scores, the increase in the overlapping redundancy between reads provided by MRC does not contribute

to the compression. How to convert *quality scores* in a reads set into a characteristic feature vector is a topic for investigation.

Overall, as shown in Table 6, our proposed MRC method outperforms over compressing reads sets separately in all cases.

3.2. *3D visualization for the clusters of reads data sets after transformation*

By the feature extraction and selection of MRC, a reads set is converted into a vector with m dimensions. To assess the effectiveness of our clustering method, we apply Multi-Dimensional Scaling [9] on the original feature spaces of database-17, database-21, database-50, database-100 (see Figure 1 a, c, e, g), in comparison with the feature vectors constructed by MRC (the cluster labels marked in Figure 1 b, d, f, h). It can be seen that the clusters from MRC matches the distance space computed by Multi-Dimensional Scaling, suggesting that MRC is effective for selecting subgroups of high similarity for compression of multiple reads sets. Full scale version of Figure 1 are shown in supplementary file.

3.3. *PCA analysis on the feature vectors converted from the reads data sets*

To get deeper insights into the roles of the minimizers in the clustering, Principal Component Analysis [1] is conducted on the feature vectors converted from each database to compare the importance of each feature. Principal Component Analysis (PCA) is a method to reduce the dimension of data by maximizing the variance of each dimension [2]. By PCA, the original data (with dimension $n \times m$, each column f_j is a feature vector, $j \in [1, m]$) is transformed to a $n \times p$ matrix where $p \leq min(m, n)$, each column vector corresponds to one component $c_i = (c_{i1}, c_{i2}...c_{im}), i \in [1, p]$, c_i represents the direction of the i-th maximum variance of the original data, the absolute value of c_{ij} represents the magnitude of vector projection of feature f_j onto c_i, which can be considered as the importance of feature j for c_i. Each c_i explains an amount of the variance of original matrix, the percentage of variance explained by c_i is denoted as vr_i.

The new components are efficient for data analysis but not interpretable enough. In order to understand which features are more important for clustering, we measure the importance of feature f_j through $w_j = \sum_{a=1}^{p} |c_{aj}| \times vr_a, j \in [1, m]$. The features with the highest importance for each database and the corresponding minimizers are listed in Table 7. Interestingly, some databases share common features. For example, the features with mapped IDs 0, 2, 15, 25, 27, and 38 occurred in more than one database, which can be considered as representative minimizers of these databases worth of further investigation. Our PCA analysis results are also displayed in Figure 2.

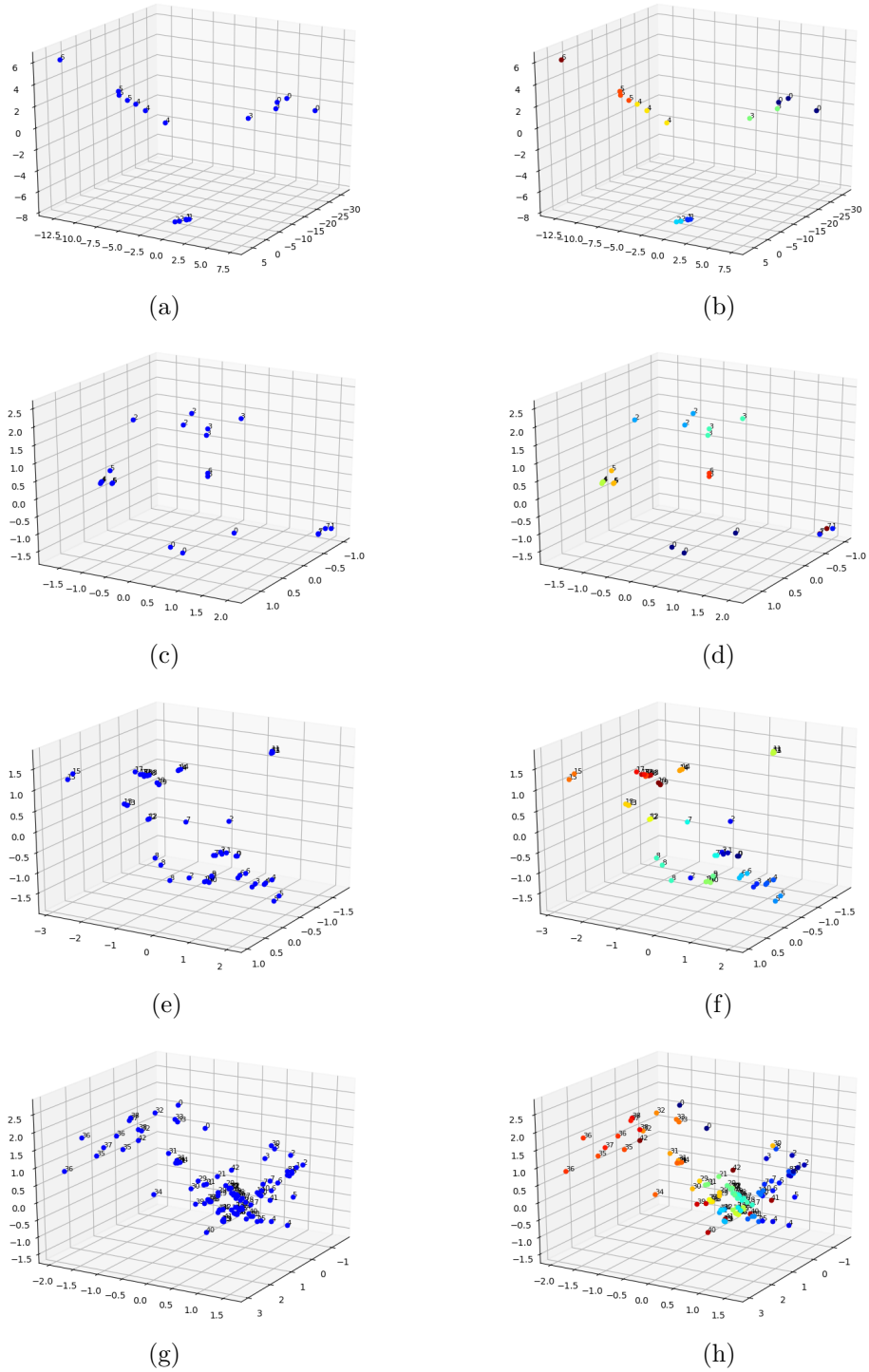


Fig. 1: 3D Visualisation of the feature space after Multi-Dimensional Scaling from (a) database-17, (b) database-17 by clustering of MRC; (c) database-21, (d) database-21 by clustering of MRC; (e) database-50, (f) database-50 by clustering of MRC; (g) database-100, (h) database-100 by clustering of MRC

database-21			database-50			database-100		
v	w	minimizer	v	w	minimizer	v	w	minimizer
12	0.725	AAAAATA	15	0.725	AAAAATT	1	0.627	AAAAAAC
6	0.723	AAAAACG	22	0.694	AAAACCG	27	0.626	AAAACGT
4	0.710	AAAAACA	25	0.650	AAAACGC	25	0.581	AAAACGC
29	0.708	AAAACTC	27	0.646	AAAACGT	2	0.577	AAAAAAG
2	0.697	AAAAAAG	0	0.621	AAAAAAA	39	0.572	AAAAGCT
17	0.694	AAAACAC	38	0.614	AAAAGCG	13	0.563	AAAAATC
15	0.665	AAAAATT	24	0.607	AAAACGA	26	0.558	AAAACGG
18	0.664	AAAAAGT	11	0.601	AAAAAGT	38	0.552	AAAAGCG
0	0.657	AAAAAAA	37	0.599	AAAAGCC	20	0.551	AAAACCA
13	0.655	AAAAATC	21	0.577	AAAACCC	41	0.541	AAAAGGC

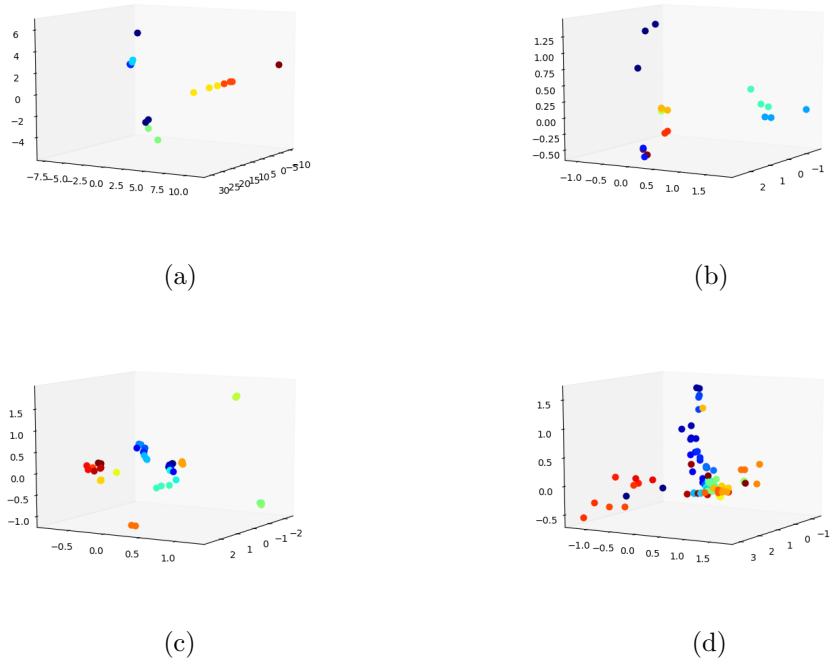
v indicates the mapped ID of a feature, w represents an importance score of a feature, a higher score indicates a more important feature.

Table 7: Minimizers of high importance in our PCA analysis

3.4. *Time complexity and speed performance*

Running time is an essential factor for measuring the applicability of an algorithm. For MRC, the time complexity of the compression depends on the compression algorithm. Here we present the time complexity for the transformation and feature vector construction. Let L and N_R denote the read length and total number of reads in a collection of reads set R. The computation of the minimizers of the reads takes $O(N_R(L-k+1))$ time, where k is the length of the minimizers. The computation of the variance of each feature and sorting based on the value of variance takes $O(m_f n + m_f \log m_f)$ time, where m_f is the number of minimizers that appears at least one time in the datasets. In the worst case, $m_f = 4^k$, the time complexity is $O(4^k n + 2k4^k \log 2)$. The standard K-means clustering takes $O(tKn_e d)$, where t is the number of iterations, n_e is the number of vectors, d is the dimension of vectors. In the worst case (assuming K-means splits the original data into two clusters with size 1 and $n_e - 1$ each time), the clustering stage will apply $n-3$ K-means with $K = 2, d = m, n_e = n, n-1..., 4$, the time complexity is $O(2\sum_{n_e=4}^{n_e=n-3} td) = O((n+1)(n-3)tm)$. As $N_R L$ and 4^k is much larger than $n^2 m$ in normal cases, the running time of stages 1 and 2 are mainly related on N_R, L and k. When k is fixed, the running time has a positive linear relationship with $N_R L$, which is the total size of database. The actual speed of our method on different databases fits the analysis.

To achieve the performance of our algorithm in the compression of multiple reads sets, we test the running time of MRC on the four databases. As mentioned before, the running time of MRC can be divided into two parts: transformation+clustering (only depending on the input databases) and compression (depending on both input set and compression algorithm).



(a) (b)

(c) (d)

Fig. 2: 3D Visualisation of the feature space after Principal Component Analysis from (a) database-17 (b) database-21; (c) database-50 (d) database-100

Stage	database-17	database-21	database-50	database-100
Transformation+Clustering	149.54	492.98	321.16	850.63
Compression (by minicom)	1866.94	18682.44	5051.15	8786.49
Compression (by PgRC1.2)	1203.26	7543.40	4168.48	8565.72
Compression(by SPRING)	1936.84	5283.13	2325.19	11207.65
Compression(by FaStore)	2963.19	18870.07	15058.386	50333.904

Table 8: Transformation and clustering time of MRC (in second), red indicates the shortest compression time of each dataset.

MRC took 149.54, 492.98, 321.16, 850.63 seconds on the transformation and clustering stage for database-17, database-21, database-50 and database-100 respectively (Table 8), while the shortest compression time taken by the four algorithms on these databases are 1203.26, 5283.13, 2325.19, 8565.72 seconds. The transformation and clustering time is much shorter than the compression time.

4. Conclusion

In this work, we introduced MRC, a clustering-based algorithm for the compression of reads databases. The key idea of this method is the measurement of the similarity between reads sets based on minimizer frequency and the application of a variant K-means clustering to group the reads sets into subsets with a high similarity between each other. Reads sets in each cluster are compressed together with additional information recorded to enable decompressing separately.

This algorithm is applicable for reads sets with the same read length. We have demonstrated that MRC achieved a substantial improvement in all cases compared to the original state-of-the-art compression algorithms. In addition, analysis of the time complexity on real databases shows that the time complexity of our algorithm is linear in practice, the time of additional preprocessing and clustering stage are much less than the compression time.

Acknowledgements

We thank Dr. Hui Peng, Dr. Chaowang Lan and Mrs. Xuan Zhang of the University of Technology Sydney (UTS) for expertise used in genome sequence dataset.
This research work was partly supported by the Australia Research Council Discovery Project (ARC DP) 180100120.

Supplementary information

Supplementary information is available for this paper at here.

References

1. Abdi H, Williams LJ, Principal component analysis, *Wiley interdisciplinary reviews: computational statistics* **2**(4):433–459, 2010.
2. Alpaydin E, *Introduction to machine learning*, MIT press, 2020.
3. Bellman R, Dynamic programming, *Science* **153**(3731):34–37, 1966.
4. Bonfield JK, Mahoney MV, Compression of fastq and sam format sequencing data, *PloS one* **8**(3):e59190, 2013.
5. Boomsma DI, Wijmenga C, Slagboom EP, Swertz MA, Karssen LC, Abdellaoui A, Ye K, Guryev V, Vermaat M, Van Dijk F, *et al.*, The genome of the netherlands: design, and project goals, *European Journal of Human Genetics* **22**(2):221–227, 2014.
6. Chandak S, Tatwawadi K, Ochoa I, Hernaez M, Weissman T, Spring: a next-generation compressor for fastq data, *Bioinformatics* **35**(15):2674–2676, 2019.
7. Chandak S, Tatwawadi K, Weissman T, Compression of genomic sequencing reads via hash-based reordering: algorithm and analysis, *Bioinformatics* **34**(4):558–567, 2018.
8. Cock PJ, Fields CJ, Goto N, Heuer ML, Rice PM, The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants, *Nucleic acids research* **38**(6):1767–1771, 2010.
9. Cox MA, Cox TF, Multidimensional scaling, in *Handbook of data visualization*, , Springer, pp. 315–347, 2008.
10. Ginart AA, Hui J, Zhu K, Numanagić I, Courtade TA, Sahinalp SC, David NT, Op-

timal compressed representation of high throughput sequence data via light assembly, *Nature communications* **9**(1):1–9, 2018.
11. Grabowski S, Deorowicz S, Roguski Ł, Disk-based compression of data from genome sequencing, *Bioinformatics* **31**(9):1389–1395, 2015.
12. Hach F, Numanagić I, Alkan C, Sahinalp SC, Scalce: boosting sequence compression algorithms using locally consistent encoding, *Bioinformatics* **28**(23):3051–3057, 2012.
13. Jain AK, Data clustering: 50 years beyond k-means, *Pattern recognition letters* **31**(8):651–666, 2010.
14. Jones DC, Ruzzo WL, Peng X, Katze MG, Compression of next-generation sequencing reads aided by highly efficient de novo assembly, *Nucleic acids research* **40**(22):e171–e171, 2012.
15. Kowalski T, Grabowski S, Deorowicz S, Indexing arbitrary-length k-mers in sequencing reads, *PloS one* **10**(7):e0133198, 2015.
16. Kowalski T, Grabowski SP, Engineering the compression of sequencing reads, *bioRxiv* , 2020.
17. Kowalski TM, Grabowski S, Pgrc: pseudogenome-based read compressor, *Bioinformatics* **36**(7):2082–2089, 2020.
18. Liu Y, Wong L, Li J, Allowing mutations in maximal matches boosts genome compression performance, *Bioinformatics* , 2020. doi:10.1093/bioinformatics/btaa572, URL `https://doi.org/10.1093/bioinformatics/btaa572`, btaa572.
19. Liu Y, Yu Z, Dinger ME, Li J, Index suffix–prefix overlaps by (w, k)-minimizer to generate long contigs for reads compression, *Bioinformatics* **35**(12):2066–2074, 2019.
20. Liu Y, Zhang X, Zou Q, Zeng X, Minirmd: accurate and fast duplicate removal tool for short reads via multiple minimizers, *Bioinformatics* , 2020. doi:10.1093/bioinformatics/btaa915, btaa915.
21. Margulies M, Egholm M, Altman WE, Attiya S, Bader JS, Bemben LA, Berka J, Braverman MS, Chen YJ, Chen Z, *et al.*, Genome sequencing in microfabricated high-density picolitre reactors, *Nature* **437**(7057):376–380, 2005.
22. Metzker ML, Sequencing technologies—the next generation, *Nature reviews genetics* **11**(1):31–46, 2010.
23. Patro R, Kingsford C, Data-dependent bucketing improves reference-free compression of sequencing reads, *Bioinformatics* **31**(17):2770–2777, 2015.
24. Roberts M, Hayes W, Hunt BR, Mount SM, Yorke JA, Reducing storage requirements for biological sequence comparison, *Bioinformatics* **20**(18):3363–3369, 2004.
25. Roguski Ł, Ochoa I, Hernaez M, Deorowicz S, Fastore: a space-saving solution for raw sequencing data, *Bioinformatics* **34**(16):2748–2756, 2018.
26. Shendure J, Ji H, Next-generation dna sequencing, *Nature biotechnology* **26**(10):1135–1145, 2008.
27. Tetko IV, Livingstone DJ, Luik AI, Neural network studies. 1. comparison of overfitting and overtraining, *Journal of chemical information and computer sciences* **35**(5):826–833, 1995.
28. Yanovsky V, Recoil-an algorithm for compression of extremely large datasets of dna data, *Algorithms for Molecular Biology* **6**(1):23, 2011.
29. Ziv J, Lempel A, A universal algorithm for sequential data compression, *IEEE Transactions on information theory* **23**(3):337–343, 1977.